ILLINOIS INSTITUTE OF TECHNOLOGY

Driver Drowsiness Detection

Karan Sathiayan
A20469856
(ksathiayan@hawk.iit.edu)

Praveen Surendran
A20496784
(s5@hawk.iit.edu)

July 31st 2022

Prof. Jawahar Panchal
Data Preparation and Analysis – CSP 571

Abstract

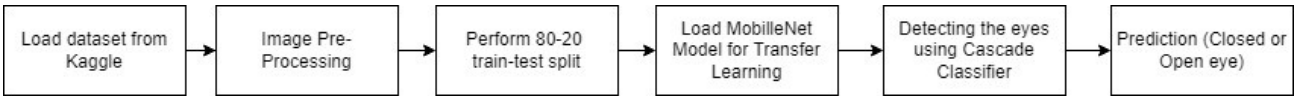
Driver drowsiness detection systems may indeed be a significant application of machine learning and image processing given its high importance. Numerous research efforts have recently been mentioned in the literature in this area. In this study, we want to forecast the driver's drowsiness based on the condition of his or her eyes. Drowsiness detecting systems face numerous difficulties. The existence of spectacles, a beard, and changes in intensity caused by lighting conditions are a few of the crucial factors. In this project, we suggest and put into practice a pre-processing method that can be utilized to fix these issues. Following the face detection step, the facial elements in the proposed method that are more significant and thought to be the most beneficial for drowsiness are retrieved and employed for prediction. The method has been evaluated and put into practice in a real environment.

Overview

Humans have always built machines and devised tactics to make their lives easier and safer, whether for ordinary goals like going to work or for more intriguing ones like flying. Modes of transportation advanced with technological advancements, and our reliance on them grew enormously. It has had a significant impact on our lives as we know it. We may now go to destinations at a speed that even our grandparents could not have imagined. Almost everyone in the modern world uses some form of transportation on a daily basis. Some people are wealthy enough to possess cars, while others rely on public transportation. Regardless of social position, however, there are some regulations and codes of behavior for individuals who drive.

Staying attentive and active while driving is one of them. Every year, hundreds of thousands of tragedies are related to this wonderful technology due to our failure to fulfill our responsibilities toward safer travel. Following the rules and regulations on the road may appear to most people to be a minor detail, but it is critical. While on the road, an automobile has the most power, and in the hands of irresponsible drivers, it may be harmful, and negligence can occasionally endanger the lives of other road users. One type of irresponsibility is failing to admit when we are too fatigued to drive. Many scholars have written research papers on driver sleepiness detection systems in order to monitor and prevent the negative consequences of such neglect. However, some of the system's points and observations aren't always exact enough. As a result, this project will provide data and a different viewpoint on the problem at hand, in order to improve their implementations and better optimize the solution.

Flow of the Project



- First acquire the dataset from Kaggle which contains images of the eye.
- We then pre-process the images.
- Then a 80-20 train-test split is performed.
- We then fit a CNN Model along with Transfer learning.
- For test images, we first perform object detection which acquires the coordinates for the eye.
- We use this eye image to predict if the eye is closed or open.

Initial Preparation

The dataset is around 45MB with a total of 4000 images, out of which 2000 are closed eyed and the rest are open eyed images.

Here is an example of a closed eye/ open eye image from the dataset.

Link - https://www.kaggle.com/datasets/prasadvpatil/mrl-dataset

Image Pre-Processing

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it.

These are crucial steps to be performed before they are used for model training and inference.

OpenCV

Known as OpenCV, this software library for computer vision and machine learning is free and open source.

A standard infrastructure for computer vision applications was created with OpenCV in order to speed up

the incorporation of artificial intelligence into products. OpenCV makes it simple for businesses to consume and alter the code because it is a BSD-licensed product.

An image is loaded using the cv2.imread() method from the given file. This method produces an empty matrix if the picture cannot be read (due to a missing file, poor permissions, an unsupported or invalid format).

Grayscaling

Grayscaling is the process of converting an image from other color spaces e.g., RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white.

