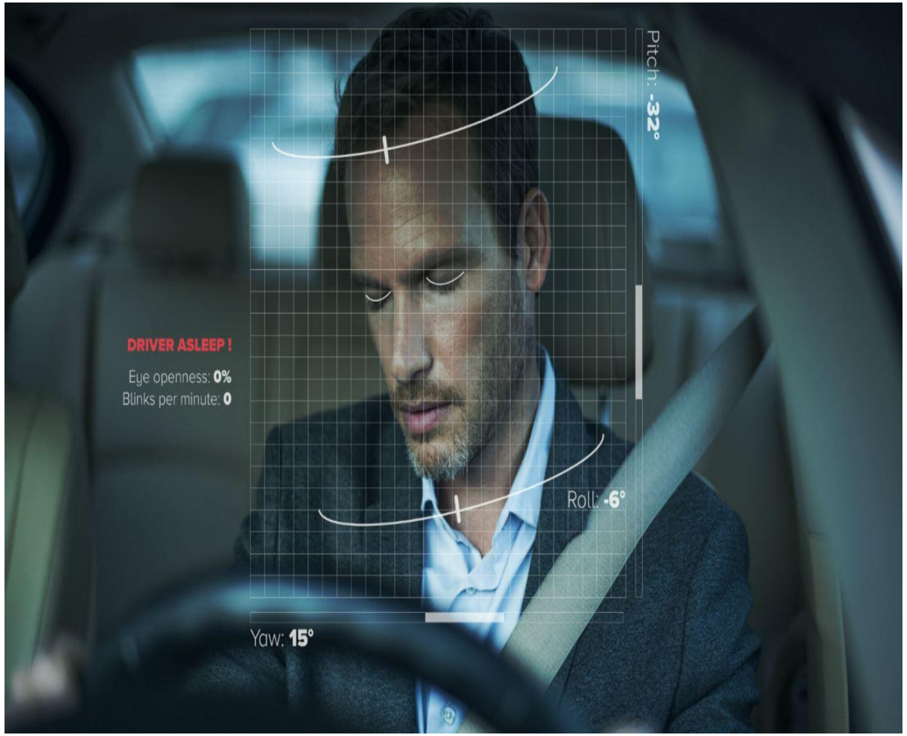
Capstone Project

Face Recognition and Drowsiness Detection

1. Lalit Ahirrao
2. Aniket Gajmal
3. Rushikesh Pawar
4. Prasad Ghegade
5. Samarth Gangurde

Table of Content

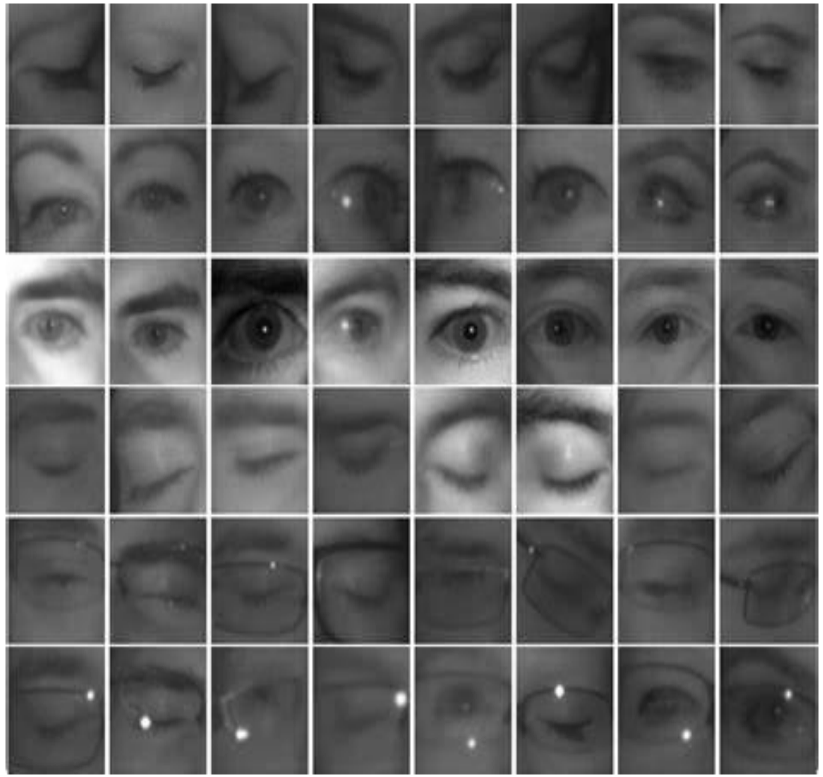


Reason Behind Project

National Highway Traffic Safety Administration (NHTSA), in 2017 drowsy driving led to at least 91,000 crashes, resulting in roughly 50,000 injuries and 800 deaths3.This data likely underestimates the impact of drowsy driving because it’s often impossible to definitively determine whether drowsy driving caused an accident, especially after fatal crashes other studies calculate that drowsy driving causes up to 6,000 deadly crashes every year

