REAL TIME DROWSINESS DETECTION USING CASCADE OBJECT DETECTION AND YOLO ALGORITHM

A PROJECT REPORT

Submitted by

G. ESWARAN	**712220104010**
RAJNISH KUMAR	**712220104030**
R. SWETHA	**712220104050**
T. SWETHA	**712220104051**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PARK COLLEGE OF ENGINEERING AND TECHNOLOGY

KANIYUR, COIMBATORE-641659

ANNA UNIVERSITY :: CHENNAI 600025

MAY 2024

BONAFIDE CERTIFICATE

Certified that this project report **"REAL TIME DROWSINESS DETECTION USING CASCADE OBJECT DETECTION AND YOLO ALGORITHM"** is the Bonafide work of **"G. ESWARAN (712220104010), RAJNISH KUMAR (712220104030), R. SWETHA (712220104050), T. SWETHA (712220104051)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. V. Saranya,M.Tech,Ph.D .,	Mrs.S.Princy M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor	Assistant Professor
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
Park College of Engineering and Technology, Kaniyur, Coimbatore - 641659	Park College of Engineering and Technology, Kaniyur, Coimbatore - 641659

Submitted for the University Project VIVA - VOCE examination held on _____

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

First and foremost, we praise and thank God and our parents, from the bottom of our heart for bestowing his benediction on us throughout the course of this project.

We wish to express our sincere thanks to our beloved Chairman **Dr. P. V. RAVI Ph.D., MISTE**, for providing an opportunity to work on this project.

We would like to express our deep sense of gratitude to the Chief Executive Officer **Dr. ANUSHA.R B.E., M.S.(USA), Ph.D.,** for her valuable support and encouragement.

We extend sincere thanks to our Principal **Dr. D. LAKSHMANAN, M.E., Ph.D,** for supporting and encouraging to carry out our project.

We are highly indebted and graceful to our beloved Head of the Department **Dr. V SARANYA, M.Tech, Ph.D.,** for her valuable and her constant suggestions and persistent encouragement.

We express our gracious gratitude to our beloved guide **Mrs. S. PRINCY M.E.,** for her guidance and support throughout our project.

We also thank all the faculty members, teaching and non-teaching staff and lab in charge of our department who helped us to complete our project successfully.

TABLEOFCONTENT

ABSTRACT

Driver drowsiness has currently been a severe issue threatening road safety, hence it is vital to develop an effective drowsiness recognition algorithm to avoid traffic accidents. However, recognizing drowsiness is still very challenging, due to the large intra-class variations in facial expression, head pose and illumination condition. In this paper, a new deep learning framework based on the hybrid Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving. A real-time driver drowsiness detection and traffic analysis system for driving safety. Based on computer vision techniques, the driver's face is located from a color video captured in a car. Then, face detection is employed to locate the regions of the driver's eyes, which are used as the templates for eye tracking in subsequent frames. Finally, the tracked eye's images are used for drowsiness detection in order to generate warning alarms. Also Camera capture image in front of road, and analysis road traffics.

CHAPTER 1

INTRODUCTION

Most of the road accidents are caused because of drowsiness and drunk driving and also working environments, reduced sleep and time factor. Driver drowsiness and fatigue drunk driving reduces the driver decision making capability and perception level. These two situations affect the ability to control the vehicle. There are some techniques which are used to detect drowsiness in drivers like by sensing of driver operation or physiological characteristics of driver like or vehicle movement etc.

A new approach towards automobile safety and security with autonomous region primarily based automatic automotive system is projected during this conception. We have a tendency to propose 3 distinct however closely connected ideas viz. Drowsy Driver Detection system and a traffic detection system with external vehicle intrusion dodging primarily based conception.

In recent time's automobile fatigue connected crashes have very enlarged. so as to attenuate these problems, we've incorporated driver alert system by watching each the driver's eyes still as sensing still because the driver state of affairs based primarily based native setting recognition based AI system is projected.

1.1 DRIVER DROWSINESS ACCIDENTS

Driver drowsiness is a significant cause of road accidents worldwide. When a driver becomes drowsy, their attention, reaction time, and decision-making abilities are impaired, which increases the risk of accidents. According to the National Highway Traffic Safety Administration (NHTSA), drowsy driving is responsible for an estimated 100,000 crashes, 71,000 injuries, and 1,550 deaths annually in the United States alone. Drowsy driving is particularly common among long-haul truck drivers, shift workers, and people with untreated sleep disorders such as sleep apnea. The risk of drowsy driving also increases during the night, especially between midnight and 6 a.m., and during mid-afternoon hours when people may experience a natural dip in alertness. To address this issue, various driver drowsiness detection systems have been developed, including physiological, behavioral, and vision-based approaches. These systems use sensors, cameras, and machine learning algorithms to monitor the driver's state and alert them if they appear to be becoming drowsy. Overall, the development and deployment of driver drowsiness detection systems have the potential to significantly reduce the number of accidents caused by drowsy driving. However, more research is needed to optimize these systems and make them more accessible to all drivers, especially those who are most at risk.

Fig.No.-1 : DRIVER DROWSINESS

CHAPTER 2

LITERATURE SURVEY

2.1 TITLE: Impact of the face angle to traveling trajectory during the riding standing-type personal mobility device

AUTHOR: Jeyeon Kim, Kenta Sato, Naohisa Hashimoto

YEAR: 2019

DESCRIPTION:

We investigate the impact of face direction during traveling by Standing-Type Personal Mobility Device (PMD). The use of PMD devices has been a popular choice for recreational activities in the developed countries such as in the USA and the countries in Europe. These devices are not completely risk free and various accidents have been reported. Since that, the risk factors leading to accidents have to be investigated. Unfortunately, the research studies on the risk factors on riding PMD devices have not been matured as much as the studies on driving cars. We evaluate the impacts of face angle on travelling trajectory during travelling in a PMD. We showed by experiments that, the face direction is an important factor in risk assessment for traveling by a PMD. In addition, it showed that the deviation of the running locus occurred in the direction opposite to the direction the subject was closely watching. These results showed how distracted driving is dangerous and causes the risks of undeliberate trajectory changing.

2.2 TITLE: Using real-life alert-based data to analyse drowsiness and distraction of commercial drivers

AUTHOR: SaraFerreira, ZafeirisKokkinogenis, AntónioCouto

YEAR: 2019

DESCRIPTION:

Professional drivers are particularly exposed to drowsiness and distraction inasmuch as they drive for long periods of time and as a daily routine. Therefore, several studies have been conducted to investigate drivers´ behavior, supported by controlled experiments (e.g. naturalistic and driving simulator studies). However, due to emerging technologies, new study methods can be developed to complement existing studies. In this study, retrospective data gathered from a driver monitoring system (DMS), which monitored 70 professional drivers from different companies, was used to investigate the effect of journey characteristics on the number of alerts due to either distraction or drowsiness. Two separate negative binomial models were developed, including explanatory variables describing the continuous driving time (sub-journey time), the journey time (a set of sub-journeys), the number of breaks and the breaking duration time. Dummy variables were also included. Interesting results were observed such as increasing continuous driving time, the number of distraction and drowsiness alerts increase too.

2.3 TITLE: Smartphones as an integrated platform for monitoring driver behavior: The role of sensor fusion and connectivity

AUTHOR: Stratis Kanarachosa, Stavros-Richard G. Christopoulosa, Alexander Chroneosa

YEAR: 2020

DESCRIPTION:

Nowadays, more than half of the world's web traffic comes from mobile phones, and by 2020 approximately 70 percent of the world's population will be using smartphones. The unprecedented market penetration of smartphones combined with the connectivity and embedded sensing capability of smartphones is an enabler for the large-scale deployment of Intelligent Transportation Systems (ITS). On the downside, smartphones have inherent limitations such as relatively limited energy capacity, processing power, and accuracy. These shortcomings may potentially limit their role as an integrated platform for monitoring driver behaviour in the context of ITS. This study examines this hypothesis by reviewing recent scientific contributions. The Cybernetics theoretical framework was employed to allow a systematic comparison. First, only a few studies consider the smartphone as an integrated platform. Second, a lack of consistency between the approaches and metrics used in the literature is noted.