Importance of grayscaling:

- **Dimension reduction:** For example, In RGB images there are three color channels and has three dimensions while grayscale images are single-dimensional.
- **Reduces model complexity:** Consider training neural article on RGB images of 10x10x3 pixel. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for grayscale images.
- **For other algorithms to work:** Many algorithms are customized to work only on grayscale images e.g., Canny edge detection function pre-implemented in OpenCV library works on Grayscale images only. [1]

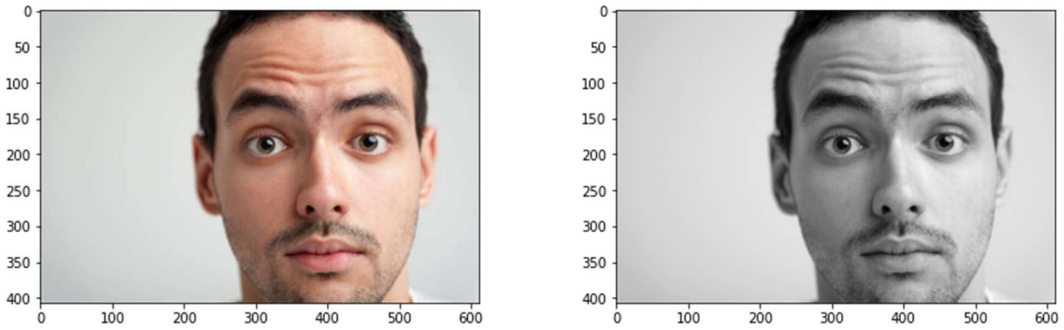


Image Enhancement

Histogram equalization is a basic image processing technique that adjusts the global contrast of an image by updating the image histogram's pixel intensity distribution. Doing so enables areas of low contrast to obtain higher contrast in the output image.

Essentially, histogram equalization works by:

- Computing a histogram of image pixel intensities
- Evenly spreading out and distributing the most frequent pixel values (i.e., the ones with the largest counts in the histogram)
- Giving a linear trend to the cumulative distribution function (CDF)

The result of applying histogram equalization is an image with higher global contrast. [2]

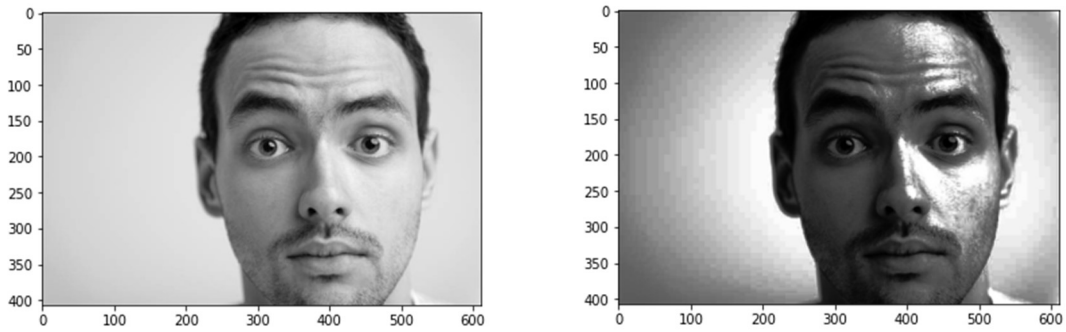


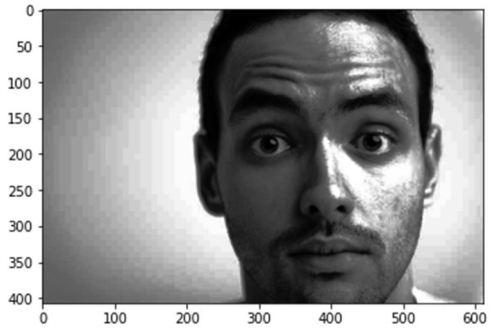
Image Resizing

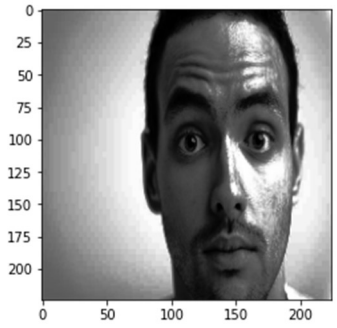
Image resizing refers to the scaling of images. Scaling comes in handy in many image processing as well as machine learning applications. It helps in reducing the number of pixels from an image and that has several advantages e.g. It can reduce the time of training of a neural network as more is the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model. It also helps in zooming in images. Many times, we need to resize the image i.e., either shrink it or scale up to meet the size requirements. OpenCV provides us several interpolation methods for resizing an image.

