

DRIVER ALERT SYSTEM

By

Uppanapalli Lakshmi Sowjanya (20BAI1289)

Mandla Sheshi Kiran Reddy (20BAI1061)

A project report submitted to

Dr. G. BHARADWAJA KUMAR

COMPUTER SCIENCE IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

in partial fulfilment of the requirements for the course of

CSE1015 – MACHINE LEARNING ESSENTIALS

in

B.Tech. COMPUTER SCIENCE AND ENGINEERING



Vandalur – Kelambakkam Road

Chennai – 600127

APRIL 2022

ABSTRACT

In a country like India there are many accidents happening every-day in the highways and on roads due to overspeeding, drunk and drive and also due to the sleep of drivers in the night times.

This project aims to alert the driver whenever he is drowsy by predicting whether he is drowsy by the eyes recognition and check whether they are opened or not and after certain timer specified by the car company the alert system goes on in the car which alerts the driver.

Built an eye recognition model with the help of the MLR-EYE dataset and with the help of deep learning architecture with the Transfer learning technique and the model used is the MobileNet.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. G. Bharadwaja Kumar**, Senior Professor, School of Computer Science Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our **Head of The Dept Dr. Priyadarshini J (for B.Tech – Artificial Intelligence and Machine Learning)** for her support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

SHESHI KIRAN REDDY

LAKSHMI SOWJANYA

TABLE OF CONTENTS

1. Abstract
2. Introduction
3. Steps Of Proposed Methodology
4. Dataset Description
5. Description Of Methods Used
6. Exploratory Analysis along with the Code explanation with Steps
7. Results
8. Conclusion
9. References

