DRIVER DROWSINESS MONITORING USING CONVOLUTIONAL NEURAL NETWORKS (ConvNet)

A PROJECT REPORT

Submitted by

GURUPRABHA V **312316104053**

HEMASREE S **312316104057**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE

St. JOSEPH'S COLLEGE OF ENGINEERING, CHENNAI-119

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2020

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**DRIVER DROWSINESS MONITORING USING CONVOLUTIONAL NEURAL NETWORKS (ConvNet)"** is the bonafide work of **GURUPRABHA V (312316104053)** and **HEMASREE S (312316104057)** who carried out the project work under my supervision, for the partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science.

SIGNATURE

Dr. A. Chandrasekar M.E., Ph.D.,
Professor

HEAD OF THE DEPARTMENT

Department of Computer Science and Engineering
St. Joseph's College of Engineering
Old Mamallapuram Road
Chennai-600119

SIGNATURE

Dr.D.Rosy Salomi Victoria, M.E., M.S., Ph.D.,
Associate Professor

SUPERVISOR

Department of Computer Science and Engineering
St. Joseph's College of Engineering
Old Mamallapuram Road
Chennai-600119

CERTIFICATE OF EVALUATION

College Name **:** St. Joseph's College of Engineering

Branch & Semester **:** Computer Science (VIII)

S.NO	NAME OF THE STUDENTS	TITLE OF THE PROJECT	NAME OF THE SUPERVISOR WITH DESIGNATION
1. 2.	GURUPRABHA V (312316104053) HEMASREE S (312316104057)	DRIVER DROWSINESS MONITORING USING CONVOLUTIONAL NEURAL NETWORK(ConvNet)	Dr. D.Rosy Salomi Victoria M.E., M.S., Ph.D., Associate Professor

The report of the project work submitted by the above students in partial fulfillment for the award of Bachelor of Engineering degree in Computer Science of Anna University were evaluated and confirmed to be reports of the work done by the above students.

Submitted for the Viva Voce held on: _____.

(INTERNAL EXAMINER) **(EXTERNAL EXAMINER)**

ACKNOWLEDGEMENT

At the outset we would like to express my sincere gratitude to our beloved **Chairman, Dr.Babu Manoharan, M.A., M.B.A., Ph.D.,** for his constant guidance and support.

We would like to express our heartfelt thanks to our respected **Managing Director Mrs. S. Jessie Priya, M.Com.** for her kind encouragement and blessings.

We wish to express our sincere thanks to **Director Mr. B. Shashi Sekar , M.Sc.,** for providing ample facilities in the institution.

We express our deepest gratitude and thanks to our beloved **Principal Dr.Vaddi Seshagiri Rao, M.E., M.B.A., Ph.D., F.I.E.,** for his inspirational ideas during the course of the project.

We wish to express our sincere thanks and gratitude to **Dr. A. Chandrasekar, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering and my project guide **Dr. D. Rosy Salomi Victoria, M.E., M.S., Ph.D.,** Associate Professor, Department of Computer Science and Engineering, St. Joseph's College of Engineering for their guidance and assistance in solving the various intricacies involved in the project.

Our special thanks to Project coordinator **Mrs. P.N Jeipratha, M.E.,(Ph.D.),** Assistant Professor, Department of Computer Science and Engineering for the deluge of ideas and assistance she has provided to us all through this project.

Finally we thank our parents and friends who helped us in the successful completion of this project.

ABSTRACT

The advancement in computer vision has provided assistance to drivers in the form of automatic self-driving cars etc. Statistics have shown that 20% of all road accidents are caused by drivers fatigue and drowsiness. It pose a serious problem for which several approaches were proposed. However, they are not suitable for real time processing. The major challenges faced by these methods are robustness to handle variation in human face and lightning conditions. Our aim is to implement an intelligent processing system that can reduce road accidents drastically. This approach enables us to identify driver's facial features such as PERCLOS, EAR, and MAR, blink rate, yawning, head movement etc. In this system, the driver is continuously monitored by using a webcam. The driver's face and eye is detected using haar cascade classifiers. Eye images are extracted and fed to Custom designed CNN for classifying whether both left and right eye are closed. Based on the classification, eye closure score is calculated. If the driver is found to be drowsy, an alarm will be triggered.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

ACRONYM	EXPANSION
AI	Artificial Intelligence
CNN	Convolutional Neural Network
RELU	Rectified Linear Unit
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
PyPI	Python Package Index
EAR	Eye Aspect Ratio
MAR	Mouth Aspect Ratio
MT-DMF	Multi Task Driver Monitoring Framework
UTA-RLDD	The University of Texas At Arlington Real Life Drowsiness Dataset

CHAPTER 1

INTRODUCTION

This chapter gives an overview of the proposed system and scope of the project. Fatigue driving is the main cause in 10%–20% of all traffic accidents and causes nearly 60% of fatal accidents. In Germany, several studies conducted by Volkswagen AG in 2005 indicate that driver falling asleep causes 5-25% of all collisions. As a result, it is necessary for us to develop intelligent assistance systems that can detect fatigue-driving status and provide alerts for the drivers.

Drowsiness detection is studied by monitoring vehicle-based measurements, behavioral measurements, and physiological measurements. Vehicle-based measurements are from steering wheel movements, driving speed, brake patterns, and standard deviation of lane positions. Behavioral measurements are obtained from driver eye/face movement using a camera. Physiological measurements such as heart rate, electrocardiogram, electromyogram, electroencephalogram and electrooculogram can be used to monitor drowsiness. This project is focused on detecting drowsiness by monitoring eye closure of the driver. PERCLOS (percentage of eyelid closure over the pupil overtime) is considered a reliable measure for predicting drowsiness and has been incorporated in commercial products. Until recently, most vision-based drowsiness detection has relied on handcrafted features for monitoring facial and head movements. In general, handcrafted features have shown limited effectiveness in real-world scenarios. On the other hand, features learned based on deep learning have been more effective in real-world scenarios.

1.1 SYSTEM OVERVIEW

Over the years, various safety-related driving assistant systems have been proposed to reduce the risk of car accidents, and statistics have shown fatigue to be a leading cause of car accidents. In fact, the American Automobile Association released a figure in 2010 that 17% of all fatal crashes in the USA could be attributed to tired drivers. This seems to be a global trend.

The system continuously monitors the driver by using a webcam. Face and eyes are detected by using predefined classifiers available in opencv called Haar Cascade classifiers. Left and right eye images of the driver are extracted and fed into a trained custom designed CNN for prediction of eye closure. If the eyes are closed in consecutive frames then an alarm triggers to alert the driver.

1.2 SCOPE OF THE PROJECT

The main goal of this project is to develop an intelligent processing system to avoid road accidents. This can be done by real time monitoring of driver's drowsiness and warning the driver of inattention to prevent accidents. Drivers Face and eye is detected continuously by a webcam using Haar Cascade classifiers. Face is fixed as the region of interest, eye images are extracted and fed into a Convolutional Neural Network for prediction. Based on eye closure, the system detects whether the driver is alert or drowsy and triggers an alarm.

CHAPTER 2
LITERATURE SURVEY

This chapter details about literature survey in the fields to detect driver drowsiness through facial features and the use of CNN for image classification.

[5] Mandalapu Sarada Devi and Dr. Preeti R Bajaj "Driver Fatigue Detection Based on Eye Tracking "First International Conference (2008)

Video file is converted into frames. It uses a simple approach of detecting eyes by measuring the distances between the intensity changes in the eye area, one can determine whether the eyes are open or closed. If the eyes are found closed for five consecutive frames, the system draws the conclusion that the driver is falling asleep and issues a warning signal.

It does not take into account other physiological factors such as blink rate, yawning, head movement etc. It is not suitable for real time processing.The basis of the method used by authors was the horizontal intensity variation on the face. One similarity among all faces is that eyebrows are significantly different from the skin in intensity, and that the next significant change in intensity, in the y- direction, is the eyes. This facial characteristic is the center of finding the eyes on the face, which will allow the system to monitor the eyes and detect long periods of eye closure.