Choice of Interpolation Method for Resizing –

- cv2.INTER_AREA: This is used when we need to shrink an image.
- cv2.INTER_CUBIC: This is slow but more efficient.
- cv2.INTER_LINEAR: This is primarily used when zooming is required. This is the default interpolation technique in OpenCV. [3]

In our project, we used the default interpolation method for resizing.





Train-Test Split

The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a sufficiently large dataset available.

We have used 80-20 split in our project.

Building the Neural Network Model

Convolutional Neural Network

Convolutional Neural Network is a Deep Learning algorithm specially designed for working with Images and videos. It takes images as inputs, extracts and learns the features of the image, and classifies them based on the learned features.

This algorithm is inspired by the working of a part of the human brain which is the Visual Cortex. The visual Cortex is a part of the human brain which is responsible for processing visual information from the outside world. It has various layers and each layer has its own functioning i.e. each layer extracts some information from the image or any visual and at last all the information received from each layer is combined and the image/visual is interpreted or classified.

Similarly, CNN has various filters, and each filter extracts some information from the image such as edges, different kinds of shapes (vertical, horizontal, round), and then all of these are combined to identify the image.

Transfer Learning

Transfer learning is about leveraging feature representations from a pre-trained model, so you don't have to train a new model from scratch. The pre-trained models are usually trained on massive datasets that are a standard benchmark in the computer vision frontier. The weights obtained from the models can be reused in other computer vision tasks. These models can be used directly in making predictions on new tasks or integrated into the process of training a new model. Including the pre-trained models in a new model leads to lower training time and lower generalization error.
Transfer learning is particularly very useful when you have a small training dataset. In this case, you can, for example, use the weights from the pre-trained models to initialize the weights of the new model. [5]

MobileNet function

Mobilenet is a model which does the same convolution as done by CNN to filter images but in a different way than those done by the previous CNN. It uses the idea of Depth convolution and point convolution which is different from the normal convolution as done by normal CNNs. This increases the efficiency of CNN to predict images and hence they can be able to compete in the mobile systems as well. Since these ways of convolution reduce the comparison and recognition time a lot, so it provides a better response in

a very short time and hence we are using them as our image recognition model. This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet.

There are a total of 3,229,889 parameters, out of which 3,208,001 are trainable and 21,888 are non-trainable parameters.

Total params: 3,229,889
Trainable params: 3,208,001
Non-trainable params: 21,888

The **model architecture** is given below:

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 56, 56, 64)	256
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_pw_2_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1152
conv_dw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_dw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	16384
conv_pw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 57, 57, 128)	0
conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	1152
conv_dw_4_bn (BatchNormalization)	(None, 28, 28, 128)	512
conv_dw_4_relu (ReLU)	(None, 28, 28, 128)	0
conv_pw_4 (Conv2D)	(None, 28, 28, 256)	32768
conv_pw_4_bn (BatchNormalization)	(None, 28, 28, 256)	1024
conv_pw_4_relu (ReLU)	(None, 28, 28, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	2304
conv_dw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1024
conv_dw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	65536
conv_pw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1024
conv_pw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 29, 29, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	2304
conv_dw_6_bn (BatchNormalization)	(None, 14, 14, 256)	1024
conv_dw_6_relu (ReLU)	(None, 14, 14, 256)	0
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	131072
conv_pw_6_bn (BatchNormalization)	(None, 14, 14, 512)	2048
conv_pw_6_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_dw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2048
conv_dw_7_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	262144
conv_pw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2048

```
conv_pw_7_relu (ReLU)       (None, 14, 14, 512)       0
conv_dw_8 (DepthwiseConv2D)  (None, 14, 14, 512)      4608
conv_dw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)
conv_dw_8_relu (ReLU)       (None, 14, 14, 512)       0
conv_pw_8 (Conv2D)          (None, 14, 14, 512)       262144
conv_pw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)
conv_pw_8_relu (ReLU)       (None, 14, 14, 512)       0
conv_dw_9 (DepthwiseConv2D)  (None, 14, 14, 512)      4608
conv_dw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)
conv_dw_9_relu (ReLU)       (None, 14, 14, 512)       0
conv_pw_9 (Conv2D)          (None, 14, 14, 512)       262144
conv_pw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)
conv_pw_9_relu (ReLU)       (None, 14, 14, 512)       0
conv_dw_10 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)
conv_dw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)
conv_dw_10_relu (ReLU)      (None, 14, 14, 512)       0
conv_pw_10 (Conv2D)         (None, 14, 14, 512)       262144
conv_pw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)
conv_pw_10_relu (ReLU)      (None, 14, 14, 512)       0
conv_dw_11 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)
conv_dw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)
conv_dw_11_relu (ReLU)      (None, 14, 14, 512)       0
conv_pw_11 (Conv2D)         (None, 14, 14, 512)       262144
conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)
```

```
conv_pw_11_relu (ReLU)      (None, 14, 14, 512)       0
conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0
conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)
conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
zation)
conv_dw_12_relu (ReLU)      (None, 7, 7, 512)         0
conv_pw_12 (Conv2D)         (None, 7, 7, 1024)        524288
conv_pw_12_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)
conv_pw_12_relu (ReLU)      (None, 7, 7, 1024)        0
conv_dw_13 (DepthwiseConv2D  (None, 7, 7, 1024)       9216
)
conv_dw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)
conv_dw_13_relu (ReLU)      (None, 7, 7, 1024)        0
conv_pw_13 (Conv2D)         (None, 7, 7, 1024)        1048576
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
activation (Activation)     (None, 1)                 0
=================================================================
Total params: 3,229,889
Trainable params: 3,208,001
Non-trainable params: 21,888
_________________________________________________________________
```

Train & Test Accuracy

```
drowsiness_model.fit(X_train, y_train, epochs = 3, validation_data = (X_test, y_test))
#Note: Increase the number of epoch to get more appropriate result, accuracy.

Epoch 1/3
100/100 [==============================] - 36s 213ms/step - loss: 0.0255 - accuracy: 0.9891 - val_loss: 0.0121 - val_accuracy: 0.9975
Epoch 2/3
100/100 [==============================] - 20s 197ms/step - loss: 0.0041 - accuracy: 0.9991 - val_loss: 1.6829e-10 - val_accuracy: 1.0000
Epoch 3/3
100/100 [==============================] - 20s 202ms/step - loss: 4.4885e-04 - accuracy: 1.0000 - val_loss: 6.9015e-06 - val_accuracy: 1.0000
<keras.callbacks.History at 0x7f41d01326d0>
```

```
[ ]  print("Train Accuracy: ", accuracy_score(y_train, predict_train_label))

     Train Accuracy:  1.0

The training accuracy of the model was found to be 1.0
```

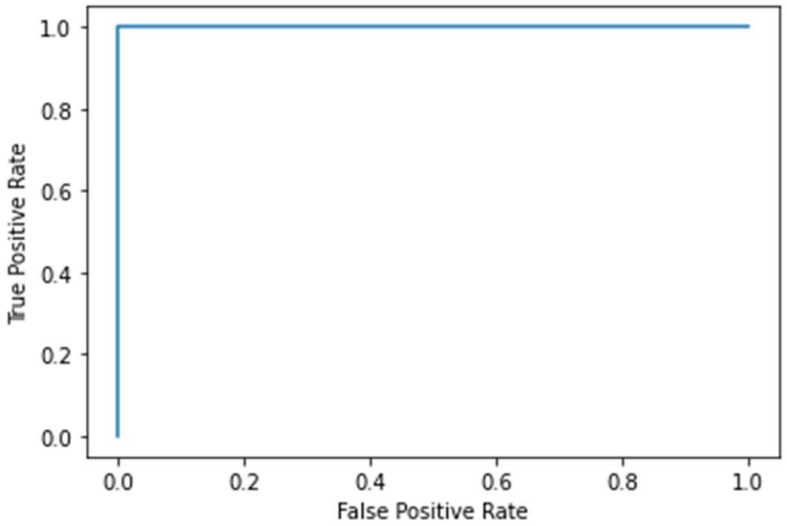
```
[ ]  print("Test Accuracy: ", accuracy_score(y_test, predict_test_label))

     Test Accuracy:  1.0

The test accuracy of the model was found to be 1.0
```

Since, the image dataset from Kaggle was well organized, on performing transfer learning, after 3 epochs, we get a training accuracy of 1.0 and a validation accuracy of 1.0.

