المندوبية السامية للتخطيط
HAUT-COMMISSARIAT AU PLAN

INSEA

RAPPORT DE STAGE D'APPLICATION

Driver drowsiness detection

Préparé par : Oussama Yousr

Filière : **DSE**

Sous la direction de : Mme. Imane Hilal (Laboratoire SI2M)

Année universitaire : 2020/2021

ABSTRACT:

Driver fatigue becomes one of the most important reason for road accidents that can cause deaths or serious injuries, Various Studies prove that if a driver is early identified as drowsy and be alarmed , the cases of accidents can be remarkably reduced , in this project , I propose some reliable methods toward drowsiness detection by analyzing images of the Driver's Face , using a large real-life dataset of 60 subjects , consists of around 30 hours of video.

Through this Project, Different methods will be implemented to extract meaningful features from Faces, the most important one is using spatiotemporal features of the faces , it will remarkably shows an increase compared to the existing accuracy.

Dedication

Before any development on this professional experience, it is necessary to start this report by addressing my thanks to Miss Imane Hilal, I would also like to thank my family for all the support they have given me throughout this internship, especially my Friend El Mehdi Soummer for his Support and brilliant Ideas, My sincere thanks also go to all the professors and administrators of the National Institute of Statistics and Applied Economics (INSEA) for their efforts, advice, and support.

Thanks, once again, to all those who have contributed in any way to the realization of this project.

Key words:

1. Deep learning
2. LSTM
3. Sequential Model
4. Computer Vision
5. CNN
6. Machine learning

Abbreviations

1. SWM: Steering Wheel Movement
2. SDLP: Standard Deviation of Lane Position
3. ECG: electrocardiogram
4. EMG: electromyogram
5. EEG: electroencephalogram
6. FC: – Fully-connected
7. CNN: A Convolutional Neural Network
8. KNN: K-nearest neighbors
9. LSTM: Long short-term memory
10. EAR: Eye Aspect Ratio
11. RNN: Recurrent neural network

Table of Content

Table of figures

Introduction

Driving involves the performance of a particular sequence of actions with situational awareness, as well as, quick and accurate decision making, situational awareness is critical in driving, as direct attention is required to process the perceived cues, monitoring attention status, is one of the most important parameters for safe driving.

According to the National Sleep Foundation, about half of U.S. adult drivers admit to consistently getting behind the wheel while feeling drowsy, about 20% admit to falling asleep behind the wheel at some point in the past year – with more than 40% admitting this has happened at least once in their driving careers.

Driving while drowsy is similar to driving under influence of alcohol, Driver's reaction times awareness of hazards and ability to sustain attention all worsen the drowsier the driver is .

Driving after Going more than 20 hours without sleep is the equivalent of driving with blood-alcohol concentration of 0.08%, we are more three times more likely to be in a car crash if you are fatigued, extremely dangerous, isn't it!?

A study by The AAA Foundation for traffic safety estimated that 328 000 drowsy driving crashes occur annually , that's more than three time the police-reported number , the same study found that 109 000 of those drowsy driving crashes resulted in an injury about 6 400 were fatal , the researchers suggest the prevalence of drowsy driving fatalities is more than 350% greater than reported , beyond the human toll is the economic one , NHTSA estimated fatigue-related crashes resulting in injury or death cost society 109$ billion annually , not including property damage.

Fatigue symptoms include : yawning , slow reaction time , eyelid closure (Blinks) , Variations in Driver Behavior , in order to deal with this major challenge , I propose in this work a method based on facial image analysis and tested on the realistic Dataset , there are now two major facial feature , EYE Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) , in this project I won't be using the MAR , the reason behind that is the confusion we're going to face to distinguish between a yawning and speaking (pronouncing A for example) so that's why I will use only the EYE Aspect Ratio , three Machine learning models are chosen in this work in order to order to predict whether the driver is drowsy or not.

Chapiter 1: LITERATURE REVIEW

I. EXISTING METHODS OF DROWSINESS DETECTION:

I.1 Mathematical Models:

Bio-mathematical models offer a quantitative analysis of the effect of sleep cycle on individual performance. Types of inputs used typically are circadian cycles, duration of sleep, duration of wakefulness and sleep history to predict risk of fatigue and performance quality.

One of the earliest models is the Two Process Model. This model is based on the interaction of two processes, i.e., the circadian Process 'C' and the homeostatic Process 'S'. These processes predict performance and fatigue levels. An upgrade to the two-process model is the Three Process Model of Alertness, which utilizes the duration of sleep and wakefulness as input to predict fatigue risk and alertness. This computer-based model considers both circadian and homeostatic components.

Various other models are developed in continuation of this which incorporate various combinations of possible inputs to predict fatigue.

I.2 Machine Learning Based Models:

Machine learning-based implementations are data-driven algorithms that have been trained on a large amount of driving data collected in the lab and on the road. Based on the levels of representation and the approach employed for feature derivation, the category can be classified into shallow and deep models.

I.2.1 Shallow Model:

Shallow models provide reasonable predictive ability with minimal complexity. Shallow models consist of a few layers and require limited training data, but they require predefined discriminative features. Well known techniques in this domain include Artificial Neural Networks (ANN) with one hidden layer and Support Vector Machine (SVM). In various techniques, ANNs and SVMs are trained on features like PERCLOS (% of eyelid closure), EEG and ECG signals of the driver, and other features, using in various permutations and combinations, to classify a driver as alert, drowsy or dangerous. However, as mentioned earlier, these techniques use precomputed features. Due to inability to determine these factors accurately, deep models are used.

I.2.2 DEEP MODELS:

Deep learning models are machine learning techniques which incorporate learning representation of data instead of task specific methods. In contrast to shallow models, deep models have the ability to extract the features from the training data. Convolutional neural networks (CNN) based models are primarily used for driver fatigue detection.

II. Features Used for Drowsiness Detection:

Researchers have used various methods to measure driver drowsiness such as:

II.1 Vehicle-Based Measures:

One of the most important method is vehicle-based Measures , these measurements are determined in a simulated environment by placing sensors on various vehicle components, including the steering wheel and the acceleration pedal; the signals sent by the sensors are then analyzed to determine the level of drowsiness. Some researchers found that sleep deprivation can result in a larger variability in the driving speed , However, the two most commonly used vehicle-based measures are the steering wheel movement and the standard deviation of lane position :

Steering Wheel Movement (SWM) : is measured using steering angle sensor and it is a widely used vehicle-based measure for detecting the level of driver drowsiness When drowsy, the number of micro-corrections on the steering wheel reduces compared to normal driving, Fairclough and Graham found that sleep deprived drivers made fewer steering wheel reversals than normal drivers .

Standard Deviation of Lane Position (SDLP) : is another measure through which the level of driver drowsiness can be evaluated . In a simulated environment, the software itself gives the SDLP and in case of field experiments the position of lane is tracked using an external camera

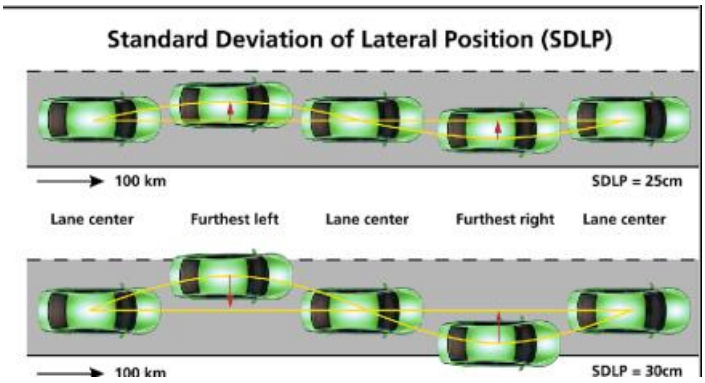


Figure 1 - Standard Deviation of Lane Position

II.2 Behavioral Measures:

A drowsy person displays a number of characteristic facial movements, including rapid and constant blinking, nodding or swinging their head, and frequent yawning , Some researchers used multiple facial actions, including inner brow rise, outer brow rise, lip stretch, jaw drop and eye blink, to detect drowsiness , these methods are still in progress .

most of the methods have been tested on data obtained from drivers mimicking drowsy behavior rather than on real video data in which the driver gets naturally drowsy , which not the case in this project , it is based on a real video Dataset .

List of previous works on driver drowsiness detection using behavioral measures.

Sensor used	Drowsiness Measure	Detection techniques	Feature Extraction	Classification	Positive Detection rate
CCD micro camera with Infra-Red Illuminator	Pupil	Ada-boost	Red eye effect, Texture detection method	Ratio of eye-height and eye-width	92%
Camera and Infra-Red Illuminator	PERCLOS, eye closure duration, blink frequency, and 3 other	Two Kalman filters for pupil detection	Modification of the algebraic distance algorithm for conics Approximation & Finite State Machine	Fuzzy Classifier	Close to 100%
CCD camera	Yawning	Gravity-center template and grey projection	Gabor wavelets	LDA	91.97%
Digital Video camera	Facial action	Gabor filter	Wavelet Decomposition	SVM	96%
Fire wire camera and webcam	Eye Closure Duration & Freq of eye closure	Hough Transform	Discrete Wavelet Transform	Neural Classifier	95%
Camera	Multi Scale dynamic features	Gabor filter	Local Binary Pattern	Ada boost	98.33%
IR Camera	Eye State	Gabor filter	Condensation algorithm	SVM	93%
Simple Camera	Eye blink	Cascaded Classifiers Algorithm detects face and Diamond searching Igorithm to trace the face	Duration of eyelid closure, No. of continuous blinks, Frequency of eye blink	Region Mark Algorithm	98%
Camera with IR Illuminator	PERCLOS	Haar Algorithm to detect face	Unscented Kalman filter algorithm	SVM	99%

Figure 2 - List of previous works on driver drowsiness detection using behavioral measures.

II.3 Physiological Measures:

As drivers become drowsy, their head begins to sway and the vehicle may wander away from the center of the lane. The previously described vehicle-based and vision based measures become apparent only after the driver starts to sleep, which is often too late to prevent an accident. However, physiological signals start to change in earlier stages of drowsiness. Hence, physiological signals are more suitable to detect drowsiness with few false positives.