[2] Kartik Dwivedi, Kumar, Biswaranjan and Amit Sethi "Drowsy Driver Detection using Representation Learning "2014 IEEE

It proposes a vision based intelligent algorithm to detect driver drowsiness. Previous approaches are generally based on blink rate, eye closure, yawning, eyebrow shape and other hand engineered facial features. The proposed algorithm makes use of features learnt using convolutional neural network so as to explicitly capture various latent facial features and the complex non-linear feature interactions. A softmax layer is used to classify the driver as drowsy or non- drowsy. The feature maps produced by convolving the learnt weights with input image act as the features for driver drowsiness detection. Using these set of features a soft-max layer classifier is used to finally classify the frames extracted as drowsy or non-drowsy.

The merit is that it makes decision on frame level by application of 2D convolutional neural networks on each frame for feature extraction. The demerit is that it does not take into the account spatial-temporal relationship. It uses simulated datasets and does not consider features such as head movement.

[6] Sanghyuk Park, Fei Pan, Sunghun Kang and Chang D. Yoo" Driver drowsiness detection system based on feature representation learning using various deep networks"

Deep network based feature representation learning approaches have been providing an automated and efficient set of learned features which help us to classify the driver as drowsy or non-drowsy. It uses a deep architecture referred to as deep drowsiness detection (DDD) network for learning effective features and detecting drowsiness given a RGB input video of a driver. The proposed DDD

network consists of three deep networks AlexNet, VGG-FaceNet and FlowImageNet. Given an image sequences, the AlexNet is fine tuned to learn features related to drowsiness. The VGG-FaceNet is trained to learn facial feature related to drowsiness, which is robust to genders, ethnicity, hairstyle and various accessories adornment. FlowImageNet takes dense optical flow image that is extracted from consecutive image sequences and is trained to learn behaviour features related to drowsiness such as facial and head movements. Each three networks are independently finetuned for multi-class drowsiness classification given the following four classes non-drowsiness, drowsiness with eye blinking, nodding and yawning. The outputs of the three networks are integrated and fed to a softmax classifier for drowsiness detection.

Due to the lack of ground truth label of test dataset, they substituted evaluation dataset for test dataset. It uses a complex structure for feature extraction. All the extracted image sequences from the evaluation video dataset were labelled manually are some of its demerits.

[8] Xiaoxi Ma, Lap-Pui Chau and Kim-Hui Yap "Depth Video-based Two-stream Convolutional Neural Networks for Driver Fatigue Detection"2017 IEEE

Driver fatigue detection system based on CNN using depth video sequences, which helps to provide alerts properly to fatigue drivers during the night time. Specifically, the two-stream CNN architecture incorporates spatial information of current depth frame and temporal information of neighboring depth frames which is represented by motion vectors. Besides, the recent development of Kinect makes depth images acquired from it available and reliable during both daytime and

nighttime. It incorporates spatial information and temporal information of depth frames which is represented by motion vectors. But it does not represent the explicit temporal motion information within two consecutive depth frames better.

[3] Ki Wan Kim, Hyung Gil Hong, Gi Pyo Nam and Kang Ryoung Park "A Study of Deep CNN-Based Classification of Open and Closed Eyes Using a Visible Light Camera Sensor" (2017)

This paper proposes a method to classify open and closed eye images with different conditions, acquired by a visible light camera, using a deep residual convolutional neural network. First, the image of an eye is used as input, and it is resized into a standard 224×224 pixel image via size normalization. Additionally, we perform zero-center normalization, which does not use the pixel values of the input images as they are, but instead uses the values with the mean image of the training set subtracted. The normalized image is then used as the input for the pre-trained CNN and the classification of open and closed eyes is performed based on the output of the CNN. However, the CNN is operated only on single image based method.

[9] Weiwei Zhang, Jinya Su" Driver Yawning Detection based on Long Short Term Memory Networks" (2017) IEEE

This system employs a Convolutional Neural Networks to extract spatial image features and a Long Short Term Memory Network to analyze temporal characteristics. GoogleNet is used to process individual video frames. It uses a network-in-network approach, stacking Inception modules to form a network that is substantially different from previous CNN. GoogleNet takes a single frame of size 220 x 220. Multiple Inception modules then pass this frame, which applies 1x1, 3x3, and 5 x 5 convolution and max-pooling operations respectively. Finally,

the activations are average-pooled and output as a 1000-dimensional vector. A recurrent neural network is then used to explicitly consider sequences of CNN activations. Since yawning is a dynamic, continuous event that usually lasts 4-5 seconds. Adjacent frames have a certain causal relationship, which could be useful in making more accurate predictions. But it focuses only on yawning. It uses frame by frame prediction. Therefore, it requires a lot of memory.

[1] Dr. Priya Gupta, Nidhi Saxena, Meetika Sharma, Jagriti Tripathi"Deep Neural Network for Human Face Recognition" (2018)

Convolutional Neural Networks (CovNets), a type of deep networks has been proved to be successful for FR. For real-time systems, some preprocessing steps like sampling needs to be done before using to CovNets. Deep neural network (another type of deep network) for face recognition. In this approach, instead of providing raw pixel values as input, only the extracted facial features are provided. The conventional face recognition pipeline consists of four stages face detection, face alignment, face representation (or feature extraction), and classification. It extracts facial features from input images and feeds them to deep neural networks for training and classification (softmax layer is used).

[4] Luigi Celona, Lorenzo Mammana, Simone Bianco, Raimondo Schettini "A Multi-Task CNN Framework for Driver Face Monitoring" (2018)

A vision-based Multi-Task Driver Monitoring Framework (MTDMF) that simultaneously analyzes head pose eyes and mouth status, and drowsiness level of the driver. It simultaneously analyzes head pose, eyes and mouth status.But it does

not take into account the temporal dependencies. In the first step, given an input frame, the face region is detected and cropped from the whole image, and face landmarks are estimated for generating a mask of eye regions. Given the cropped face image and the eyes mask, the Multi-Task Driver Monitoring Network (MT-DMN) predicts labels describing respectively the status of eyes, mouth, head, and drowsiness. DLib's CNN-based face detector and landmark estimator are used to localize the face region and to compute 68 facial key points. The classification is performed with a 10 way fully connected layer, which provides an output for each status of the four addressed tasks: eyes (stillness or sleepy eyes), mouth (stillness, yawning, or talking/laughing), head (stillness, nodding, or looking aside), and drowsiness (stillness or drowsy).

[10] Zhongke Gao, Xinmin Wang, Yuxuan Yang, Chaoxu Mu, Qing Cai, Weidong Dang, and Siyang Zuo "EEG-Based Spatial-Temporal Convolutional Neural Network for Driver Fatigue Evaluation"2019 IEEE

It uses a spatial–temporal structure of multichannel electroencephalogram (EEG) signals and develops a novel EEG-based spatial–temporal convolutional neural network (ESTCNN) to detect driver fatigue. First, it introduce the core block to extract temporal dependencies from EEG signals. Then, it employs dense layers to fuse spatial features and realize classification. The developed network could automatically learn valid features from EEG signals. Temporal convolution can be used as a feature extraction module to process time series, and when compared with recurrent models, it has better computational efficiency on time series classification tasks.

It attaches importance to temporal dependencies learning for each electrode and also strengthens spatial information extraction. However, EEG signal is extremely weak with low signal-to-noise ratios, which makes it fairly difficult to develop computational algorithms for fatigue detection.

[7] Tawsin Uddin Ahmed ,Sazzad Hossain,Mohammad Shahadat Hossain,Raihan Ul Islam ,Karl Andersson"Facial Expression Recognition using Convolutional Neural Network with Data Augmentation"(2019)

The objective of this research is to develop a facial expression recognition system based on convolutional neural network with data- augmentation Convolutional neural network with data augmentation leads to higher validation accuracy than the other existing models. This paper come up with the idea of CNN with data augmentation and combined dataset collected from several datasets. Deep convolutional neural network, which is a combination of convolutional neural network, coupled with the deep residual blocks. First, the model takes an image from the dataset and detects face from the image by Cascade Classifier. If face is found, then it is sent for preprocessing. Data have been augmented by ImageDataGenerator function offered by the Keras API. At last, the augmented dataset is fed into CNN in order to predict the class.

CHAPTER 3
SYSTEM ANALYSIS

This chapter provides an overview of the existing system, its disadvantages, proposed system and its advantages.