2.4 TITLE: Sentio: Driver-in-the-Loop Forward Collision Warning Using Multisample Reinforcement Learning

AUTHOR: Salma Elmalaki, Huey-Ru Tsai

YEAR: 2020

DESCRIPTION:

On one side, a common thread in ADAS applications is to focus entirely on the context of the vehicle and its surrounding vehicles leaving the human (driver) context out of consideration. On the other side, and due to the increasing sensing capabilities in mobile phones and wearable technologies, monitoring complex human context becomes feasible which paves the way to develop driver-in-the-loop context-aware ADAS that provide personalized driving experience. In this paper, we propose Sentio1; a Reinforcement Learning based algorithm to enhance the Forward Collision Warning (FCW) system leading to Driver-in-the-Loop FCW system. Since the human driving preference is unknown a priori, varies between different drivers, and moreover, varies across time for the same driver, the proposed Sentio algorithm needs to take into account all these variabilities which are not handled by the standard reinforcement learning algorithms. We verified the proposed algorithm against several human drivers.

2.5 TITLE: A Novel Model-Based Driving Behavior Recognition System Using Motion Sensors.

AUTHOR: Wu M, Zhang S, Dong Y.

YEAR: 2019

DESCRIPTION:

A novel driving behavior recognition system based on a specific physical model and motion sensory data is developed to promote traffic safety. Based on the theory of rigid body kinematics, we build a specific physical model to reveal the data change rule during the vehicle moving process. In this work, we adopt a nine-axis motion sensor including a three-axis accelerometer, a three-axis gyroscope and a three-axis magnetometer, and apply a Kalman filter for noise elimination and an adaptive time window for data extraction. Based on the feature extraction guided by the built physical model, various classifiers are accomplished to recognize different driving behaviors. Leveraging the system, normal driving behaviors (such as accelerating, braking, and lane changing and turning with caution) and aggressive driving behaviors (such as accelerating, braking, lane changing and turning with a sudden) can be classified with a high accuracy of 93.25%. Compared with traditional driving behavior recognition methods using machine learning only, the proposed system possesses a solid theoretical basis, performs better and has good prospects.

2.6 TITLE: Driver Drowsiness Monitoring System using Visual Behavior and Machine Learning

AUTHOR: Ashish Kumar, Rusha Patra

YEAR: 2019

DESCRIPTION:

Drowsy driving is one of the major causes of road accidents and death. Hence, detection of driver's fatigue and its indication is an active research area. Most of the conventional methods are either vehicle based, or behavioral based or physiological based. Few methods are intrusive and distract the driver, some require expensive sensors and data handling. Therefore, in this study, a low cost, real time driver's drowsiness detection system is developed with acceptable accuracy. In the developed system, a webcam records the video and driver's face is detected in each frame employing image processing techniques. Facial landmarks on the detected face are pointed and subsequently the eye aspect ratio, mouth opening ratio and nose length ratio are computed and depending on their values, drowsiness is detected based on developed adaptive thresholding. Machine learning algorithms have been implemented as well in an offline manner. A sensitivity of 95.58% and specificity of 100% has been achieved in Support Vector Machine based classification.

2.7 TITLE: Real Time Driver Drowsiness Detection Based on Driver's Face Image Behavior Using a System of Human Computer Interaction Implemented in a Smartphone

AUTHOR: Eddie E. Galarza, Fabricio D. Egas, Franklin M. Silva, Paola M. Velasco, Eddie D. Galarza

YEAR: 2020

DESCRIPTION:

The main reason for motor vehicular accidents is the driver drowsiness. This work shows a surveillance system developed to detect and alert the vehicle driver about the presence of drowsiness. It is used a smartphone like small computer with a mobile application using Android operating system to implement the Human Computer Interaction System. For the detection of drowsiness, the most relevant visual indicators that reflect the driver's condition are the behavior of the eyes, the lateral and frontal assent of the head and the yawn. The system works adequately under natural lighting conditions and no matter the use of driver accessories like glasses, hearing aids or a cap. Due to a large number of traffic accidents when driver has fallen asleep this proposal was developed in order to prevent them by providing a non-invasive system, easy to use and without the necessity of purchasing specialized devices. The method gets 93.37% of drowsiness detections.

2.8 TITLE: Estimation of the Driving Style Based on the Users' Activity and Environment Influence

AUTHOR: Mikhail Sysoev, Andrej Kos, Jože Guna, Matevž Pogačnik

YEAR: 2020

DESCRIPTION:

New models and methods have been designed to predict the influence of the user's environment and activity information to the driving style in standard automotive environments. For these purposes, an experiment was conducted providing two types of analysis: (i) the evaluation of a self-assessment of the driving style; (ii) the prediction of aggressive driving style based on drivers' activity and environment parameters. Sixty seven h of driving data from 10 drivers were collected for analysis in this study. The new parameters used in the experiment are the car door opening and closing manner, which were applied to improve the prediction accuracy. An Android application called *Sensoric* was developed to collect low-level smartphone data about the users' activity. The driving style was predicted from the user's environment and activity data collected before driving. The prediction was tested against the actual driving style, calculated from objective driving data. The prediction has shown encouraging results, with precision values ranging from 0.727 up to 0.909 for aggressive driving recognition rate.

2.9 TITLE: Drowsiness Detection and Alert System: A Review

AUTHOR: Jyotsna Gabhane, Dhanashri Dixit, Pranali Mankar, Ruchika Kamble, Sayantani Gupta

YEAR: 2019

DESCRIPTION:

Truck drivers, company car drivers and shift workers are the most at risk of falling asleep while driving. Majority of the accidents occur due to the drunkenness of the driver. The burden of which lies on the company owner as they are made liable. It can lead to economic loss. In this presentation we present an adaptive driver and company owner alert system and an application that provides driving behavior to the company owner. One of the major cause of traffic accident is Driver‘s drowsiness. It is a serious highway safety problem. If drivers could be warned before they became too drowsy to drive safely, some of these crashes could be prevented. In order to reliably detect the drowsiness, it depends on the presentation of timely warnings of drowsiness. To date, the effectiveness of drowsiness detection methods has been limited by their failure to consider individual differences. Based on the type of data used, drowsiness detection can be conveniently separated into the two categories of intrusive and non-intrusive methods.

2.10 TITLE: On Developing a Driver Identification Methodology Using In-Vehicle Data Recorders

AUTHOR: Luis Moreira-Matias

YEAR: 2020

DESCRIPTION:

Recently, cutting edge technologies to facilitate data collection have emerged on a large scale. One of the most prominent is the in-vehicle data recorder (IVDR). There are multiple ways to assign the IVDR's data to the different drivers who share the same vehicle. Irrespective of the level of sophistication, all of these technologies still suffer considerable limitations in their accuracy. The purpose of this paper is to propose a methodology, which can identify the driver of a given trip using historical trip based data. To do so, an advanced machine learning pipeline is proposed. The main goal is to take advantage of highly available data—such as driver-labeled floating car data collected by an IVDR—to build a pattern-based algorithm able to identify the trip's driver category when its true identity is unknown. This stepwise process includes feature generation/selection, multiple heterogeneous explanatory models, and an ensemble approach (i.e., stacked generalization) to reduce their generalization error.

CHAPTER 3

SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

In the existing approaches, the cascaded objection algorithm has been utilized in order to extract the face and eyes in the Given input.Driver drowsiness detection is a technique used to identify if a driver is experiencing drowsiness or fatigue while driving.The VGG (Visual Geometry Group) Network ResNet (Residual Network) Inception Network DenseNet (Densely Connected Convolutional Network) MobileNet (Mobile Network).Drowsiness is a state of reduced alertness and impaired cognitive function that can lead to accidents, especially when driving. Eye joints are used to find the state of eye. Based on joints Drowsiness is defined.

