Sleep Detection Sensor

1 Abstract

Distracted driving, driving under influence, unskilled driving are well-known hazards, but few people think twice about getting behind the wheel when feeling drowsy. Each year around 100,000 traffic crashes occur because of sleep disorder in the United States. Following the former, researches based on computer vision application has devised high pixel cameras for sleep detection. In this work, a sensor is developed using deep learning to determine the drowsiness of driver by continuosly observing his/her eye states. To detect the eye region, OpenCV haar cascade classifier is used. Convolution neural network is built to extract the eye feature from dynamically captured camera frames. Fully connected softmax layer of CNN classifying the driver as sleep or non-sleep. This system alerts the driver with buzzer if sensor detects him/her in sleep mood. The model is caliberated on collected dataset consist of open/close eye images under different lighting conditions and clocked the accuracy of more than 90% during testing phase.

2 Introduction and project idea

For a safe and secure automobile like car, truck, bus etc driver's sleep detection is very important to forestall any pertinent road accidents. Especially truck drivers whose job requires the driving for long stretches regardless of day/night often resort to sleep deprivation which may cause life harming incidents. In the United States, as per the study conducted by National Safety Council (NSC), every year in total 100,000 road accidents, 71,000 injuries and 1,550 fatalities occur due to sleepy driving. In India, around 1,36,118/year fatality rate within the last one decade was reported due to road crashes[1]. Patrolling squads and police authorities of the expressways/highways disclosed that majority of the accidents happened between 2 am and 5 am due to driver's dormancy.

In order to thwart such mishap, a real time system is needed to supervise the driver's dormant state continuously. Various metrics can be used to enumerate the state of asleep. These are Physiological metrics ,Behavioral metrics and Vehicle-based metrics. In this study, we are building a system which will detect whether a driver is sleepy or non sleepy based on behavioral features. We make use of a video camera which will consistently focus and monitor the state of the driver's eyes. If the driver's eyes remain close for more than the defined threshold then the detection system will trigger the alarm to alert the driver/passengers about dormancy. Fig.1 depicting the idea with a beam(Higlighted red) acting as an eye's CCTV.

The proposed method identifies and extract the eye region of the driver through a smart camera using a object detection algorithm(OpenCV and Haar

Cascade classifier) and sends it to the classification algorithm. Next, the classification algorithm(Convolution neural network) will then extract the eye features from the captured images and declares the driver's dormancy post classifying the eyelids(either open or closed) as per defined threshold. For better insight, general architecture of system is shown in Fig.2.

Fig.1. Picture showing the driver(eyes) under surveillance

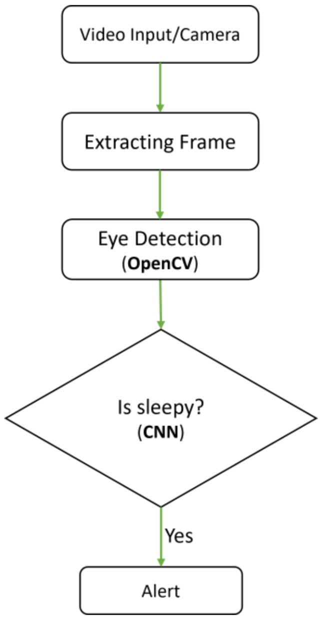


Fig.2. Architecture of sleep detection sensor.

3 Background

There are few high class motor vehicles which possess a built-in sleep detection sensors. Several tools for android or iOS platforms have been devised for driver's sleep deprivation[10]. Anti-sleep pilot sporadically calls the driver to interact with the app so the driver remains sober. Sleepy driver plus, a tool notifies the driver with his/her exertion level through the alarm clock. [11] proposed driver safety mechanism CarSafe which used dual-camera phones to spot deleterious driving occurrences like tailgating, zig-zag or rough driving, distracted driving.

[1]Devised sleep detection using convolution neural networks(CNN). The model extracted the facial features like eye state to determine the drowsiness. This method secured 96.42% accuracy. [5]Proposed a system for sleep detection based on EEG patterns/signals. Neural networks was used as classification strategy and clocked accuracy of 83.3%. [4]Built model for sleep detection based on steering wheel patterns. Steering wheel patterns was extracted in the form of features using signal processing methods. The model recorded the accuracy of 86%.

[3]developed deep learning based android application to classify drowsiness or non drowsiness using facial landmark points. The algorithm clocked the accuracy beyond 80%. Sleep detection based on integration of brain activity and visual activity was proposed by [7]. The model used Electroencephalography (EEG) signals to monitor brain activity and facial features for visual activity. This amalgamation achieved an accuracy of 80.6%. [2]Proposed the idea to identify sleep deprivation thorough the eye blinks. The neural network algorithm was used to locate the pupil and to the clock number of eye blink per minutes followed by classification. [10]Presented Sober-Drive system that leveraged PERCLOS to identify on-board drowsy operators using smart phones. They implemented Back Propagation Neural Network as an Eye States Classifier.

Hidden Markov Model was used by [9]. Viola Jones algorithm and Gabor wavelet decomposition for the face region and Adaboost learning algorithm for selecting features was used. The classification of sleepy or non-sleepy was done by HMM. [8] proposed eye surveillance based drowsiness detection system. The system triggers sleep alert in case it finds the driver's dormancy beyond the specified threshold. This work make use of Viola Jones algorithm for detecting the eye and face region. The devised work achieved an outdoor accuracy of 72.8% and an indoor accuracy of 82%. Detection system for the drowsiness developed by [6] which used both Driving Quality Signals and brain activity in combination. The idea was to achieve the better accuracy by monitoring both EEG signal for brain activity and ride characteristics for driving signals.