Analysis

Receiver Operating Characteristics (ROC)

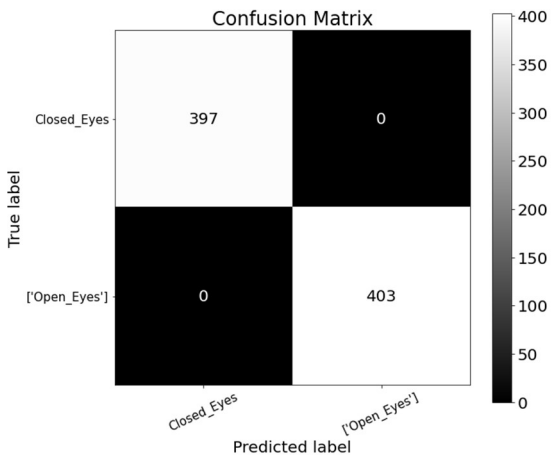
The receiver operating characteristic (ROC) curve is frequently used for evaluating the performance of binary classification algorithms. It provides a graphical representation of a classifier's performance, rather than a single value like most other metrics. The ROC curve is produced by calculating and plotting the true positive rate against the false positive rate for a single classifier at a variety of thresholds. [9]
The ROC for our model is given below.



Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives you insights not only on the errors being made by your classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone. [8]

The Confusion Matrix of our model is given below:



Object Detection (Eyes)

The images that are found in the dataset are all close-up images of the eyes, and the Neural Network Model is trained on them. If we are to use any other image that is not a close-up image of the eye, the Model would find it difficult to locate the eye. To overcome this issue, we use Haar-cascade Detection to help with identifying the coordinates of the eye.

Haar-cascade Detection in OpenCV

A Haar classifier, or a Haar cascade classifier, is a machine learning object detection program that identifies objects in an image and video. OpenCV provides a training method (see Cascade Classifier Training) or pretrained models, that can be read using the CascadeClassifier - load method. The pretrained models are located in the data folder in the OpenCV installation or can be found here.

The following code example will use pretrained Haar cascade models to detect faces and eyes in an image. First, a CascadeClassifier is created and the necessary XML file is loaded using CascadeClassifier - load method. Afterwards, the detection is done using CascadeClassifier - detectMultiScale method, which returns boundary rectangles for the detected faces or eyes. [7]

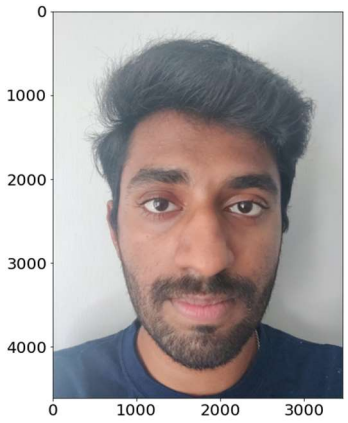
Prediction

From the coordinates obtained by the Haar-Cascade Classifier, we get the eye image. This image matrix is then used as an input to the Neural Network Model. The Neural Network Model gives a probabilistic prediction for the input image. If the predicted output is greater than 0.5, we classify the image as “Eyes are OPEN – Driver Awake” and “Eyes are CLOSED – Driver Tired” otherwise.

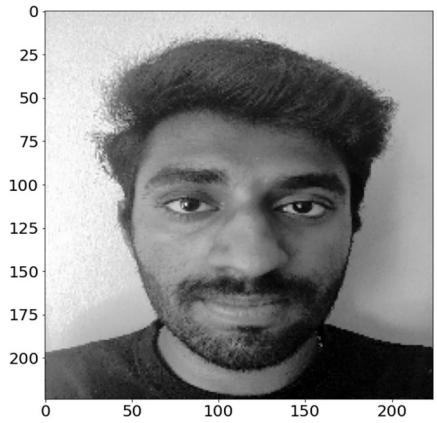
Results

To verify the predictions, a few of our open and closed eyed images are used.