3.1.1. DISADVANTAGE

- Need to improve the performance of the overall system.
- Accuracy of the models is need to be improved.
- The operation time is more.

3.2 PROPOSED SYSTEM

In this proposed system Face recognition presents a challenging problem in the field of image analysis and computer vision.Face recognition is a biometric system used to identify or verify a person from a digital image. Face Recognition system is widely used in security. Face recognition system should be able to automatically detect a face in an image. This involves extracting the features from the image and then recognize the face, regardless of lighting, expression, illumination, ageing, transformations (translate, rotate and scale image) and posture, which is a difficult task.

The proposed system for driver drowsiness detection is a deep CNN models-based ensemble approach. The system consists of four deep CNN models, each trained on a different subset of the data. The training data consists of a large dataset of driver drowsiness videos.In the proposed method, eye state is detected by cascade object detection method. The detected features are processed through machine learning models. Once train the model, model will be stored for model deployment purpose. The proposed system for driver drowsiness detection is a deep CNN models-based ensemble approach that aims to improve the accuracy of driver drowsiness detection using multiple models and a weighted average approach.The accuracy of the model is calculated by using sklearn python package.

3.2.1 ADVANTAGE

- Detection time is low
- More accuracy.
- Python is environment friendly, hence easy to implememented in any device.
- Multiple work executed at same time
- Better services to passengers.
- Driver or conductor no need to shout always

BLOCK DIAGRAM

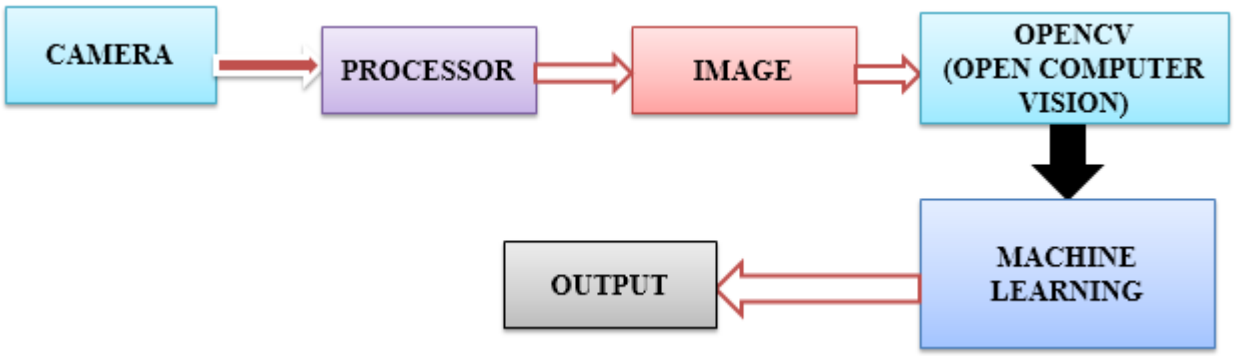

Fig.No. - 3.1 : Proposed System

Processor:

Use a high-performance processor to handle the real-time image processing and analysis required for drowsiness detection.

Image Processing with OpenCV:

Use OpenCV functions to crop and normalize facial regions, ensuring consistent input for the deep learning models.

Machine Learning Process:

Divide the dataset into training, validation, and test sets. Train the deep YOLO models on the training set, fine-tuning them for the drowsiness detection task. Validate the models on the validation set and adjust hyper parameters to avoid overfitting. Evaluate the ensemble's performance on the test set to assess its ability to generalize to new data.

Output:

Integrate the drowsiness detection system with an alert mechanism, such as sound alarms, haptic feedback, or visual warnings, to notify the driver in real-time. Ensure a seamless and immediate response to prevent potential accidents caused by driver drowsiness.

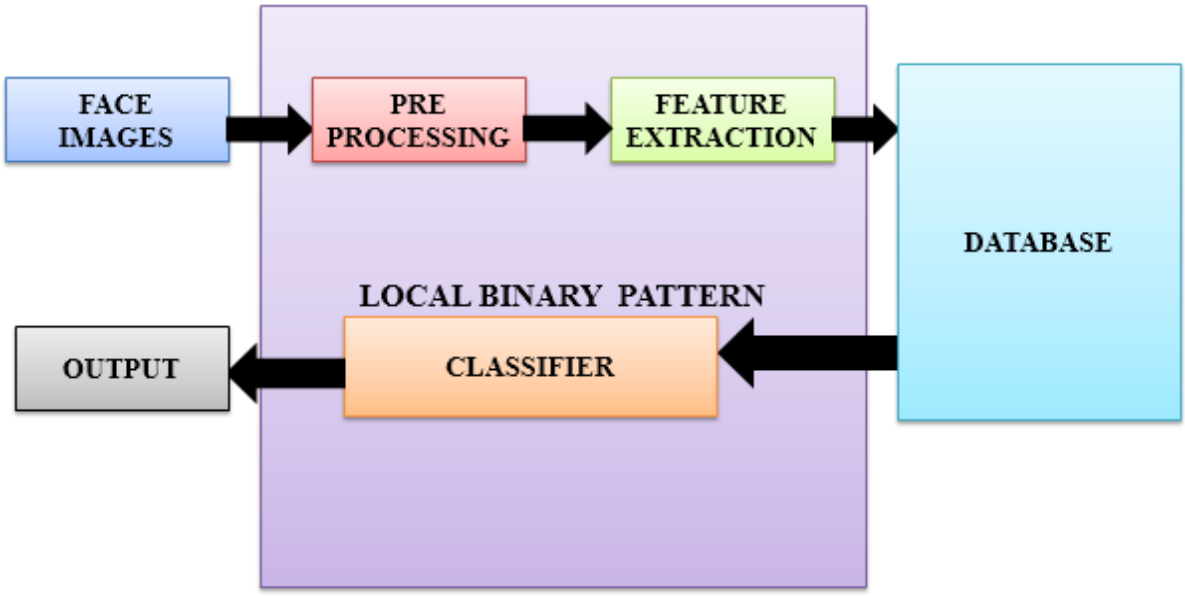

Fig.No .- 3.2 : Machine Learning Process

EXPLANATION:

Input Image:

The process begins with capturing real-time images or video frames of the driver's face using a camera placed within the vehicle.

Pre-processing:

Use a pre-trained face detection algorithm to locate and extract the face region from the input image.Identify facial landmarks, such as eyesto provide spatial context for feature extraction.Adjust the brightness, contrast, and scale of the facial region to ensure consistent input for the subsequent stages.

Feature Extraction:

The pre-trained models can capture hierarchical features, learning high-level representations of facial expressions, eye movements, and other relevant features.Extract features that are indicative of both alert and drowsy states.

Database:

Maintain a database containing labeled examples of drowsy and alert states for training and validating the models.Ensure the dataset is diverse and representative of real-world scenarios to enhance the generalization capabilities of the models.

Classification:

Use a binary classification model to predict whether the driver is drowsy or alert based on the learned features.Implement an ensemble approach to combine predictions from multiple models for improved accuracy and robustness.

Output Explanation:

If the model predicts drowsiness, initiate an alert system.Provide a confidence score or probability to convey the certainty of the prediction.Establish a threshold for decision-making based on the confidence score.