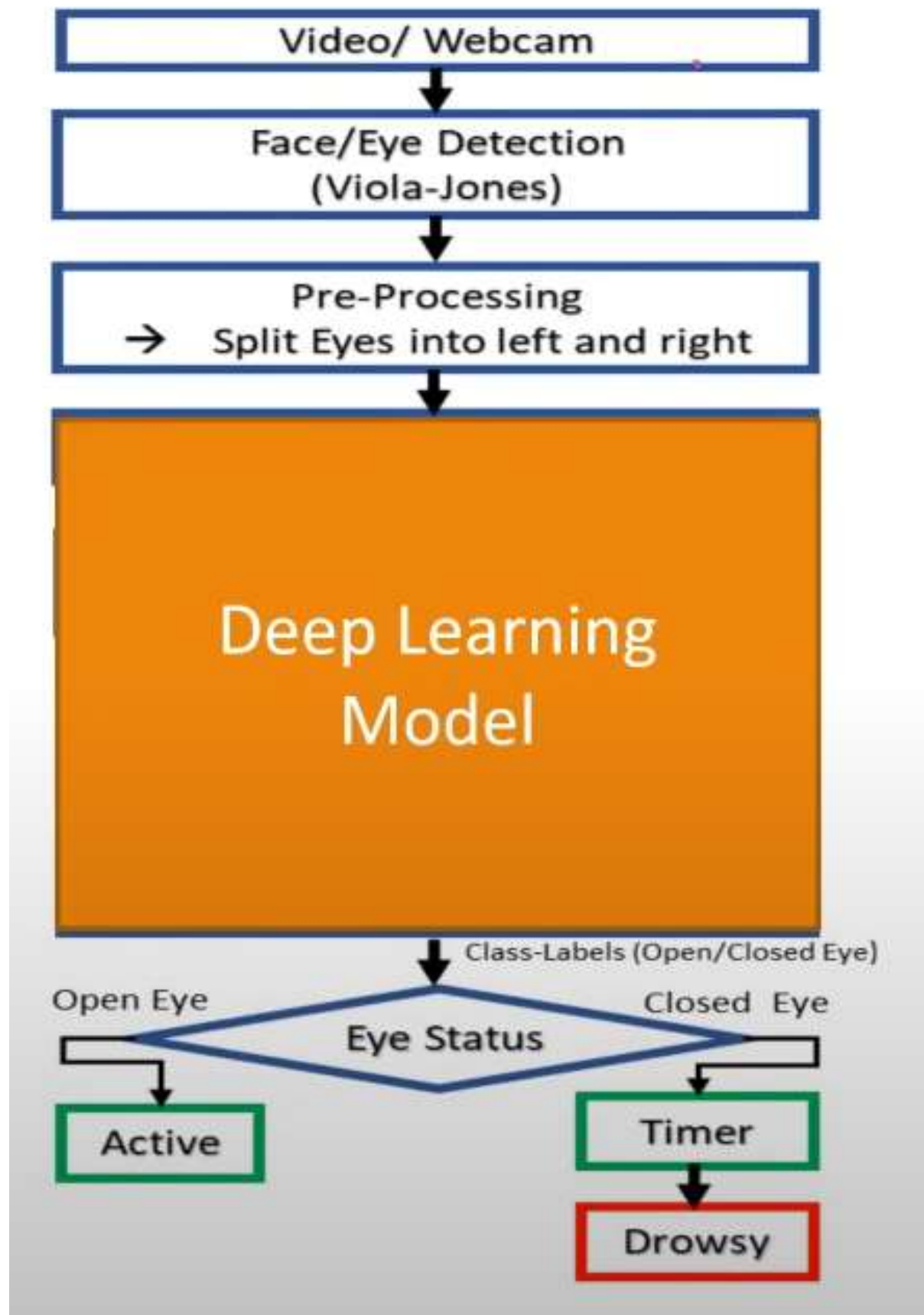
INTRODUCTION

Driver drowsiness detection is a car safety technology which helps prevent accidents caused by the driver getting drowsy. Various studies have suggested that around 20% of all road accidents are fatigue-related, up to 50% on certain roads. 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. Some of the current systems learn driver patterns and can detect when a driver is becoming drowsy.

With this Python project, we will be making a drowsiness detection system using Python, OpenCV, Keras, with Transfer Learning technique and using MobileNet model which will alert the driver when he feels sleepy.

STEPS FOR PROPOSED METHODOLOGY

- Changes In The Dataset / Dataclening
- Performing Transfer Learning Technique
- Training The Mobilenet Model
- Model Evaluation
- Prediction Model