4 Theoretical Framework

4.1 Eye region detection

OpenCV library of python is used in the detection system for extracting eye region from a video(realtime) footage. It is dedicated to solving computer vision problems and has a bunch of pre-trained HAAR classifiers stored as XML files that can be used to identify objects such as trees, number plates, faces, eyes, etc. In this work, eye detection algorithm of OpenCV is deployed to identify and extract the eye region from dynamic frames. Fig.3 representing the way it extracts the region of interest(ROI) i.e eye from detection boundary(Highlighted green). This extracted information is then fed to convolution neural networks for determining the state of drowsiness.

Fig.3. Working of OpenCV(eye detection and extraction)

4.2 Feature extraction and classification

The proposed model is mounted over an idea of convolution neural network. The network composed of multiple layers named convolution layer, pooling layer and fully connected layer. Convolution layers and fully connected layers are followed by their corresponding activation/transfer function. If an image is given as input to the network, it will go all the way across these layers to reach the output. As shown in Fig.5, an image(24x24x1) dataset is given as input to the network which then able to classify these as open/close at the output. Each image act as an input for first convolution layer. Convolution layer perform the computations over the image using filters of specified kernel size(4x4) to calculate the feature maps for the respective filters. Fig.6 shows the 16 sample filters of kernel size 4x4 which convolution layer scans over the input images to evaluate the feature map. Feature map size is calculated using following equation.

$$\textit{Featuremap} = \frac{n + 2p - f}{s} + 1 \qquad (1)$$

where n is input image size, p is padding, f is filter(kernel) size and s is stride.

Next, these feature maps go to pooling layer having kernel size(2x2) as input.
The pooling layer will reduce the dimension of each feature map to accelerate the calculations. Fig.7 shows the output(2 feature maps) produce by convolution layer by scanning 2 different filters on the input image. Then, the output produce by max-pooling layer for the 2 input feature maps. The main purpose of showcasing the visuals(Filters, Feature maps and output of max pooling layer) is to provide the basic understanding of convolution and max-pooling layer. Again the output from pooling layer will stream to second convolution and pooling layer. Output of second pooling layer will feed as input to fully connected layers after going through flatten layer. Fully connected layers carries out the process of over-imposition and yield the output based on which further enumerations pertinent to loss estimation, back propagation, forward propagation takes place accordingly. The entire set of operations prior to fully connected layers helps in preprocessing of an

image/dataset. The main idea behind the pretreatment is to subvert the computational overhead. CNN perquisites need to be fulfill before initializing the model training cycle. These perquisites are called as hyperparameters. Hyperparameters are are more like configurations which assists explicit adjustment of convolution network according to the input dataset. Hyperparameters are composed of activation function(Rectified linear uni, Softmax etc), optimizer(Adam), loss function(Categorical cross entropy), checkpoints, horizontal flip, batch size, epochs, training and testing split, stepsize, learning rate, number of layers, filter size, number of filters etc. Checkpoint deploys to save the highest validation/testing accuracy model to the desired location. Suppose we have 4 epochs(1,2,3,4) with corresponding validation/testing accuracies(80%,74%,85%,79%). In this case, checkpoint save the model having 85% accuracy.

Horizontal flip is a data augmentation feature which play significant role in enhancing the overall accuracy of classifier. Convolution layer scans the same image again after flipping it horizontally if ”Horizontal flip” is set to true. Fig.4 shows the image with its horizontal flip. Convolution layer scans both ’AI’ and ’IA’ separately to produce 2 feature maps. Thus, called data augmentation feature.

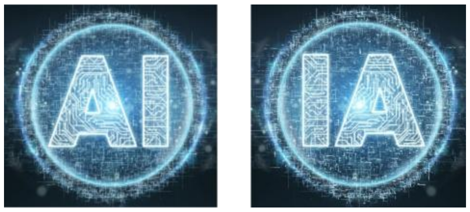


Fig.4. Horizontal flip

Mathematical notations associated to CNN are as follows:-

Rectified Linear unit(Activation function):

$$f(x) = max(0,x) \tag{2}$$

This function returns 0 if it receives any negative input, but for any positive value x it returns that value back. Softmax(Activation function):

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

$\vec{\ }$ ^{z_i}is standard exponential function where σ is the

sofmax, z is input vector, e

for input vector, K is number of classes in the multi-class classifier, e^{z_j} is standard exponential function for output vector. Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels. It returns the probability of each class. When the number of classes is two, it becomes the same as the sigmoid activation function. In other words, sigmoid is simply a variant of the softmax function.

Categorical cross-entropy(Loss Function)[1]:

$$Loss = -\sum_{i=1}^{outputsize} y_i \cdot \log \hat{y}_i \tag{4}$$

where $\hat{y}_i$ is the $i-th$ scalar value in the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output. This loss is a very good measure of how distinguishable two discrete probability distributions are from each other. In this context, y_i is the probability that event i occurs and the sum of all y_i is 1, meaning that exactly one event may occur. The minus sign ensures that the loss gets smaller when the distributions get closer to each other.

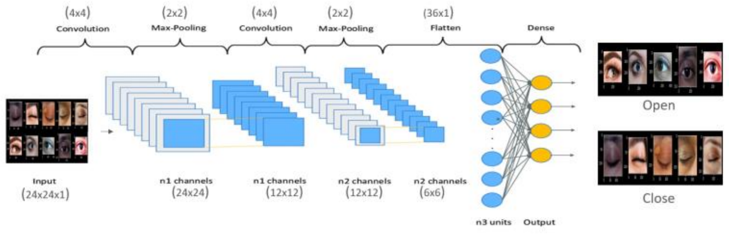


Fig.5. Architecture of CNN

[1] https://peltarion.com/knowledge-center/documentation/modeling-view/build-anai-model/loss-functions/categorical-crossentropy

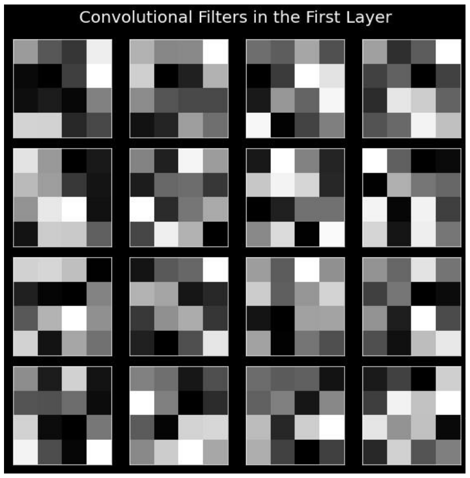


Fig.6. 16 filters(4x4) of convolution layer

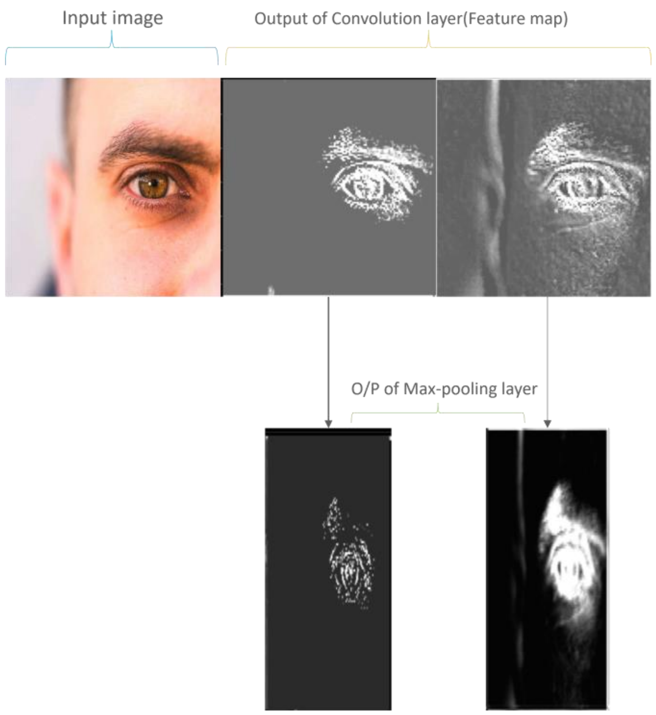


Fig.7. Sample output of convolution and max-pooling layer

4.3 Development Environments/Tools

1. **Tensorflow-** Platform for Keras.
2. **Keras-** To build classification model.
3. **OpenCV-** For object(Eye, Face, trees etc) detection.
4. **Anaconda-** To execute the main file(sleepsensor.py)
5. **Google Colab-** To train the CNN model on GPU.
6. **Pygame-** To play the buzzer/alarm sound.

5 Implementation

In the proposed method(Similar to [1]), the model is first configured with the best combination of hyperparamters in order to achieve the maximum validation/testing accuracy. Table 1 contains the listing of different combinations realized for the model training. Out of these, the model which gives the highest accuracy of 96% is equipped with 3 convolutional layers and 2 fully connected layer. Extracted eye(Both close eye and open eye) images(sample dataset shown in fig.4) with size of 256 X 256 are passed as input to the first convolution layer (Conv2d). In Conv2d, input image is convolved with 16 filters of size 4x4. After Conv2d, batch Normalization, ReLU transfer function, Max pooling2d(2×2) are incorporated in the architecture. Conv2d required 272 parameters. The output of Max pooling2d is fed in to the second convolution layer(Conv2d 1). In Conv2d 1, input is convolved with 32 filters with size 4x4 each. After Conv2d 1, batch Normalization, ReLU transfer function, MaxPooling2d 1(2x2) applied. Conv2d 1 required 8224 parameters. The output of MaxPooling2d 1 is fed in to the third convolution layer(Conv2d 2). In Conv2d 2, input is convolved with 128 filters with size 4x4 each. After Conv2d 2, Batch Normalization, ReLU transfer function, MaxPooling2d 2(2x2) followed by dropout(0.25) applied. Conv2d 2 required 36992 parameters.

Fully connected layer(dense) followed by dropout(0.5) required 4194336 parameters. Proposed CNN model required 4,239,890 trainable parameters. Fig.9 illustrates the total number parameters for each convolution and fully connected layer. The output of classifier is two state, so output layer having only two outputs. Adam method is used for Optimization. Here softmax activation function is used for classification. CNN's layer plot is shown in Fig.8.

CNN training algorithm:-

Step 1- Import packages/libraries
  a. import keras
  b. import tensorflow
Step 2- Load dataset
    a. dataset path
    b. "ImageDataGenerator()" - Lables the dataset automatically.

c. flow from directory() - To convert dataset into network format.

Step 3- Defining layers of CNN

a. ”Conv2D(), MaxPooling2D()” - First convolution and max-pooling layer.
b. ”Conv2D(), MaxPooling2D()” - Second convolution and max-pooling layer.
c. Conv2D(), MaxPooling2D() - Third convolution and max-pooling layer.
d. ’Dropout(0.25)’ - To prevent over-fitting.
e. Flatten() - Flatten layer.
f. Dense() - Fully connected layer
g. ’Dropout(0.5)’ - To prevent over-fitting.
h. Dense() - Output layer

Step 4- Define hyperparameters

a. ’categorical crossentropy’ -Loss function
b. ’adam’ - Optimizer
c. epochs, batch size
d. ModelCheckpoint- Saves the best training model

Step 5 - Training execution model.fit generator() -

To start the training.

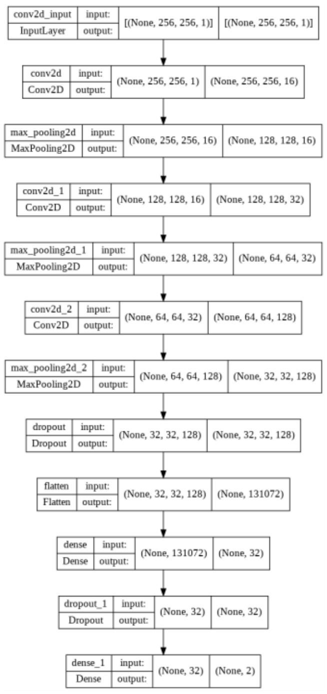
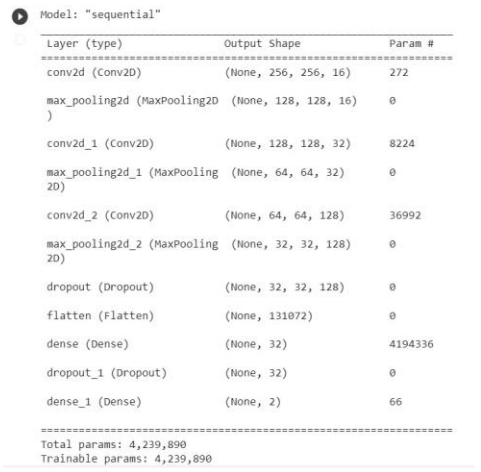

conv2d_input InputLayer input: output: [(None, 256, 256, 1)] [(None, 256, 256, 1)]
conv2d Conv2D input: output: (None, 256, 256, 1) (None, 256, 256, 16)
max_pooling2d MaxPooling2D input: output: (None, 256, 256, 16) (None, 128, 128, 16)
conv2d_1 Conv2D input: output: (None, 128, 128, 16) (None, 128, 128, 32)
max_pooling2d_1 MaxPooling2D input: output: (None, 128, 128, 32) (None, 64, 64, 32)
conv2d_2 Conv2D input: output: (None, 64, 64, 32) (None, 64, 64, 128)
max_pooling2d_2 MaxPooling2D input: output: (None, 64, 64, 128) (None, 32, 32, 128)
dropout Dropout input: output: (None, 32, 32, 128) (None, 32, 32, 128)
flatten Flatten input: output: (None, 32, 32, 128) (None, 131072)
dense Dense input: output: (None, 131072) (None, 32)
dropout_1 Dropout input: output: (None, 32) (None, 32)
dense_1 Dense input: output: (None, 32) (None, 2)

Fig.8. CNN’s layer plot



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      272

 max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      8224

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       36992

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 128)      0
 2D)

 dropout (Dropout)           (None, 32, 32, 128)       0

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 32)                4194336

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 4,239,890
Trainable params: 4,239,890
```

Fig.9. CNN parameters(Weights and biases)

6 Experiments and Results

Alike [1], we conducted 2 types of experiments. First experiment is carried out on collected dataset. Second experiment is performed on video(dynamic input). In first experiment, we compiled a dataset of around 1452 images. Few samples from the dataset are shown in Fig.10. The images of the dataset does not belongs to any specific format. One can customize or enhance the size of dataset by adding any type of eye images. Out of 1452 images, 726 images are sleepy images and remaining are non-sleepy images. For the experiment, a total of 1234 images are used for training out of which 617 images are sleepy images and another 617 images are non-sleepy images. In total, 218 images are used for validation out of which 109 images are of closed eye and rest 109 are of open eye. After performing the hyper-parameter analysis, proposed model has achieved an accuracy of 96% on validation dataset. Table 1 shows accuracy of the proposed model with respect to corresponding set of hyper-parameters. 'Output' column in Table 1 mapped with 'fig' referring to the associated plots and snapshots. Confusion matrix.1 and the classification report.1 for the model with highest validation accuracy(96%) are shown in Fig.23 and Fig.24. Confusion matrix.2 and the classification report.2 for the model with validation accuracy of 91% are shown in Fig.25 and Fig.26.

In second type of experiment, first we have trained our model with 1234 samples. In testing stage, we extracted the video frames via webcam and set the buzzer trigger when the model predicts close eye state for specified threshold. OpenCV function cv2.VideoCapture(0) is used to access video camera. During training spell, we used static eye images but during testing key frames are extracted from dynamic video inputs and predicted against the trained static images. cap.read() function is reading and storing each frame in desired variable. Given below the succinct algorithm description of the same.

Algorithm Description(2nd expirement[2]):

Step 1- Take input image from video camera/Webcam
cv2.VideoCapture(0), an OpenCV library function, is used to access the webcam. cap.read() function reading and storing the frames in specified variable.

Step 2 - Detect eyes region
OpenCV's eye detection algorithm is used to detect/read eye region consistently from live video footage. The algorithm takes gray scale image as input. Thus we are converting the image into required format using "cv2.cvtColor(frame, cv2.COLOR BGR2GRAY)". Then eye region extraction is performed by "re1=frame[y:y+h,x:x+w]".

Step 3 - Push the captured eyes data into the classification model Extracted eye region push to classifier(CNN). Classifier will decide whether eyes are open or closed. cv2.resize(re1,(24,24)) re-scaling the image as per the compatibility of training model. load model('CNN/classifier1.h5')- Loads the trained model. model.predict(re1)- Performs the prediction on extracted eye.

Step 4 - If eyes are closed for a specified time interval then the model will predict the driver as sleepy and play the alarm sound. cv2.putText(frame, "Open", (12, height-23), font, 1, (255,255,255), 1, cv2.LINE AA) - Display real time status of the person. sound.play()- Trigger the buzzer if person is sleepy. Experimental flow diagram is shown in Fig.27. Results in 2nd type of experiment are given in Fig.28 and Fig.29.

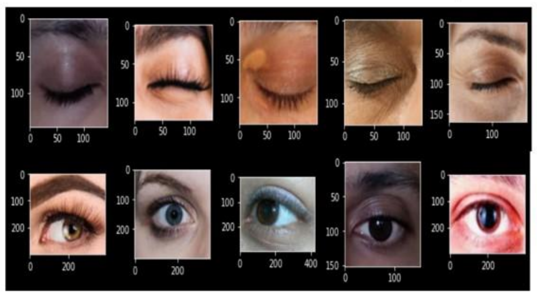


Fig.10. Sample Dataset(Both sleepy and non sleepy).

[2] https://data-flair.training/blogs/python-project-driver-drowsiness-detectionsystem/

Table 1. Accuracy v/s Hyperparameters

Training samples	Testing samples	Epochs	Batch size	kernel size	Dropouts	Accuracy	Output
1234	218	3x3	10	128	None	84%	Fig.11,Fig.12
1234	218	3x3	10	128	2	93%	Fig.13,Fig.14
1234	218	3x3	10	100	2	94%	Fig.15,Fig.16
1234	218	3x3	15	100	2	95%	Fig.17,Fig.18
1234	218	4x4	15	100	2	96%	Fig.19,Fig.20
741	711	4x4	15	100	2	91%	Fig.21,Fig.22

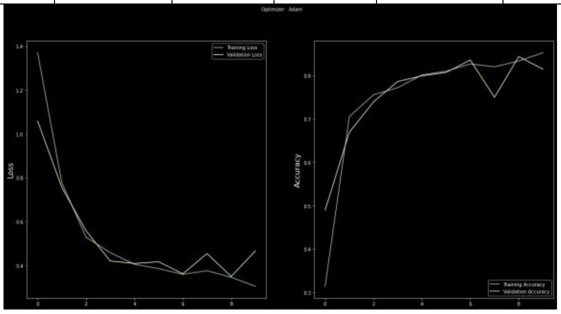
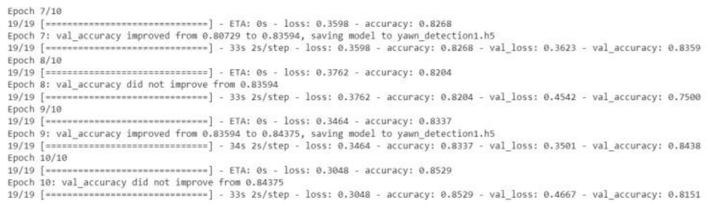


Fig.11. Accuracy(84%) and loss plot



```
Epoch 7/10
19/19 [==============================] - ETA: 0s - loss: 0.3598 - accuracy: 0.8268
Epoch 7: val_accuracy improved from 0.80729 to 0.83594, saving model to yawn_detection1.h5
19/19 [==============================] - 33s 2s/step - loss: 0.3598 - accuracy: 0.8268 - val_loss: 0.3623 - val_accuracy: 0.8359
Epoch 8/10
19/19 [==============================] - ETA: 0s - loss: 0.3762 - accuracy: 0.8204
Epoch 8: val_accuracy did not improve from 0.83594
19/19 [==============================] - 33s 2s/step - loss: 0.3762 - accuracy: 0.8204 - val_loss: 0.4542 - val_accuracy: 0.7500
Epoch 9/10
19/19 [==============================] - ETA: 0s - loss: 0.3464 - accuracy: 0.8337
Epoch 9: val_accuracy improved from 0.83594 to 0.84375, saving model to yawn_detection1.h5
19/19 [==============================] - 34s 2s/step - loss: 0.3464 - accuracy: 0.8337 - val_loss: 0.3501 - val_accuracy: 0.8438
Epoch 10/10
19/19 [==============================] - ETA: 0s - loss: 0.3048 - accuracy: 0.8529
Epoch 10: val_accuracy did not improve from 0.84375
19/19 [==============================] - 33s 2s/step - loss: 0.3048 - accuracy: 0.8529 - val_loss: 0.4667 - val_accuracy: 0.8151
```

Fig.12. Training output(*val accuracy* = 84%)

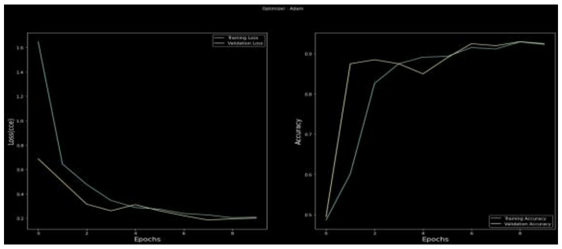
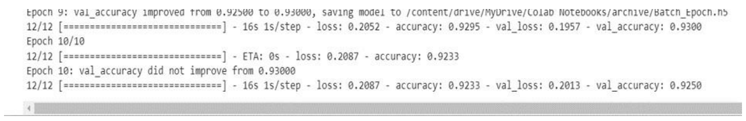


Fig.13. Accuracy(93%) and loss plot



```
Epoch 9: val_accuracy improved from 0.92500 to 0.93000, saving model to /content/drive/MyDrive/Colab Notebooks/archive/Batch_Epoch.h5
12/12 [==============================] - 16s 1s/step - loss: 0.2052 - accuracy: 0.9295 - val_loss: 0.1957 - val_accuracy: 0.9300
Epoch 10/10
12/12 [==============================] - ETA: 0s - loss: 0.2087 - accuracy: 0.9233
Epoch 10: val_accuracy did not improve from 0.93000
12/12 [==============================] - 16s 1s/step - loss: 0.2087 - accuracy: 0.9233 - val_loss: 0.2013 - val_accuracy: 0.9250
```

Fig.14. Training output(*val accuracy* = 93%)

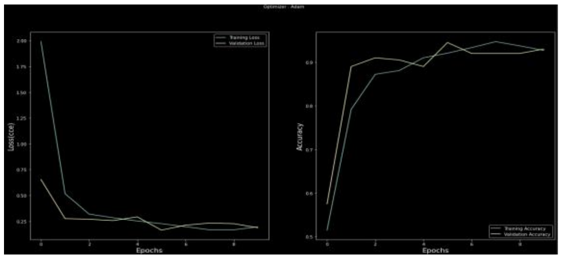
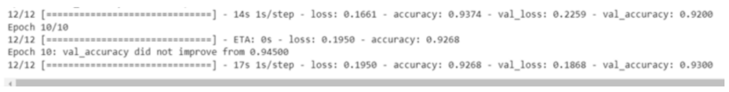


Fig.15. Accuracy(94%) and loss plot



```
12/12 [==============================] - 14s 1s/step - loss: 0.1661 - accuracy: 0.9374 - val_loss: 0.2259 - val_accuracy: 0.9200
Epoch 10/10
12/12 [==============================] - ETA: 0s - loss: 0.1950 - accuracy: 0.9268
Epoch 10: val_accuracy did not improve from 0.94500
12/12 [==============================] - 17s 1s/step - loss: 0.1950 - accuracy: 0.9268 - val_loss: 0.1868 - val_accuracy: 0.9300
```

Fig.16. Training output(*val accuracy* = 94%)

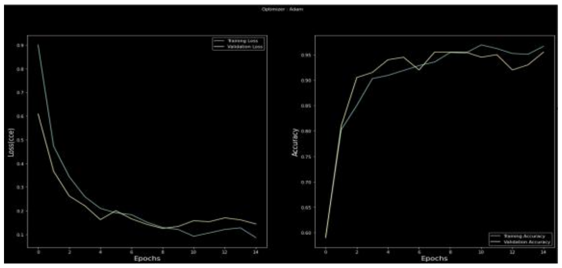
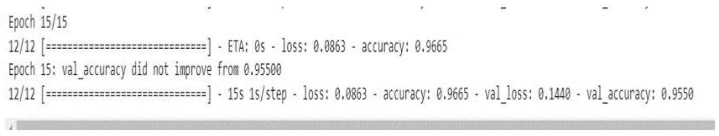


Fig.17. Accuracy(95%) and loss plot



```
Epoch 15/15
12/12 [==============================] - ETA: 0s - loss: 0.0863 - accuracy: 0.9665
Epoch 15: val_accuracy did not improve from 0.95500
12/12 [==============================] - 15s 1s/step - loss: 0.0863 - accuracy: 0.9665 - val_loss: 0.1440 - val_accuracy: 0.9550
```

Fig.18. Training output(*val accuracy* = 95%)

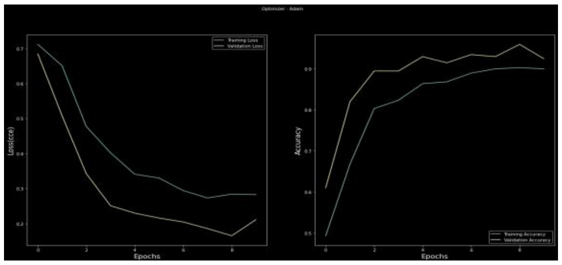
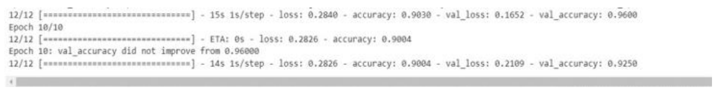


Fig.19. Accuracy(96%) and loss plot



```
12/12 [==============================] - 15s 1s/step - loss: 0.2840 - accuracy: 0.9030 - val_loss: 0.1652 - val_accuracy: 0.9600
Epoch 10/10
12/12 [==============================] - ETA: 0s - loss: 0.2826 - accuracy: 0.9004
Epoch 10: val_accuracy did not improve from 0.96000
12/12 [==============================] - 14s 1s/step - loss: 0.2826 - accuracy: 0.9004 - val_loss: 0.2109 - val_accuracy: 0.9250
```

Fig.20. Training output(*val accuracy* = 96%)

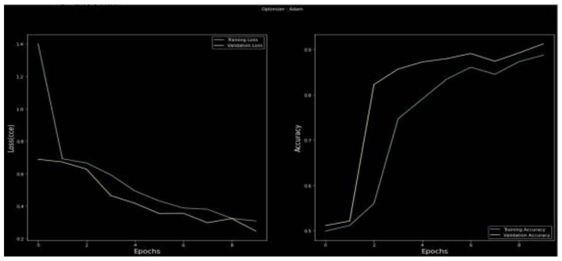
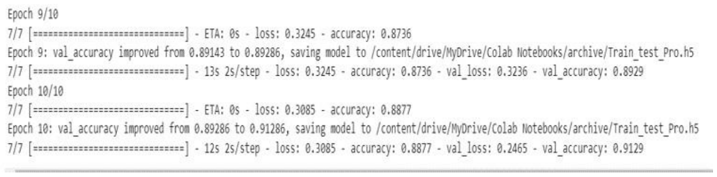


Fig.21. Accuracy(91%) and loss plot



```
Epoch 9/10
7/7 [==============================] - ETA: 0s - loss: 0.3245 - accuracy: 0.8736
Epoch 9: val_accuracy improved from 0.89143 to 0.89286, saving model to /content/drive/MyDrive/Colab Notebooks/archive/Train_test_Pro.h5
7/7 [==============================] - 13s 2s/step - loss: 0.3245 - accuracy: 0.8736 - val_loss: 0.3236 - val_accuracy: 0.8929
Epoch 10/10
7/7 [==============================] - ETA: 0s - loss: 0.3085 - accuracy: 0.8877
Epoch 10: val_accuracy improved from 0.89286 to 0.91286, saving model to /content/drive/MyDrive/Colab Notebooks/archive/Train_test_Pro.h5
7/7 [==============================] - 12s 2s/step - loss: 0.3085 - accuracy: 0.8877 - val_loss: 0.2465 - val_accuracy: 0.9129
```

Fig.22. Training output(*val accuracy* = 91%)

```
[[105   4]
 [  5 104]]
```

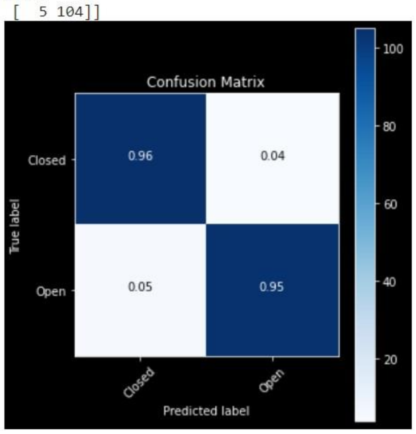


Fig.23. Confusion matrix.1

```
Classification Report
              precision    recall  f1-score   support

      Closed       0.95      0.96      0.96       109
        Open       0.96      0.95      0.96       109

    accuracy                           0.96       218
   macro avg       0.96      0.96      0.96       218
weighted avg       0.96      0.96      0.96       218
```

Fig.24. Classification report.1

```
[[104   5]
 [ 14  95]]
```

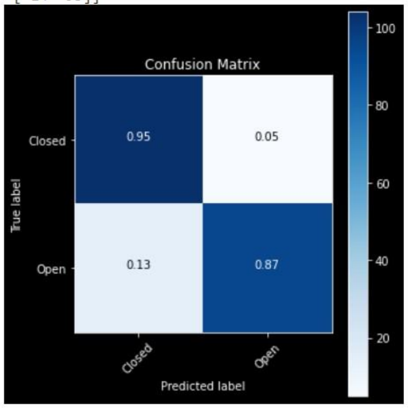
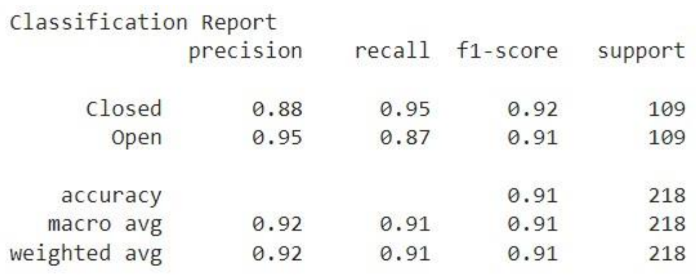


Fig.25. Confusion matrix.2

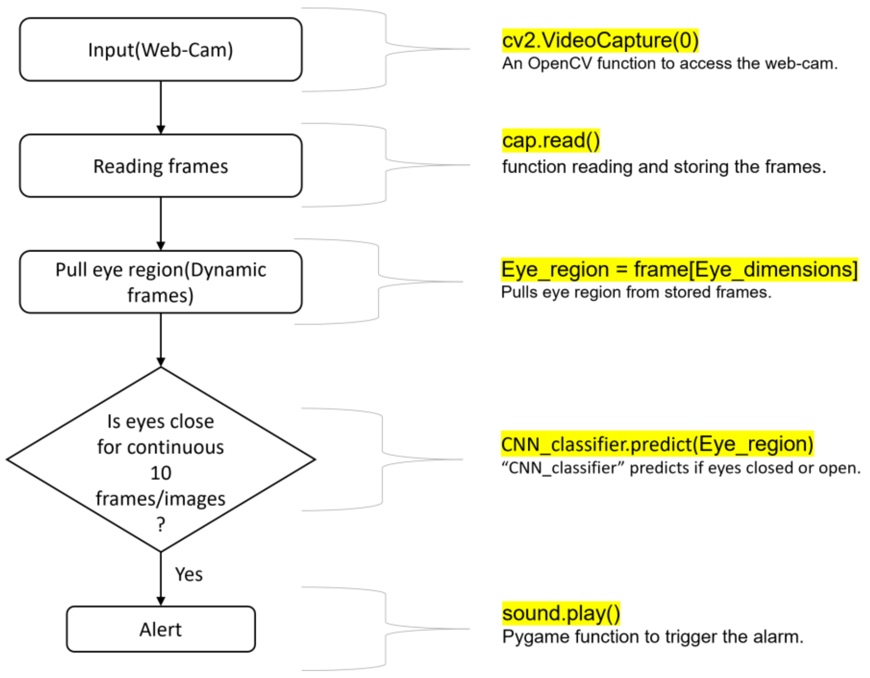