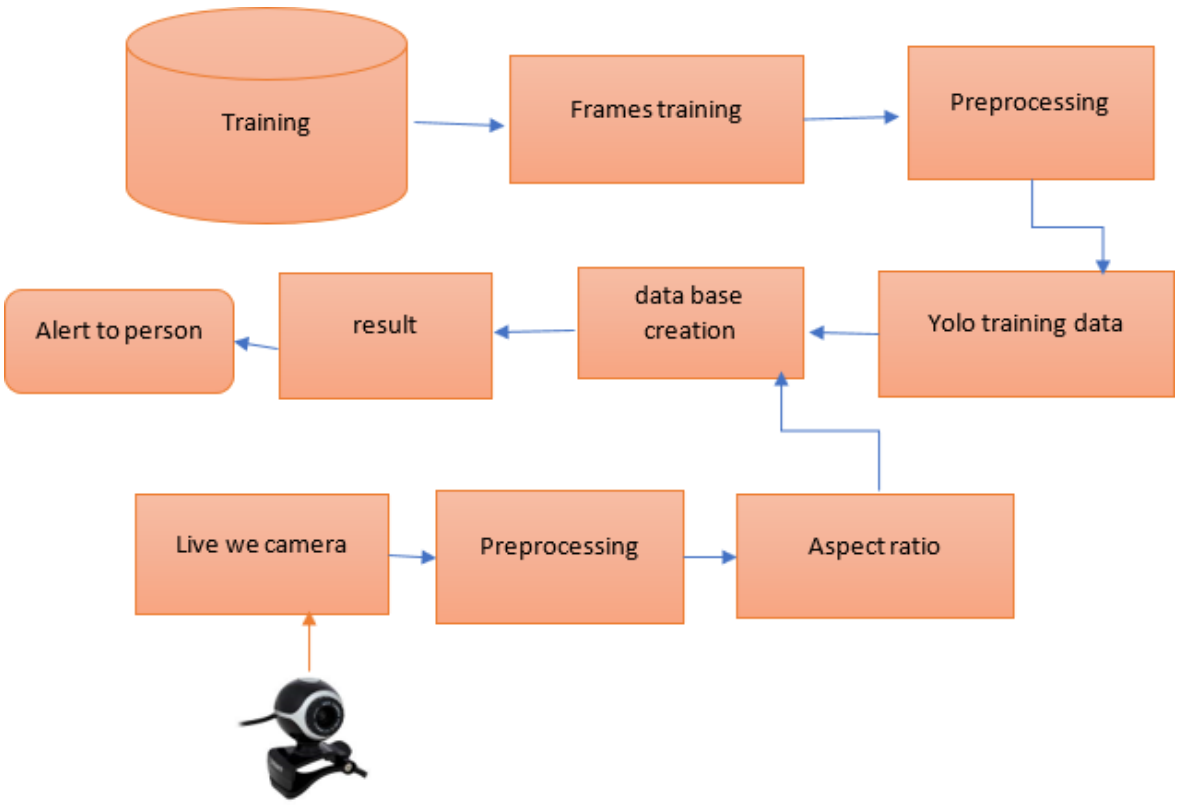

Fig.No .- 6.2.1 : Data Base & Output Explanation

EXPLANATION

Frame Training:

Begin with capturing frames from the camera feed. Each frame contains the driver's face and surroundings.

Pre-processing:

Resize frames to a consistent input size required for YOLO training. Normalize pixel values to a suitable range (e.g., [0, 1]). Handle aspect ratio to maintain the correct proportions of objects in the image.

YOLO Training Data Preparation:

Annotate the training data with bounding boxes around the driver's face. Create a YOLO-compatible dataset with labeled images and corresponding bounding box coordinates. Split the dataset into training and validation sets for model training and evaluation.

Database Creation:

Include labels indicating whether the driver is drowsy or alert. Ensure a diverse and representative dataset to improve the model's generalization.

YOLO Training:

Utilize the preprocessed frames as input during the training process.Fine-tune the YOLO model to detect the driver's face and classify drowsy/alert states.

Monitor training metrics, such as loss and accuracy, to assess the model's performance.

Result Analysis:

Analyze metrics such as precision, recall, and F1-score to gauge the model's effectiveness. Adjust hyperparameters or augment the dataset if needed to improve performance.

Alert Person Explanation:

Detect the driver's face and classify the drowsy/alert state based on the model's predictions. This could involve sound alarms, haptic feedback, or visual warnings to prompt the driver to stay alert.

CHAPTER 4

MODULE DESCRIPTION

MODULES

4.1. Dataset collection

4.2. Take the image input

4.3. Detect face in the image and create a region of interest (ROI)

4.4. Detect the eyes from roi and feed it to the classifier

4.5. Classifier will Categorize whether Eyes are Open or Closed

4.1. Dataset collection

The dataset used for this model is created by us. To create the dataset, we wrote a script that captures eyes from a camera and stores in our local disk. We separated them into their respective labels ‘Open’ or ‘Closed’. The data was manually cleaned by removing the unwanted images which were not necessary for building the model. The data comprises around 7000 images of people’s eyes under different lighting conditions. After training the model on our dataset, we have attached the final weights and model architecture file “models/cnnCat2.h5”

The dataset used in the research project or paper that proposed the deep CNN models-based ensemble approach to driver drowsiness detection will depend on the specific study. However, a common dataset used for driver drowsiness detection research is the "Drowsy Driver Detection Dataset" created by the University of Texas at Arlington.

Other datasets that could potentially be used for driver drowsiness detection include the "PupilLabs Dataset" and the "State Farm Distracted Driver Detection Dataset," among others. The specific dataset used will depend on the research goals and the availability and suitability of the data for the task at hand.

4.2. Take Image as Input from a Camera

With a webcam, we will take images as input. So to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV, cv2.VideoCapture(0) to access the camera and set the capture object (cap). cap.read() will read each frame and we store the image in a frame variable

4.3. Detect Face in the Image and Create a Region of Interest (ROI)

To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes gray images in the input. We don't need color information to detect the objects. We will be using haar cascade classifier to detect faces. This line is used to set our classifier face = cv2.CascadeClassifier(' path to our haar cascade xml file'). Then we perform the detection using faces = face.detectMultiScale(gray). It returns an array of detections with x,y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

4.4. Detect the eyes from ROI and feed it to the classifier

The same procedure to detect faces is used to detect eyes. First, we set the cascade classifier for eyes in leye and reye respectively then detect the eyes using left_eye = leye.detectMultiScale(gray). Now we need to extract only the eyes data from the full image. This can be achieved by extracting the boundary box of the eye and then we can pull out the eye image from the frame with this code.

4.5. Classifier will Categorize whether Eyes are Open or Closed

We are using cnn classifier for predicting classifier for predicting the eye status. To feed our image into the model, we need to perform certain operations because the model needs the correct dimensions to start with. First, we convert the color image into grayscale using Then, we resize the image to 24*24 pixels as our model was trained on 24*24-pixel images We normalize our data for better convergence **r_eye = r_eye/255** (All values will be between 0-1). Expand the dimensions to feed into our classifier. We loaded our model using. Now we predict each eye with our model If the value of lpred[0] = 1, it states that eyes are open, if value of lpred[0] = 0 then, it states that eyes are closed.

CHAPTER 5

SOFTWARE REQUIREMENT

5.1 HARDWARE SYSTEM CONFIGURATION:

- Processor – i3
- RAM - 4 GB (min)
- Hard Disk - 20 GB

5.2 SOFRWARE SYSTEM CONFIGURATION:

- Operating System : Windows 10
- Software : Python Idle

CHAPTER 6

SOFTWARE ENVIRONMENT

6.1. Python Technology:

Python is an interpreter, high-level, general-purpose programming language. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. **Python** is often described as a "batteries included" language due to its comprehensive standard library.