3.1 EXISTING SYSTEM

Technologies in drowsiness detection can be classified into three main categories. The first category involves measuring cerebral and muscular signals and cardiovascular activity. The second category includes techniques of measuring overall driver behavior from the vehicle patterns.The third category consists of using computer vision techniques as a non-invasive way to monitor driver's drowsiness. The baseline model uses Real-Life Drowsiness Data set (UTA-RLDD) and trains a model that first extracts blink related features, including blink duration, amplitude, and eye opening velocity. The existing system uses the position of key facial features, including the eyes, nose, mouth, and chin, as well as the eye-aspect ratio (EAR), and mouth-aspect ratio (MAR) to detect drowsiness. The importance of these features are learned as a time series within a sequential model.

3.1.1 Disadvantages of the Existing System

1. EEG signal is extremely weak signal with low signal to noise ratio. Therefore, it is difficult to develop computation algorithm for fatigue detection. In addition, these techniques are invasive and not commercially viable.

2. The techniques of measuring overall driver behavior from the vehicle patterns do not work with micro-sleeps since measuring many of these parameters requires a significant amount of time and user data.
3. The issue with the baseline model is that it relies on handcrafted blink features rather than an end-to-end learning system. This work also ignores many informative facial cues that signal drowsiness.
4. Facial landmark features detected using dlib library suffers from normalization issues. It is found that everybody has a different baseline for eye and mouth aspect ratios and normalizing for each participant was necessary. Data pre-processing and feature extraction/normalization requires a lot of time.
5. The existing system was struggling with new faces and the primary reason for this struggle was the fact that each individual has different core features in their default alert state.

3.2 PROPOSED SYSTEM

Our proposed system will provide solution for monitoring driver's drowsiness. The cons of the existing system in extracting only selected hand crafted features is overcome by using custom designed CNN for learning effective features and detecting drowsiness given an input image of a driver. The driver will be continuously monitored by a webcam. The video captured is converted into a sequence of frames. For each frame the face and eye is detected using predefined classifiers available in opencv called haar cascade classifiers. Eye images are extracted and it is passed through a series of 2D convolutional layers, max-pooling layers and finally the fully connected dense layer classifies whether eyes are closed or not. A score is calculated based on eye closure. If both eyes are closed

consecutively in 15 frames then the system predicts as drowsy and an alarm sound is triggered to alert the driver.

3.2.1 Advantages of the Proposed System

1. Previous approaches could only make decisions based on carefully handcrafted features such as eye blinks and head gestures for detecting driver drowsiness.
2. CNN based feature representation learning approaches have been providing an automated and efficient set of learned features which help us to classify the driver as drowsy or non-drowsy very accurately.
3. The normalization issues in existing model is eliminated by using custom designed CNN.

3.3 REQUIREMENTS SPECIFICATION

3.3.1 Hardware Requirements

- Hard disk: 200GB & above
- RAM: 8GB
- Processor: Pentium IV & above

3.3.2 Software Requirements

- Python 3.7 or above versions
- Anaconda software
- Open cv
- Keras
- Tensor flow

3.4 LANGUAGE SPECIFICATION

3.4.1 ANACONDA DISTRIBUTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, data science, machine learning applications, large-scale data processing, predictive analytics, etc. that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS. Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda. Anaconda Navigator is a desktop Graphical User Interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels with command line commands.

Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, MacOS, and Linux.

The following applications are available by default in Navigator

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

Conda is open source, cross platform and environment management system that installs, runs, and updates packages and their dependencies. Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages. Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications.

Python is platform independent. Python balances high-level and low-level programming. Python has sets, lists, dictionaries, tuples, thread- safe queues, strings, etc. The open source libraries of Python cover the needs of almost any AI project. By using the libraries available, the developer does not have to write code from scratch and can simply use the libraries instead. Some of the most popular Python libraries are NumPy, pandas, NLTK, Tensorflow, Scikit-Learn and Matplotlib.

NumPy stands for Numerical python. High performance numerical and scientific calculations can be performed on multidimensional data. It provides a

high-level abstraction for numerical calculations. Numpy also consists of many smaller sub-packages for various mathematical tasks like linear algebra, FFTs, generating a random value, and polynomial manipulation. SciPy, another python library is built on numpy.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA.

CHAPTER 4

SYSTEM DESIGN

This chapter gives a description about system design, the components involved and various UML diagrams of the system.

4.1 SYSTEM ARCHITECTURE

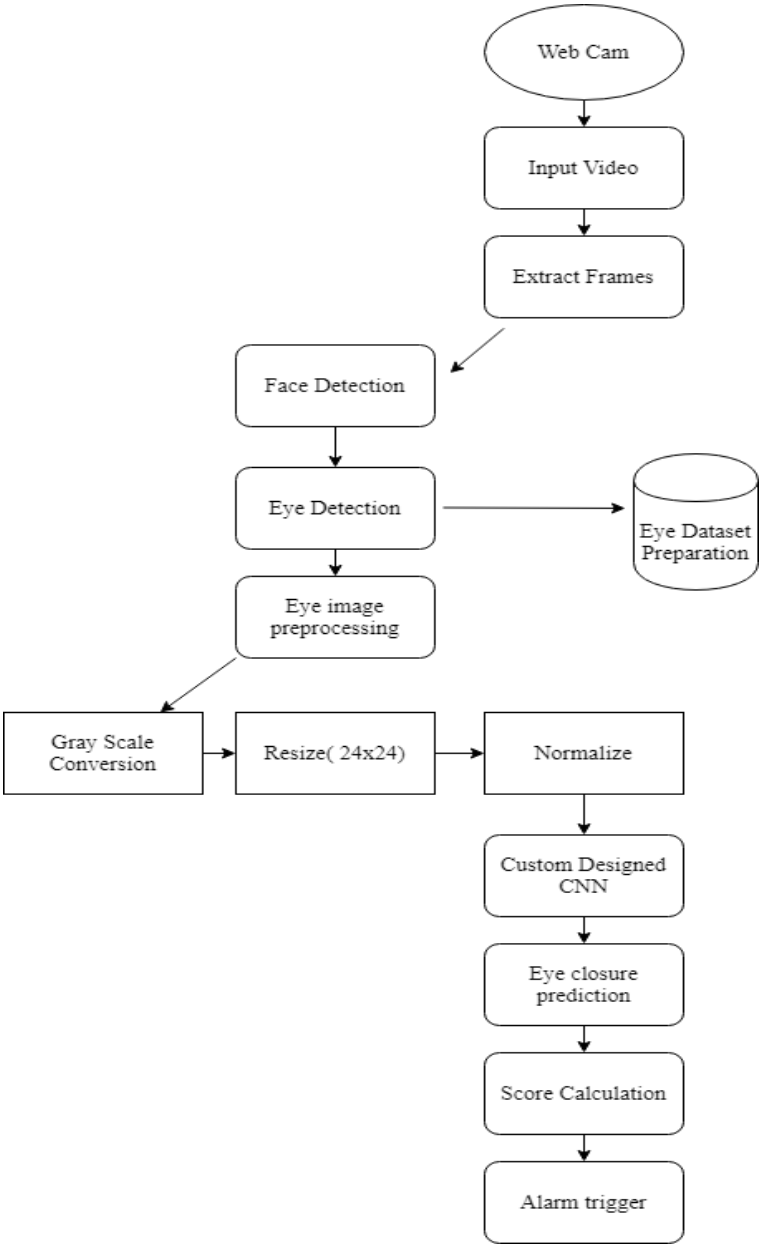


FIGURE 4.1.1 ARCHITECTURE OF PROPOSED SYSTEM

Figure 4.1.1 depicts the flow of the system to be created. In the initial step, driver is monitored by using a webcam. The video input is converted into a sequence of frames. In each frame drivers face and eyes are detected by using haar cascade classifiers. To extract image feature, which is robust to various backgrounds and environment changes from the input images, we adopt a custom designed CNN model. The detected eyes are stored as images to form dataset for CNN. The system also provides provision for the preparation of eye dataset to train CNN model. To train the image and to increase the number of datasets Data augmentation is done. The images of both eyes are then subjected to series of image preprocessing steps such as gray scale conversion, resizing and normalizing etc. It is then fed to pre-trained CNN model consisting of convolution layers, max- pooling layers and dense layers to predict eye closure. Based on the prediction, a score is calculated. If the system finds the driver as drowsy, then an alarm will be triggered to alert the driver.