Many researchers have considered the following physiological signals to detect drowsiness: electrocardiogram (ECG), electromyogram (EMG), electroencephalogram (EEG) and electro-oculogram (EoG) (**Table 3**). Some researchers have used the EoG signal to identify driver drowsiness through eye movements:

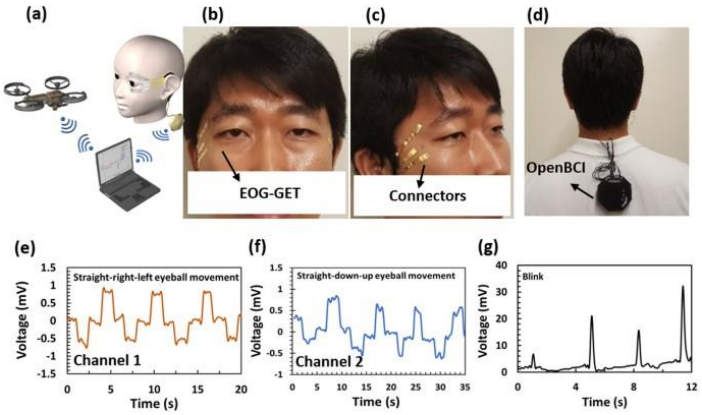


Figure 3 - EOG signals

EOG signals: Electrooculography (EOG) is a technique for measuring the cornea-retinal standing potential that exists between the front and the back of the human eye. The resulting signal is called the electrooculogram .

The heart rate (HR) also varies significantly between the different stages of drowsiness, such as alertness and fatigue. Therefore, heart rate, which can be easily determined by the ECG signal, can also be used to detect drowsiness.

Advantages and limitations of various measures:

Measures	Parameters	Advantages	Limitations
Vehicle based measures	Deviation from the lane position Loss of control over the steering wheel movements	Nonintrusive	Unreliable
Behavioral Measures	Yawning Eye closure Eye blink Head pose	Non-intrusive; Ease of use	Lighting condition Background
Physiological measures	Statistical & energy features derived from ECG EoG EEG	Reliable; Accurate	Intrusive

Figure 4 - Table of different measures

Hybrid measures: in some cases, there is no variation in vehicle-based parameters, which makes the drowsiness detector useless, so it would be beneficial to use multiple features:

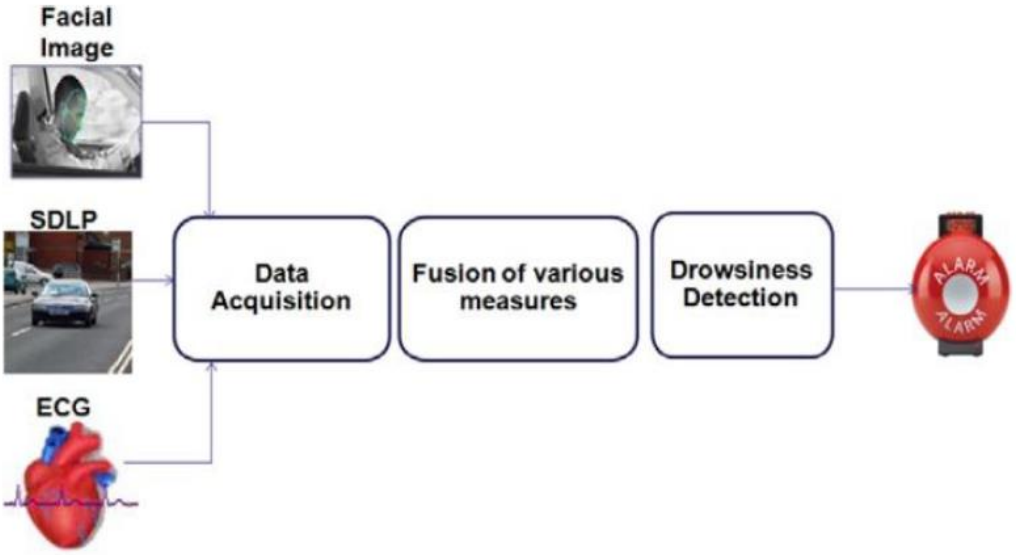


Figure 5 - Hybrid measures

Chapter 2: Data collection

There are numerous works in drowsiness detection , but none of them uses a dataset that a realistic , that was taken in the right body state and the accurate situation , so that was the purpose of the project , is to train a deep learning Model or even extract some useful features from faces using a realistic Dataset , UTA REAL-LIFE Drowsiness Dataset did the work perfectly , the RLDD (real life drowsiness Dataset) dataset consists of around 30 hours of RGB videos of 60 healthy participants , for each participant 3 videos , each one of them belong to a different class , so In total there is 180 videos .

participants were instructed to take three videos of themselves by their phone/Web camera in three different drowsiness States (ALERT, LOW VIGILANT, DROWSY) .

for around 10 minutes each, all videos were recorder in such an angle that both eyes were visible, and the camera was placed within one arm length away from the subject, taking in consideration the distance between the driver and the steering wheel, and also the participant were asked to do some additional tasks (reading, talking, yawning) .

The Dataset consists of 180 RGB Videos, Each Video is around 10 Minutes long, labeled as belonging to one of three classes: alert (labeled as 0) low vigilant (labeled as 5) and drowsy (labeled as 10) , the dataset was annotated by the participants themselves so it can be accurate.

Index of /~athitsos/projects/drowsiness

Name	Last modified	Size	Description
Parent Directory		-	
Fold1_part1.zip	20-Sep-2020 17:03	12G	
Fold1_part2.zip	20-Sep-2020 18:39	11G	
Fold2_part1.zip	19-Sep-2020 22:20	11G	
Fold2_part2.zip	19-Sep-2020 22:39	11G	
Fold3_part1.zip	20-Sep-2020 19:14	12G	
Fold3_part2.zip	20-Sep-2020 19:40	7.5G	
Fold4_part1.zip	20-Sep-2020 22:23	12G	
Fold4_part2.zip	21-Sep-2020 14:24	13G	
Fold5_part1.zip	21-Sep-2020 17:27	10G	
Fold5_part2.zip	21-Sep-2020 17:58	12G	
uta.jpg	19-Sep-2020 22:02	685K	

Apache/2.2.3 (Red Hat) Server at vlm1.uta.edu Port 80

Figure 6 - The whole realistic dataset (100Gb)

Chapter 3: Technologies and tools

I. Python :

Python is the most widely used open-source programming language for computer science. This language has propelled itself to the forefront of infrastructure management, data analysis and software development. Indeed, among its qualities, Python allows developers to focus on what they do rather than how they do it. It has freed developers from the constraints of form that occupied their time with older languages. Thus, developing code with Python is faster than with other languages.

II. google colab:

Colaboratory, often shortened to "Colab", is a product of Google Research. Colab allows anyone to write and run Python code of their choice through the browser. It is an environment particularly suited for machine learning, data analysis and education. In more technical terms, Colab is a hosted Jupyter notebook service that requires no configuration and provides free access to computing resources, including GPUs.

III. keras:

Keras is an open-source library written in Python (under the MIT license) based primarily on the work of Google developer François Chollet as part of the ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) project. A first version of the cross-platform software was released on March 28, 2015. The goal of this library is to allow the rapid constitution

of neural networks. In this context, Keras does not function as its own framework but as an application programming interface (API) for accessing and programming different machine learning frameworks. Theano, Microsoft Cognitive Toolkit (formerly CNTK) and TensorFlow are among the frameworks supported by Keras.

IV. Matpotlib :

Matplotlib is an open source Python library, originally developed by neurobiologist John Hunter in 2002. The goal was to visualize the electrical signals of the brain of people with epilepsy. To achieve this, he wanted to replicate the graphical creation features of MATLAB with Python.

Following the death of John Hunter in 2012, Matplotlib has been improved over time by many contributors from the open source community. It is used to create graphs, and high quality diagrams. It is an open source alternative to MATLAB.

V. Tensorflow:

TensorFlow is an open source machine learning tool developed by Google. The source code was opened on November 9, 2015 by Google and released under the Apache license.

It is based on the DistBelief infrastructure, initiated by Google in 2011, and has an interface for Python, Julia and R2

TensorFlow is one of the most widely used tools in AI in the field of machine learning3.

VI. Scikit Learn :

Scikit-learn is an artificial intelligence (AI) tool for data science projects. This machine learning framework includes an initial free library, with the possibility to integrate other free libraries. This is notably the case of SciPy and NumPy .

VII. Numpy :

NumPy is a library for the Python programming language, designed to manipulate matrices or multidimensional arrays as well as mathematical functions operating on these arrays.

Numpy :

More precisely, this free and open-source software library provides multiple functions allowing to create directly an array from a file or to save an array in a file, and to manipulate vectors, matrices and polynomials.

Chapter 4: Models

In this project , I have used three different Models , Convolutional neural network (CNN) , long short-term memory (LSTM) and a Machine Learning Model (KNN) .

In order to find the most optimized and accurate approach we can adopt to distinguish between a drowsy person and an alert one.

I. Convolutional Neural Networks:

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, CNNs have three main types of layers which are :

– convolutional layer

– pooling layer

– Fully-connected (FC) layer

the pre-processing is less required in a convolutional network ConvNET , it has the ability to learn features/characteristics from images or Multi-channel Matrix .

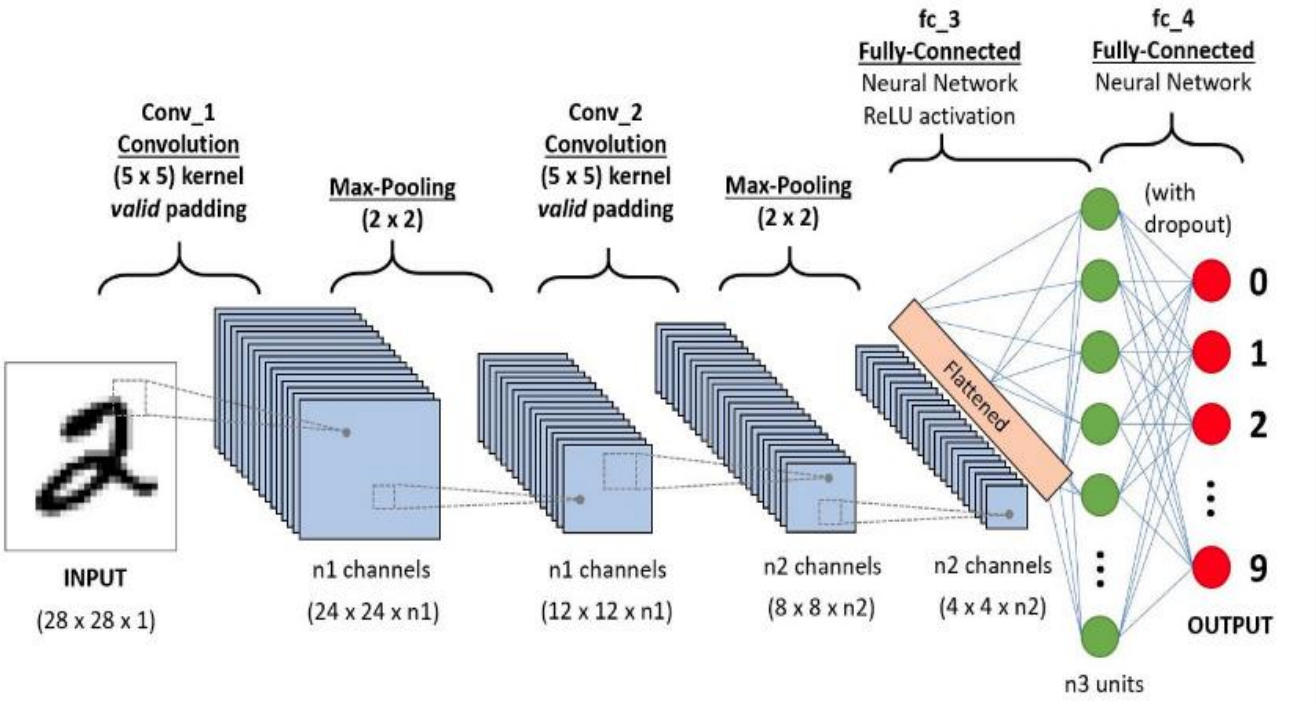


Figure 7 - A CNN sequence to classify handwritten digits (MNIST Dataset)

why convolutional neural network better than a simple dense neural network in processing images?

The reason that makes a CNN do so much better and out-perform a classic dense neural network Is that Conv Layers take advantage of inherent or inseparable proprieties of images .

– a simple FeedForward neural networks don't see any order in their inputs .

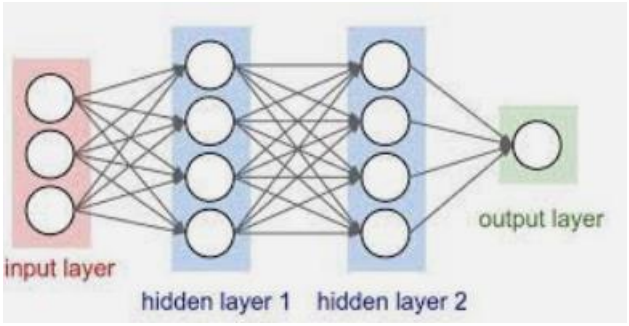


Figure 8 – Dense neural network

so, it will be no impact on the performance in case of shuffling input data .

– CNN in opposition, take advantage of local spatial coherence of images , they are able to reduce the number of operations needed to process an image using convolution on patches of adjacent pixels.

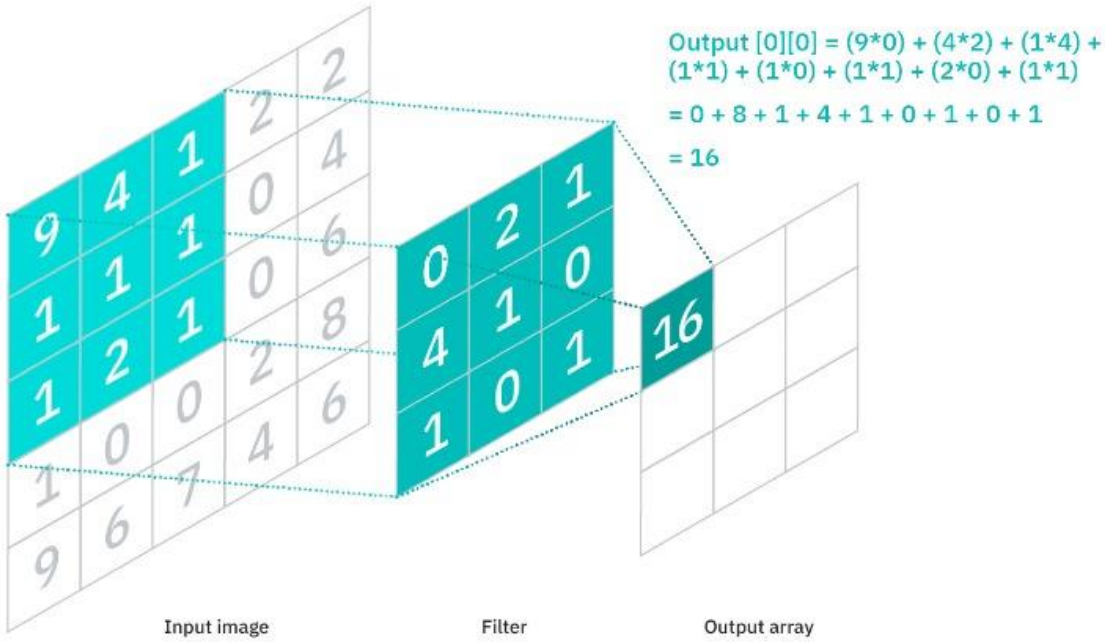


Figure 9 – Convolution product

I.1 Pre-processing & Data Generation:

I.1.1 pre-processing:

As I already mentioned, I've worked with videos Dataset , each one is 10 min video , and as we know Convolutional neural network has only the ability of process numeric Data , which means the necessity to work with frames captured from the videos .

Figure 10 - Captured frame from a Video of a drowsy participant

the frames captured from the videos may be considered as Noisy Images , the Model Will have multiple tasks if give him the complete image , by trying to extract different features , shoulders , backgrounds and even understand the Body Movements made by each participant , which is not good for Our Model performance , so it would be more beneficial to use a pre-trained Model , that can detects faces , then we can Separate Faces from the whole image and work on it directly .

Two proposed Model Would perfectly Fit the task :

– cnn_face_detection_model_v1 from DLIB

– MTCNN and FaceNet Model

cnn_face_detection_model_v1:

I worked on my project with both the **cnn__face_detection_model_v1** Model and **The MTCNN** model then I've compered them to choose the most accurate and robust one.

here is the cnn_face_detection in the image below :

```
import dlib
```

```
In [ ]: cnn_face_detector = dlib.cnn_face_detection_model_v1('/content/
        mmod_human_face_detector.dat')
```

Figure 11 - cnn_face_detection model

MTCNN :

in this part of the project , I used the Multi-task Cascaded Convolutional Networks (MTCNN) to detect faces in frames and then detect features from those Faces using the FaceNet model , which takes too long comparing to Dlib method , so the main purpose here is to compare the result (using whole face features) to the one we are going to see using the face landmarks .

```
In [ ]: # confirm mtcnn was installed correctly
        import mtcnn
        # print version
        print(mtcnn.__version__)
```

Figure 12 – MTCNN import

Here is how we declare the MTCNN model to detect faces from frames:

```
In [ ]: arr_1 = os.listdir('/content/gdrive/MyDrive/variables/Fold3_par
        t1/25')
        detector = MTCNN()
        # Generate data
        for path_3 in arr_1 :
          label_train  = np.array([])
          Frame_train = np.array([]).reshape(0,128)
          #).reshape(0,1).astype(np.int64)
          label = int(os.path.splitext(path_3)[0])
          full_path_2 = f"/content/gdrive/MyDrive/variables/Fold3_part
        1/25/{path_3}"
          #print(label)
          vs = cv2.VideoCapture(full_path_2)
          nb_frame = 0
          #detector = MTCNN()
```

Figure 13 - MTCNN declaration

After Declaring the **MTCNN** model as **detector**, we will need to localize the bounding box around the face, this model itself returns 4 values, 1 Point (x,y) , the Width and the height so we can draw a box around the face.

Here is the function that implement the **MTCNN** model:

```
from PIL import Image
from numpy import asarray
from mtcnn.mtcnn import MTCNN

# extract a single face from a given photograph
def extract_face(pixels, required_size=(160, 160)):
    # create the detector, using default weights
    # detect faces in the image
    results = detector.detect_faces(pixels)
    if len(results) == 0 :
      return np.array([]).reshape(0, 160, 160, 3)
    # extract the bounding box from the first face
    x1, y1, width, height = results[0]['box']
    # bug fix
    x1, y1 = abs(x1), abs(y1)
    x2, y2 = x1 + width, y1 + height
    # extract the face
    face = pixels[y1:y2, x1:x2]
    # resize pixels to the model size
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = asarray(image)
    return face_array

# load the photo and extract the face
```

Figure 14 - place

After Detecting the face and crop the frames to face-frames one , The face embedding task Comes to work , let's first define the face embedding Concept : it Is the process of converting a face image into numerical data , that data is then represented as a vector in a latent semantic space , by loading the **facenet** Model, the work will be done perfectly.

```
In [50]: model_facenet = tf.keras.models.load_model('./facenet_keras.h5')
         print('Loaded Model')

         Loaded Model
```

Figure 15 – facenet model

here is the function that implement the embedding task :

```
# get the face embedding for one face
def get_embedding(model_facenet , face_pixels):
    # scale pixel values
    face_pixels = face_pixels.astype('float32')
    # standardize pixel values across channels (global)
    mean, std = face_pixels.mean(), face_pixels.std()
    face_pixels = (face_pixels - mean) / std
    # transform face into one sample
    samples = expand_dims(face_pixels, axis=0)
    # make prediction to get embedding
    yhat = model_facenet.predict(samples)
    return yhat[0]
```

Figure 16 – face embedding method

After testing the Both methods on the RDD , I can say that the second method where we implement the FaceNet Model is more accurate and Robust , since we're getting one Dimension Data , it will be easy to train it and Reach a Good result Comparing to the first model ,Training Image Data would take more time and the model may find some difficulties to extract useful features from the face.

After Vectorizing all the face Frames , I chose to use a baseline model , a simple one since the data are only 1-D :

```
model = models.Sequential([
          layers.Dense(256,activation='relu'),
          layers.Dense(128,activation='relu'),
          layers.Dense(64,activation='relu'),
          layers.Dense(32,activation='relu'),
          layers.Dense(16,activation='relu'),
          layers.Dense(11,activation='softmax'),

])
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=opt,
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
history = model.fit(training_generator , validation_data= valid
ation_generator, epochs = 2 )
history
```

Figure 17 – baseline dense model

I.1.2 Data Generation:

there are multiple methods to Generate Data for the Model , but before doing this task , we need to transform our dataset from Videos to Data frames , which is not evident , since we are dealing with 120GB of Videos , mostly high quality , the deep learning frameworks propose some useful methods , generate frames from the videos , store them in some specific directories named by the given classes ,

```
tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    labels="inferred",
    label_mode="int",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    **kwargs
)
```

Figure 18 – image generator from directories

the Directory must be structured as the following example :

```
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
```

Figure 19 – Directory classes

we are talking here about more than 200GB of frames , Too much space , neither the local Computer nor the Free Cloud platform Could afford such Storage Space ?

Generators aren't the best solution !

We can clearly See that the most optimal method is to Do the Training task and Frames Generating .

Simultaneously , and the best way to do that is **Sequence ,** a TensorFlow API that enables the Generating Temporal Data train once on each sample per epoch which not the case with Generators .

```
from skimage.io import imread
from skimage.transform import resize
import numpy as np
import math

# Here, `x_set` is list of path to the images
# and `y_set` are the associated classes.

class CIFAR10Sequence(Sequence):

    def __init__(self, x_set, y_set, batch_size):
        self.x, self.y = x_set, y_set
        self.batch_size = batch_size

    def __len__(self):
        return math.ceil(len(self.x) / self.batch_size)

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) *
        self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) *
        self.batch_size]

        return np.array([
            resize(imread(file_name), (200, 200))
               for file_name in batch_x]), np.array(batch_y)
```

Figure 20 – customized class for data generation

every Sequence must implement the __getitem__ and the __len__ to get a complete batch and to calculate the number the number of batches in the Sequence , if we want to modify the Dataset between epochs we implement on_epoch_end .

```
#next step is to collect all the videos in one folder and take the first character of their name which is the label
import numpy as np
import keras
#path to video's folder
path = '/content/gdrive/MyDrive/variables/Fold2_part2'
#
train_ID = os.listdir(path)[1:]
class VideoFrameGenerator(data_utils.Sequence):
    'Generates data for Keras'
    def __init__(self,train_ID, batch_size=1, dim=(64,64), shuffle=True ):
        'Initialization'
        self.batch_size = batch_size
        self.train_ID = train_ID
        self.shuffle = shuffle
        self.dim = dim
        self.on_epoch_end()
    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.train_ID) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_temp = [self.train_ID[k] for k in indexes]
        #list_IDs_temp = [20]
        # Generate data
        X, y = self.__data_generation(list_IDs_temp)
        X =  np.array([X])[0]
        y =  np.array([y])
        y = y.reshape(y.shape[1])
        randomize = np.arange(y.shape[0])
        np.random.shuffle(randomize)
        X = X[randomize]
        y = y[randomize]
        return X , y
```

Figure 21 – Images data generator

Here is The Sequence Generator I made to Generate frames from the Videos :

Let's take a look on the Code :

the path Variable is the absolute path of the Videos in Google Drive , Since I've used Google Colab as a Work Notebook , Google Drive was the best option to store the Video dataset.

The __len__ function gives the number of time the Data will be generated, once __gititem__ is triggered , it triggered itself the __data_generation function where we pre-process and capture the frames from the Videos Dataset.

```
def __data_generation(self, list_IDs_temp):

    'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
    # Initialization
    #X = np.empty((self.batch_size, *self.dim, self.n_channels))
    #y = np.empty((self.batch_size), dtype=int)
    arr_1 = os.listdir(f'/content/gdrive/MyDrive/variables/Fold2_part2/{list_IDs_temp[0]}')
    path_1  = f'/content/gdrive/MyDrive/variables/Fold2_part2/{list_IDs_temp[0]}'
    isDirectory = os.path.isdir(path_1)
    if isDirectory :
      label_train  = []
      Frame_train = []
      # Generate data
      for path_3 in arr_1 :
        if path_3 != '.ipynb_checkpoints' :

          label = int(os.path.splitext(path_3)[0])
          full_path_2 = f"/content/gdrive/MyDrive/variables/Fold2_part2/{list_IDs_temp[0]}/{path_3}"
          vs = cv2.VideoCapture(full_path_2)
          nb_frame = 0
          while True :
            ret, image = vs.read()
            if ret == False  or nb_frame == 5000 :
                break
            #rects = detector(image,0)
            rects = detector(image, 0)
            for face in rects:
              x1 = face.left() # left point
              y1 = face.top() # top point
              x2 = face.right() # right point
              y2 = face.bottom() # bottom point
              img = image[y1:y2,x1:x2]
              if img.size != 0 :
                img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                img = cv2.resize(image, self.dim)
                #image = image.img_to_array(image)
                label_train.append(label)
                Frame_train.append(img)
                nb_frame += 1
          vs.release()
          cv2.destroyAllWindows()

    return Frame_train, label_train
```

Figure 22 – data generator method

as we can see , the __data_generation Generates video by video , it Captures frames , detect faces , then use the face Bounding Boxe to return only the face , resize the final Image then append it to a Frame_train Numpy array .

I.2 VGG16 Network:

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR (Imagenet) competition in 2014. It is regarded as one of the excellent vision model architectures till date. Most unique characteristic about VGG16 is that instead of having a large number of hyper-parameters it focuses on having convolution layers of 3x3 filter with a stride 1 and always used

the same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network, and it has about 138 million (approx) parameters. The architecture for VGG16 is shown in figure below.

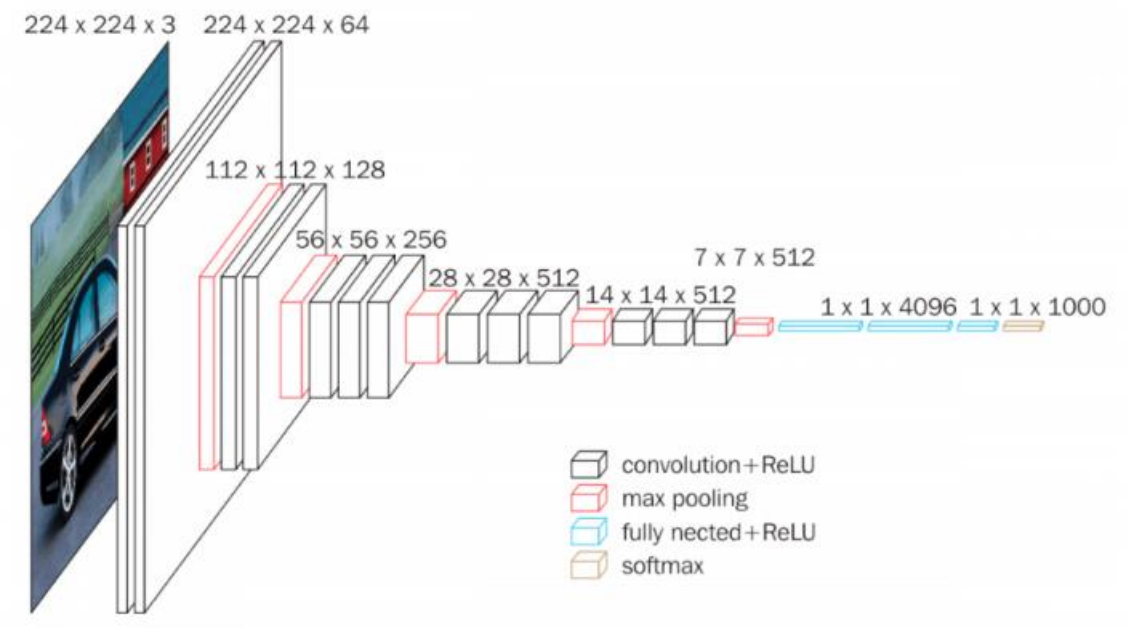


Figure 23 – VGG16 architecture

The input to conv1 layer is of fixed size 224*224 RGB image , the image is passed through a stack of convolutional (conv) layers , three Fully-Connected (Fc) layers follow a stack of convolutional layers (which had a different depth in different architectures) , the final layer is the SoftMax layer , The configuration of the fully connected layers is the same in all networks .

Transfer learning : Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks .

Usually in pre-trained models, core convolutional layers are trained on the larger dataset and their weights are fixed. After this, the final layers of the network (dense fully connected layers) are fine-tuned using our target dataset (smaller) .

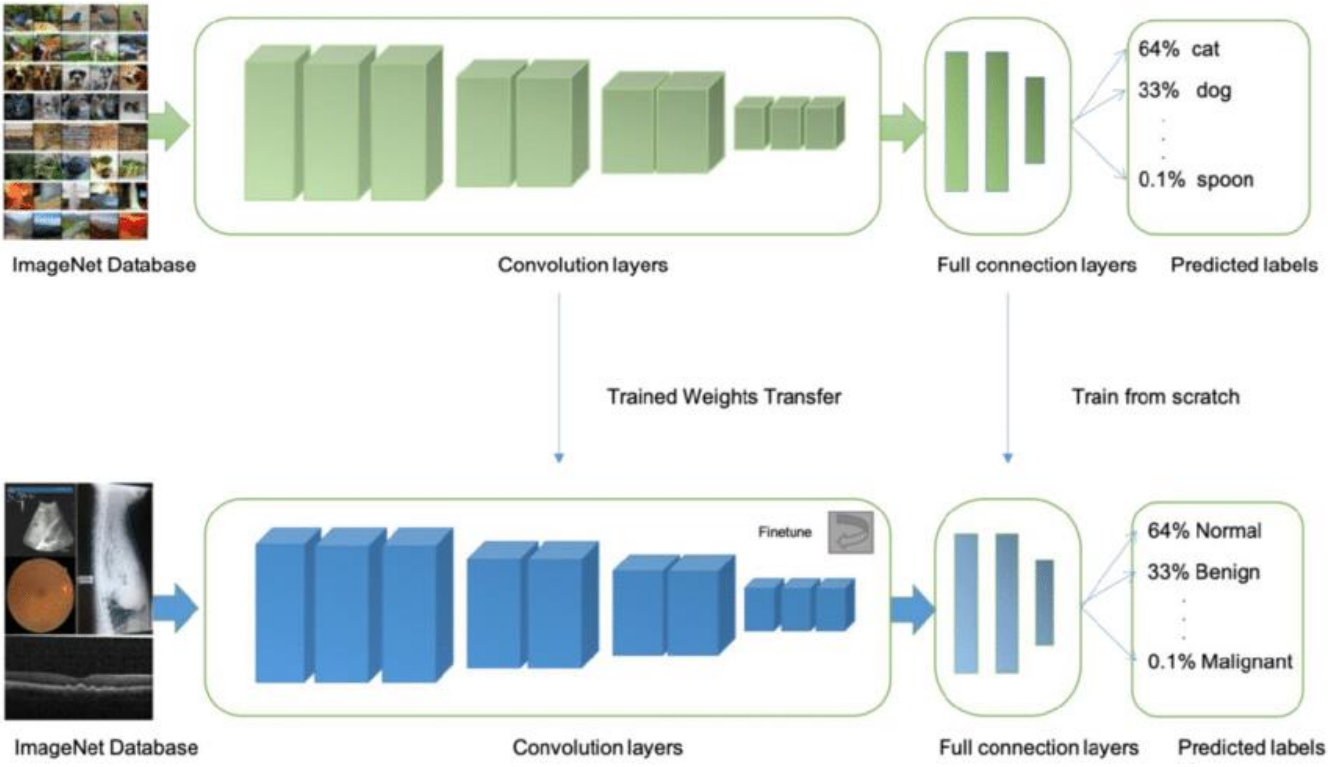


Figure 24 - Transfer Learning Example

In my Case I used the Pre-Trained VGG16 Model , even if it is an outdated Model , but I still has a good Performance and not the Heavy for the Memory , I Found out that it's beneficial to initial weights as the pre-trained weights on the ImageNet Dataset, since we're talking about the same task , classification of images in the Image Below we Can see some transfer Learning , some Modification on the Model , by changing the last layer by a Dense layer with SoftMax activation , So I can Predict if the participant is drowsy (10) , low vigilant(5) or Alert(0) .

```
In [30]: vgg16 = VGG16( include_top=False, input_shape = (64,64, 3) , weights = 'imagenet' )
```

```
In [31]: model = Sequential()

         for layer in vgg16.layers[:-1]: # this is where I changed your code
             model.add(layer)
         # Freeze the layers
         #for layer in model.layers:
         #    layer.trainable = False
```

```
In [32]: model.add(Flatten())
         model.add(Dense(11, activation='softmax'))
```

Figure 25 – transfer learning on VGG16

In this image , we can see the VGG16 Architecture with some modification in the last or the Head layer , by flattening the Layers , A Flatten layer in Keras reshapes the tensor to have a shape that is equal to the number of elements contained in the tensor , this is the same thing as making a 1d-array of elements.

```
In [33]: model.build((0,64,64,3))
         model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 11)                90123
=================================================================
Total params: 14,804,811
Trainable params: 14,804,811
Non-trainable params: 0
_________________________________________________________________
```

Figure 26 – the main model architecture

The image below shows the Compile Method that configures the model for training , it contains as parameters :

optimizer : String (name of optimizer) or optimizer instance , here in the notebook I chose the instance by implement the Adam Optimizer .

Loss : In Machine learning , **Loss** function is used to find error or deviation in the learning process , keras requires loss function during compilation process , here in this model I've used the **Sparse_categorical_crossentropy** for two reasons , the first is that I'm try to classify more than two classes , so there is no way I can use the **binary_crossentropy** and the second reason is that the data is not hot encoded .

```
In [34]: opt = tf.keras.optimizers.Adam(learning_rate=1e-5)
         model.compile(optimizer = opt, loss='sparse_categorical_crossen
         tropy', metrics=['accuracy'])
```

Figure 27 – Model optimizer

Metrics : in ML , **Metrics** is used to evaluate the performance of your model , it is similar to loss function but not used in training process , we use the **Accuracy** metric as the standard one .

in the image below , the **fit()** Method is taking place , The main purpose of this **fit()** function is used to evaluate your model on training .

```
In [ ]: history = model.fit(training_generator, validation_data=validat
        ion_generator, epochs = 10 )
```

Figure 28 – Model training

II. Sequential Model and KNN:

Sequence models are the machine learning models that input or output sequences of data. Sequential data includes text streams, audio clips, video clips, time-series data and etc. Recurrent Neural Networks (RNNs) and LSTM are the most popular algorithm used in sequence models .

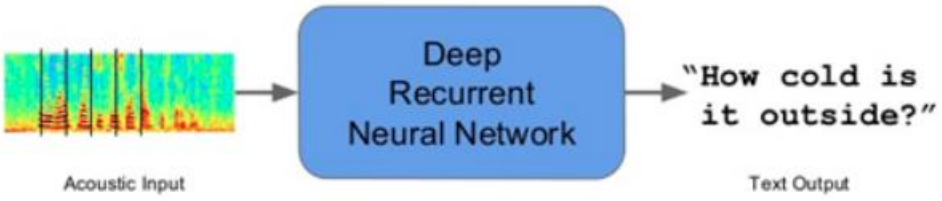


Figure 29 - deep sequential model

II.1 Features Extraction /pre-processing:

Facial feature extraction is the process of extracting face component features like eyes, nose, mouth, etc from human face image. Facial feature extraction is very much important for the initialization of processing techniques like face tracking, facial expression recognition or face recognition.

as far as we know The best way to analyse if the driver is drowsy is the blinks , In General yawning or mouth behavior isn't a good choice since the driver may speake or eat during driving

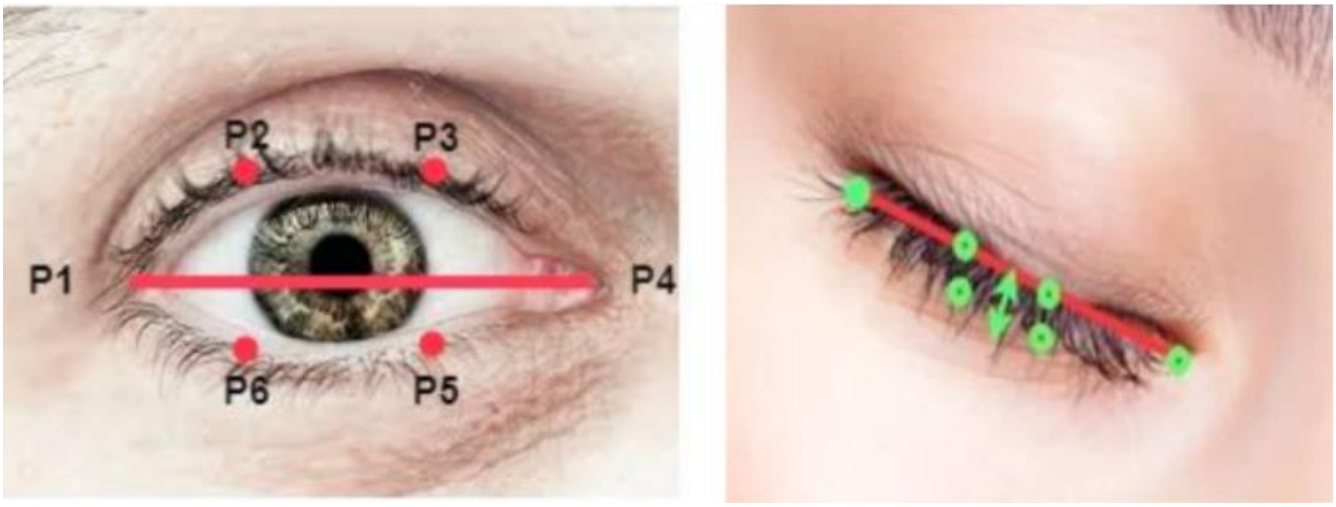


Figure 30 – eyes landmarks

The motivation behind using blink-related features such as blink duration, amplitude, and eye opening velocity, was to capture temporal patterns that appear naturally in human eyes and might be overlooked by spatial feature detectors like CNNs .

Open and closed eye with landmarks detected

The Question here is how the model will be able to distinguish between a drowsy driver and an alert driver , how can we calculate those features , Blink duration , amplitude and eye opening velocity.

In a real-world application of drowsiness detection, where a decision should be made every few minutes, the input could simply consist of the last few minutes of video. The output of the blink detection module is a sequence of blink events {blink 1 , ..., blink K }. Each blink is a four-dimensional vector containing four features describing the blink: duration, amplitude, eye opening velocity, and frequency. For each blink event blink, we defined the start of the blink , The bottom of the blink , it is the moment when the distance between the two Eyelids reach the minimum , and The end of the blink.

For each frame K , we denoted the Eye Aspect Ratio as :

$$EAR[k] = \frac{||\vec{p_2} - \vec{p_6}|| + ||\vec{p_3} - \vec{p_5}||}{||\vec{p_1} - \vec{p_4}||}$$

where the vector P is the 2D location of a facial landmark from the eye region .

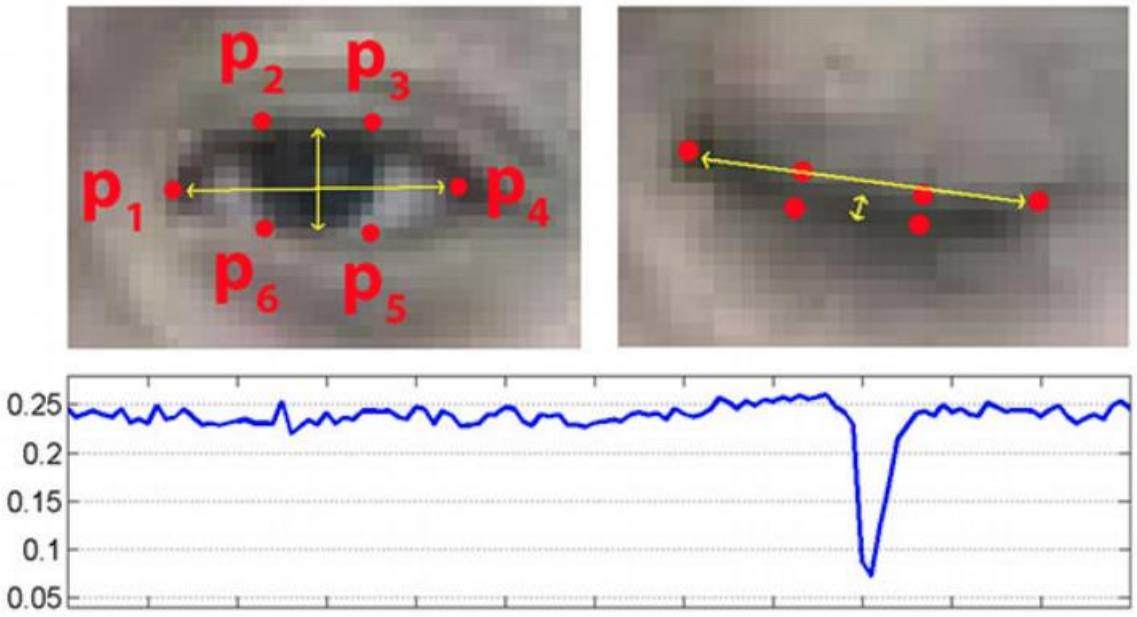


Figure 31 – eyes landmarks

Using the previous Notation , we define four main scale invariant features extracted from a certain Blink :

1- Duration : it is a temporal Variable that defines the duration of a blink :

$$Duration(blink(i)) = end(blink(i)) - start(blink(i)) + 1$$

2- Amplitude : this variable defines an important factor of fatigue , in reality when we feel tired , the process of opening the eyes after a Blink becomes slow , so the amplitude goes higher and higher

$$Amplitude(Blink(i)) = (EAR[start(blink(i))] - 2EAR[bottom(blink(i))] + EAR[end(blink(i))]) / 2$$

3 - Eye Opening Velocity : this defines the speed of the opening of the eyes :

$$Eye\ Opening\ Velocity = (EAR[end(blink(i))] - EAR[bottom(blink(i))]) / (end(blink(i)) - bottom(blink(i))$$

4 Frequencies : the Frequencies of blinks in a certain number of frames :

$$frequency = 100*(Number\ of\ blinks\ up\ to\ blink(i)) / (number\ of\ frames\ up\ to\ last\ frame)$$

as we can see in the image below, I defined an implementation of the EAR Function by calculating the Euclidean distance between the eye landmarks :

```
def eye_aspect_ratio(eye):
        # compute the euclidean distances between the two sets of
        # vertical eye landmarks (x, y)-coordinates
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])
        # compute the euclidean distance between the horizontal
        # eye landmark (x, y)-coordinates
        C = dist.euclidean(eye[0], eye[3])
        # compute the eye aspect ratio
        ear = (A + B) / (2.0 * C)
        # return the eye aspect ratio
        return ear
```

Figure 32 – eye aspect ratio method

here is how I extracted the Eyes landmarks , it's by using the most common face detection library.

```
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

Figure 33 – eyes landmarks indexes

As we can see , the figure ... and both show two steps , detecting the Face using the DLIB Library , then localize the eyes landmarks and finally calculate EAR (eye-aspect-ratio) .

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(datFile)
```

Figure 34 – detector and predictor

```
shape = predictor(gray, rect)
shape = face_utils.shape_to_np(shape)
# extract the left and right eye coordinates, then use the
# coordinates to compute the eye aspect ratio for both eyes
leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
leftEAR = eye_aspect_ratio(leftEye)
rightEAR = eye_aspect_ratio(rightEye)
```

Figure 35 – calculating the EAR

after calculating the Eye-Aspect-Ratio , we need to Find the minimum peaks so we can Determine the Eyes Blinks since we're just talking about a minimal distance between the Eyelids .

After going through multiple functions and libraries , I found an elegant and simple function , it is called :

scipy.signal.argrelextrema() . It Finds all the local peaks in a list or a NumPy array and put them in the same order in a python list

```
In [ ]: from scipy.signal import argrelextrema

        index_peak = argrelextrema(ear, np.less)
        index_peak = np.asarray(index_peak)[0]
        index_peak

Out[ ]: (array([    2,     8,    13, ..., 18218, 18221, 18226]),)
```

Figure 36 – The minimum peaks

After Finding all the Needed peaks that define the driver Blinks , I will Need To find the Beginning and the End of each Blink , here is the two functions that perfectly Do the Work :

```
def start_check(blink_index_list, ear) :
  start = np.array([] , dtype=np.int64)
  Bool = True
  for i in blink_index_list :
    Bool = True
    j = i -1
    while Bool and  j > 0 :
      if ear[j] <= ear[j-1]  :
        j -= 1
      else :
        start = np.append(start,j)
        Bool = False
    #this condition is to make sure that we can save the end value even if it is that last value , since in the while condit
ion
    #we're going to stop before the last value to avoid the index out of bounds error
    if j == 0 :
      start  = np.append(start , j)
  return start
```

Figure 37 – start of blink method

```
def end_check(blink_index_list, ear) :

  end = np.array([] , dtype=np.int64)
  Bool = True
  for i in blink_index_list :
    Bool = True
    j = i+1

    while Bool and j< len(ear)-1 :
      if ear[j] <= ear[j+1] :

        j += 1
      else :
        end  = np.append(end , j)
        Bool = False
    #this condition is to make sure that we can save the end value even if it is that last value , since in the while condit
ion
    #we're going to stop before the last value to avoid the index out of bounds error
    if j == len(ear)-1 :
      end  = np.append(end , j)

  return end
```

Figure 38 – end of blink method

II.2 LSTM Model :

Long Short-Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) (and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn! All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

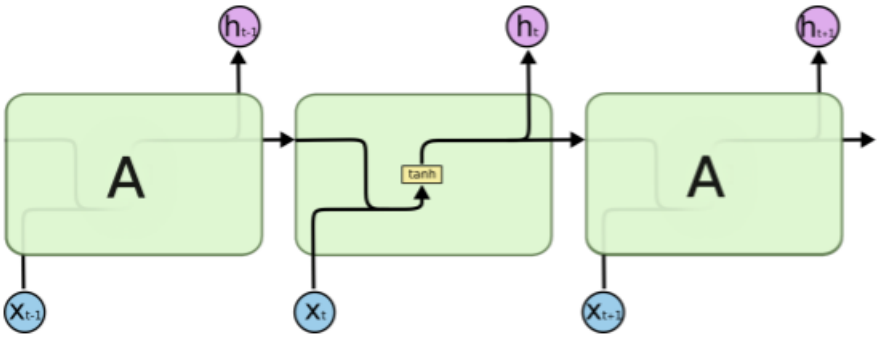


Figure 39 - The repeating module in a standard RNN contains a single layer .

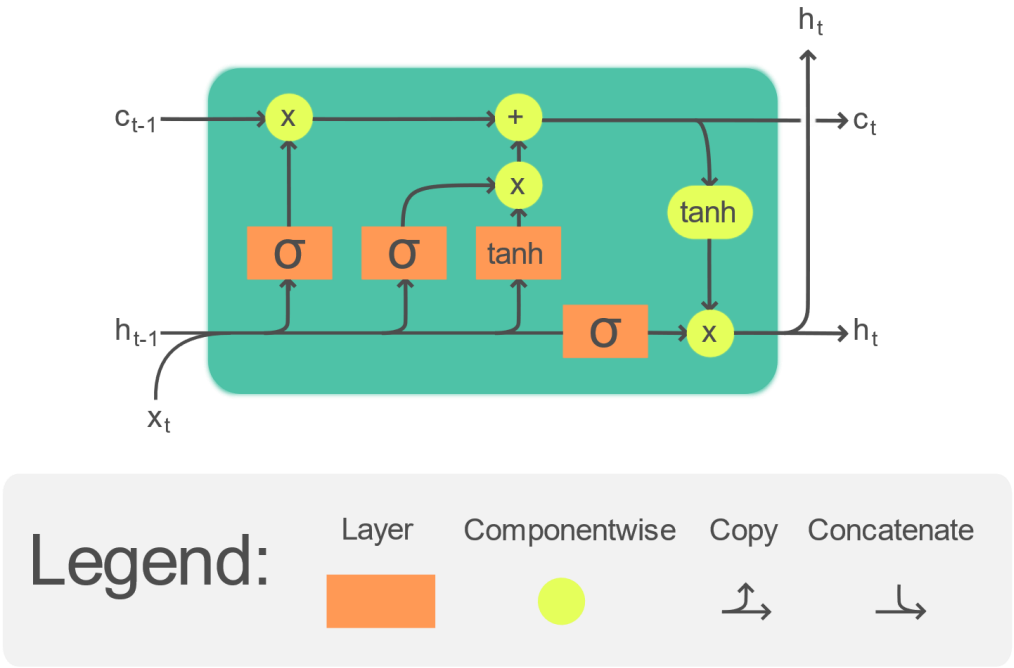


Figure 40 – LSTM layer

The module in an LSTM Contains four interacting layers:

The compact forms of the equations for the forward pass of an LSTM unit with a forget gate are:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Variables :

X(t) : Input vector to the LSTM unit .

F(t) : forget gate's activation vector .

i(t) : input/update gate's activation vector

o(t) : output gate's activation vector

h(t) : hidden state vector also known as output vector of the LSTM unit .

Č(t) : cell input activation vector .

C(t) : cell state vector .

W , b : weight matrices and bias vector parameters which need to be learned during training

Activation functions :

σ_g : sigmoid function

σ_c : hyperbolic tangent function .

σ_h : hyperbolic tangent function .

```
In [6]: Blinks_train = np.array([]).reshape(0,30,4)
        label_train = np.array([]).reshape(0,1)
        for i in range(1,6) :
            arr = np.load(f'./Early-Drowsiness-Detection/Blinks_30_Fold{i}.npy')
            arr_2 = np.load(f"./Early-Drowsiness-Detection/Labels_30_Fold{i}.npy")

            Blinks_train = np.vstack([Blinks_train,arr])
            label_train = np.append(label_train,arr_2)

In [7]: Test_Blinks_train = np.array([]).reshape(0,30,4)
        Test_label_train = np.array([]).reshape(0,1)
        for i in range(1,6) :
            arr = np.load(f'./Early-Drowsiness-Detection/BlinksTest_30_Fold{i}.npy')
            arr_2 = np.load(f"./Early-Drowsiness-Detection/LabelsTest_30_Fold{i}.np
        y")

            Test_Blinks_train = np.vstack([Blinks_train,arr])
            Test_label_train = np.append(label_train,arr_2)

In [8]: Blinks_train.shape

Out[8]: (34592, 30, 4)
```

Figure 41 – regrouping data

As known , The LSTM or the RNN model are often used in the NLP field , because of their capability to understand the Context and extract some useful features from the Context itself. but how can we do that , it's by embedding or Vectorizing the Corpus or the Text data to Vectors and feed it to the model sequence Vectors , here in my Project , I made I feed my LSTM Model 30 Vectors by 30.

In The image below , I defined an LSTM Model , Bidirectional One:

```
data_dim = 4
timesteps = 30
num_classes = 11

# expected input data shape: (batch_size, timesteps, data_dim)
model = Sequential()
model.add(Bidirectional(LSTM(64, return_sequences=True,
                  input_shape=(timesteps, data_dim)))) # returns a sequence of v
ectors of dimension 32
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(32, return_sequences=True)))  # returns a sequen
ce of vectors of dimension 32
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(16)))  # return a single vector of dimension 32
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Figure 42 – main model architecture

Here is the training Step with 20 Epochs :

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accurac
y',
                                            patience = 2,
                                            verbose=1,
                                            factor=0.1,
                                            min_lr=0.000001)
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

```
model.fit(Blinks_train[:34500], label_train[:34500],
          batch_size=64, epochs=20,
          validation_data=(Test_Blinks_train, Test_label_train))
```

Figure 43 – model training

in the image below , the model iterates over The whole dataset 20 Consecutive times :

```
Epoch 1/20
540/540 [==============================] - 71s 108ms/step - loss: 0.9800 - accuracy: 0.5730 - val_loss: 0.5258 - val_accurac
y: 0.7498
Epoch 2/20
540/540 [==============================] - 55s 102ms/step - loss: 0.4943 - accuracy: 0.7789 - val_loss: 0.3612 - val_accurac
y: 0.8403
Epoch 3/20
540/540 [==============================] - 55s 102ms/step - loss: 0.3571 - accuracy: 0.8404 - val_loss: 0.2847 - val_accurac
y: 0.8780
Epoch 4/20
540/540 [==============================] - 55s 102ms/step - loss: 0.2821 - accuracy: 0.8768 - val_loss: 0.1828 - val_accurac
y: 0.9214
Epoch 5/20
540/540 [==============================] - 55s 102ms/step - loss: 0.2180 - accuracy: 0.9126 - val_loss: 0.1637 - val_accurac
y: 0.9288
Epoch 6/20
540/540 [==============================] - 54s 101ms/step - loss: 0.1776 - accuracy: 0.9291 - val_loss: 0.1566 - val_accurac
y: 0.9399
Epoch 7/20
540/540 [==============================] - 55s 101ms/step - loss: 0.1421 - accuracy: 0.9474 - val_loss: 0.1023 - val_accurac
y: 0.9592
Epoch 8/20
540/540 [==============================] - 55s 102ms/step - loss: 0.1149 - accuracy: 0.9573 - val_loss: 0.0851 - val_accurac
y: 0.9675
Epoch 9/20
540/540 [==============================] - 55s 101ms/step - loss: 0.0947 - accuracy: 0.9665 - val_loss: 0.0779 - val_accurac
y: 0.9722
Epoch 10/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0979 - accuracy: 0.9643 - val_loss: 0.0578 - val_accurac
y: 0.9771
Epoch 11/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0812 - accuracy: 0.9699 - val_loss: 0.0892 - val_accurac
y: 0.9678
Epoch 12/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0756 - accuracy: 0.9735 - val_loss: 0.0287 - val_accurac
y: 0.9877
Epoch 13/20
540/540 [==============================] - 57s 105ms/step - loss: 0.0692 - accuracy: 0.9752 - val_loss: 0.0303 - val_accurac
y: 0.9898
Epoch 14/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0440 - accuracy: 0.9844 - val_loss: 0.0252 - val_accurac
y: 0.9908
Epoch 15/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0419 - accuracy: 0.9844 - val_loss: 0.0198 - val_accurac
y: 0.9910
Epoch 16/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0424 - accuracy: 0.9852 - val_loss: 0.0277 - val_accurac
y: 0.9910
Epoch 17/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0405 - accuracy: 0.9862 - val_loss: 0.0165 - val_accurac
y: 0.9940
Epoch 18/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0232 - accuracy: 0.9917 - val_loss: 0.0127 - val_accurac
y: 0.9951
Epoch 19/20
540/540 [==============================] - 55s 101ms/step - loss: 0.0324 - accuracy: 0.9889 - val_loss: 0.0345 - val_accurac
y: 0.9894
Epoch 20/20
540/540 [==============================] - 55s 102ms/step - loss: 0.0311 - accuracy: 0.9893 - val_loss: 0.0372 - val_accurac
y: 0.9852
<tensorflow.python.keras.callbacks.History at 0x7f189a634b50>
```

Figure 44 – training result

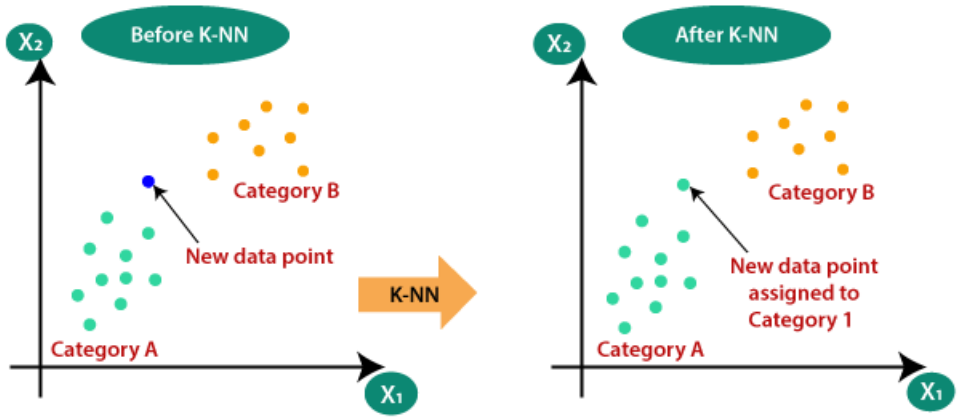
The Final Step is the Evaluation of the model , using the evaluate() method by Keras Model Training APIs.

```
model.evaluate(Blinks_train[34500:], label_train[34500:])
```

```
3/3 [==============================] - 0s 16ms/step - loss: 0.1197 - accuracy:
0.9674
[0.11972814798355103, 0.967391312122345]
```

Figure 45 – model evaluation

II.3 KNN Model :



I used Scikit Learn , The machine learning Library to Define a KNN Model , and then I chose 3 as an Hyper-parameter for Number of neighbors .

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
        neigh = KNeighborsClassifier(n_neighbors=3)
        neigh.fit(train_dataset, y_Data)

Out[ ]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                             weights='uniform')
```

Figure 46 – KNN model

Chapter 5: Results

LSTM Model Result:

After Extracting the face Features and Calculate all the needed Variables , Feed it to LSTM Model here is the Result :

```
In [12]: model.evaluate(Blinks_train[34500:], label_train[34500:])

         3/3 [==============================] - 0s 16ms/step - loss: 0.1197 - accuracy: 0.9
         674
Out[12]: [0.11972814798355103, 0.967391312122345]
```

Figure 47 – LSTM evaluation

KNN Model Result :

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
        neigh = KNeighborsClassifier(n_neighbors=3)
        neigh.fit(train_dataset, y_Data)

Out[ ]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                             weights='uniform')

In [ ]: neigh.score(X_test_set,y_test_set)

Out[ ]: 0.9996299820276985

In [ ]: neigh.score(validation_dataset[0], validation_dataset[1])

Out[ ]: 0.9982328733962721
```

Figure 48 – KNN Score

Dense Model Result:

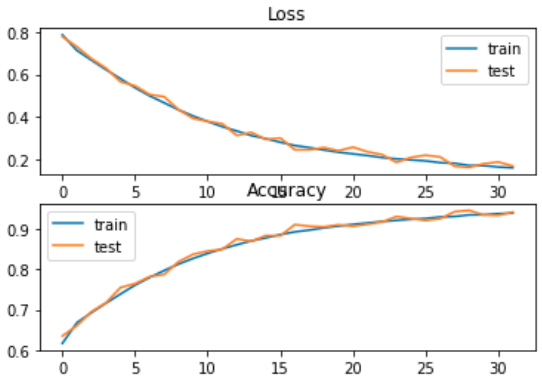
```
In [ ]: print("Evaluate on test data")
        results = model.evaluate(x = X_test_set , y = y_test_set )
        print("test loss, test acc:", results)
        print("Accuracy of the model is - " , results[1]*100 , "%")
```

```
Evaluate on test data
2956/2956 [==============================] - 11s 3ms/step - loss: 0.1695 - accurac
y: 0.9303
test loss, test acc: [0.16946080327033997, 0.9302991628646851]
Accuracy of the model is -  93.0299162864685 %
```

Figure 49 -

```
In [ ]: from matplotlib import pyplot

        # plot loss during training
        pyplot.subplot(211)
        pyplot.title('Loss')
        pyplot.plot(history.history['loss'], label='train')
        pyplot.plot(history.history['val_loss'], label='test')
        pyplot.legend()
        # plot accuracy during training
        pyplot.subplot(212)
        pyplot.title('Accuracy')
        pyplot.plot(history.history['accuracy'], label='train')
        pyplot.plot(history.history['val_accuracy'], label='test')
        pyplot.legend()
        pyplot.show()
```



Conclusion

One of the main issues within the framework of video-based drowsiness detection is how to choose the efficient feature extraction technique , in this work , we detect Drowsy state by The most accurate facial Feature (EAR) , Moreover , unlike other methods that adopt predefined thresholds to decide the state of drowsiness , we made a Deep learning Model that could understand The Context and the face state of the driver , based on it we can decide whether the driver is drowsy or not , The proposed method has been validated on a very challenging Video Dataset (RLDD) , which simulates real driving conditions l , night driving, wearing glasses ... , the preliminary results are very promising , these proposed methods had some limitations, especially when the driver has sunglasses , In Future Works , efforts should be focused on how to overcome these limitations , overcome all the external disturbances .

Annex

Real Time Drowsiness Detector Code:

183 lines (158 sloc) | 5.69 KB

```
from scipy.spatial import distance as dist
from imutils.video import FileVideoStream
from scipy.signal import argrelextrema
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import tensorflow as tf
from tensorflow import keras
import numpy as np
import argparse
import imutils
import joblib
import time
import dlib
import cv2
import os