6.2. Python Programing Language:

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and met objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python packages with a wide range of functionality, including:

- Easy to Learn and Use
- Expressive Language
- Interpreted Language
- Cross-platform Language
- Free and Open Source
- Object-Oriented Language
- Extensible
- Large Standard Library
- GUI Programming Support

- Integrated

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

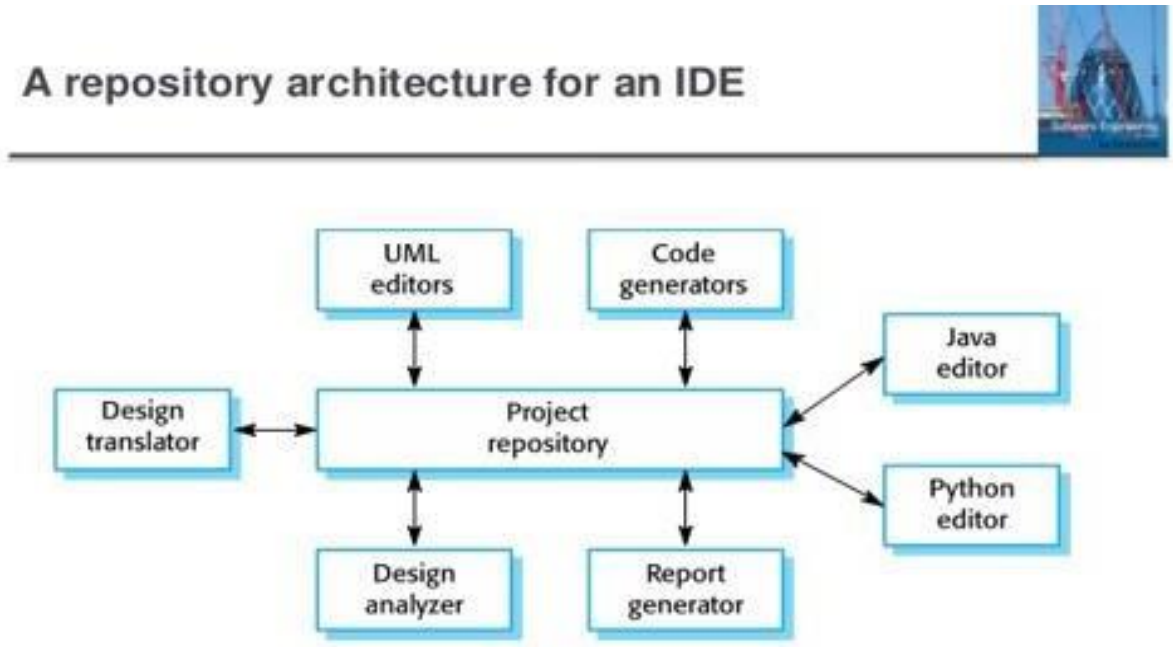

Fig.No .- 6.2.1 : A Repository Architecture For An IDE

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one and preferably only one obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the Python reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Python is also available,

which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

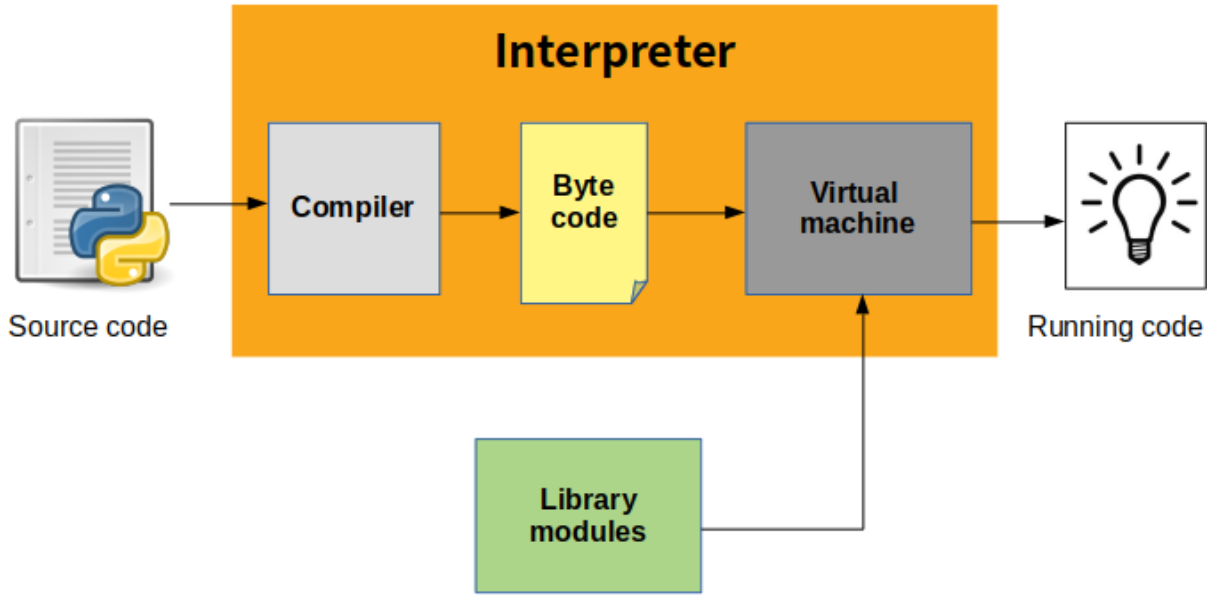

Fig.No. – 6.2.2 : Python Interpreter

6.2.1 The Python Platform:

The platform module in Python is used to access the underlying platform's data, such as, hardware, operating system, and interpreter version information. The platform module includes tools to see the platform's hardware, operating system, and interpreter version information where the program is running.

There are four functions for getting information about the current Python interpreter. python_version() and python_version_tuple() return different forms of the interpreter version with major, minor, and patch level components. python_compiler() reports on the compiler used to build the interpreter. And python_build() gives a version string for the build of the interpreter.

Platform() returns string containing a general purpose platform identifier. The function accepts two optional Boolean arguments. If aliased is true, the names in the return value are converted from a formal name to their more common form. When terse is true, returns a minimal value with some parts dropped.

What does python technology do?

Python is quite popular among programmers, but the practice shows that business owners are also Python development believers and for good reason. Software developers love it for its straightforward syntax and reputation as one of the easiest programming languages to learn. Business owners or CTOs appreciate the fact that there’s a framework for pretty much anything – from web apps to machine learning.

Moreover, it is not just a language but more a technology platform that has come together through a gigantic collaboration from thousands of individual professional developers forming a huge and peculiar community of aficionados.

So what are the tangible benefits the language brings to those who decided to use it as a core technology? Below you will find just some of those reasons.

6.2.2 Productivity And Speed

It is a widespread theory within development circles that developing Python applications is approximately up to 10 times faster than developing the same application in Java or C/C++. The impressive benefit in terms of time saving can be explained by the clean object-oriented design, enhanced process control capabilities, and strong integration and text processing capacities. Moreover, its own unit testing framework contributes substantially to its speed and productivity.

6.2.3 Python Is Popular For Web Apps

Web development shows no signs of slowing down, so technologies for rapid and productive web development still prevail within the market. Along with JavaScript and Ruby, Python, with its most popular web framework Django, has great support for building web apps and is rather popular within the web development community.

6.2.4 Open-Source And Friendly Community

As stated on the official website, it is developed under an OSI-approved open source license, making it freely usable and distributable. Additionally, the development is driven by the community, actively participating and organizing conference, meet-ups, hackathons, etc. fostering friendliness and knowledge-sharing.

6.2.5 Python Is Quick To Learn

It is said that the language is relatively simple so you can get pretty quick results without actually wasting too much time on constant improvements and digging into the complex engineering insights of the technology. Even though Python programmers are really in high demand these days, its friendliness and attractiveness only help to increase number of those eager to master this programming language.

6.2.6 Broad Application

It is used for the broadest spectrum of activities and applications for nearly all possible industries. It ranges from simple automation tasks to gaming, web development, and even complex enterprise systems. These arz the areas where this technology is still the king with no or little competence:

- Machine learning as it has a plethora of libraries implementing machine learning algorithms.
- Web development as it provides back end for a website or an app.
- Cloud computing as Python is also known to be among one of the most popular cloud-enabled languages even used by Google in numerous enterprise-level software apps.
- Scripting.
- Desktop GUI applications.

6.2.7 Python Compiler

The Python compiler package is a tool for analyzing Python source code and generating Python bytecode. The compiler contains libraries to generate an abstract syntax tree from Python source code and to generate Python bytecode from the tree.

The compiler package is a Python source to bytecode translator written in Python. It uses the built-in parser and standard parser module to generate a concrete syntax tree. This tree is used to generate an abstract syntax tree (AST) and then Python bytecode.