```
Classification Report
              precision    recall  f1-score   support

      Closed       0.88      0.95      0.92       109
        Open       0.95      0.87      0.91       109

    accuracy                           0.91       218
   macro avg       0.92      0.91      0.91       218
weighted avg       0.92      0.91      0.91       218
```

Fig.26. Classification report.2


Input(Web-Cam)
Reading frames
Pull eye region(Dynamic frames)
Is eyes close for continuous 10 frames/images ?
Yes
Alert

cv2.VideoCapture(0)
An OpenCV function to access the web-cam.

cap.read()
function reading and storing the frames.

Eye_region = frame[Eye_dimensions]
Pulls eye region from stored frames.

CNN_classifier.predict(Eye_region)
"CNN_classifier" predicts if eyes closed or open.

sound.play()
Pygame function to trigger the alarm.

Fig.27. Sensor's workflow

Fig.28. Visuals of detection sensor(Without glasses)

Fig.29. Visuals of detection sensor(With glasses)

7 Conclusion

The proposed idea of sleep detection sensor is built over the concepts of convolution neural networks(CNN). The system exploiting python's OPENCV libraries for detecting and extracting the eye region from the stored frames. Depending upon the input(Eye images), CNN is developed to generate the prediction regarding the sleep or non-sleep. The model trained and tested with various combinations of hyperparamerters. According to the results, proposed system has clocked the validation accuracy of 96%. This concludes that the network is efficient enough for the real world driver-assistance systems. The model also furnishes excellent response even the driver is observed wearing glasses(not sunglasses). In addition, the classifier is built in such a way that it can be trained on any random eye dataset. Thus, the input format is not a problem in presence of its rescalling/reshaping property. Nevertheless, there is still some room for the performance elevation. Our future work constitute the mobile(Android/iOS) version of sensor system with a supplemental feature of spotting the activities causing distraction like using cell phone while driving. This also incorporates the automated hyperparameter optimization to select the best configurations there by eliminating the manual efforts for maximum validation/testing accuracy.

References

1. Chirra, V.R.R., ReddyUyyala, S., Kolli, V.K.K.: Deep cnn: A machine learningapproach for driver drowsiness detection based on eye state. Rev. d'Intelligence Artif. **33**(6), 461–466 (2019)
2. Danisman, T., Bilasco, I.M., Djeraba, C., Ihaddadene, N.: Drowsy driver detectionsystem using eye blink patterns. In: 2010 International Conference on Machine and Web Intelligence. pp. 230–233. IEEE (2010)
3. Jabbar, R., Al-Khalifa, K., Kharbeche, M., Alhajyaseen, W., Jafari, M., Jiang, S.: Real-time driver drowsiness detection for android application using deep neural networks techniques. Procedia computer science **130**, 400–407 (2018)
4. Krajewski, J., Sommer, D., Trutschel, U., Edwards, D., Golz, M.: Steering wheelbehavior based estimation of fatigue. In: Proceedings of the fifth international driving symposium on human factors in driver assessment, training and vehicle design. pp. 118–124. Public Policy Center, University of Iowa (2009)
5. Mardi, Z., Ashtiani, S.N.M., Mikaili, M.: Eeg-based drowsiness detection for safedriving using chaotic features and statistical tests. Journal of medical signals and sensors **1**(2), 130 (2011)
6. Noori, S.M.R., Mikaeili, M.: Driving drowsiness detection using fusion of electroencephalography, electrooculography, and driving quality signals. Journal of medical signals and sensors **6**(1), 39 (2016)

7. Picot, A., Charbonnier, S., Caplier, A.: On-line detection of drowsiness using brainand visual information. IEEE Transactions on systems, man, and cybernetics-part A: systems and humans **42**(3), 764–775 (2011)
8. Said, S., AlKork, S., Beyrouthy, T., Hassan, M., Abdellatif, O., Abdraboo, M.: Realtime eye tracking and detection—a driving assistance system. Adv. Sci. Technol. Eng. Syst. J **3**(6), 446–454 (2018)
9. Tadesse, E., Sheng, W., Liu, M.: Driver drowsiness detection through hmm baseddynamic modeling. In: 2014 IEEE International conference on robotics and automation (ICRA). pp. 4003–4008. IEEE (2014)
10. Xu, L., Li, S., Bian, K., Zhao, T., Yan, W.: Sober-drive: A smartphone-assisteddrowsy driving detection system. In: 2014 International conference on computing, networking and communications (ICNC). pp. 398–402. IEEE (2014)
11. You, C.W., Lane, N.D., Chen, F., Wang, R., Chen, Z., Bao, T.J., Montes-de Oca,M., Cheng, Y., Lin, M., Torresani, L., et al.: Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In: Proceeding of the 11th annual international conference on Mobile systems, applications, and services. pp.
13–26 (2013)