STEAM MEMBERS:

ARCHANA S **(20S005)**

SHIVAYAVASHILAXMIPRIYA S **(20S030)**

TITLE : STUDENT-CARE (SLEEPINESS DETECTION SYSTEM)

1. Problem identification:

The problem identified is sleepiness of the students. It is taken and addressed in this project for the benefit of the students.Nowadays students need to spend a lot of time on projects and assignments which make them tired and drowsy.The solution for the sleepiness problem aims to predict the sleepy state of students and to alert them.This will help them concentrate and take necessary breaks from the work. This problem aims to develop a sleepiness detection model with deep learning models.Addressing this problem will surely help the students.

Abstract:

The sleepiness detection model is developed with python programming language.Convolutional Neural Networks is a type of artificial neural network which is most popularly used for image and object classification.The detection system is built using 3 models - dlib and open-cv , 3 layer CNN model ,4 layer CNN model. These models are developed in visual studio and then the best model is found to be the 4 layer CNN model which gives the maximum accuracy of 96% with low training and validation loss.

2.Uniqueness and motivation of the problem:

As part of their studies, students must deal with a lot of work like projects and assignments everyday.They must manage days when a project or assignment deadline is approaching. There may be times when they must attend and concentrate on important class meetings or sessions , but due to tiredness they end up in deep sleep.Sometimes in order to complete the assignments, students tend to spend extra hours which makes them extremely tired and makes them sleep. In this project, a unique drowsiness detection system is implemented to detect drowsiness of the students .The motive of this problem is to deal with sleepiness of students

using CNN models. Most of the time, when we sleep, our eyes are closed and our heads are bobbing.This is where machine learning and deep learning helps us.Using the eye movement and state, drowsiness can be detected. This type of problem is already dealt with in the driver drowsiness system. It is taken as the reference for our problem.

Technologies used:

1.Kaggle kernel- It is used to build the CNN models for the sleepiness detection system.

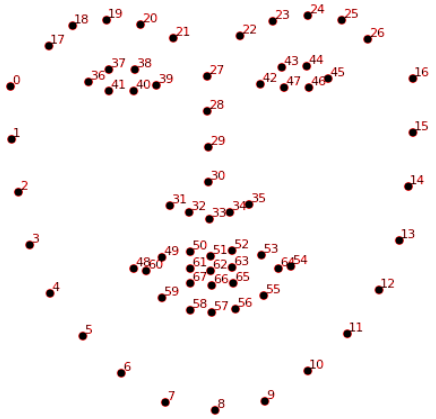
2.Visual studio - It is used to build the 3 py models for the System.

3.Python 3.9.6

3.Generation/Identification of dataset:

Model 1- Dlib and opencv:

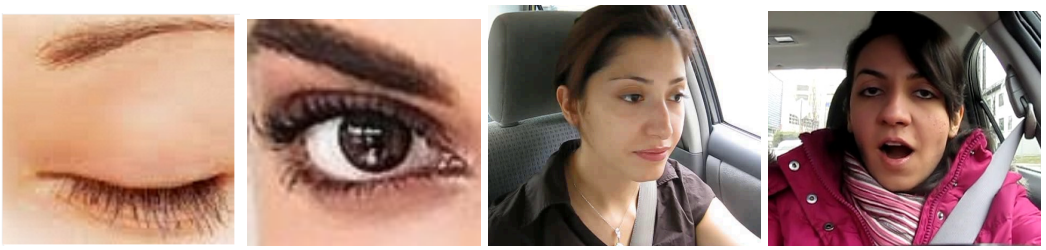
Dlib is a landmark's facial detector with pre-trained models, it is used to estimate the location of 68 coordinates (x, y) that map the facial points on a person's face. Opencv is an open-source library used for computer vision and image classification and face detection. There are no specific datasets used for this model. But to detect the face , we use the haar cascade classifier which is implemented by the open-cv library.This opencv provides pretrained face detection models and provides an interface to train a model.This haar cascade classifier uses the famous Viola Jones algorithm which consists of coded lists of decision trees in which each vertex one Haar feature claims the face and not face features.Also the shape_predictor_68_face_landmarks.dat file downloaded from kaggle is used to detect and predict the 68 landmarks in the face.



Model 2&3 - 3 layer and 4 layer CNN:

For developing these CNN models we use the yawn_eye_dataset_new dataset available in kaggle. This dataset is identified from the solution of the driver drowsiness system. This dataset contains 2900 images splitted into two directories: train and test. There are 433 images in the testing dataset and 2467 images in the training dataset. The 2 directories consist of images of eyes opened, closed and mouth yawn and no yawn images.These images are used for the development of both the CNN models.