The full functionality of the package duplicates the built-in compiler provided with the Python interpreter. It is intended to match its behavior almost exactly. Why implement another compiler that does the same thing? The package is useful for a variety of purposes. It can be modified more easily than the built-in compiler. The AST it generates is useful for analyzing Python source code.

6.2.8 The Basic Interface

The top-level of the package defines four functions. If you import compiler, you will get these functions and a collection of modules contained in the package.

compiler.parse(buf)

Returns an abstract syntax tree for the Python source code in buf. The function raises Syntax Error if there is an error in the source code. The return value is a compiler.ast. Module instance that contains the tree.

compiler.parseFile(path)

Return an abstract syntax tree for the Python source code in the file specified by path. It is equivalent to parse(open(path).read()).

6.2.9 Limitations

There are some problems with the error checking of the compiler package. The interpreter detects syntax errors in two distinct phases. One set of errors is detected by the interpreter's parser, the other set by the compiler. The compiler package relies on the interpreter's parser, so it get the first phases of error checking

for free. It implements the second phase itself, and that implementation is incomplete. For example, the compiler package does not raise an error if a name appears more than once in an argument list: def f(x, x): ...

A future version of the compiler should fix these problems.

6.2.10 Python Abstract Syntax

The compiler.ast module defines an abstract syntax for Python. In the abstract syntax tree, each node represents a syntactic construct. The root of the tree is Module object.

The abstract syntax offers a higher level interface to parsed Python source code. The parser module and the compiler written in C for the Python interpreter use a concrete syntax tree. The concrete syntax is tied closely to the grammar description used for the Python parser. Instead of a single node for a construct, there are often several levels of nested nodes that are introduced by Python's precedence rules.

The abstract syntax tree is created by the compiler.transformer module. The transformer relies on the built-in Python parser to generate a concrete syntax tree. It generates an abstract syntax tree from the concrete tree.

The transformer module was created by Greg Stein and Bill Tutt for an experimental Python-to-C compiler. The current version contains a number of modifications and improvements, but the basic form of the abstract syntax and of the transformer are due to Stein and Tutt.

6.2.11 AST Nodes

The compiler.ast module is generated from a text file that describes each node type and its elements. Each node type is represented as a class that inherits from the

abstract base class compiler.ast.Node and defines a set of named attributes for child nodes.

classcompiler.ast.Node

The Node instances are created automatically by the parser generator. The recommended interface for specific Node instances is to use the public attributes to access child nodes. A public attribute may be bound to a single node or to a sequence of nodes, depending on the Node type. For example, the bases attribute of the Class node, is bound to a list of base class nodes, and the doc attribute is bound to a single node.

Each Node instance has a lineno attribute which may be None. XXX Not sure what the rules are for which nodes will have a useful lineno.

All Node objects offer the following methods:

getChildren()

Returns a flattened list of the child nodes and objects in the order they occur. Specifically, the order of the nodes is the order in which they appear in the Python grammar. Not all of the children are Node instances. The names of functions and classes, for example, are plain strings.

getChildNodes()

Returns a flattened list of the child nodes in the order they occur. This method is like getChildren(), except that it only returns those children that are Node instances.

The While node has three attributes: test, body, and else_. (If the natural name for an attribute is also a Python reserved word, it can’t be used as an attribute name.

An underscore is appended to the word to make it a legal identifier, hence else_ instead of else.)

The if statement is more complicated because it can include several tests.

The If node only defines two attributes: tests and else_. The tests attribute is a sequence of test expression, consequent body pairs. There is one pair for each if/elif clause. The first element of the pair is the test expression. The second elements is a Stmt node that contains the code to execute if the test is true.

The getChildren() method of If returns a flat list of child nodes. If there are three if/elif clauses and no else clause, then getChildren() will return a list of six elements: the first test expression, the first Stmt, the second text expression, etc.

The following table lists each of the Node subclasses defined in compiler.ast and each of the public attributes available on their instances. The values of most of the attributes are themselves Node instances or sequences of instances. When the value is something other than an instance, the type is noted in the comment. The attributes are listed in the order in which they are returned by getChildren() and getChildNodes().

6.2.12 Development Environments:

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting.

6.2.13 Implementations

Reference implementation

CPython is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features. It compiles Python programs into an intermediate bytecode which is then executed by its virtual machine. CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.

Other Implementations

PyPy is a fast, compliant interpreter of Python 2.7 and 3.5. Its just-in-time compiler brings a significant speed improvement over CPython but several libraries written in C cannot be used with it.

Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version.

MicroPython and CircuitPython are Python 3 variants optimized for microcontrollers. This includes Lego Mindstorms EV3.

RustPython is a Python 3 interpreter written in Rust.

Unsupported Implementations

Other just-in-time Python compilers have been developed, but are now unsupported:

Google began a project named Unladen Swallow in 2009, with the aim of speeding up the Python interpreter five-fold by using the LLVM, and of improving its multithreading ability to scale to thousands of cores, while ordinary implementations suffer from the global interpreter lock.

Psyco is a just-in-time specialising compiler that integrates with CPython and transforms bytecode to machine code at runtime. The emitted code is specialized for certain data types and is faster than standard Python code.

In 2005, Nokia released a Python interpreter for the Series 60 mobile phones named PyS60. It includes many of the modules from the CPython implementations and some additional modules to integrate with the Symbian operating system. The project has been kept up-to-date to run on all variants of the S60 platform, and several third-party modules are available. The Nokia N900 also supports Python with GTK widget libraries, enabling programs to be written and run on the target device.

6.2.14 Cross-Compilers To Other Languages

There are several compilers to high-level object languages, with either unrestricted Python, a restricted subset of Python, or a language similar to Python as the source language:

- Jython enables the use of the Java class library from a Python program.
- IronPython follows a similar approach in order to run Python programs on the .NET Common Language Runtime.
- The RPython language can be compiled to C, and is used to build the PyPy interpreter of Python.
- Pyjs compiles Python to JavaScript.
- Cython compiles Python to C and C++.
- Numba uses LLVM to compile Python to machine code.
- Pythran compiles Python to C++.
- Somewhat dated Pyrex (latest release in 2010) and Shed Skin (latest release in 2013) compile to C and C++ respectively.

- Google's Grumpy compiles Python to Go.
- MyHDL compiles Python to VHDL.

6.2.15 Performance

A performance comparison of various Python implementations on a non-numerical (combinatorial) workload was presented at EuroSciPy '13.

6.2.16 API Documentation Generators

Python API documentation generators include:

- Sphinx
- Epydoc
- HeaderDoc
- Pydoc

6.2.17 PANDAS

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

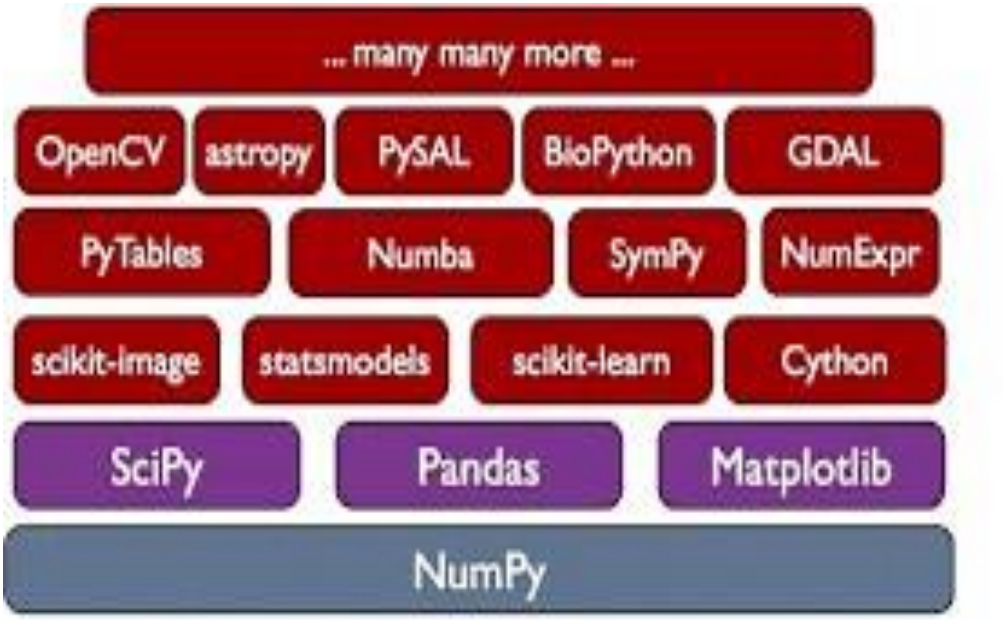

Fig.No. – 6.2.17.1 : Pandas Software Library

6.2.18 CSV Reader

CSV (Comma Separated Values) is a simple file format used to store tabular data, such as a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

For working CSV files in python, there is an inbuilt module called csv.

