# SYNOPSIS

Name: Lokita Varma

SAPID: 60004200140

Title of Synopsis: Computer Vision – Drowsiness detector

## 1. Introduction

One of the biggest reasons for deaths around the world is auto collisions. According to statistics, around1.3 million people breathe their last due to these accidents. A major reason for these mishaps is due to some sort of interruption or due to the drowsiness of the driver. As we are advancing in the technical industry in mankind, we have developed highways that allow inter-city or inter-state travel very easily and efficiently but what we did not realise is that it compromises on the safety of the driver. There are so many people who drive continuously for a long period of time which tends to the absence of rest. In addition to this, various other disturbances like a phone call, talking to another traveller, and so on may prompt a mishap. To avoid such adversities, we propose a framework which alerts the driver if he seems distracted or heavy-eyed. In other words, the target of this moderate Python venture is to conduct a drowsiness identification framework which will recognize if the driver's eyes are shut for a couple of moments and if true, then alert the driver. The basic approach to solve this issue is to extract the image of the face of the driver and classify them to determine of the individual's eyes are 'open' or 'closed'. If the eyes are shut for a significant amount of time, then it would start the alarming sound which we set.

## 2. Problem Definition and Algorithm

### 2.1 Task Definition

We are given a dataset of human eye images. After taking the imagesof the driver from the dashboard of a car, we are supposed to detect whether the driver is drowsy or not. If the eyes seem more closed than open, we can classify that driver as sleepy. For this problem the input is the image of the face of the driver (as mentioned earlier) and after the computer has read over the image, using the algorithm, it will give the output with the

image being categorized in one of two classes of ‘closed’ or ‘open’ where, as the name suggests, the ‘closed’ class will have images of eyes which are more closed than open and vice versa.

## 2.2 Algorithm Definition

### 2.2.1 Stage 1 – Take image as an input from a camera

We take the input of the image of the driver with the help of a camera(webcam). We can utilize the technique given by OpenCV called VideoCapture(). It is a class for video capturing from video files, image sequences or cameras. In our case we will be using cameras. It provides a C++ API for capturing video from cameras. Hence, we use cv2.VideoCapture () to take the image of the driver and store it in a variable.

### 2.2.2 Stage 2 – Detect the face structure in the image and create an ROI

To recognize the eyes from the face of the driver, we have to change over the picture into grayscale. This makes it a single-dimensional image and it only has different shades of gray. This single-dimensional property of the grayscale images is used to decrease the model’s training complexity. Hence the grayscale representation simplifies the algorithm and reduces computational requirements.

After converting the image to grayscale, we can use cv2.CascadeClassifier(). The deep cascade learning algorithm splits the network into its layers and trains each layer one by one until all the layers in the input architecture have been trained, however, if no architecture is given, one can use cascade learning to train as many layers as desired. Haar feature-based cascade classifiers is an effectual machine learning based approach, in which a cascade function is trained using a sample that contains a lot of positive and negative images. It is then used to detect objects in other images. In our case, the positive images will be images of faces and negative images will be images without faces. These positive and negative images are used to train the classifier.

We use v2.CascadeClassifier.detectMultiScale() to find faces or eyes. It returns a rectangle surrounding the face in the image and it is defined like so:
Cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbours[, minSize[, maxSize]]]])

Where the parameters are:

- **image**: image stands for matrix of the type CV_8U which contains an image where objects are detected.
- **scaleFactor**: it is a parameter which specifies how much should the image size be reduces by at each image scale. It is used to create a scale pyramid. If we reduce the size a little bit after resizing, we increase the chance of detecting the face.
- **minNeighbours**: Parameter specifying how many neighbours each candidate rectangle should have to retain it.
- **minSize**: Minimum possible object size. Objects smaller than that are ignored.
- **maxSize**: Maximum possible object size. Objects larger than that are ignored.

Hence, we can create a region of interest (ROI) for the face and apply eye detection on this ROI.

### 2.2.3 Stage 3 – Detect the eyes from the ROI and feed it to the classifier

A similar method used to detect faces is used to detect the eyes from the region of interest. 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.
We feed the information to the Convolutional Neural Network (CNN). A CNN is a multilayered neural network with a special architecture to detect complex features in data. CNNs have been used in image recognition, powering vision in robots, and for self-driving vehicles.

### 2.2.4 Stage 4 – Classifier will categorize whether the eyes are open or closed

We need to alter the image of the eye extracted in the previous step to fit the specifications that the model needs. To begin with, we convert the shading picture into grayscale using:
r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY)**.**

After this, we resize the picture to 24*24 pixels as our model was trained on 24*24 pixel images using:
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.

### 2.2.5 Stage 5 – Calculate score to check whether the person is drowsy

The score is basically a value we will use to decide to what extent the individual has shut 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.

For instance, if the score gets more prominent than 15 that means the individual's eyes are shut for a significant period. This is the point at which we start the caution sound utilizing sound.play()

## 3. Conclusion

In this synopsis, I have talked about building a drowsy-driver-alert system. I have used OpenCV to detect the faces and eyes using haar cascade classifier and then we used a CNN model to predict the status.

The result of the project is used to avoid accidents that are made by the drowsiness of the driver it avoided by using this project by alerting the driver when he/she feels sleepy

## 4. Bibliography


**https://docs.opencv.org/master/**

**https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php**

**https://heartbeat.fritz.ai/a-beginners-guide-to-convolutional-neural-networks-cnn-cf26c5ee17ed**

**https://www.cs.utexas.edu/~mooney/cs391L/paper-template.html**