Sample data:

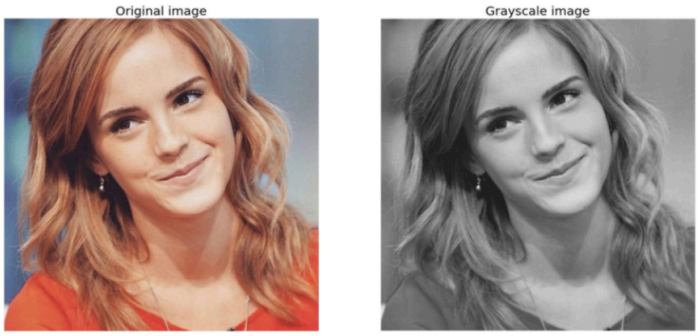


We predict sleepiness with the closed and open images of eyes for the new testing data.This dataset is used already for driver drowsiness and it is used here for the student sleepiness detection system.

4.Data preprocessing techniques as compared with your literature:

The purpose of using preprocessing steps in a face detection system is to speed up the detection process and reduce false positives. A preprocessing step should reject an acceptable amount of non-face windows.

I. Grayscaling:



The grayscaling is used to convert an image from a color space to shades of gray. It varies between white and black colors.

It is used mainly for :

1.Dimensionality reduction:

Grayscale images are single dimensional whereas other color spaces are 3 dimensional.

2.Reducing model complexity

3.For many algorithms to work.

The opencv's cv2.cvtColor() function is used to convert rgb images into grayscale images.

II. shape_to_np:

The dlib face landmark detector will return a shape object containing the 68 *(x, y)*-coordinates of the facial landmark regions.Using the shape_to_np function, we can convert this object to a NumPy array.

III. labelbinarizer:

In CNN models we use this function for preprocessing.The preprocessing is mainly done with the dataset collected, here to convert the multiclass labels into binary class labels we use labelbinarizer.It will turn a list into a matrix, where the number of columns in the target matrix is exactly as many as unique value in the input set.

```
from sklearn.preprocessing import LabelBinarizer
label_bin = LabelBinarizer()
y = label_bin.fit_transform(y)
```

IV. Resizing the images:

For resizing the images we use ImageDataGenerator which is used for getting the original data and transforming it on a random basis. It is also used to carry out the data augmentation where we can get the increment in the generalization of our model. It supports operations such as rotations, translations,shearin,scale changes and flips. In the 1st CNN model we use it to rescale the images and also for grayscaling. In the 2nd CNN model we use it to rescale and horizontally flip the images and rotate it to 30 degrees.

```
def generator(dir, gen=image.ImageDataGenerator(rescale=1./255), shuffle=True,batch_size=1,target_size=(24,24),class_mode='categorical' ):

    return gen.flow_from_directory(dir,batch_size=batch_size,shuffle=shuffle,color_mode='grayscale',class_mode=class_mode,target_size=target_size)
```

```
train_generator = ImageDataGenerator(rescale=1/255, zoom_range=0.2, horizontal_flip=True, rotation_range=30)
test_generator = ImageDataGenerator(rescale=1/255)

train_generator = train_generator.flow(np.array(X_train), y_train, shuffle=False)
test_generator = test_generator.flow(np.array(X_test), y_test, shuffle=False)
```

V. Splitting the dataset:

The dataset is splitted into train and testing data before model building. The first model of CNN splits data into two batches of train and valid batch that uses the images in the 2 directories of our dataset. The 2nd model splits the entire dataset into a 70:30 ratio for training and testing the data.

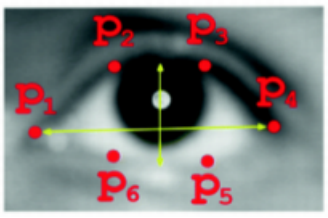
```
from sklearn.model_selection import train_test_split
seed = 42
test_size = 0.30
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=seed, test_size=test_size)
```

5.Experiments / Algorithms applied on the data:

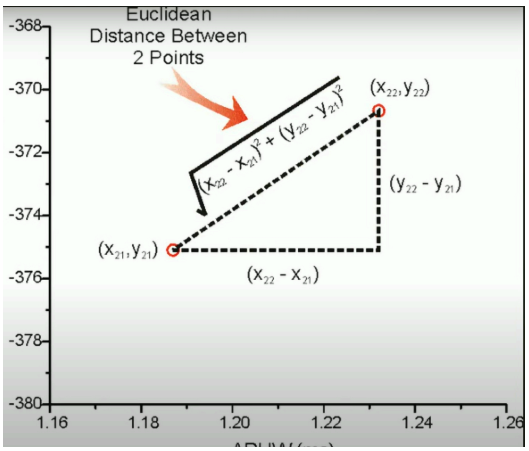
Totally 3 models are developed :

1.Dlibs and CNN model:

Overview: Here we use Open-cv and dlib to build the model. The 68 landmarks in the dataset are detected first. Then the euclidean distance among the landmarks of the eyes is computed and then the EAR ratio i.e. Eye Aspect Ratio is used to detect whether the eyes are open or close.The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks



$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$



If the ratio is greater than 0.25 then the student is said to be in an active state. If the ratio is between 0.21 and 0.25 then the user is in a drowsy state and else the person is in a sleepy state.