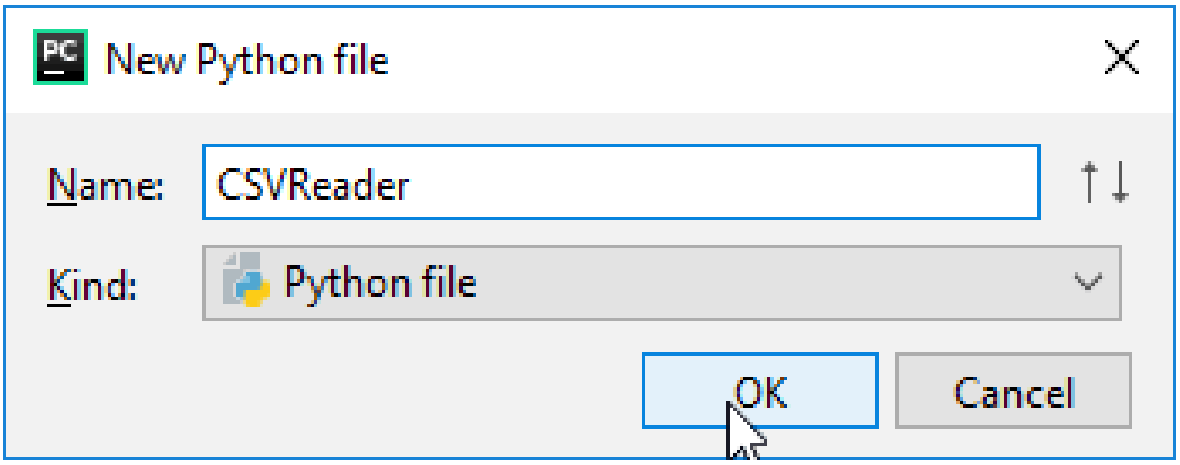

Fig.No. – 6.2.18.1 : CSV Reader

CHAPTER 7

7.1 CONCLUSION AND FUTURE WORKS

In conclusion, the deep CNN models-based ensemble approach is a promising method for driver drowsiness detection. By combining multiple CNN models, the ensemble approach can improve the accuracy and robustness of the detection system. This approach can also handle various types of input data, such as images, videos, and physiological signals.

However, there are still some challenges in implementing this approach. One of the major challenges is the large number of parameters and the high computational cost required to train multiple deep CNN models. Therefore, it is important to carefully design the architecture and optimize the training process to achieve the best performance with limited computational resources. Another challenge is the need for large and diverse datasets for training the deep CNN models. The quality and size of the dataset have a significant impact on the performance of the models, and it is important to collect and label data that are representative of the target population.

Providing owners with weekly reports on drivers' behavior, including images of when they feel drowsy and the times they feel drowsy, is a proactive approach to ensuring road safety. Analyzing this data can help improve driver performance and prevent potential accidents caused by drowsiness. It's a smart move towards better safety and efficiency.

CHAPTER 8

APPENDIX

8.1 SOURCE CODE

```
## import packages

import cv2
import os
from keras.models import load_model
import tensorflow as tf
import numpy as np
from pygame import mixer
import time
from mail import report_send_mail
import serial


path = os.getcwd()



'''ser = serial.Serial(
   port='COM3', # Device name
   baudrate=9600, # Baud rate such as 9600 or 115200 etc.
   parity=serial.PARITY_NONE, # Enable parity checking
   stopbits=serial.STOPBITS_ONE, # Number of stop bits
   bytesize=serial.EIGHTBITS, # Number of data bits.
   timeout=.1, # Set a read timeout value.
   rtscts=0  # Enable hardware (RTS/CTS) flow control.
)'''




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

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


## number of class

lbl=['Close','Open']


## load model
#model = load_model('models/drowsiness_model.h5')
model = tf.keras.models.load_model('models/drowsiness_model.h5')

## to get web cam capture
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_COMPLEX_SMALL

## initialize the some parameter

count=0
score=0
thicc=2
rpred=[99]
lpred=[99]

while(True):
    ret, frame = cap.read()
    height,width = frame.shape[:2]

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces =
face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25))
    left_eye = leye.detectMultiScale(gray)
    right_eye=  reye.detectMultiScale(gray)

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

```
for (x,y,w,h) in faces:
    cv2.rectangle(frame, (x,y) , (x+w,y+h) , (100,100,100) , 1 )