Dataset Information

MRL Eye Dataset, the large-scale dataset of human eye images. This dataset contains infrared images in low and high resolution, all captured in various lightning conditions and by different devices. The dataset is suitable for testing several features or trainable classifiers. In order to simplify the comparison of algorithms, the images are divided into several categories, which also makes them suitable for training and testing classifiers.



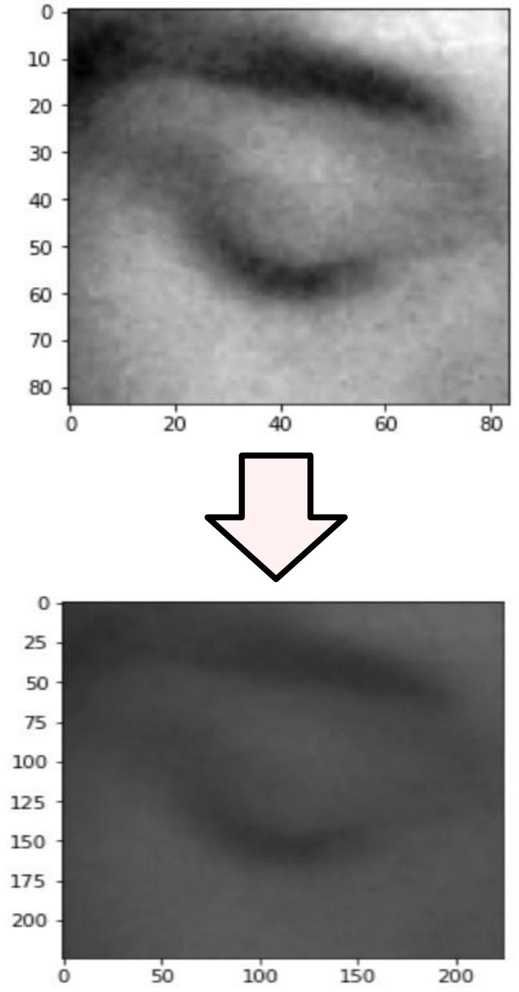
Preprocessing

- **Reshape**

MRL Images we used for the project had size 86-86 pixels ,we changed them to 225-225 pixels so we can feed it to our algorithms