2.CNN models:

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The steps involved in CNN models are defined as:

ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer. ReLU performs an element-wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a rectified feature map.

Pooling is a down-sampling operation that reduces the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.

Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.The flattened matrix is fed as input to the fully connected layer to classify the image.

Epochs-indicates the number of passes of the entire training dataset the machine learning

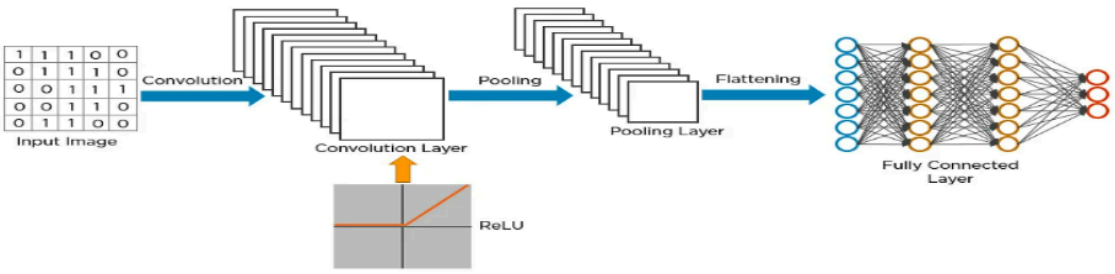
algorithm has completed.

Softmax-the function at the last layer of the neural network which calculates the probability distribution of the event over 'n' different events.

Batch- The number of samples to work through before updating the internal model parameters.

The common steps of a basic CNN model are:

- The pixels from the images of the dataset are fed to the convolutional layer that performs the convolution operation
- It results in a convolved map
- The convolved map is applied to a ReLU function to generate a rectified feature map
- The image is processed with multiple convolutions and ReLU layers for locating the features
- Different pooling layers with various filters are used to identify specific parts of the image
- The pooled feature map is flattened and fed to a fully connected layer to get the final output



3 layer CNN model:

In this model the dataset is preprocessed and then built with 3 layers and then we use 30 epochs to build the CNN model and evaluate accuracy.

```
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(24,24,1)),
    MaxPooling2D(pool_size=(1,1)),
    Conv2D(32,(3,3),activation='relu'),
    MaxPooling2D(pool_size=(1,1)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(1,1)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

4 layer CNN model:

In this model the dataset is preprocessed and then built with 4 layers and then we use 50 epochs to build the model and evaluate accuracy.

```
model = Sequential()

model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(145,145,3)))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="softmax"))

model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer="adam")

model.summary()
```

6.Results obtained:

1) Dlib and opencv model:

i) Importing the necessary libraries:

```
import cv2
import numpy as np
import dlib
from imutils import face_utils
```

ii) Open a webcam using the VideoCapture function and then define the functions for computing the euclidean distance and the EAR ratio.

```
cap = cv2.VideoCapture(0)
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
sleep = 0
drowsy = 0
active = 0
status = ""
color = (0, 0, 0)
def compute(ptA, ptB):
    dist = np.linalg.norm(ptA - ptB)
    return dist
def blinked(a, b, c, d, e, f):
    up = compute(b, d) + compute(c, e)
    down = compute(a, f)
    ratio = up/(2.0*down)
    if(ratio > 0.25):
        return 2
    elif(ratio > 0.21 and ratio <= 0.25):
        return 1
    else:
        return 0
```

iii) Using while loop we detect the faces and then compute the EAR ratio of the detected eyes

```
while True:
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)
    face_frame = frame.copy()
    for face in faces:
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()
        cv2.rectangle(face_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        landmarks = predictor(gray, face)
        landmarks = face_utils.shape_to_np(landmarks)
        left_blink = blinked(landmarks[36], landmarks[37],
                             landmarks[38], landmarks[41], landmarks[40], landmarks[39])
        right_blink = blinked(landmarks[42], landmarks[43],
                              landmarks[44], landmarks[47], landmarks[46], landmarks[45])
```

iv) Using the computed EAR ratio we set the status of the person as drowsy, active and sleepy states.

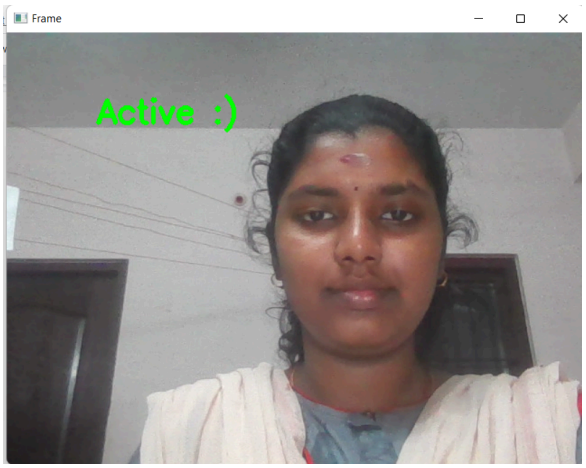
```
if(left_blink == 0 or right_blink == 0):
    sleep += 1
    drowsy = 0
    active = 0
    if(sleep > 6):
        status = "SLEEPING !!!"
        color = (255, 0, 0)

elif(left_blink == 1 or right_blink == 1):
    sleep = 0
    active = 0
    drowsy += 1
    if(drowsy > 6):
        status = "Drowsy !"
        color = (0, 0, 255)

else:
    drowsy = 0
    sleep = 0
    active += 1
    if(active > 6):
        status = "Active :)"
        color = (0, 255, 0)
```

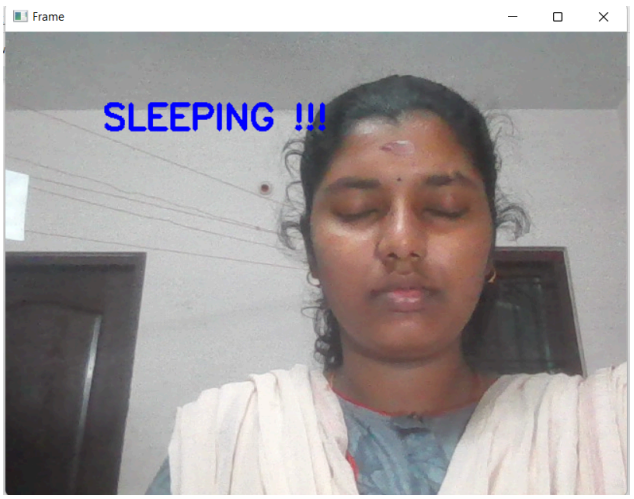
v) Display the frames of the detected faces with the status.

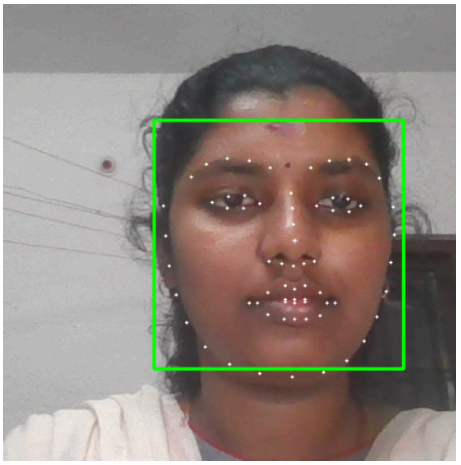
```
    cv2.putText(frame, status, (100, 100),
                cv2.FONT_HERSHEY_SIMPLEX, 1.2, color, 3)
    for n in range(0, 68):
        (x, y) = landmarks[n]
        cv2.circle(face_frame, (x, y), 1, (255, 255, 255), -1)
cv2.imshow("Frame", frame)
cv2.imshow("Result of detector", face_frame)
key = cv2.waitKey(1)
if key == 27:
    break
```

The results obtained from the dlib model developed is :

The first three figures denote the states of the person according to their face reactions. The last figure shows the 68 landmarks of the detected face.







2) The 3 layer CNN model with 30 epochs:

i) Building a CNN model and downloading it as “cnnCat2.h5” file.

Here first the model is built using kaggle kernel and downloaded as a h5 file and then the model is loaded in the python file created in the visual studio to detect the status of the person.In the kaggle kernel the 3 layer model is developed with 30 epochs and then fitted and downloaded.

```
77/77 [==============================] - 13s 167ms/step - loss: 0.0219 - accuracy: 0.9930 - val_loss: 0.1800 - val_accuracy: 0.9599
Epoch 25/30
77/77 [==============================] - 13s 168ms/step - loss: 0.0154 - accuracy: 0.9938 - val_loss: 0.1709 - val_accuracy: 0.9615
Epoch 26/30
77/77 [==============================] - 13s 168ms/step - loss: 0.0131 - accuracy: 0.9947 - val_loss: 0.2248 - val_accuracy: 0.9519
Epoch 27/30
77/77 [==============================] - 13s 169ms/step - loss: 0.0256 - accuracy: 0.9914 - val_loss: 0.1814 - val_accuracy: 0.9567
Epoch 28/30
77/77 [==============================] - 13s 173ms/step - loss: 0.0208 - accuracy: 0.9910 - val_loss: 0.1875 - val_accuracy: 0.9495
Epoch 29/30
77/77 [==============================] - 13s 165ms/step - loss: 0.0185 - accuracy: 0.9926 - val_loss: 0.1771 - val_accuracy: 0.9615
Epoch 30/30
77/77 [==============================] - 13s 175ms/step - loss: 0.0296 - accuracy: 0.9889 - val_loss: 0.1804 - val_accuracy: 0.9471
```

ii) Importing libraries in the python file used for loading the developed model and the sleepiness detection.

```
import cv2
import os
from keras.models import load_model
import numpy as np
from pygame import mixer
import time
```

iii) Including the paths of haar cascade classifier xml files for the frontal face, left and right eyes detection and also including the alarm sound file for alerting the student.

```
mixer.init()
sound = mixer.Sound('alarm.wav')
face = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
leye = cv2.CascadeClassifier('haarcascade_lefteye_2splits.xml')
reye = cv2.CascadeClassifier('haarcascade_righteye_2splits.xml')
lbl=['Close','Open']
model = load_model('cnnCat2.h5')
path = os.getcwd()
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0
thicc=2
rpred=[99]
lpred=[99]
```

iv)Reading the face from the webcam and then converting the images into grayscale and then detecting the face, left and right eyes.

```
while(True):
    ret, frame = cap.read()
    height,width = frame.shape[:2]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25))
    left_eye = leye.detectMultiScale(gray)
    right_eye =  reye.detectMultiScale(gray)
    cv2.rectangle(frame, (0,height-50) , (200,height) , (0,0,0) , thickness=cv2.FILLED )
    for (x,y,w,h) in faces:
        cv2.rectangle(frame, (x,y) , (x+w,y+h) , (100,100,100) , 1 )
```

v)For both the left and right eyes, the status is labeled as open and close based on the maximum probability value of the class it belongs to.

```
    for (x,y,w,h) in right_eye:
        r_eye=frame[y:y+h,x:x+w]
        count=count+1
        r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
        r_eye = cv2.resize(r_eye,(24,24))
        r_eye= r_eye/255
        r_eye=  r_eye.reshape(24,24,-1)
        r_eye = np.expand_dims(r_eye,axis=0)
        rpred = np.argmax(model.predict(r_eye), axis=-1)
        if(rpred[0]==1):
            lbl='Open'
        if(rpred[0]==0):
            lbl='Closed'
        break

    for (x,y,w,h) in left_eye:
        l_eye=frame[y:y+h,x:x+w]
        count=count+1
        l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
        l_eye = cv2.resize(l_eye,(24,24))
        l_eye= l_eye/255
        l_eye=l_eye.reshape(24,24,-1)
        l_eye = np.expand_dims(l_eye,axis=0)
        lpred = np.argmax(model.predict(l_eye), axis=-1)
        if(lpred[0]==1):
            lbl='Open'
        if(lpred[0]==0):
            lbl='Closed'
        break
```

vi)In every iteration of the while loop if the eyes are closed continuously the score is incremented and if the score value exceeds 15 then the person is alerted with the alarm sound.Else the score is zero and the alarm is not used.

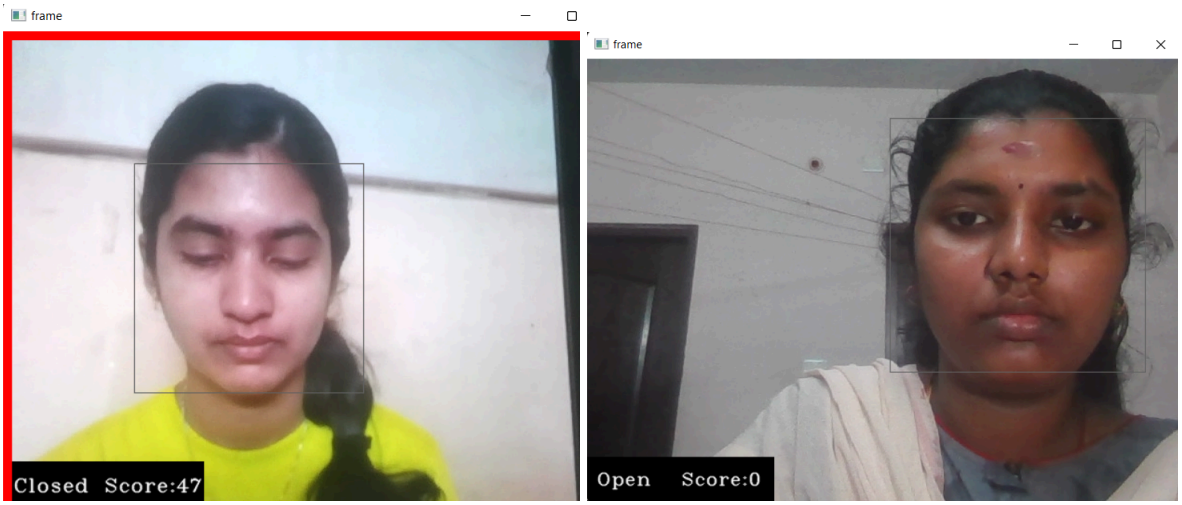
```
    if(rpred[0]==0 and lpred[0]==0):
        score=score+1
        cv2.putText(frame,"Closed",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
    # if(rpred[0]==1 or lpred[0]==1):
    else:
        score=score-1
        cv2.putText(frame,"Open",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
    if(score<0):
        score=0
    cv2.putText(frame,'Score:'+str(score),(100,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
    if(score>15):
        #person is feeling sleepy so we beep the alarm
        cv2.imwrite(os.path.join(path,'image.jpg'),frame)
        try:
            sound.play()
        except:  # isplaying = False
            pass
```

vii) Displaying the frame of the status shown by the face of the person.

```
81          if(thicc<16):
82              thicc= thicc+2
83          else:
84              thicc=thicc-2
85              if(thicc<2):
86                  thicc=2
87          cv2.rectangle(frame,(0,0),(width,height),(0,0,255),thicc)
88      cv2.imshow('frame',frame)
89      if cv2.waitKey(1) & 0xFF == ord('q'):
90          break
91  cap.release()
92  cv2.destroyAllWindows()
```

The result obtained from the 3 layer CNN model is:

The alarm sound plays when the first face is shown in the camera and the second images shows the open image with the score of 0.



3) 4 layer CNN model with 50 epochs:

i) Building a CNN model and downloading it as "drowsiness_new7.h5" file.

Here first the model is built using kaggle kernel and downloaded as a h5 file and then the model is loaded in the python file created in the visual studio to detect the status of the person.In the kaggle kernel the 4 layer model is developed with 50 epochs and then fitted and downloaded.

```
43/43 [==============================] - 7s 169ms/step - loss: 0.1102 - accuracy: 0.9564 - val_loss: 0.1052 - val_accuracy: 0.9619
Epoch 44/50
43/43 [==============================] - 7s 170ms/step - loss: 0.1078 - accuracy: 0.9555 - val_loss: 0.0940 - val_accuracy: 0.9706
Epoch 45/50
43/43 [==============================] - 7s 166ms/step - loss: 0.0929 - accuracy: 0.9659 - val_loss: 0.1021 - val_accuracy: 0.9689
Epoch 46/50
43/43 [==============================] - 7s 164ms/step - loss: 0.0860 - accuracy: 0.9636 - val_loss: 0.0851 - val_accuracy: 0.9706
Epoch 47/50
43/43 [==============================] - 7s 160ms/step - loss: 0.1041 - accuracy: 0.9569 - val_loss: 0.0955 - val_accuracy: 0.9706
Epoch 48/50
43/43 [==============================] - 7s 167ms/step - loss: 0.0982 - accuracy: 0.9614 - val_loss: 0.1365 - val_accuracy: 0.9602
Epoch 49/50
43/43 [==============================] - 8s 176ms/step - loss: 0.1205 - accuracy: 0.9503 - val_loss: 0.1008 - val_accuracy: 0.9671
Epoch 50/50
43/43 [==============================] - 7s 160ms/step - loss: 0.0874 - accuracy: 0.9636 - val_loss: 0.0823 - val_accuracy: 0.9740
```

ii) Importing necessary libraries and including alarm sound and detecting the frontal face, left and right eyes using the classifier.

```python
import cv2
import numpy as np
from keras.models import load_model
from tensorflow.keras.utils import img_to_array
from playsound import playsound
from threading import Thread
def start_alarm(sound):
    """Play the alarm sound"""
    playsound('alarm.wav')

classes = ['Closed', 'Open']
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_frontalface_default.xml')
left_eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_lefteye_2splits.xml')
right_eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_righteye_2splits.xml')
cap = cv2.VideoCapture(0)
model = load_model("drowiness_new7.h5")
count = 0
alarm_on = False
alarm_sound = "alarm.wav"
status1 = ''
status2 = ''
```

iii) Detecting the faces through the webcam and resizing it and preprocessing the image captured and detecting the face and eyes using haar cascade classifier. The class of the left eye is detected with the argmax function.

```python
while True:
    _, frame = cap.read()
    height = frame.shape[0]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 1)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        left_eye = left_eye_cascade.detectMultiScale(roi_gray)
        right_eye = right_eye_cascade.detectMultiScale(roi_gray)
        for (x1, y1, w1, h1) in left_eye:
            cv2.rectangle(roi_color, (x1, y1), (x1 + w1, y1 + h1), (0, 255, 0), 1)
            eye1 = roi_color[y1:y1+h1, x1:x1+w1]
            eye1 = cv2.resize(eye1, (145, 145))
            eye1 = eye1.astype('float') / 255.0
            eye1 = img_to_array(eye1)
            eye1 = np.expand_dims(eye1, axis=0)
            pred1 = model.predict(eye1)
            status1=np.argmax(pred1)
            #print(status1)
            #status1 = classes[pred1.argmax(axis=-1)[0]]
            break
```

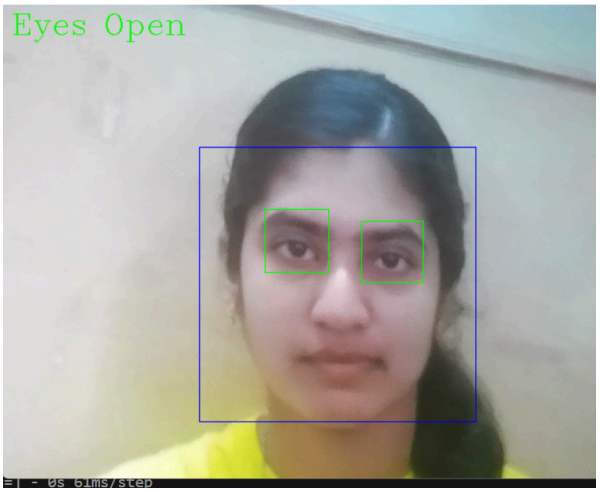
iv) The status of the right eye is detected and then the alarm is played if the count is greater than 10 here and the status is marked as closed else the eyes are open and the status is active.

```
        for (x2, y2, w2, h2) in right_eye:
            cv2.rectangle(roi_color, (x2, y2), (x2 + w2, y2 + h2), (0, 255, 0), 1)
            eye2 = roi_color[y2:y2 + h2, x2:x2 + w2]
            eye2 = cv2.resize(eye2, (145, 145))
            eye2 = eye2.astype('float') / 255.0
            eye2 = img_to_array(eye2)
            eye2 = np.expand_dims(eye2, axis=0)
            pred2 = model.predict(eye2)
            status2=np.argmax(pred2)
            #print(status2)
            #status2 = classes[pred2.argmax(axis=-1)[0]]
            break
        # If the eyes are closed, start counting
        if status1 == 2 and status2 == 2:
        #if pred1 == 2 and pred2 == 2:
            count += 1
            cv2.putText(frame, "Eyes Closed, Frame count: " + str(count), (10, 30), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 1)
            # if eyes are closed for 10 consecutive frames, start the alarm
            if count >= 10:
                cv2.putText(frame, "Drowsiness Alert!!!", (100, height-20), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)
                if not alarm_on:
                    alarm_on = True
                    # play the alarm sound in a new thread
                    t = Thread(target=start_alarm, args=(alarm_sound,))
                    t.daemon = True
                    t.start()
        else:
            cv2.putText(frame, "Eyes Open", (10, 30), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 1)
            count = 0
            alarm_on = False
```

v) The frame is displayed and the window is released.

```
        cv2.imshow("Drowsiness Detector", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

The result obtained from the 4 layer CNN model is:



The first image in the above figure shows the opened eyes and the second figure shows the sleeping face.

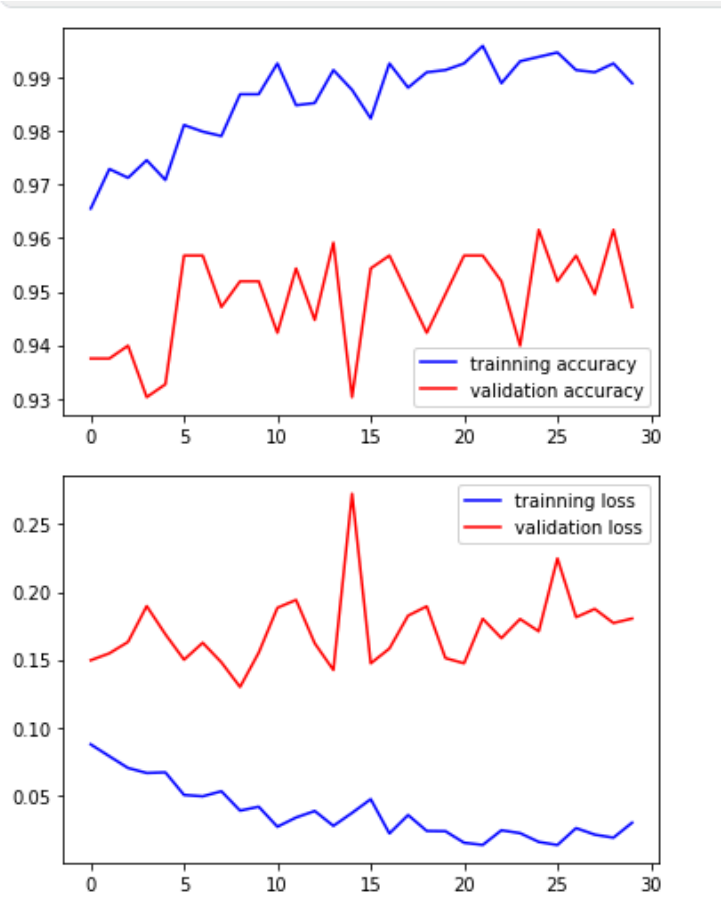
7.Significant results as compared with your literature survey:

Dlib model:

Dlib's shape detector is used to map the coordinates of the facial landmarks of the input video and drowsiness detected by monitoring aspect ratios of eyes and mouth.The maximum accuracy obtained for the Dlib model is 95%.

3 layer CNN model:

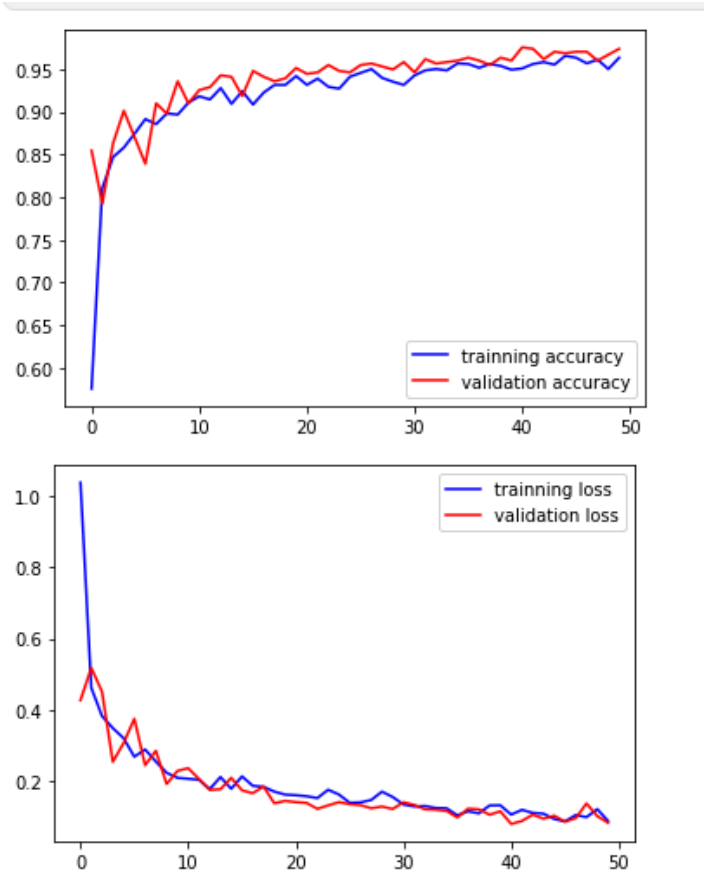
The maximum accuracy obtained from the 3 layer Sequential CNN model is 99% for the training dataset and 95% for the testing data.



The above graph shows the training and testing accuracy of the model developed using CNN. The 2nd part of the image shows the loss of the training and testing data used.We can infer that the training data shows more accuracy than the testing data.

4-layer CNN model:

The maximum accuracy obtained by the 4 layer Sequential CNN model for both the training and testing data is 97% and on an average the accuracy was 96%.



The classification report for the CNN model is:

```
from sklearn.metrics import classification_report
print(classification_report(np.argmax(y_test, axis=1), prediction, target_names=labels_new))
```

```
              precision    recall  f1-score   support

        yawn       0.93      0.84      0.88        63
     no_yawn       0.88      0.95      0.91        74
      Closed       1.00      0.94      0.97       215
        Open       0.95      1.00      0.97       226

    accuracy                           0.96       578
   macro avg       0.94      0.93      0.93       578
weighted avg       0.96      0.96      0.95       578
```

The above graph shows the accuracy and loss of the model developed with the 4 layer CNN model.

The accuracy of the models developed :

Model	Accuracy
Dlib and opencv model	95%
3 layer CNN model	Training data-99% Testing data - 95%
4 layer CNN model	Training data-97% Testing data - 97%

The literature survey showed the highest accuracy obtained with CNN models. In our project too the 4 layer model is found to be the best among the 3 models developed with an accuracy of 96%. This model is better than the other 2 models.

8.Future directions:

The model developed using CNN is a sequential model that gives the maximum accuracy among the deep learning models developed. Our future works include the development of other CNN models with more layers and more models like MobileNet. In future the aim of this project will be expanded to a web app to help students track their working and sleeping schedule. This would really help them to manage work and take enough breaks for nurturing their brain and their health properly.

Student life teaches everyone the most important lessons and requires proper management of all these work. This system developed will surely benefit the students in managing the work loads and help them to organize the work timings.

The other challenges faced while observing results is the model couldn’t detect the faces in low light areas.

In future, we will develop drowsiness detection using mouth state to improve the performance of the system. Also advanced sequential models and more CNN models will be included. Our project will be developed into an app or website in future works.