DATASET DESCRIPTION

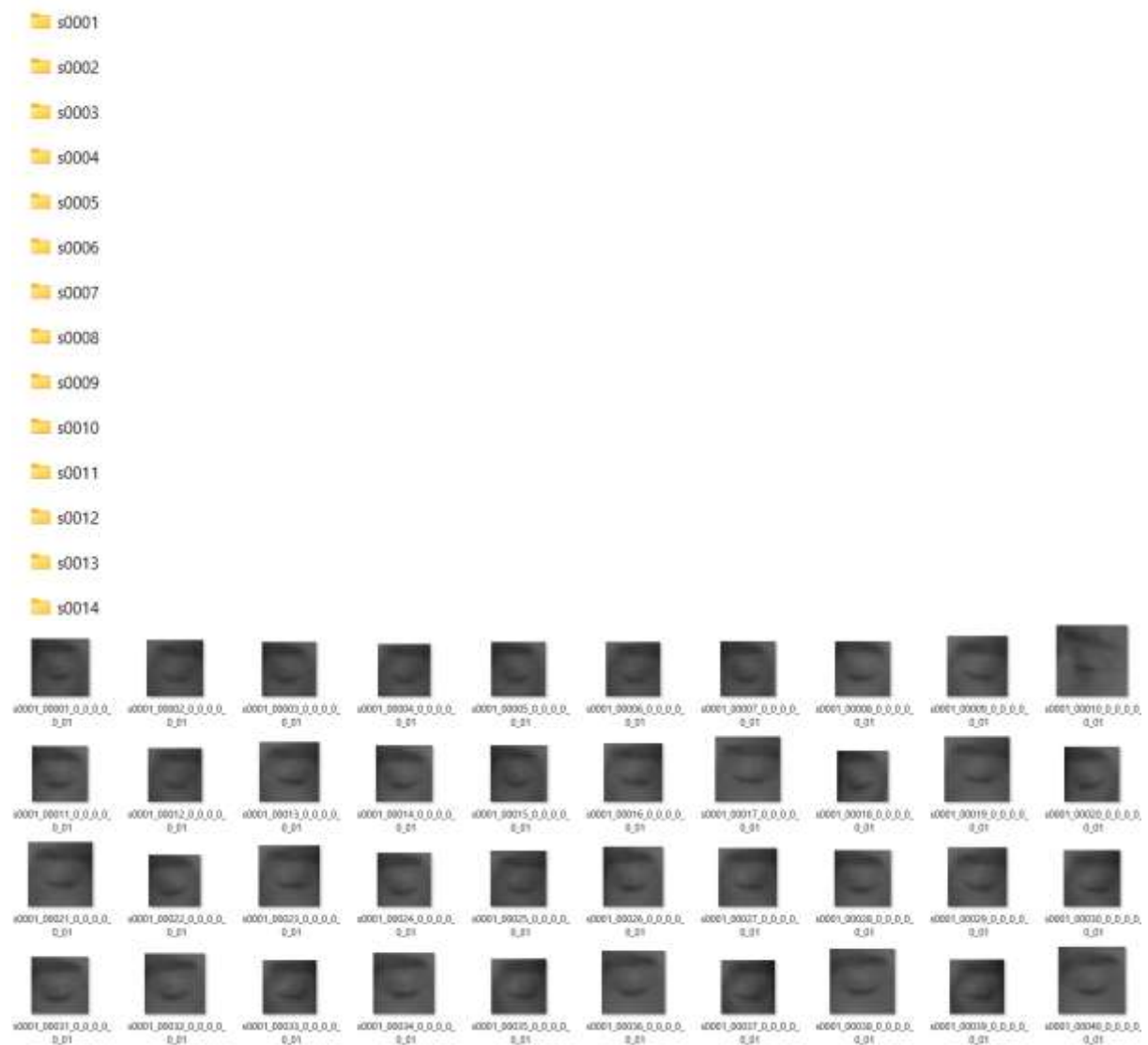
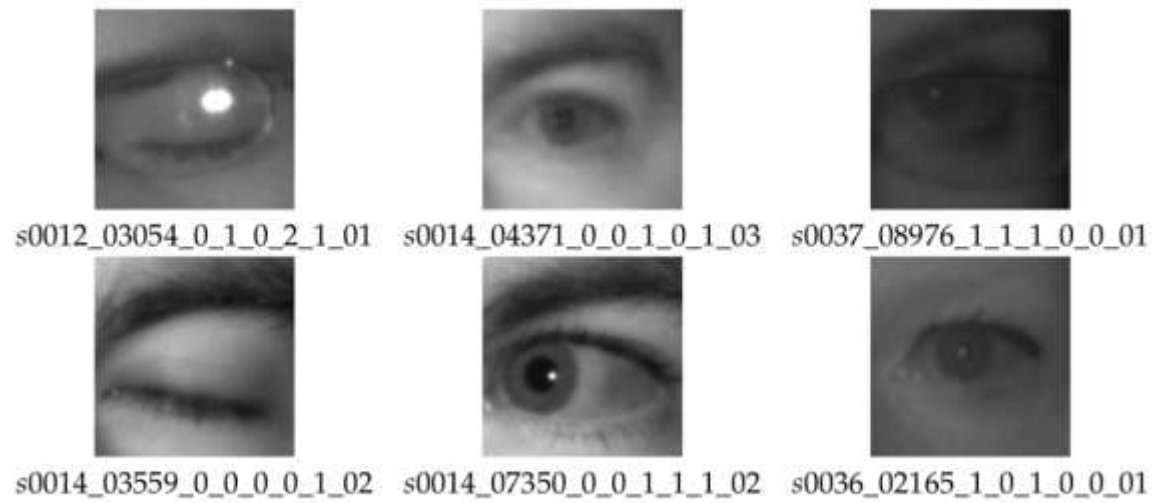
The detection of eyes and their parts, gaze estimation, and eye-blinking frequency are important tasks in computer vision. In last years, we have been solving these tasks in the area of driver's behaviour, which causes the acquiring of a lot of testing data that was acquired in real conditions. Therefore, we introduce the 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.

In the dataset, we annotated the following properties (the properties are indicated in the following order):

subject ID; in the dataset, we collected the data of 37 different persons (33 men and 4 women)

1. **image ID**: the dataset consists of 84,898 images
2. **gender [0 - man, 1 - woman]**: the dataset contains the information about gender for each image (man, woman)
3. **glasses [0 - no, 1 - yes]**: the information if the eye image contains glasses is also provided for each image (with and without the glasses)
4. **eye state [0 - closed, 1 - open]**: this property contains the information about two eye states (open, close)
5. **reflections [0 - none, 1 - small, 2 - big]**: we annotated three reflection states based on the size of reflections (none, small, and big reflections)
6. **lighting conditions [0 - bad, 1 - good]**: each image has two states (bad, good) based on the amount of light during capturing the videos
7. **sensor ID [01 - RealSense, 02 - IDS, 03 - Aptina]**: at this moment, the dataset contains the images captured by three different sensors (Intel RealSense RS 300 sensor with 640 x 480 resolution, IDS Imaging sensor with 1280 x 1024 resolution, and Aptina sensor with 752 x 480 resolution)

An examples of image annotations of the proposed dataset:



DESCRIPTION OF METHODS USED

Tensorflow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

OpenCV:

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

Numpy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas:

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

OS:

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The `*os*` and `*os.path*` modules include many functions to interact with the file system.

Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Transfer Learning:



Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

The reuse of a previously learned model on a new problem is known as transfer learning. It's particularly popular in deep learning right now since it can train deep neural networks with a small amount of data. This is particularly valuable in the field of data science, as most real-world situations do not require millions of labelled data points to train complicated models.

Overview

- What is Transfer Learning and it's Working?
- How Transfer Learning Works?

- Why Should You Use Transfer Learning?
- When to use Transfer Learning?
- Models That Have Been Pre-Trained?

The reuse of a previously learned model on a new problem is known as transfer learning. It's particularly popular in deep learning right now since it can train deep neural networks with a small amount of data. This is particularly valuable in the field of data science, as most real-world situations do not require millions of labelled data points to train complicated models.

The knowledge of an already trained machine learning model is transferred to a different but closely linked problem throughout transfer learning. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the model's training knowledge to identify other objects such as sunglasses.

MobileNet:

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings, and segmentation.

The MobileNet model is designed to be used in mobile applications, and it is TensorFlow's first mobile computer vision model.

MobileNet uses depthwise separable convolutions. It significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks.

A depthwise separable convolution is made from two operations.

1. Depthwise convolution.
2. Pointwise convolution.

MobileNet is a class of CNN that was open-sourced by Google, and therefore, this gives us an excellent starting point for training our classifiers that are insanely small and insanely fast.

EXPLORATORY DATA ANALYSIS & CODE DESCRIPTION WITH STEPS:

Importing The Libraries:

```
In [1]: 1 import tensorflow as tf
2 import cv2
3 import os
4 import matplotlib.pyplot as plt
5 import numpy as np
6 !pip install opencv-python
```

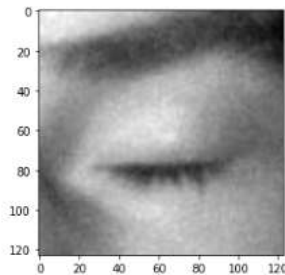
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.21.6)

PLOTTING AN IMAGE FROM THE DATASET AND CHECKING ITS SIZE:

```
In [3]: 1 img_array=cv2.imread("/content/Eye_dataset_m1/Closed_Eyes/s0005_00219_0_0_0_1_01.png",cv2.IMREAD_GRAYSCALE)
```

```
In [4]: 1 import matplotlib.pyplot as plt
2 plt.imshow(img_array,cmap="gray")
```

Out[4]: <matplotlib.image.AxesImage at 0x7fb013ceda10>



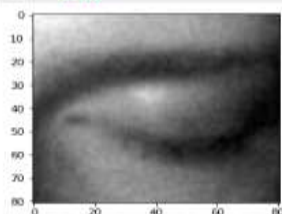
```
In [5]: 1 img_array.shape
```

Out[5]: (123, 123)

Inference: Here the size of the image is 123x123 dimension size. For, applying the transfer learning algorithm and bicubic interpolation we need to resize the image to 224x224 size dimensions.

CLASSIFYING THE IMAGES INTO CLOSED EYES AND OPEN EYES FOR THE TRAINING DATASET:

```
In [6]: 1 Datadirectory="/content/Eye_dataset_m1"
2 Classes=["Closed_Eyes","Open_Eyes"]
3 for category in Classes:
4     path=os.path.join(Datadirectory,category)
5     for img in os.listdir(path):
6         img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
7         backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
8         plt.imshow(img_array,cmap="gray")
9         plt.show()
10        break
11 break
```



RESIZING THE IMAGES IN THE DATASET TO 224 X 224 SIZE:

```
In [7]: 1 img_size=224
        2 new_array=cv2.resize(backtorgb,(img_size,img_size))
        3 plt.imshow(new_array,cmap="gray")
        4 plt.show()
```



READING ALL THE IMAGES AND CONVERTING THEM INTO AN ARRAY FOR DATA AND LABELS:

```
In [9]: 1 training_Data=[]
        2 def create_training_Data():
        3     for category in Classes:
        4         path=os.path.join(Datadirectory,category)
        5         class_num=Classes.index(category)
        6         for img in os.listdir(path):
        7             try:
        8                 img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
        9                 backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
       10                 new_array=cv2.resize(backtorgb,(img_size,img_size))
       11                 training_Data.append([new_array,class_num])
       12             except Exception as e:
       13                 pass
```

TRAINING DATA DIMENSIONS NORMALIZING:

```
In [10]: 1 create_training_Data()
        2

In [11]: 1 print(len(training_Data))
        2
1854

In [12]: 1 import random
        2 random.shuffle(training_Data)

In [13]: 1 X=[]
        2 y=[]
        3
        4 for features,label in training_Data:
        5     X.append(features)
        6     y.append(label)
        7
        8 X=np.array(X).reshape(-1,img_size,img_size,3)

In [14]: 1 X.shape
Out[14]: (1854, 224, 224, 3)

In [15]: 1 #normalize the data
        2 X=X/255.0; #we are normalizing
```

Inference: Here there are totally 1854 images and each image is of size 224 x 224 and we randomly shuffled the data to avoid overfitting and normalized the data array.

STORING THE DATA:

```
In [17]: 1 import pickle
2
3 pickle_out=open("X.pickle","wb")
4 pickle.dump(X,pickle_out)
5 pickle_out.close()
6
7 pickle_out=open("y.pickle","wb")
8 pickle.dump(y,pickle_out)
9 pickle_out.close()
```

```
In [18]: 1 pickle_in=open("X.pickle","rb")
2 X=pickle.load(pickle_in)
3
4 pickle_in=open("y.pickle","rb")
5 y=pickle.load(pickle_in)
```

DEEP LEARNING ARCHITECTURE – TRANSFER LEARNING AND USING THE MobileNet MODEL:

```
In [20]: 1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.keras import layers
```

```
In [21]: 1 model=tf.keras.applications.mobilenet.MobileNet()

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5
17227776/17225924 [=====] - 0s 0us/step
17235968/17225924 [=====] - 0s 0us/step
```

```
In [22]: 1 model.summary()
```

conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1, 1, 1024)	0
dropout (Dropout)	(None, 1, 1, 1024)	0
conv_preds (Conv2D)	(None, 1, 1, 1000)	1025000
reshape_2 (Reshape)	(None, 1000)	0
predictions (Activation)	(None, 1000)	0

=====
Total params: 4,253,864
Trainable params: 4,231,976
Non-trainable params: 21,888

```

In [24]: 1 base_input=model.layers[0].input

In [25]: 1 base_output=model.layers[-4].output

In [26]: 1 flat_layer=layers.Flatten()(base_output)
          2 final_output=layers.Dense(1)(flat_layer)
          3 final_output=layers.Activation('sigmoid')(final_output)

In [27]: 1 new_model=keras.Model(inputs=base_input,outputs=final_output)

In [28]: 1 new_model.summary()
conv_pw_13_bn (BatchNormali (None, 7, 7, 1024) 4096
zation)
conv_pw_13_relu (ReLU) (None, 7, 7, 1024) 0
global_average_pooling2d (G (None, 1, 1, 1024) 0
lobalAveragePooling2D)
dropout (Dropout) (None, 1, 1, 1024) 0
flatten (Flatten) (None, 1024) 0
dense (Dense) (None, 1) 1025
=====
Total params: 3,229,889
Trainable params: 3,208,001
Non-trainable params: 21,888

In [29]: 1 #setting for binary classification(open/closed)

In [30]: 1 new_model.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])

In [32]: 1 new_model.fit(X,Y,epochs=1,validation_split=0.1) ##training
53/53 [=====] - 328s 6s/step - loss: 0.8256 - accuracy: 0.9442 - val_loss: 1.5757 - val_accuracy: 0.89
78
Out[32]: <keras.callbacks.History at 0x7fb00dd4cd50>

In [33]: 1 new_model.save('my_model.h5')

```

Inference: Here we are using the transfer learning algorithm which uses the MobileNet model, here this model considers the nearly 4million params. But we use the binary classification for the data 0-opened eyes and 1-closed eyes. So, we have 2 classes and we modify the classification layer and the last FC layer and using the sigmoid function. So, we decreased the params to nearly 3million.In this way we created the new model and will use it for model evaluation and predictions.

RUNNING THE EPOCHS:

```

In [30]: 1 new_model.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])

In [32]: 1 new_model.fit(X,Y,epochs=1,validation_split=0.1) ##training
53/53 [=====] - 328s 6s/step - loss: 0.8256 - accuracy: 0.9442 - val_loss: 1.5757 - val_accuracy: 0.89
78
Out[32]: <keras.callbacks.History at 0x7fb00dd4cd50>

In [33]: 1 new_model.save('my_model.h5')

```

Inference: Here we ran 53 epochs and we got the training-loss=0.8256;Training accuracy=0.9442 ; validation-loss=1.5757 and validation-accuracy=0.89.

MODEL EVALUATION ON BOTH CLOSED AND OPEN EYE IMAGES FROM THE DATASET:

```
In [35]: 1 img_array=cv2.imread('/content/s0012_00029_0_0_0_1_01.png',cv2.IMREAD_GRAYSCALE)
          2 backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
          3 new_array=cv2.resize(backtorgb,(img_size,img_size))
```

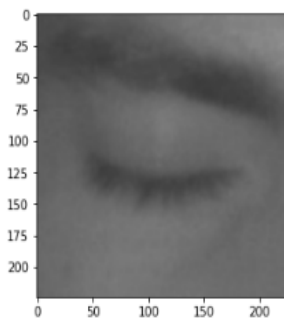
```
In [36]: 1 X_input=np.array(new_array).reshape(1,img_size,img_size,3)
```

```
In [37]: 1 X_input.shape
```

```
Out[37]: (1, 224, 224, 3)
```

```
In [38]: 1 plt.imshow(new_array)
```

```
Out[38]: <matplotlib.image.AxesImage at 0x7fb0137e2a10>
```



```
In [39]: 1 X_input=X_input/255.0
```

```
In [40]: 1 prediction=new_model.predict(X_input)
```

```
In [41]: 1 prediction
```

```
Out[41]: array([[ -15.34793]], dtype=float32)
```

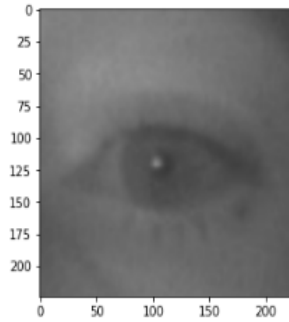
Inference: Here we took the image from the dataset but not from the training data and evaluating the model. For this closed eye we got the prediction as -15.34. Here the negative value indicates the eyes are closed. We got the prediction correct.

```
In [42]: 1 img_array=cv2.imread('/content/s0017_01639_1_0_1_0_1_01.png',cv2.IMREAD_GRAYSCALE)
2 backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
3 new_array=cv2.resize(backtorgb,(img_size,img_size))
```

```
In [43]: 1 X_input=np.array(new_array).reshape(1,img_size,img_size,3)
```

```
In [44]: 1 plt.imshow(new_array)
```

```
Out[44]: <matplotlib.image.AxesImage at 0x7fb00f635710>
```



```
In [45]: 1 X_input=X_input/255.0
```

```
In [46]: 1 prediction=new_model.predict(X_input)
```

```
In [47]: 1 prediction
```


```
Out[47]: array([[31.86922]], dtype=float32)
```

Inference: Here we took the image from the dataset but not from the training data and evaluating the model. For this opened eye we got the prediction as 31.86. Here the negative value indicates the eyes are opened. We got the prediction correct.

PREDICTION ANALYSIS ON UNKNOWN IMAGES:

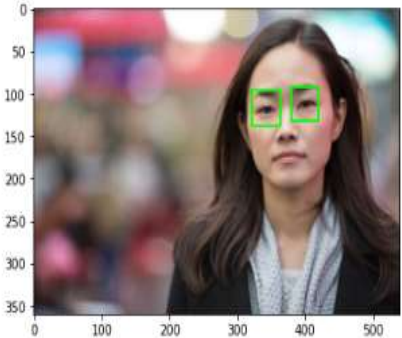
```
In [49]: 1 img=cv2.imread('/content/sad_woman.jpg')

In [50]: 1 plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
Out[50]: <matplotlib.image.AxesImage at 0x7fb00f556a18>
```



```
In [51]: 1 faceCascade=cv2.CascadeClassifier(cv2.data.harcascades+'haarcascade_frontalface_default.xml')
In [52]: 1 eye_cascade=cv2.CascadeClassifier(cv2.data.harcascades+'haarcascade_eye.xml')
In [53]: 1 gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
In [54]: 1 eyes=eye_cascade.detectMultiScale(gray,1.1,4)
In [55]: 1 for(x,y,w,h) in eyes:
2         cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

```
In [56]: 1 plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
Out[56]: <matplotlib.image.AxesImage at 0x7fb00f51e390>
```



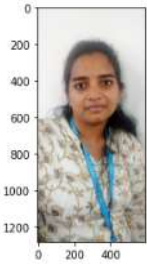
Inference: Here we correctly predicted the eyes position using the “haarcascade_frontalface_default.xml” & “haarcascade_eye.xml”.

PREDICTION ANALYSIS ON UNKNOWN IMAGES WITH EYES OPENED/CLOSED DETECTION USING ROI:

```
In [75]: 1 img=cv2.imread('/content/sowjanya_test_eye.png')

In [76]: 1 plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))

Out[76]: <matplotlib.image.AxesImage at 0x7fb00ef02690>
```



```
In [77]: 1 faceCascade=cv2.CascadeClassifier(cv2.data.harcascades+'haarcascade_frontalface_default.xml')

In [78]: 1 eye_cascade=cv2.CascadeClassifier(cv2.data.harcascades+'haarcascade_eye.xml')

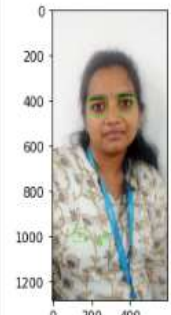
In [79]: 1 gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        2

In [80]: 1
        2
        3 eyes=eye_cascade.detectMultiScale(gray,1.1,4)
        4

In [81]: 1 for(x,y,w,h) in eyes:
        2     cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

In [82]: 1 plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))

Out[82]: <matplotlib.image.AxesImage at 0x7fb00ee67810>
```



Inference: Here we correctly predicted the eyes position using the “haarcascade_frontalface_default.xml” & “haarcascade_eye.xml”, next we are going to plot the region of interest(ROI) and detect whether the eyes are opened or not through the model we created.

```

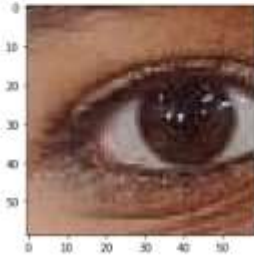
In [83]: 1 #cropping the eye image

In [84]: 1 eye_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_eye.xml')
2 gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
3 eyes=eye_cascade.detectMultiScale(gray,1.1,4)
4 for x,y,w,h in eyes:
5     roi_gray=gray[y:y+h,x:x+w]
6     roi_color=img[y:y+h,x:x+w]
7     eyess=eye_cascade.detectMultiScale(roi_gray)
8     if len(eyess)==0:
9         print("Eyes are not detected")
10    else:
11        for(ex,ey,ew,eh) in eyess:
12            eyes_roi=roi_color[ey:ey+eh,ex:ex+ew]

In [85]: 1 plt.imshow(cv2.cvtColor(eyes_roi,cv2.COLOR_BGR2RGB))

```

Out[85]: <matplotlib.image.AxesImage at 0x7fb00ee45d18>



Inference: Here we plotted the eyes and we will use it for detection.

```

In [86]: 1 eyes_roi.shape

Out[86]: (59, 59, 3)

In [87]: 1 final_image=cv2.resize(eyes_roi,(224,224))
2 final_image=np.expand_dims(final_image,axis=0)##need fourth dimension
3 final_image=final_image/255.0

In [88]: 1 final_image.shape

Out[88]: (1, 224, 224, 3)

In [89]: 1 new_model.predict(final_image)

Out[89]: array([[38.153378]], dtype=float32)

```

Inference: Here we first saw that the shape is 59 x 59. So we resized it to 224 x 224 and predicted the result as 38.153 which is positive for open eyes. So our prediction is correct.

```

In [ ]: 1 import cv2
2 from IPython.display import display, Javascript
3 from google.colab.patches import cv2_imshow
4 from google.colab.output import eval_js
5 from base64 import b64decode
6 path="haarcascade_frontalface_default.xml"
7 faceCascade=cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalface_default.xml")
8
9 cap=cv2.VideoCapture(1)
10 if not cap.isOpened():
11     cap=cv2.VideoCapture(0)
12 if not cap.isOpened():
13     raise IOError("CANNOT OPEN WEBCAM")
14
15 while True:
16     ret,frame=cap.read()
17     eye_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_eye.xml")
18     gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
19     eyes=eye_cascade.detectMultiScale(gray,1.1,4)
20     for x,y,w,h in eyes:
21         roi_gray=gray[y:y+h,x:x+w]
22         roi_color=frame[y:y+h,x:x+w]
23         cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
24         eyess=eye_cascade.detectMultiScale(roi_gray)
25         if len(eyess)==0:
26             print("eyes are not detected")
27         else:
28             for (ex,ey,ew,eh) in eyess:
29                 eyes_roi=roi_color[ey:ey+eh,ex:ex+ew]
30             final_image=cv2.resize(eyes_roi,(224,224))
31             final_image=np.expand_dims(final_image,axis=0)
32             final_image=final_image/255.0
33
34             Predictions=new_model.predict(final_image)
35             if(Predictions>0):
36                 status="Open Eyes"
37             else:
38                 status="Closed Eyes"
39             gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
40             print(faceCascade.empty())
41             faces=faceCascade.detectMultiScale(gray,1.1,4)
42
43             for(x,y,w,h) in faces:
44                 cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
45
46             font=cv2.FONT_HERSHEY_SIMPLEX
47
48             cv2.putText(frame,
49                 status,
50                 (50,50),
51                 font,3,
52                 (0,0,255),
53                 2,
54                 cv2.LINE_4)
55
56             if cv2.waitKey(2) & 0xFF == ord('q'):
57                 break
58
59 cap.release()
60 cv2.destroyAllWindows()

```

Inference: Here we wrote the code for the accessing the webcam and detecting whether the eyes are opened or not and detects the status of the eye and switch on the alert system whenever it is needed.

CONCLUSION

The driver alert system is executed successfully with the help of deep learning architecture with transfer algorithm with MobileNet model and saw the positive results. We can still increase the accuracy by increasing the epochs. Whenever the driver is feeling drowsy the system will alert the driver which decreases the accidents and it's an effective application. This will be very useful for the society as we can decrease the rate of accidents using the Driver Alert System.

REFERENCES

- <https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470>
- <https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/>
- <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>
- <https://www.youtube.com/watch?v=qwUIFKi4V48>
- <https://projectgurukul.org/driver-drowsiness-detection-system-opencv-tensorflow/>

THE END