4.2 USE CASE DIAGRAM

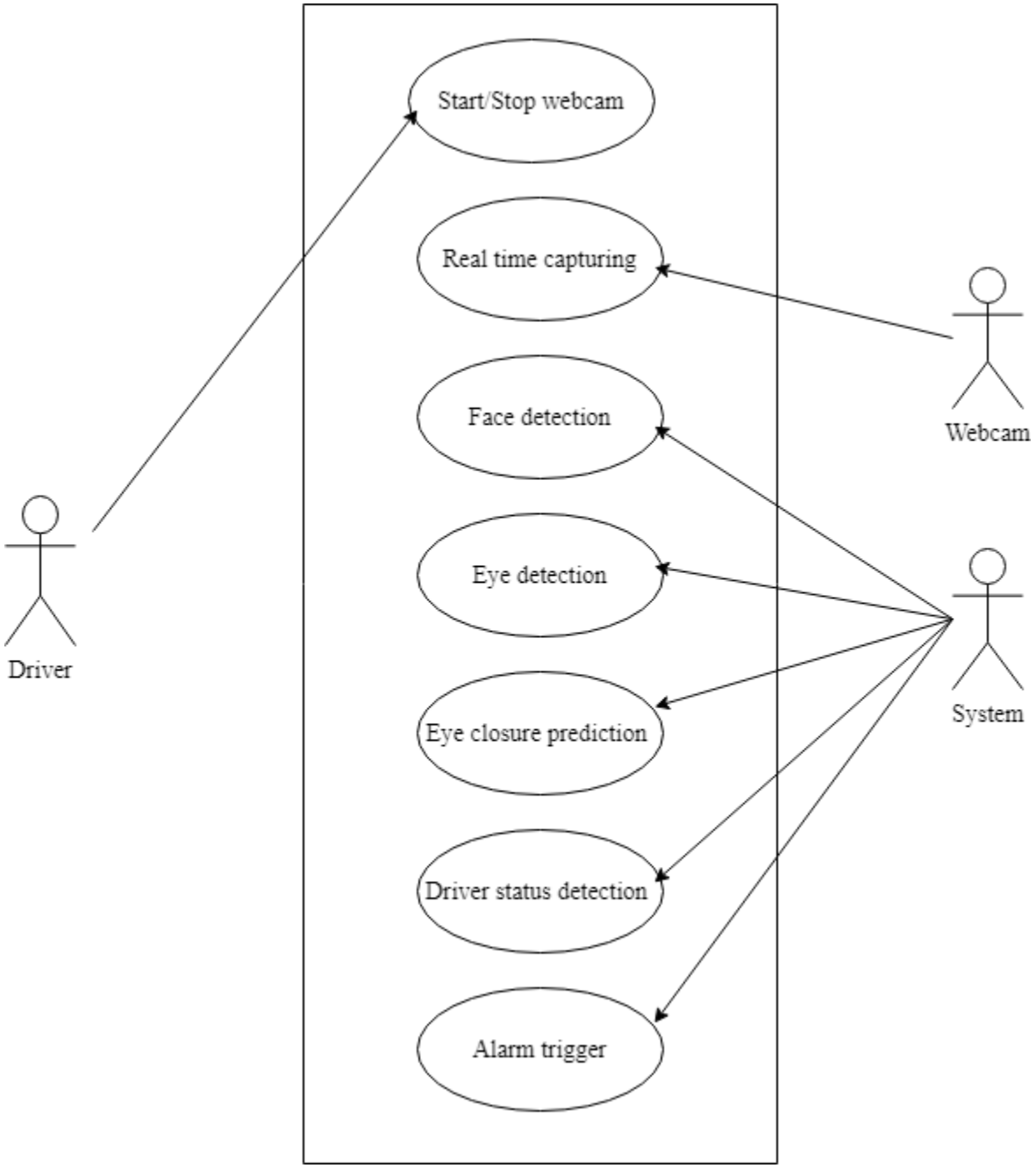


FIGURE 4.2.1 USE CASE DIAGRAM

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. A use case diagram at its core is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be followed by other types of diagrams as well. The use cases are depicted by either circles or ellipses. While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system.

Actors:

Driver: A person who drives a car.

Webcam: A device that is used to monitor and capture videos or images.

System: This is the proposed system that monitors driver’s drowsiness.

There are two major actors such as driver and system, one subordinate actor webcam to capture video input in this scenario. In the figure 4.2.1,start/stop webcam use case will start the webcam and real time capturing of video input will begin. After capturing video input face and eye detection is done by haar cascade classifiers. From the sequence of images facial and eye features are extracted and fed into the classifiers for eye closure prediction. If the eyes are closed in 15 consecutive images, the driver is detected as drowsy and an alarm will be triggered.

4.3 SEQUENCE DIAGRAM

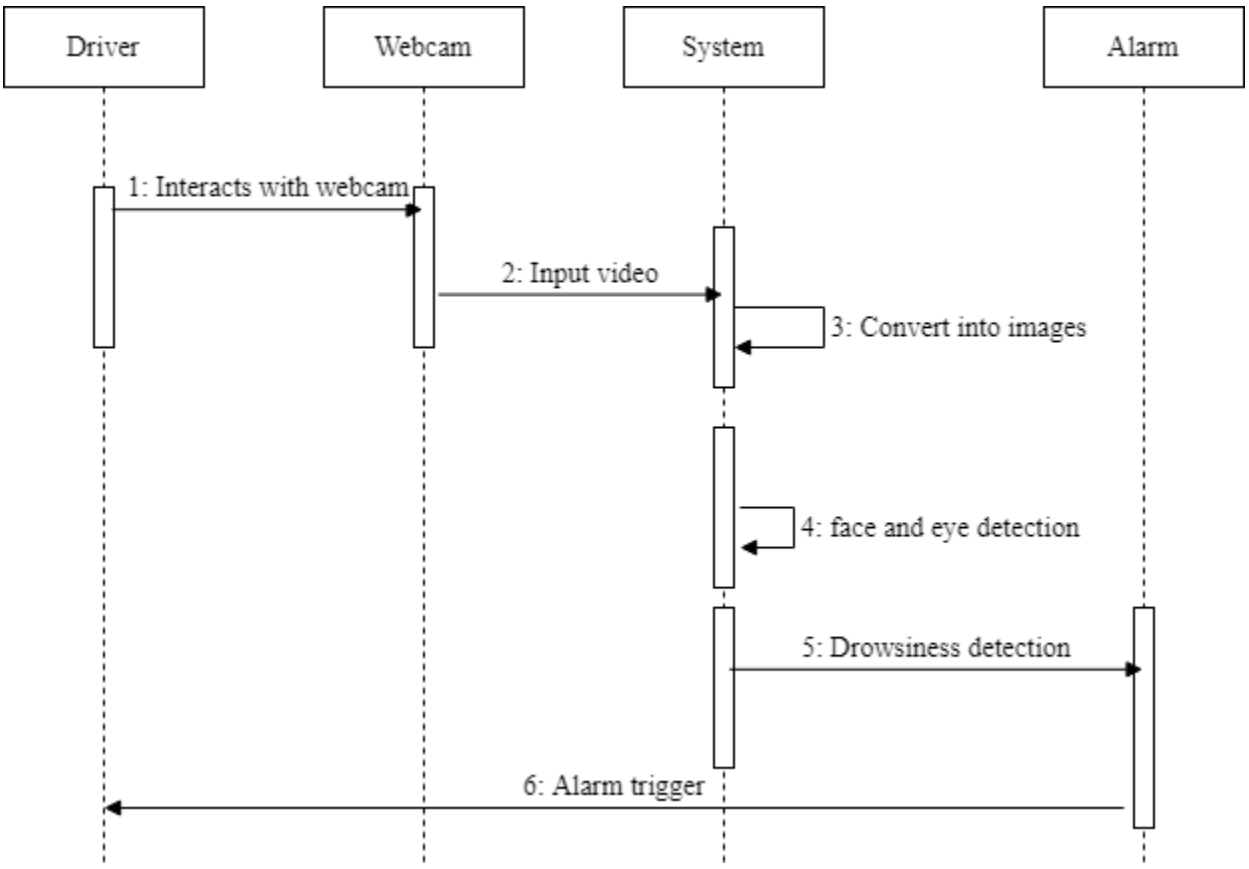


FIGURE 4.3.1 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. The figure 4.3.1 depicts the sequence of messages passed between different objects in the system. A sequence diagram illustrates use-case realizations
i.e. to show how objects interact to perform the behavior of all or part of a use case. One or more sequence diagrams may illustrate the object interactions, which enact a use case. Thus, we see the manner in which these entities interact and pass

messages to each other. The message passing for the four modules (use cases) outlined in the use case diagram is described in the sequence diagram.

4.4 STATE DIAGRAM

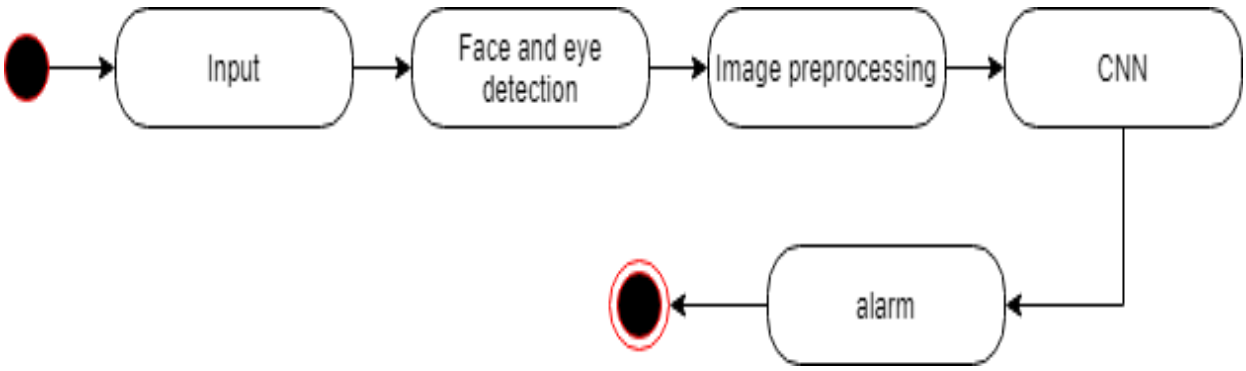


FIGURE 4.4.1 STATE DIAGRAM

In the Figure 4.4.1, the taken image frames are subjected to different states. Initially, it is subjected to input state, which takes the image for further processing. After the completion of input state, face and eye detection takes place. The eye images are subjected to image preprocessing state. At this state, the image is converted into a gray scale image, resized and normalized. The preprocessed image is passed through the layers of pre-trained CNN and it detects whether the driver is drowsy or not. The result is given to the alarm to trigger if the driver is found to be drowsy.

CHAPTER 5

MODULE DESCRIPTION

This chapter gives a description about List of modules along with the algorithms for each module.

The modules are

- Face and eye detection
- Preprocessing and labelling
- Data Augmentation
- Enhanced CNN
- Triggering Alarm

5.1 FACE AND EYE DETECTION

Haar Cascade Classifier

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in Figure 5.1.1 are used. They are just like our

convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

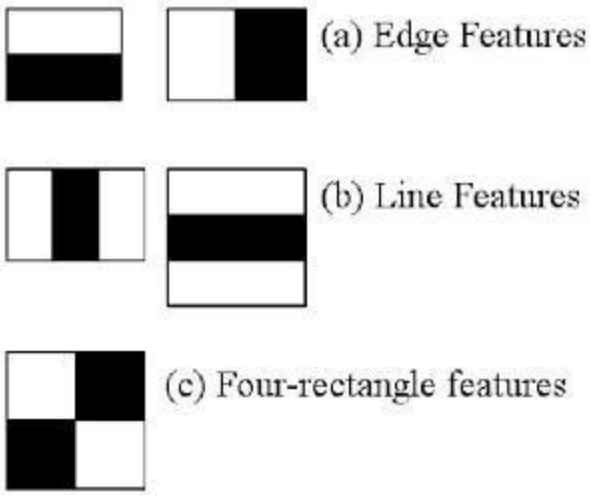


FIGURE 5.1.1 HAAR-FEATURES

OpenCV provides a training method or pretrained models, which can be read using the **cv::CascadeClassifier::load** method. The pretrained models are located in the data folder in the OpenCV installation .The system will use pretrained Haar cascade models to detect faces and eyes in an image. First, a **cv::CascadeClassifier** is created and the necessary XML file is loaded using the **cv::CascadeClassifier::load** method. Afterwards, the detection is done using the **cv::CascadeClassifier::detectMultiScale** method, which returns boundary rectangles for the detected faces or eyes.

5.2 PREPROCESSING AND LABELLING

Pre-processing is a common name for operations with images at the lowest level of abstraction. The aim of pre-processing is an improvement of the image data that suppresses unwilling distortions or enhances some image features important for further processing, although geometric transformations of images (e.g. rotation, scaling, and translation) are classified among preprocessing methods here since similar techniques are used.

Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subfield of digital signal processing, digital image processing has many advantages over analogue image processing. This allowed us to obtain a sufficient amount of data for both the alert and drowsy state. **Dataset Preparation**

Image datasets are collection of images that are used in machine learning and deep learning for training the model to find or identify particular action, the datasets are classified as training set and validation set. We prepared our own dataset by preprocessing and labelling the eyes images obtained through webcam as open and closed eyes.

5.3 DATA AUGMENTATION

Data augmentation adds value to base data by adding information derived from internal and external sources within an enterprise. Data is one of the core assets for an enterprise, making data management essential. Data augmentation can be applied to any form of data, but may be especially useful for customer data, sales patterns, product sales, where additional information can help provide more in-depth insight. Data augmentation can help reduce the manual intervention required to developed meaningful information and insight of business data, as well

as significantly enhance data quality. Data augmentation is of the last steps done in enterprise data management after monitoring, profiling and integration. Data augmentation is the process of increasing the amount and diversity of data. We do not collect new data, rather we transform the already present data

Some of the common techniques used in data augmentation include:

- **Extrapolation Technique:** Based on heuristics. The relevant fields are updated or provided with values.
- **Tagging Technique:** Common records are tagged to a group, making it easier to understand and differentiate for the group.
- **Aggregation Technique:** Using mathematical values of averages and means, values are estimated for relevant fields if needed
- **Probability Technique:** Based on heuristics and analytical statistics, values are populated based on the probability of events.

Data augmentation in Keras

Keras is a high-level machine learning framework build on top of TensorFlow. We can perform data augmentation by using the ImageDataGenerator class. It takes in various arguments like – rotation range, brightness range, shear range, zoom range etc. Figure 5.3.1 shows how images can be augmented using various parameters.

FIGURE 5.3.1 DATA AUGMENTATION OF IMAGES

5.4 ENHANCED CNN

To extract image feature which is robust to various backgrounds and environment changes from the input images, we adopt a custom designed CNN model. A layered convolutional neural networks can significantly outperform other methods for the task of large-scale image classification. Convolutional Neural Networks are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.

There are three types of layers in a convolutional neural network: convolutional layer, pooling layer, and fully connected layer. Each of these layers has different parameters that can be optimized and performs a different task on the input data. Figure 5.4.1 shows the enhanced CNN architecture used in this project.

Convolutional Layer

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling Layer

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

Fully Connected Layer

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

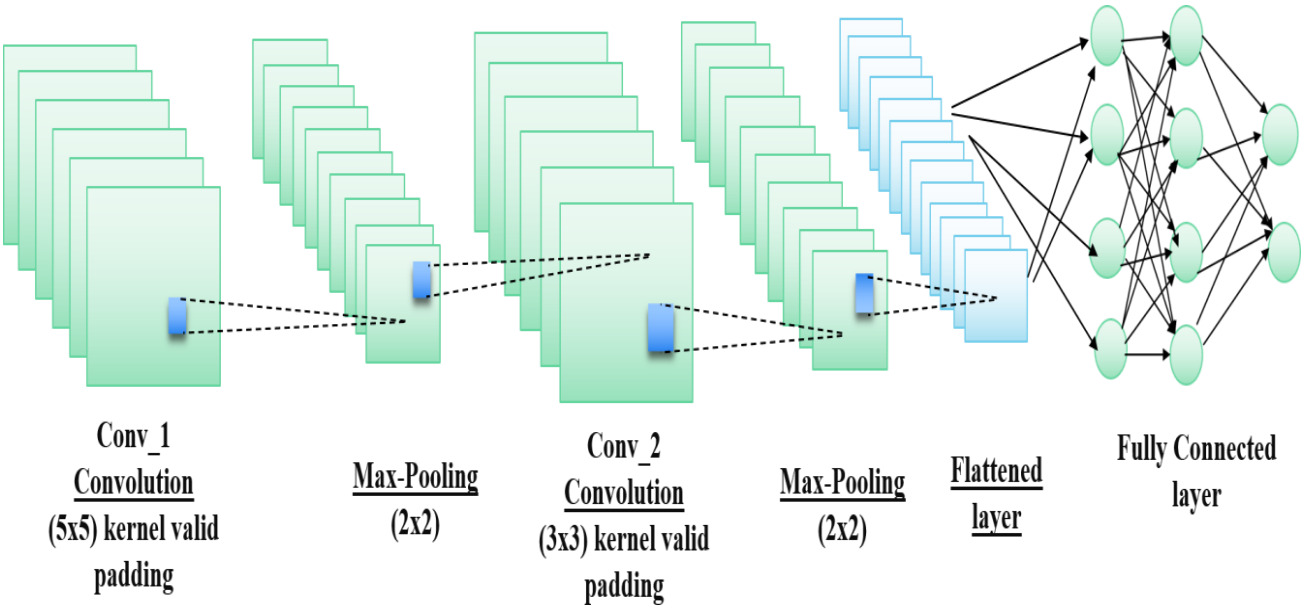


FIGURE 5.4.1 CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

5.5 TRIGGERING ALARM

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. In this project, Pygame library is used to alert the driver when he feels sleepy. The score is a value we will use to determine how long the person has closed his eyes. Therefore, if both eyes are closed, we will keep on increasing score and when eyes are open, we decrease the score. We are drawing the result on the screen using cv2.putText() function which will display real time status of the person.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

This chapter briefs about the conclusion and future enhancement of the project.

6.1 CONCLUSION

A framework for drowsiness detection based on an efficient convolutional neural network architecture, designed in order to detect drowsiness based on eye closure. Implementation starts by preparing image datasets for both open and closed eyes. 75% of the dataset is used for training the custom designed CNN and remaining 25% of the dataset is used for testing purpose. First, the input video is converted into frames and in each frame, the face and eyes are detected. The enhanced CNN have been providing an automated and efficient set of learned features that help us to classify the eyes as open or closed. If the eyes are closed in 15 consecutive frames, an alarm is triggered to alert the driver. The proposed CNN gives a training accuracy of 97% and testing accuracy of 67%.

6.2 FUTURE ENHANCEMENT

For future works, the face tracker algorithm can be applied instead of only using face detection. This system can be enhanced with additional face features instead of just eyes to give more accuracy in detection. We can also combine vehicle driving pattern information obtained using OBD (On-Board Diagnostics) sensors with the facial features extracted.

APPENDIX 1

SAMPLE CODING

FACE AND EYE DETECTION

```
import cv2
import numpy as np
from keras.models import load_model
from pygame import mixer
import time
#def generate_dataset(img,id):
# cv2.imwrite('E:\Driver_Drowsiness_project\Dataset\leye\eye'+str(id)+'.jpeg',img)
def draw_boundary(img, classifier,scaleFactor,minNeighbors,color,text):
    features = classifier.detectMultiScale(img,scaleFactor,minNeighbors)
    for(x,y,w,h)in features:
        cv2.rectangle(img,(x,y),(x+w,y+h),color,2)
        cv2.putText(img,text,(x,y-
4),cv2.FONT_HERSHEY_SIMPLEX,0.8,color,1,cv2.LINE_AA)
    return features
mixer.init()
sound = mixer.Sound(r'E:\Driver_Drowsiness_project\alarm.wav')
faceCascade=cv2.CascadeClassifier(r'C:\Users\DELL\Downloads\opencv\sources\
data\haarcascades\haarcascade_frontalface_default.xml')
lefteyeCascade=cv2.CascadeClassifier(r'C:\Users\DELL\Downloads\opencv\sourc
es\data\haarcascades\haarcascade_lefteye_2splits.xml')
righteyeCascade=cv2.CascadeClassifier(r'C:\Users\DELL\Downloads\opencv\sour
ces\data\haarcascades\haarcascade_righteye_2splits.xml')
lbl=['Close','Open']
```

```
model = load_model('E:\Driver_Drowsiness_project\eyes.h5')
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
IMG_SIZE=24
score=0
rpred=[99]
lpred=[99]
video_capture = cv2.VideoCapture(0)
while True:
    _, img = video_capture.read()
    height,width = img.shape[:2]
    color={"blue":(255,0,0),"red":(0,0,255),"green":(0,255,0)}
    #gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    coords=draw_boundary(img,faceCascade,1.1,10,color['blue'],"Face")
    lcoords=draw_boundary(img,lefteyeCascade,1.1,14,color['red'],"LEye")
    rcoords=draw_boundary(img,righteyeCascade,1.1,14,color['red'],"REye")
    for(lx,ly,lw,lh) in lcoords:
        roi_leye=img[ly:ly+lh,lx:lx+lw]
        roi_leye = cv2.resize(roi_leye, (IMG_SIZE,IMG_SIZE))
        x = np.array(roi_leye)
        x = np.expand_dims(x,axis=0)
        lpred = model.predict_classes(x)
        if(lpred[0]==1):
            lbl='Open'
        if(rpred[0]==0):
           lbl='Closed'
        break
    for(rx,ry,rw,rh) in rcoords:
```

```
        roi_reye=img[ry:ry+rh,rx:rx+rw]
        roi_reye = cv2.resize(roi_reye, (IMG_SIZE,IMG_SIZE))
        y = np.array(roi_reye)
        y = np.expand_dims(y,axis=0)
        rpred = model.predict_classes(y)
        if(rpred[0]==1):
            lbl='Open'
        if(rpred[0]==0):
           lbl='Closed'
        break
    if(rpred[0]==0 and lpred[0]==0):
        score=score+1
        cv2.putText(img,"Closed",(10,height-20),font,
1,(255,255,255),1,cv2.LINE_AA)
    else:
        score=score-1
        cv2.putText(img,"Open",(10,height-20),font,
1,(255,255,255),1,cv2.LINE_AA)
    if(score<0):
        score=0
        cv2.putText(img,'Score:'+str(score),(100,height-20),font,
1,(255,255,255),1,cv2.LINE_AA)
    if(score>15):
        cv2.putText(img,'Drowsy!!!',(200,height-20),font,
1,(255,255,255),1,cv2.LINE_AA)
        try:
            sound.play()
```

```
except:
            pass
    else:

        cv2.putText(img,'Alert',(200,height-20),font,
1,(255,255,255),1,cv2.LINE_AA)
        cv2.imshow("face detection",img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

CUSTOM CNN MODEL CODE

```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
```

```
sns.set(style='whitegrid',color_codes=True)

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
fromsklearn.metricsimport
accuracy_score,precision_score,recall_score,confusion_matrix,roc_curve,roc_auc_
score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator

#dl libraraies
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn
```

```
# specifically for manipulating zipped images and getting numpy arrays of pixel values of images.
import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image

X=[]
Z=[]
IMG_SIZE=24
c0='E:\Driver_Drowsiness_project\Dataset\closedeyes'
c1='E:\Driver_Drowsiness_project\Dataset\Openeyes'
def assign_label(img,ret_type):
    return ret_type
def make_train_data(ret_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,ret_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))
```

```
make_train_data('close',c0)
print(len(X))
make_train_data('open',c1)
print(len(X))
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,2)
X=np.array(X)
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)

model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',activation ='relu', input_shape = (IMG_SIZE,IMG_SIZE,3)))
model.add(MaxPooling2D(pool_size=(1,1)))
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(1,1)))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(1,1)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(2, activation = "softmax"))
model.summary()
datagen = ImageDataGenerator(
```

```
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # apply ZCA whitening
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.2, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.2, # randomly shift images vertically (fraction of total height)
    horizontal_flip=True, # randomly flip images
    vertical_flip=False) # randomly flip images
datagen.fit(X)
epochs=10
batch_size=16
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
History=model.fit_generator(datagen.flow(x_train,y_train,
batch_size=batch_size),epochs = epochs, validation_data = (x_test,y_test),verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)
model.save('E:\Driver_Drowsiness_project\eyes.h5', overwrite=True)
```

APPENDIX 2

SCREENSHOTS

INPUT FILE

Closed Eyes

Open Eyes

MODEL SUMMARY

```
In [12]: model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 36864)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               9437440
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 9,466,594
Trainable params: 9,466,594
Non-trainable params: 0
_________________________________________________________________
```

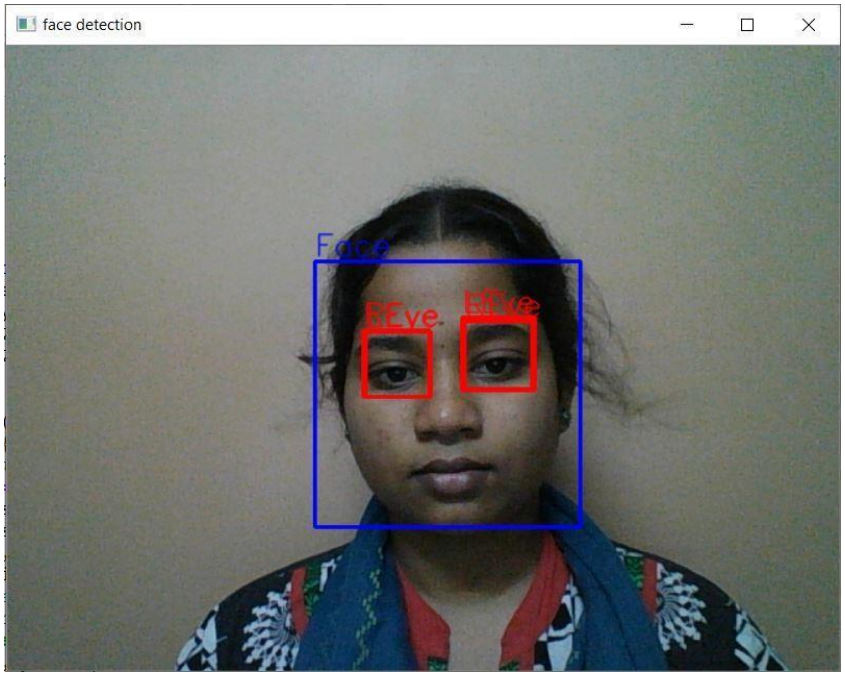
The model summary details about the layers involved in CNN, input shape, output shape and number of trainable parameters

OUTPUT

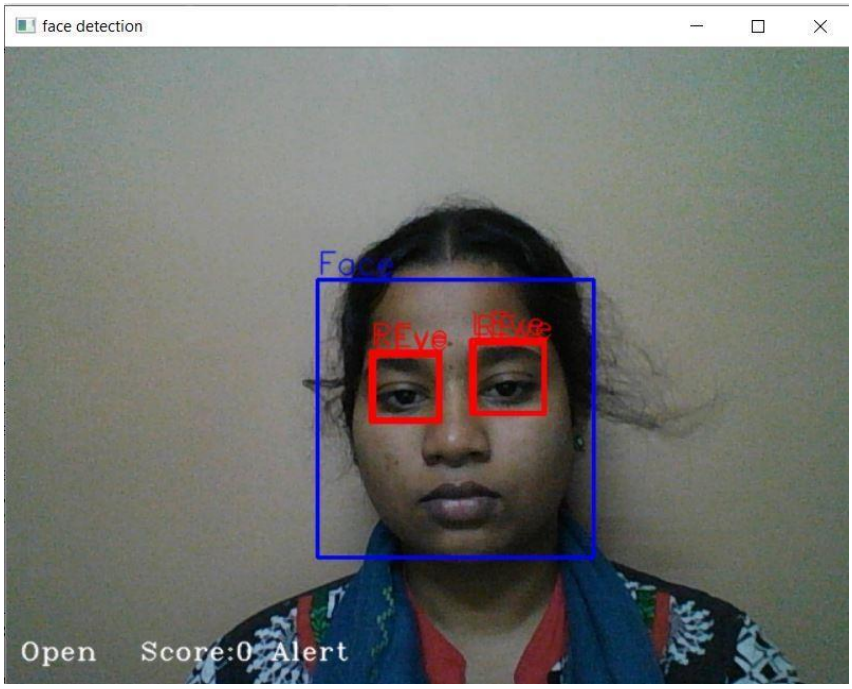
```
In [17]: History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                                       epochs = epochs, validation_data = (x_test,y_test),
                                       verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)

Epoch 1/10
187/187 [==============================] - 78s 419ms/step - loss: 3.3121 - accuracy: 0.6592 - val_loss: 0.4767 - val_accurac
y: 0.7720
Epoch 2/10
187/187 [==============================] - 75s 400ms/step - loss: 0.4647 - accuracy: 0.7842 - val_loss: 0.5745 - val_accurac
y: 0.7150
Epoch 3/10
187/187 [==============================] - 75s 403ms/step - loss: 0.6765 - accuracy: 0.5834 - val_loss: 0.6164 - val_accurac
y: 0.7560
Epoch 4/10
187/187 [==============================] - 75s 402ms/step - loss: 0.6637 - accuracy: 0.5942 - val_loss: 0.6605 - val_accurac
y: 0.5700
Epoch 5/10
187/187 [==============================] - 75s 401ms/step - loss: 0.6401 - accuracy: 0.6418 - val_loss: 0.4994 - val_accurac
y: 0.8010
Epoch 6/10
187/187 [==============================] - 75s 400ms/step - loss: 0.5622 - accuracy: 0.7162 - val_loss: 0.2739 - val_accurac
y: 0.9070
Epoch 7/10
187/187 [==============================] - 75s 401ms/step - loss: 0.4265 - accuracy: 0.8190 - val_loss: 0.2794 - val_accurac
y: 0.8710
Epoch 8/10
187/187 [==============================] - 75s 401ms/step - loss: 0.2384 - accuracy: 0.9209 - val_loss: 0.1446 - val_accurac
y: 0.9630
Epoch 9/10
187/187 [==============================] - 74s 398ms/step - loss: 0.1587 - accuracy: 0.9601 - val_loss: 0.0932 - val_accurac
y: 0.9760
Epoch 10/10
187/187 [==============================] - 74s 395ms/step - loss: 0.1139 - accuracy: 0.9729 - val_loss: 0.0321 - val_accurac
y: 0.9910
```

FACE AND EYE DETECTION

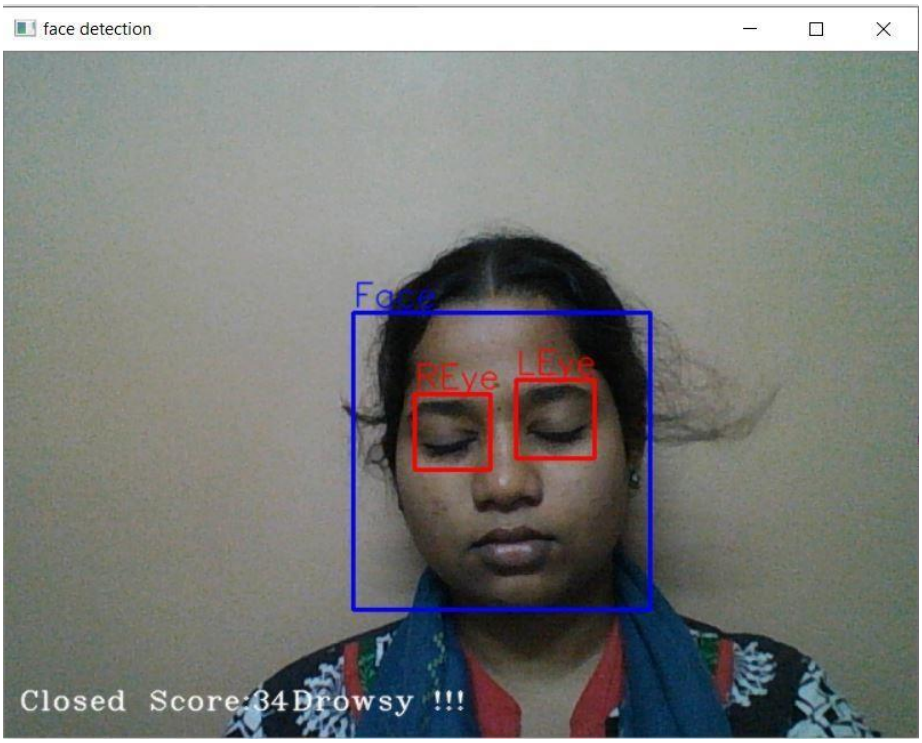
The image depicts the face and eye detection using haar-cascade classifier



The below image shows that both left and right eyes are open. Therefore, the person is in alert state.



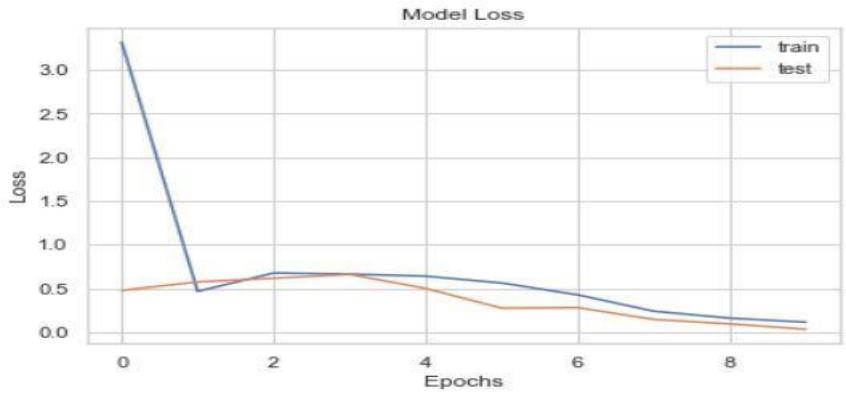
In the below image, both left and right eyes are closed. Therefore, the person is in drowsy state



PERFORMANCE MEASURES

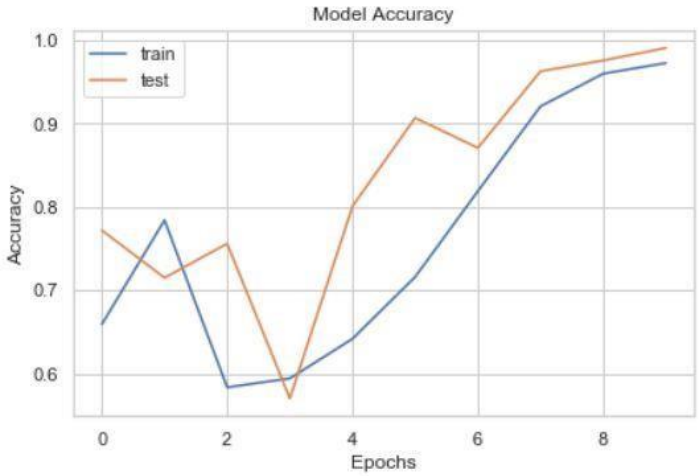
MODEL LOSS

In [18]:
```
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



MODEL ACCURACY

```
In [19]:
plt.plot(History.history['accuracy'])
plt.plot(History.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



TESTING

After training the CNN, testing is done with a dataset consisting of 1000 images of both closed and open eyes. A classification report is generated.

```
In [28]:
pred=model.predict(X)
pred_digits=np.argmax(pred,axis=1)
```

```
In [29]:
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix
report1 = metrics.classification_report(Y[:,1], pred_digits)
print(report1)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.34      0.51       500
         1.0       0.60      1.00      0.75       500

    accuracy                           0.67      1000
   macro avg       0.80      0.67      0.63      1000
weighted avg       0.80      0.67      0.63      1000
```

CONFUSION MATRIX

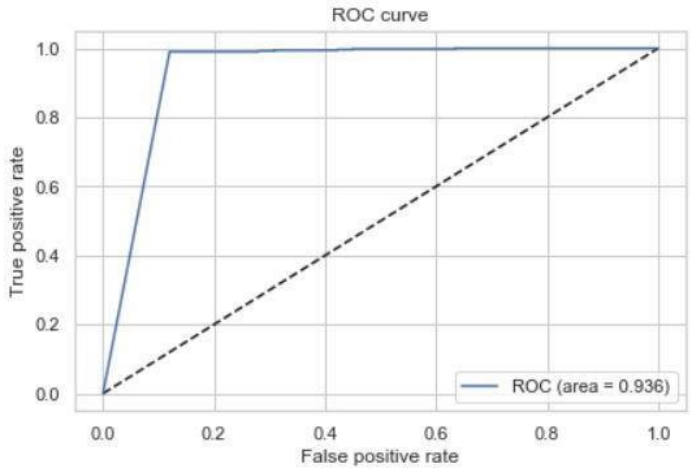
The below image shows the confusion matrix of true and false positive rate.

```
In [30]: confusion_matrix(Y[:,1], pred_digits)

Out[30]: array([[169, 331],
                [  0, 500]], dtype=int64)
```

ROC CURVE

A ROC curve is plotted with false positive rate on x-axis and true positive rate on y-axis.

```
In [31]: from sklearn.metrics import roc_curve, auc
fpr_keras, tpr_keras, thresholds_keras = roc_curve(Y[:,1], pred[:,1])
auc_rf = auc(fpr_keras, tpr_keras)
auc_rf

import matplotlib.pyplot as plt
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='ROC (area = {:.3f})'.format(auc_rf))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')

Out[31]: <matplotlib.legend.Legend at 0x1e083898f48>
```



REFERENCE

[1] Dr. Priya Gupta, Nidhi Saxena, Meetika Sharma, Jagriti Tripathi "Deep Neural Network for Human Face Recognition" Maharaja Agrasen College, University of Delhi, Vasundhara Enclave, Delhi - 110096, India January 2018, Vol.8, No.1, pp. 63-71.

[2] Kartik Dwivedi, Kumar,Biswaranjan and Amit Sethi "Drowsy Driver Detection using Representation Learning" 2014 IEEE, pp. 995-999.

[3] Ki Wan Kim, Hyung Gil Hong, Gi Pyo Nam and Kang Ryoung Park "A Study of Deep CNN-Based Classification of Open and Closed Eyes Using a Visible Light Camera Sensor"Sensors,2017

[4] Luigi Celona, Lorenzo Mammana, Simone Bianco, Raimondo Schettini "A Multi-Task CNN Framework for Driver Face Monitoring"2018 IEEE International Conference on Consumer Electronics

[5] Mandalapu Sarada Devi and Dr. Preeti R Bajaj "Driver Fatigue Detection Based on Eye Tracking "First International Conference on Emerging Trends in Engineering and Technology,2008, pp. 649-652.

[6] Sanghyuk Park, Fei Pan, Sunghun Kang and Chang D. Yoo"Driver drowsiness detection system based on feature representation learning using various deep networks"

[7] Tawsin Uddin Ahmed ,Sazzad Hossain,Mohammad Shahadat Hossain,Raihan Ul Islam ,Karl Andersson"Facial Expression Recognition using Convolutional Neural Network with Data Augmentation"(2019)

[8] Xiaoxi Ma, Lap-Pui Chau and Kim-Hui Yap "Depth Video-based Two-stream Convolutional Neural Networks for Driver Fatigue Detection"International Conference on OrangeTechnologies(ICOT) 2017 IEEE, pp. 155-158.

[9] Weiwei Zhang, Jinya Su" Driver Yawning Detection based on Long Short Term Memory Networks" (2017) IEEE

[10] Zhongke Gao , Xinmin Wang, Yuxuan Yang, Chaoxu Mu , Qing Cai, Weidong Dang , and Siyang Zuo"EEG-Based Spatial-Temporal Convolutional Neural Network for Driver Fatigue Evaluation"2019 IEEE Transactions on Neural networks and learning systems.