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

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

if(rpred[0]==0 or lpred[0]==0):
    print('sleep')
    score=score+1
```

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

    if(rpred[0]==1 or lpred[0]==1):
        #score=score-3
        score=0
        cv2.putText(frame,"Open",(10,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)
        print('No sleep')
        #ser.write(b'0')
        #time.sleep(2)
        print('---------')
        #print('0 Send')
        print('---------')
    if(score<0):
        score=0
    cv2.putText(frame,'Score:'+str(score),(100,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)
    if(score>5):

        #person is feeling sleepy so we beep the alarm
        cv2.imwrite(os.path.join(path,'image.jpg'),frame)
        cv2.imwrite('image.jpg', frame)
        report_send_mail('image.jpg')

        #ser.write(b'1')
        #time.sleep(2)
        print('---------')
        #print('1 Send')
        print('---------')
        try:
            #sound.play()
            print('')
        except:
            pass
        if(thicc<16):
            thicc= thicc+2
        else:
            thicc=thicc-2
            if(thicc<2):
```

```
            thicc=2

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

    # to plot on screen
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


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

8.2 SCREENSHOTS

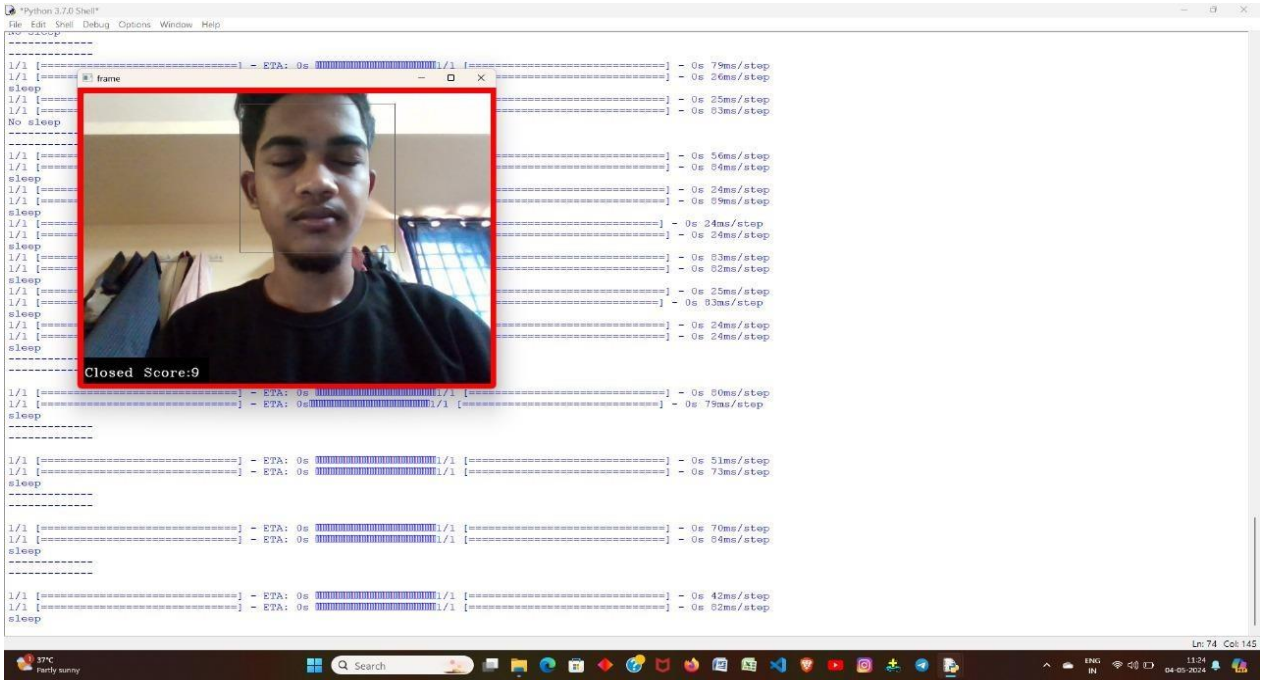

Fig.No. – 8.2.1 : Detecting Drowsiness

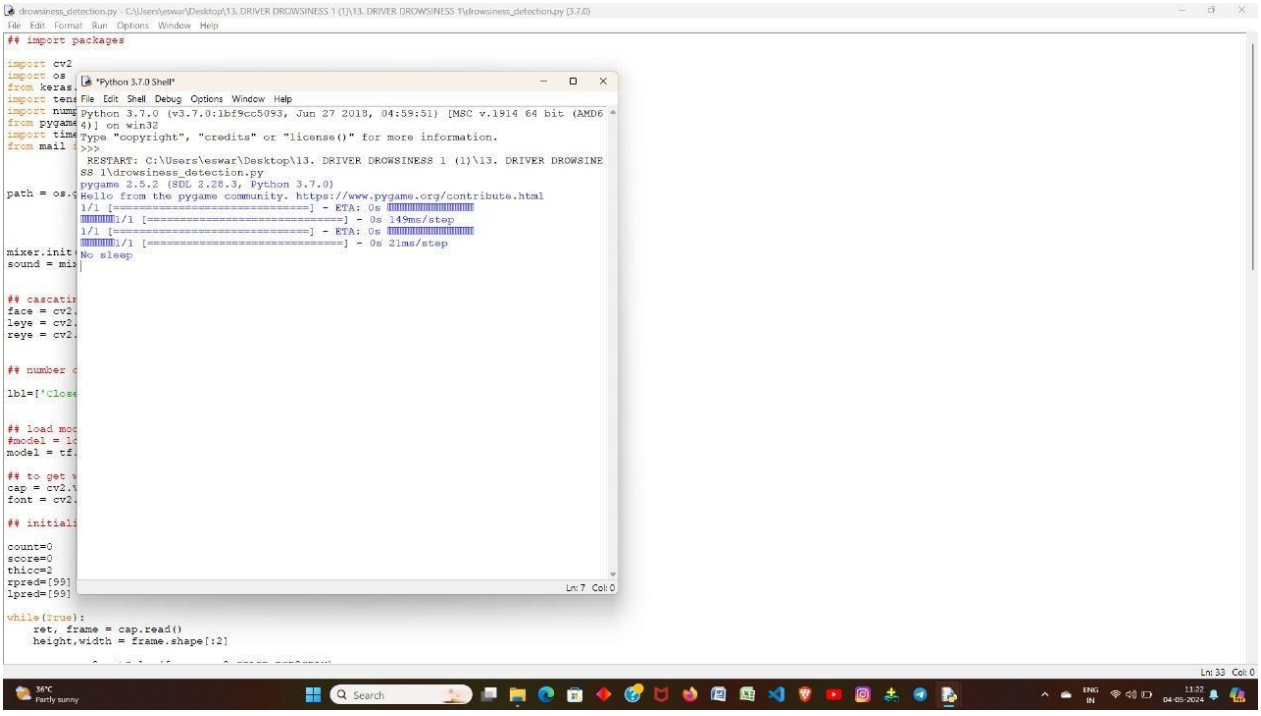

Fig.No. – 8.2.2 : Output

CHAPTER 9

9.1 REFERENCE:

[1] National Center for Statistics and Analysis.(2018). Early Estimate of motor Vehicle Traffic Fatalities for the First Half (Jan- Jun) of 2018.Accessed: Feb. 12, 2019.

[2] L. Moreira-Matias and H. Farah, “On developing a driver identification methodology using in-vehicle data recorders,” IEEE Trans. Intell. Transp. Syst., vol. 18, no. 9, pp. 2387–2396, Sep. 2017.

[3] J. Kim et al., “Impact of the face angle to traveling trajectory during the riding standing-type personal mobility device,” in Proc. Web Conf. (MATEC), vol. 161, Apr. 2018, Art. no. 003001.

[4] A. Kashevnik et al., “Context-based cyclist intelligent support: An approach to e-bike control based on smartphone sensors.” in Proc. Int. Conf. Next Gener.Wired/Wireless Netw. Cham, Switzerland: Springer, Aug. 2018, pp. 16–22.

[5] A. Smirnov, A. Kashevnik, I. Lashkov, N. Hashimoto, and A. Boyali, “Smartphone-based two-wheeled self-balancing vehicles rider assistant,” in Proc. 17th IEEE Conf. Open Innov. Assoc. (FRUCT), Apr. 2015, pp. 201–209.

[6] A. Smirnov, A. Kashevnik, and I. Lashkov, “Human-smartphone interaction for dangerous situation detection and recommendation generation while driving,” in Proc. 18th Int. Conf. Speech Comput., Aug. 2016, pp. 346–353.

[7] I. Lashkov, A. Smirnov, A. Kashevnik, and V. Parfenov, “Ontologybased approach and implementation of ADAS system for mobile device use while driving,” in Proc. 6th Int. Conf. Knowl. Eng. Semantic Web, Sep. 2015, pp. 117–131.

[8] A. Smirnov, A. Kashevnik, I. Lashkov, O. Baraniuc, and V. Parfenov, “Smartphone-based identification of dangerous driving situations: Algorithms and implementation,” in Proc. 18th IEEE Conf. Open Innov. Assoc. (FRUCT), Apr. 2016, pp. 306–313.

[9] A. Fedotov, I. Lashkov, and A. Kashevnik, “Web-service for drive safely system user analysis: Architecture and implementation,” in Proc. 22nd Conf. Open Innov. Assoc. (FRUCT), May 2018, pp. 40–47.

[10] S. Sivaraman and M. M. Trivedi, “Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis,” IEEE Trans. Intell. Transp. Syst., vol. 14, no. 4, pp. 1773–1795, Dec. 2013.

[11] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, “Survey of pedestrian detection for advanced driver assistance systems,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 7, pp. 1239–1258, Jul. 2010.

[12] D. Kim, J. Choi, H. Yoo, U. Yang, and K. Sohn, “Rear obstacle detection system with fisheye stereo camera using HCT,” Expert Syst. Appl., vol. 42, nos. 17–18, pp. 6295–6305, Oct. 2015.

[13] Z. Kim, “Robust lane detection and tracking in challenging scenarios,” IEEE Trans. Intell. Transp. Syst., vol. 9, no. 1, pp. 16–26, Mar. 2008.

[14] J. Jo, S. J. Lee, J. Kim, H. G. Jung, and K. R. Park, “Vision-based method for detecting driver drowsiness and distraction in driver monitoring system,” Proc. SPIE, vol. 50, no. 12, Dec. 2011, Art. no. 127202.

[15] H. Li, H. Wang, L. Liu, and M. Gruteser, “Automatic unusual driving event identification for dependable self-driving,” in Proc. 16th ACM Conf. Embedded Netw. Sensor Syst., New York, NY, USA, Nov. 2018, pp. 15–27.