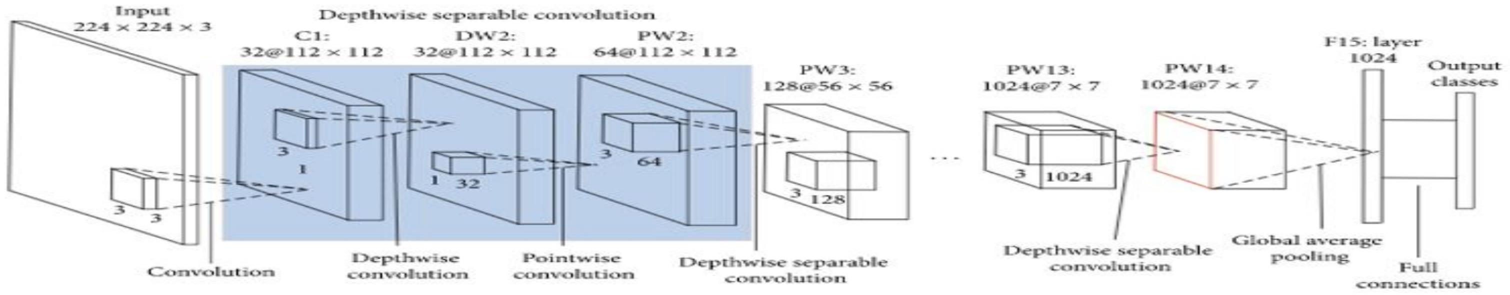
- **Normalization**

If pixel value of “0” means the pixel is white in colour and pixel value of “255” means the pixel in black in colour To normalize the scale we divide each pixel valued by 255 and the resultant value will be a value between 0 to 1



Algorithms Used

MobileNet:-



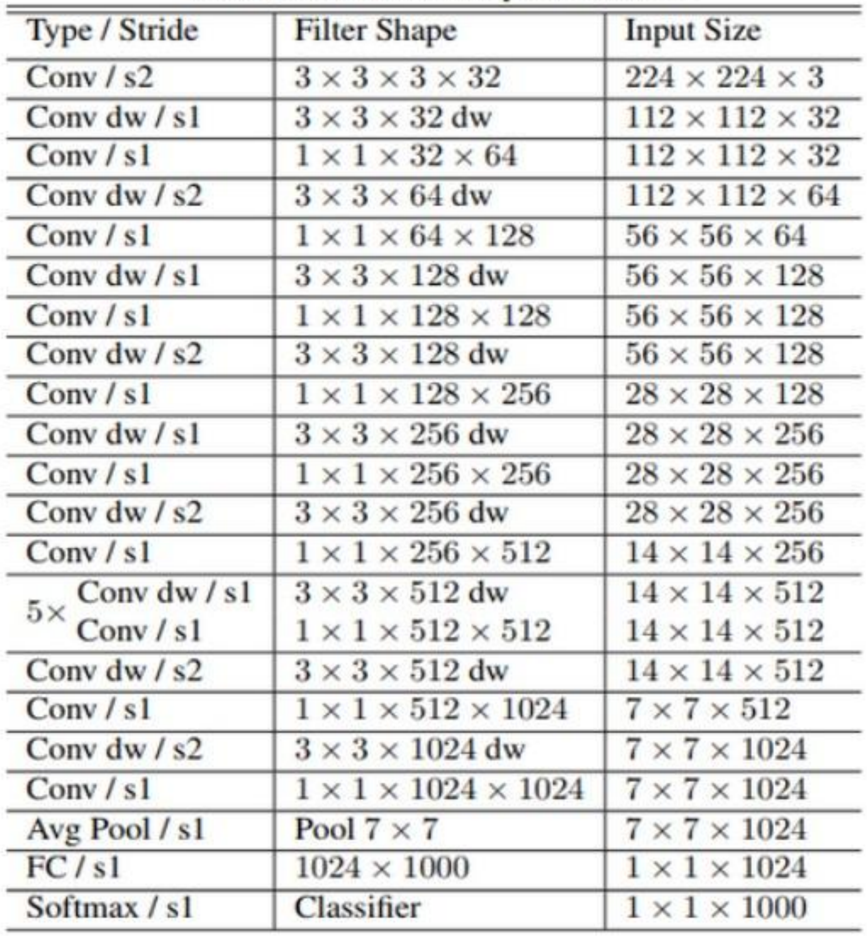
MobileNet-V2 Architecture

Chiung-Yu Chen

The MobileNet was proposed as a deep learning model by Andrew G. Howard et al of Google Research team in their research work entitled “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. This model was proposed as a family of mobile-first computer vision models for TensorFlow, designed to effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application. MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases

MobileNet Architecture

Table 1. MobileNet Body Architecture



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

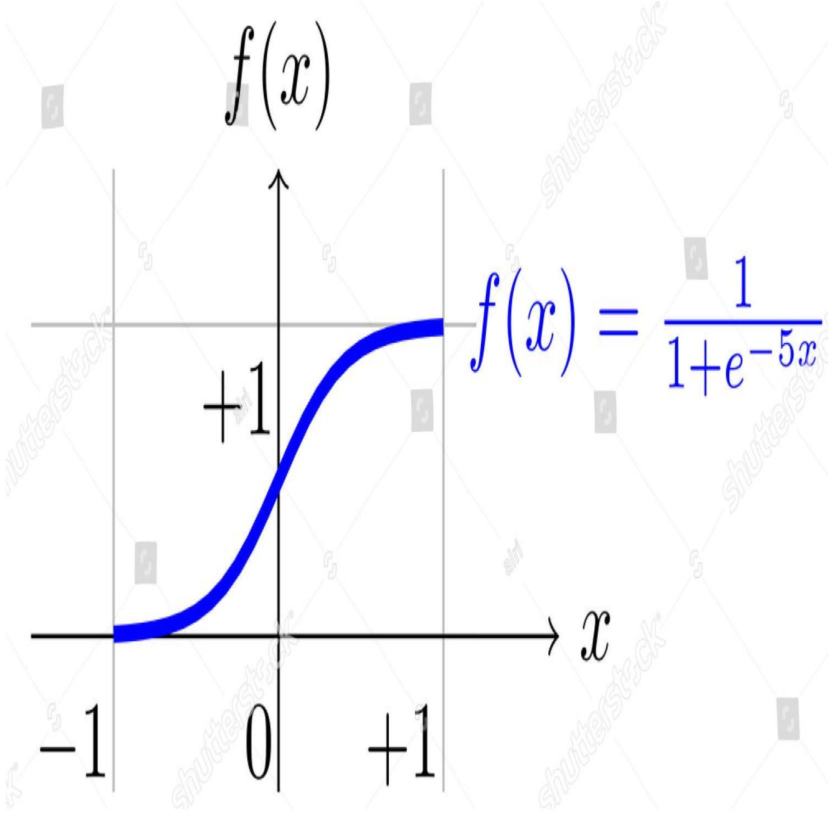
Mobilenet has 27 Convolutions layers which includes 13 depthwise Convolution, 1 Average Pool layer, 1 Fully Connected layer and 1 Softmax Layer.
In terms of Convolution layers, there are:

- 13 3x3 Depthwise Convolution
- 1 3x3 Convolution
- 13 1x1 Convolution

95% of the time is spent in 1x1 Convolution in MobileNet.

Sigmoid function:-

One of the most widely used sigmoid functions is the logistic function, which maps any real value to the range (0, 1). Note the characteristic S-shape which gave sigmoid functions their name (from the Greek letter sigma).Sigmoid functions are also useful for many machine learning applications where a real number needs to be converted to a probability. A sigmoid function placed as the last layer of a deep learning model can serve to convert the model's output into a probability score, which can be easier to work with and interpret.

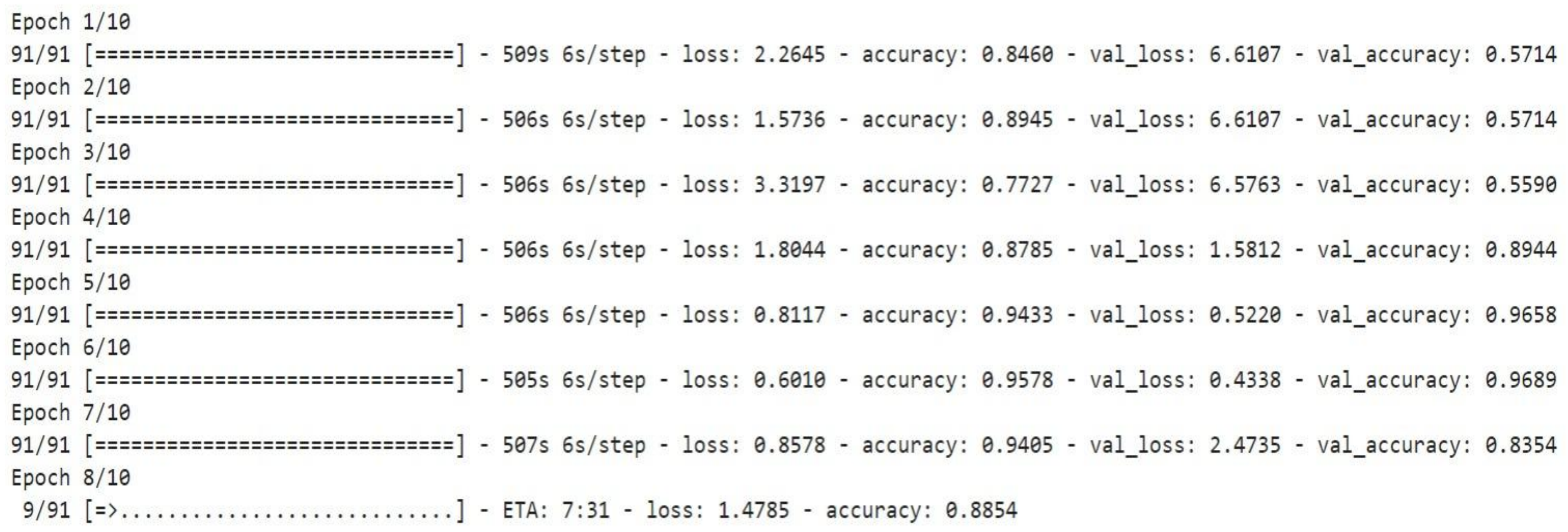


Adam Optimiser

It is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper titled Adam: A Method for Stochastic Optimization.The algorithm is called Adam. It is not an acronym and is not written as ADAM.the name Adam is derived from adaptive moment estimation.Stochastic gradient descent maintains a single learning rate for all weight updates and the learning rate does not change during training.

A learning rate is maintained for each network weight and separately adapted as learning unfolds.The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients

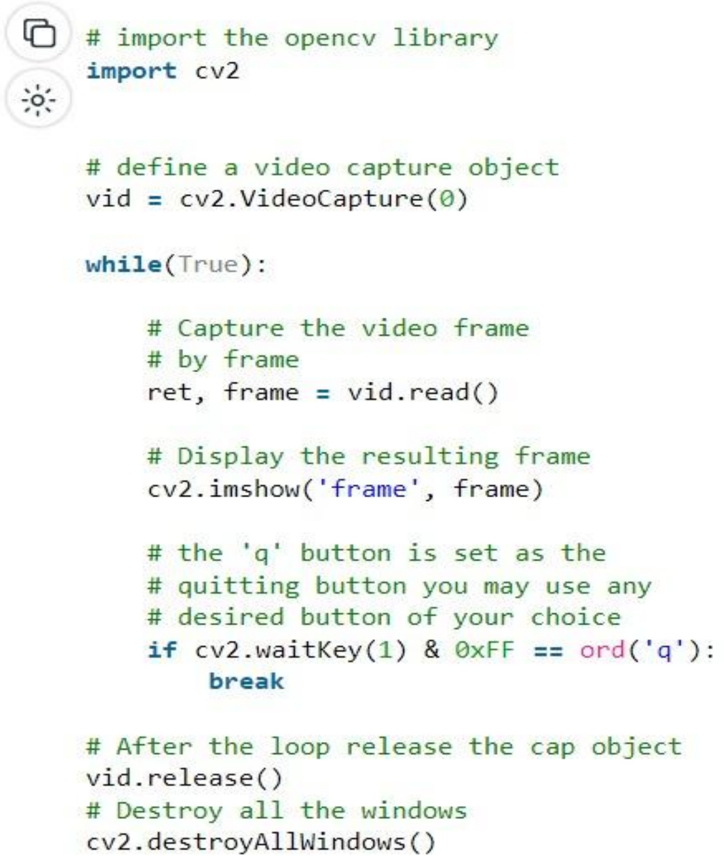
Model Accuracy :-



```
Epoch 1/10
91/91 [==============================] - 509s 6s/step - loss: 2.2645 - accuracy: 0.8460 - val_loss: 6.6107 - val_accuracy: 0.5714
Epoch 2/10
91/91 [==============================] - 506s 6s/step - loss: 1.5736 - accuracy: 0.8945 - val_loss: 6.6107 - val_accuracy: 0.5714
Epoch 3/10
91/91 [==============================] - 506s 6s/step - loss: 3.3197 - accuracy: 0.7727 - val_loss: 6.5763 - val_accuracy: 0.5590
Epoch 4/10
91/91 [==============================] - 506s 6s/step - loss: 1.8044 - accuracy: 0.8785 - val_loss: 1.5812 - val_accuracy: 0.8944
Epoch 5/10
91/91 [==============================] - 506s 6s/step - loss: 0.8117 - accuracy: 0.9433 - val_loss: 0.5220 - val_accuracy: 0.9658
Epoch 6/10
91/91 [==============================] - 505s 6s/step - loss: 0.6010 - accuracy: 0.9578 - val_loss: 0.4338 - val_accuracy: 0.9689
Epoch 7/10
91/91 [==============================] - 507s 6s/step - loss: 0.8578 - accuracy: 0.9405 - val_loss: 2.4735 - val_accuracy: 0.8354
Epoch 8/10
 9/91 [=>............................] - ETA: 7:31 - loss: 1.4785 - accuracy: 0.8854
```

CV2 VideoCapture:-

Python provides various libraries for image and video processing. One of them is OpenCV. OpenCV is a vast library that helps in providing various functions for image and video operations. With OpenCV, we can capture a video from the camera. It lets you create a video capture object which is helpful to capture videos through webcam and then you may perform desired operations on that video.I have also provided the code for cv2 VedioCapture



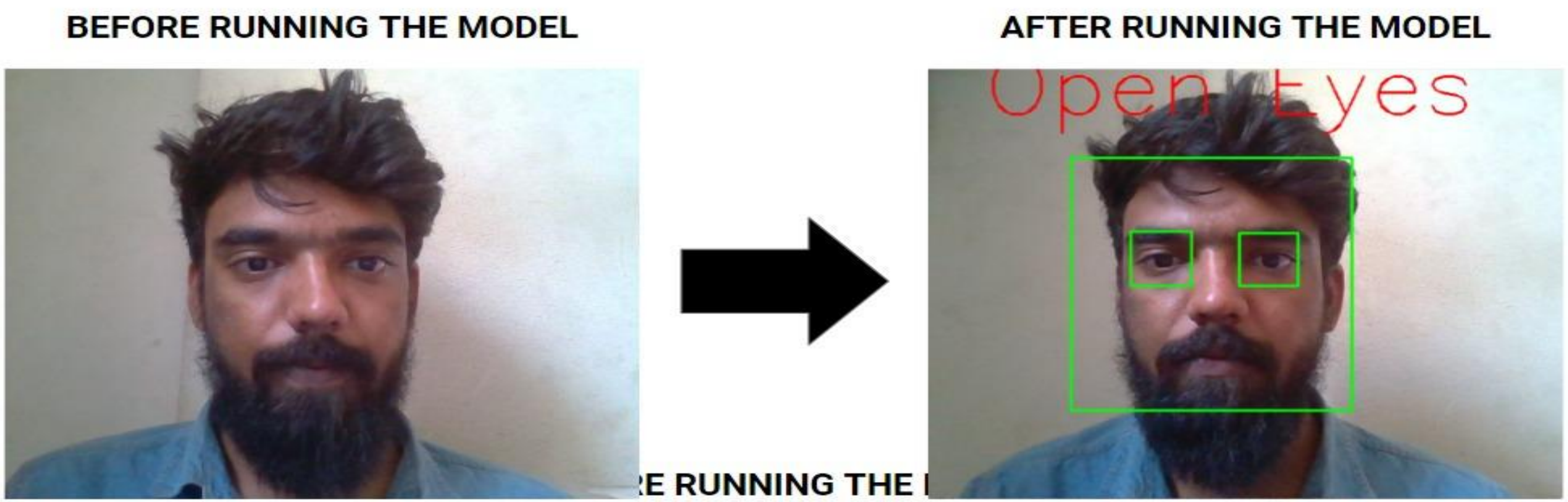
```
# import the opencv library
import cv2


# define a video capture object
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    # Display the resulting frame
    cv2.imshow('frame', frame)

    # the 'q' button is set as the
    # quitting button you may use any
    # desired button of your choice
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# After the loop release the cap object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

Haar Cascade files:-

HAAR cascade is a feature-based algorithm for object detection that was proposed in 2001 by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features. The original implementation is used to detect the frontal face and its features like Eyes, Nose, and Mouth. However, there pre-trained HAAR cascade available in their GitHub for other objects as well like for full body, upper body, lower body, smile, and many more

- **Step 1:** The image that has been sent to the classifier is divided into small parts or subwindows as shown in the illustration
- **Step 2:** We put N no of detectors in a cascading manner where each learns a combination of different types of features from images e.g. line, edge, circle, square that are passed through. Supposedly when the feature extraction is done each sub-part is assigned a confidence value.
- **Step 3:** Images or sub-images with the highest confidence are detected as face and are sent to the accumulator while the rest are rejected. Thus the cascade fetches the next frame/image if remaining and starts the process again.