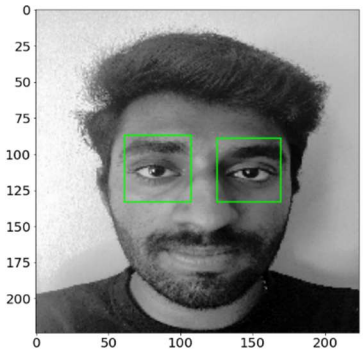
Opened Eye Detection



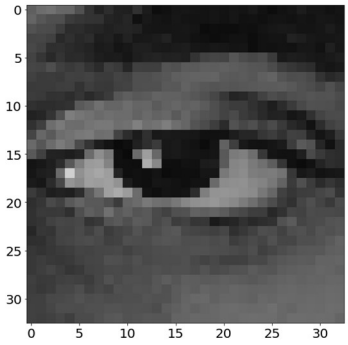
Input Image



Processed image



Eye Detection



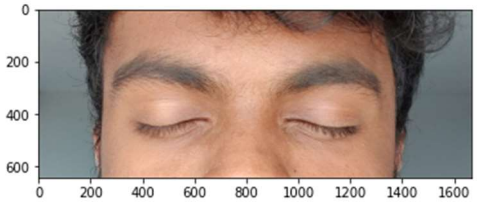
Resized image for prediction

Predicted output:

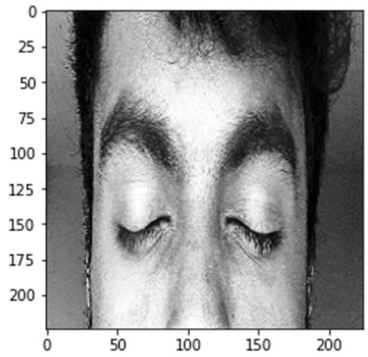
```
predict = drowsiness.predict(final_img)
if predict>0.5:
  print("Eyes are OPEN - Driver Awake")
else:
  print("Eyes are CLOSED - Driver Tired")
```

```
Eyes are OPEN - Driver Awake
```

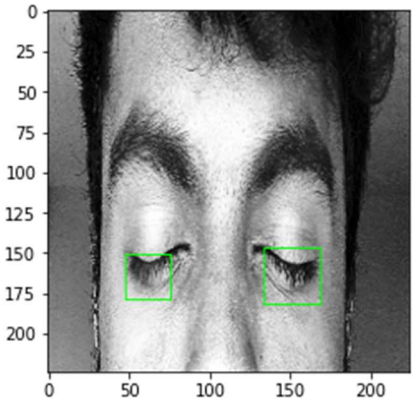
Closed Eye Detection



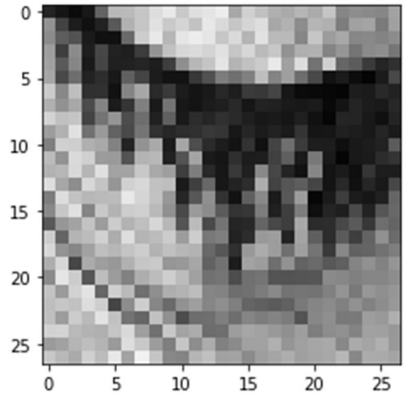
Input Image



Processed image



Eye Detection



Resized image for prediction

Predicted output:

```
predict = drowsiness.predict(final_img)
if predict>0.5:
  print("Eyes are OPEN - Driver Awake")
else:
  print("Eyes are CLOSED - Driver Tired")
```

```
Eyes are CLOSED - Driver Tired
```

Future Scope/ Real-Life Application:

We can use a webcam to take short bursts of images—say, one per second—and feed them into the model to make predictions. For instance, if three consecutive photos reveal that the driver's eyes are closed, we can be certain that they are tired. When this condition is met, we can alert the driver.

References

[1] https://www.geeksforgeeks.org/python-grayscaling-of-images-using-opencv/

[2] https://pyimagesearch.com/2021/02/01/opencv-histogram-equalization-and-adaptive-histogram-equalization-clahe/

[3] https://www.geeksforgeeks.org/image-resizing-using-opencv-python/

[4] https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

[5] https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras/

[6] https://keras.io/api/applications/mobilenet/

[7] https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html/

[8] https://machinelearningmastery.com/confusion-matrix-machine-learning/

[9] https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb/