# DRIVER DROWSINESS DETECTION

**Submitted by**

**SuriyaPrakash K(MI)**

**(31019025)**

In partial fulfillment of the requirements for the award of Master Of Science
in Computer Science with Specialization in Machine Intelligence Of


Cochin University of Science and Technology, Kochi

Conducted by


Indian Institute of Information Technology and Management-Kerala
Technopark Campus, Thiruvanathapuram-695 581

# ACKNOWLEDGEMENT

I would like to express our deepest gratitude to **Dr. Alex James**, my project guide for providing me with the necessary facilities for the completion of this project. We are thankful for the valuable discussions we had at each phase of the project and for being a very supportive and encouraging project guide. We would also like to express our sincere and heartfelt gratitude to all people who were in one way or the other involved in my project. A lot of gratitude is reserved for the Director of the Institute for facilitating and nurturing an environment where I have been able to undertake this work. We would like to express my sincere thanks to all friends and classmates whose suggestions and creative criticism.

# TABLE OF CONTENT

**1.Abstract**

**2.Introduction**

- History of face recognition
- Face recognition

**3.Model Architecture**

**4.Methodology**

**5. Output**

**6.Future Work and Implementation**

# 1.ABSTRACT:

The following document is a report on the mini project of drowsiness detection. This report describes the drowsiness detection and recognition mini-project undertaken for the Machine Intelligence which is Elective for the Third Semester. It reports the technologies available in the Open-Computer-Vision (OpenCV) library and methodology to implement them using Python. For face detection, CNN were used and for face recognition Local binary pattern histograms were used. The methodology is described including flow charts for each stage of the system. Next, the results are shown followed by a discussion of encountered challenges. The report is concluded with the authors' opinion on the project and possible applications.

## 1, INTRODUCTION:

A countless number of people drive on the highway day and night. Taxi drivers, bus drivers, truck drivers and people traveling long-distance suffer from lack of sleep. Due to which it becomes very dangerous to drive when feeling sleepy.The majority of accidents happen due to the drowsiness of the driver. So, to prevent these accidents,I built a system using Python, OpenCV, and Keras which will alert the driver when he feels sleepy.

### 1.1,HISTORY OF FACE RECOGNITION:

Face recognition began as early as 1977 with the first automated system being introduced By Kanade using a feature vector of human faces In 1983, Sirovich and Kirby introduced the principal component analysis(PCA) for feature extraction . Using PCA, Turk and Pentland Eigenface was developed in 1991 and is considered a major milestone in technology [3]. Local binary pattern analysis for texture recognition was introduced in 1994 and is improved upon for facial recognition later by incorporating Histograms(LBPH) , . In 1996 Fisherface was developed using Linear discriminant analysis (LDA) for dimensional reduction and can identify faces in different illumination conditions, which was an issue in Eigenface method . Viola and Jones introduced a face detection technique using HAAR cascades and ADABoost . In 2007, A face recognition technique was developed by Naruniec and Skarbek using Gabor Jets that are similar to mammalian eye.

### 1.3 FACE RECOGNITION :

The following sections describe the face recognition algorithms Eigenface, Fisherface, Local binary pattern histogram and how they are implemented in OpenCV.

### EIGENFACE:

Eigenface is based on PCA that classify images to extract features using a set of images. It is important that the images are in the same lighting condition and the eyes match in each image. Also, images used in this method must contain the same number of pixels and in grayscale. For this example, consider an image with n x n pixels as shown in figure 4. Each raw is concatenated to create a vector, resulting a 1×n 2 matrix. All the images in the dataset

are stored in a single matrix resulting a matrix with columns corresponding the number of images. The matrix is averaged (normalised) to get an average human face. By subtracting the average face from each image vector unique features to each face are computed. In the resulting matrix, each column is a representation of the difference each face has to the average human face.The next step is computing the covariance matrix from the result. To obtain the Eigen vectors from the data, Eigen analysis is performed using principal component analysis. From the result, where covariance matrix is diagonal, where it has the highest variance is considered the 1st Eigen vector. 2nd Eigen vector is the direction of the next highest variance, and it is in 90 degrees to the 1st vector. 3rd will be the next highest variation, and so on. Each column is considered an image and visualised, resembles a face and called Eigenfaces. When a face is required to be recognised, the image is imported, resized to match the same dimensions of the test data as mentioned above. By projecting extracted features on to each of the Eigenfaces, weights can be calculated. These weights correspond to the similarity of the features extracted from the different image sets in the dataset to the features extracted from the input image. The input image can be identified as a face by comparing with the whole dataset. By comparing with each subset, the image can be identified as to which person it belongs to. By applying a threshold detection and identification can be controlled to eliminate false detection and recognition. PCA is sensitive to large numbers and assumes that the subspace is linear. If the same face is analysed under different lighting conditions, it will mix the values when distribution is calculated and cannot be effectively classified. This makes to different lighting conditions poses a problem in matching the features as they can change dramatically.

FISHERFACE:

Fisherface technique builds upon the Eigenface and is based on LDA derived from Ronald Fishers' linear discriminant technique used for pattern recognition. However, it uses labels for

classes as well as data point information . When reducing dimensions, PCA looks at the greatest variance, while LDA, using labels, looks at an interesting dimension such that, when you project to that dimension you maximise the difference between the mean of the classes normalised by their variance . LDA maximises the ratio of the between-class scatter and within-class scatter matrices. Due to this, different lighting conditions in images has a limited effect on the classification process using LDA technique. Eigenface maximises the variations while Fisherface maximises the mean distance between and different classes and minimises variation within classes. This enables LDA to differentiate between feature classes better than PCA and can be observed . Furthermore, it takes less amount of space and is the fastest algorithm in this project. Because of these PCA is more suitable for representation of a set of data while LDA is suitable for classification.

Local Binary Pattern Histogram Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.Using the LBP combined with histograms we can represent the face images with a simple data

vector.As LBP is a visual descriptor it can also be used for face recognition tasks, as can be seen in the following step-by-step explanation.

THE MODEL ARCHITECTURE:

The model we used is built with Keras using Convolutional Neural Networks **(CNN)**. A convolutional neural network is a special type of deep neural network which performs extremely well for image classification purposes. A CNN basically consists of an input layer, an output layer and a hidden layer which can have multiple numbers of layers. A convolution operation is performed on these layers using a filter that performs 2D matrix multiplication on the layer and filter.
The CNN model architecture consists of the following layers:

- Convolutional layer; 32 nodes, kernel size 3
- Convolutional layer; 32 nodes, kernel size 3
- Convolutional layer; 64 nodes, kernel size 3
- Fully connected layer; 128 nodes

The final layer is also a fully connected layer with 2 nodes. In all the layers, a Relu activation function is used.

PREREQUISITES:

(I)OpenCV-pip-install opencv(anaconda prompt)

(II)Tensorflow –pip install tensorflow

(III)keras-pip install keras

(IV)Pygame-pip install pygame(for alarm)

METHODOLOGY:

- The “haar cascade files” folder consists of the xml files that are needed to detect objects from the image. In our case, we are detecting the face and eyes of the person.
- The models folder contains our model file “cnnCat2.h5” which was trained on convolutional neural networks.
- We have an audio clip “alarm.wav” which is played when the person is feeling drowsy.

- “Model.py” file contains the program through which we built our classification model by training on our dataset. You could see the implementation of convolutional neural network in this file.

- “Drowsiness detection.pynb” is the main file of our project. To start the detection procedure, we have to run this file.

Let’s now understand how our algorithm works step by step.

**Step 1 – 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.

**Step 2 – 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.

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

**Step 3 – 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.

```
l_eye = frame[ y : y+h, x : x+w ]
```

**l_eye** only contains the image data of the eye. This will be fed into our CNN classifier which will predict if eyes are open or closed. Similarly, we will be extracting the right eye into **r_eye**.

**Step 4 – Classifier will Categorize whether Eyes are Open or Closed**

We are using 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 r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY). Then, we resize the image to 24*24 pixels as our model was trained on 24*24 pixel images cv2.resize(r_eye, (24,24)). 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 model = load_model('models/cnnCat2.h5') . Now we predict each eye with our model lpred = model.predict_classes(l_eye). 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.

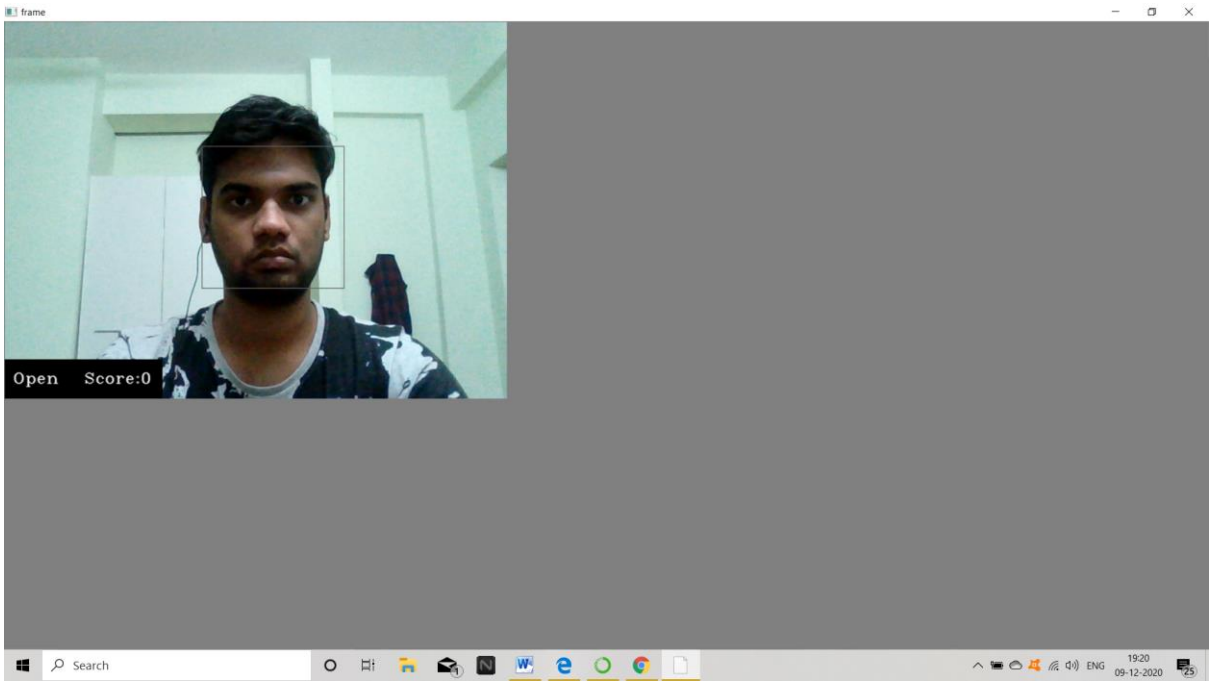
**Step 5 – Calculate Score to Check whether Person is Drowsy**

The score is basically a value we will use to determine how long the person has closed his eyes. So 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.

cv2.putText(frame, "Open", (10, height-20), font, 1, (255,255,255), 1, cv2.LINE_AA )
A threshold is defined for example if score becomes greater than 15 that means the person's eyes are closed for a long period of time. This is when we beep the alarm using sound.play()

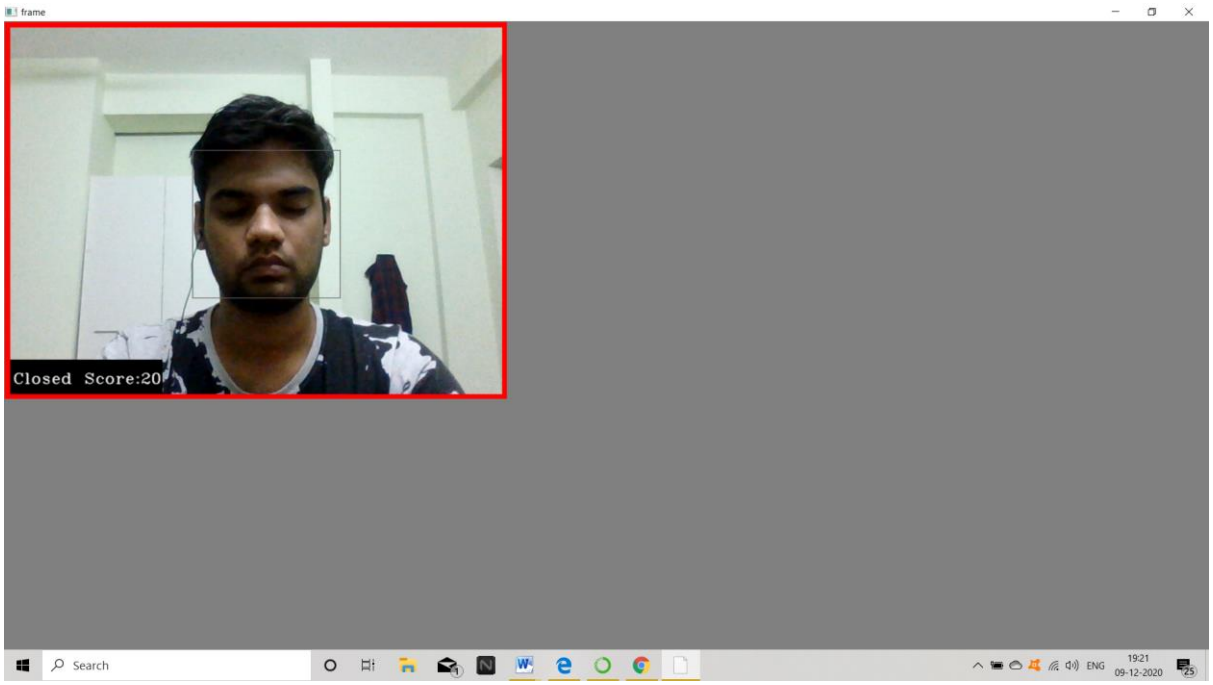
OUTPUT:

Eyes Open:



Eyes closed:

When the drowsiness score crosses the 15 second mark beeping sound will be heard till the drivers eye opens.




FUTURE WORK AND IMPLEMENTATION:

Am planning to implement gesture movements,like when the driver wakes and does a one eye close gesture ,the alarm sound will be stopped automatically and at that instant itself(Not waiting for the countdown to end).