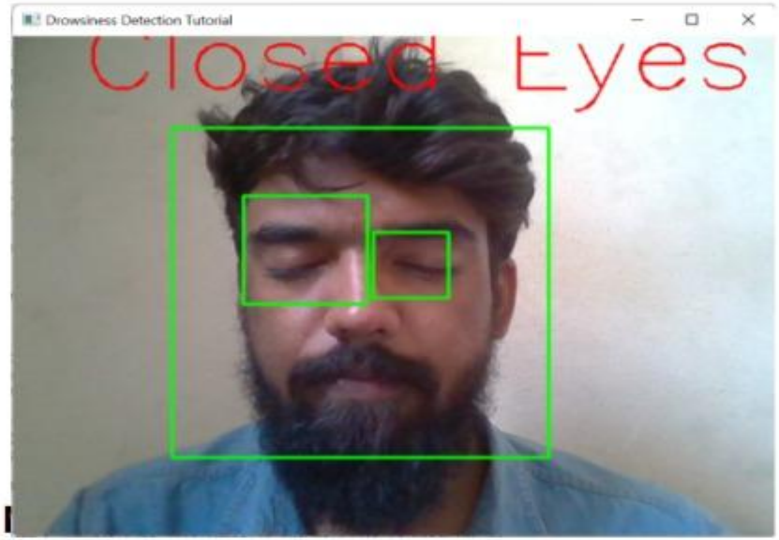
Model Deployment:-



While deploying our model we used openCV for capturing face. haar cascade files detected face and eyes by using their detection we predicted the values positive or negative if its positive its open eyes and if it's negative its close eyes

BEFORE RUNNING THE MODEL

AFTER RUNNING THE MODEL



E RUNNING THE

Here our model predicted the Negative values so our result become closed eyes

Challenges:-

- Handling large image MRL dataset with 84898 images
- Transfering image dataset to Closed_eyes and Opened_eyes category
- Changed size of image 86*86 to 225*225 pixels
- Normalization of dataset by dividing it with 225 to scale down it to 0-1 values
- Defining the input and output layer of the model
- Capturing vedioand image with open cv
- Model development, deployment on streamlit

Conclusion:-

Detectionion of driver drowsiness is a crucial problem in advanced driver assistance systems, because around 22–24% of road accidents are caused by driver being sleepy. A drowsiness detection solution should be very accurate and run in real-time. All the existing deep learning solutions for drowsiness detection are computationally intensive and cannot be easily implemented on embedded devices in this project we used MRL Dataset we started our project by importing our dataset we divided all the image in opened_eye and closed_eyes category for our easy understanding and displayed some of them,As we had images of 86-86 pixels we converted them to 225-225 pixels for feeding our model.then further we assigned all the features in X variable and all the labels in Y variable. then we normalised the X-variable by dividing it 255 so we can scale down it to 0-1 values further we moved to model training where we used mobile net as our algorithm with adam optimiser and sigmoid function which us lowest as accuracy 57% and highest accuracy as 88%, coming to or prediction system we imported opened eyes image for prediction and it gave us positive value as 23.87 that means image belongs Open_eyes category

Thank you