EYE_AR_THRESH = 0.3
EYE_AR_CONSEC_FRAMES = 3
train_blink_Final_1 = np.array([]).reshape(0,4)
labels_blink_Final_1 =  np.array([]).reshape(0,1)
# initialize the frame counters and the total number of blinks
COUNTER = 0
TOTAL = 0
EAR = []
i = 0
q = 0
model =joblib.load('/home/oussa/Desktop/finalized_model.sav')
def eye_aspect_ratio(eye):
        # compute the euclidean distances between the two sets of
        # vertical eye landmarks (x, y)-coordinates
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])
        # compute the euclidean distance between the horizontal
        # eye landmark (x, y)-coordinates
        C = dist.euclidean(eye[0], eye[3])
        # compute the eye aspect ratio
        ear = (A + B) / (2.0 * C)
        # return the eye aspect ratio
        return ear

```

```
42
43  def start_check(blink_index_list, ear) :
44    start = np.array([] , dtype=np.int64)
45    Bool = True
46    for i in blink_index_list :
47      Bool = True
48      j = i -1
49      while Bool and  j > 0 :
50        if ear[j] <= ear[j-1]  :
51          j -= 1
52        else :
53          start = np.append(start,j)
54          Bool = False
55      #this condition is to make sure that we can save the end value even if it is that last value , since in the while condition
56      #we're going to stop before the last value to avoid the index out of bounds error
57      if j == 0 :
58        start  = np.append(start , j)
59    return start
60
61  def end_check(blink_index_list, ear) :
62
63    end = np.array([] , dtype=np.int64)
64    Bool = True
65    for i in blink_index_list :
66      Bool = True
67      j = i+1
68
69      while Bool and j< len(ear)-1 :
70        if ear[j] <= ear[j+1] :
71
72          j += 1
73        else :
74          end  = np.append(end , j)
75          Bool = False
76      #this condition is to make sure that we can save the end value even if it is that last value , since in the while condition
77      #we're going to stop before the last value to avoid the index out of bounds error
78      if j == len(ear)-1 :
79        end  = np.append(end , j)
80
```

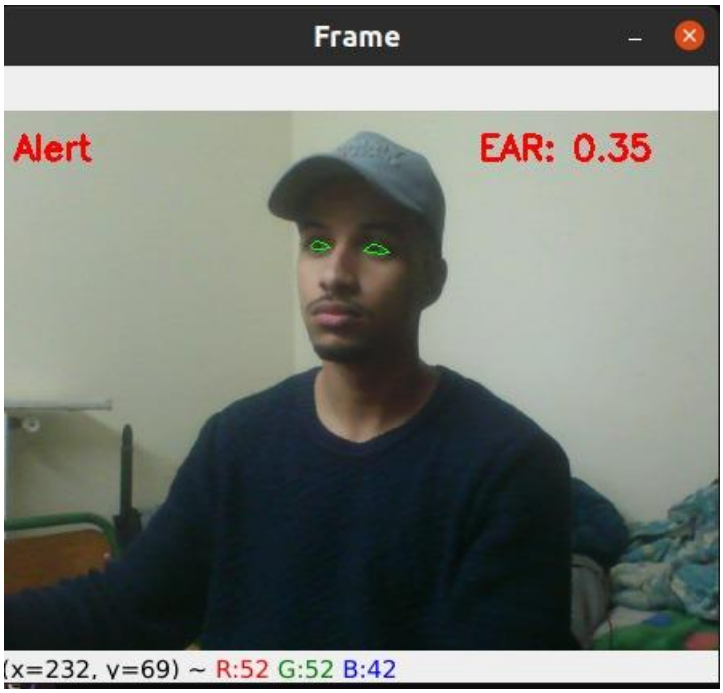
```
82
83  datFile =  "/home/oussa/Desktop/shape_predictor_68_face_landmarks.dat"
84  detector = dlib.get_frontal_face_detector()
85  predictor = dlib.shape_predictor(datFile)
86  (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
87  (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
88
89  vs = cv2.VideoCapture(-1)
90  while True :
91      ret, frame = vs.read()
92
93      if ret == False:
94          break
95      frame = imutils.resize(frame, width=450)
96
97
98   #   cv2_imshow(frame)
99
100     #frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
101     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
102     # detect faces in the grayscale frame
103     rects = detector(gray, 0)
104     for rect in rects:
105         COUNTER += 1
106         shape = predictor(gray, rect)
107         shape = face_utils.shape_to_np(shape)
108         leftEye = shape[lStart:lEnd]
109         rightEye = shape[rStart:rEnd]
110         leftEAR = eye_aspect_ratio(leftEye)
111         rightEAR = eye_aspect_ratio(rightEye)
112         EAR.append((leftEAR + rightEAR) / 2.0 )
113         leftEyeHull = cv2.convexHull(leftEye)
114         rightEyeHull = cv2.convexHull(rightEye)
115         cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
116         cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
117         cv2.putText(frame, "EAR: {:.2f}".format((leftEAR + rightEAR) / 2.0 ), (300, 30),
118             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

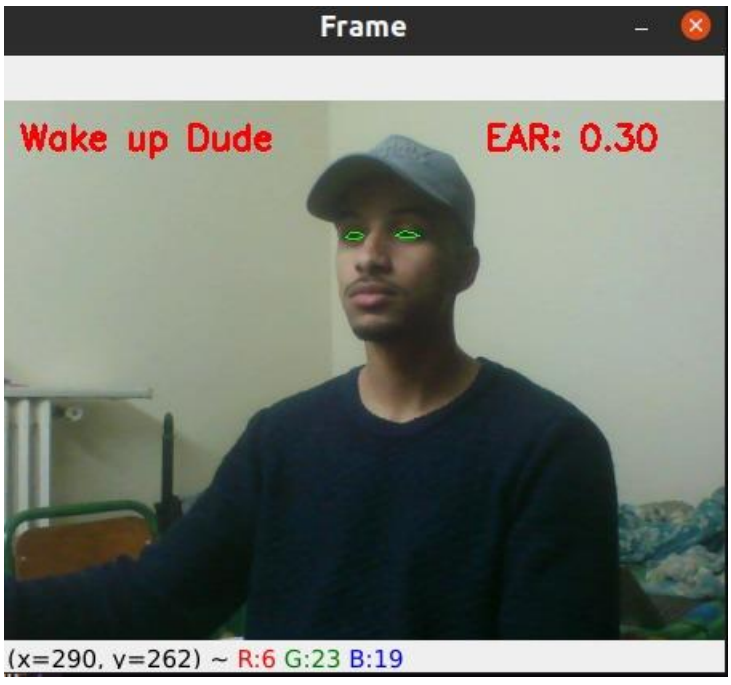
```
    if COUNTER == 100 :
        ear = np.array(EAR)
        ear.reshape(ear.shape[0])
        index_peak = argrelextrema(ear, np.less)
        index_peak_blink = np.array([])
        for i in index_peak[0] :
            if EAR[i] < 0.3 :
                index_peak_blink = np.append(index_peak_blink , i)
        index_peak_blink = index_peak_blink.astype(np.int64)
        start = start_check(index_peak_blink,ear)
        #end of blinks
        end = end_check(index_peak_blink,ear )
        #duration of blinks
        Duration = end - start + 1

        Am = ear[end] - 2*ear[index_peak_blink] + ear[start]
        #amplitude of blinks
        Ampl = Am / (end - start)
        #velocity of blinks
        Eye_Open_Speed = (ear[end] - ear[index_peak_blink])/(end - index_peak_blink )
        Frames_nmbr = np.arange(1 , len(ear)+1)
        #frequency of blinks
        Freq = 100*(np.arange(1 , len(index_peak_blink)+1)/Frames_nmbr[end])
        feature_blink = np.column_stack((Duration, Ampl,Eye_Open_Speed,Freq))
        result_array = model.predict(feature_blink)
        drowsy_array = []
        low_vigilant = []
        for i in range(len(result_array)) :
            max = result_array[i]
            if max == 10 :
                drowsy_array.append(max)
            elif max == 5 :
                low_vigilant.append(max)
```

```
153
154         if len(drowsy_array) > 10 :
155
156
157             q = 1
158         elif len(low_vigilant) > 10 :
159
160
161               q = 1
162         else :
163
164
165               q = 0
166         COUNTER = 0
167         EAR = []
168    #  cv2_imshow(frame)
169      if q == 1 :
170        cv2.putText(frame, "Wake up Dude", (10, 30),
171                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
172        cv2.imshow("Frame", frame)
173      else :
174         cv2.putText(frame, "Alert", (10, 30),
175                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
176         cv2.imshow("Frame", frame)
177      key = cv2.waitKey(1) & 0xFF
178      if key == ord("q"):
179              break
180  cv2.release()
181  cv2.destroyAllWindows()
182
183
```

Test :


Frame
Alert
EAR: 0.35
(x=232, y=69) ~ R:52 G:52 B:42


Frame
Wake up Dude
EAR: 0.30
(x=290, y=262) ~ R:6 G:23 B:19

References:

[1] M. R. Endsley, “Toward a theory of situation awareness in dynamic systems,” Hum. Factors, J. Hum. Factors Ergonom. Soc., vol. 37, no. 1, pp. 32–64, 1995.

[2] Y. Xing, C. Lv, D. Cao, H. Wang, and Y. Zhao, “Driver workload estimation using a novel hybrid method of error reduction ratio causality and support vector machine,” Measurement, vol. 114, pp. 390–397, Jan. 2018.

[3] Y. Xing et al., “Identification and analysis of driver postures for in vehicle driving activities and secondary tasks recognition,” IEEE Trans. Comput. Social Syst., vol. 5, no. 1, pp. 95–108, Mar. 2018.

[4] Y. Zhao et al., “An orientation sensor-based head tracking system for driver behaviour monitoring,” Sensors, vol. 17, no. 11, p. 2692, 2017.

[5] C. M. Martinez, M. Heucke, F.-Y. Wang, B. Gao, and D. Cao, “Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey,” IEEE Trans. Intell. Transp. Syst., vol. 19, no. 13, pp. 666–676, Mar. 2016.

[6] Y. Dong, Z. Hu, K. Uchimura, and N. Murayama, “Driver inattention monitoring system for intelligent vehicles: A review,” IEEE Trans. Intell. Transp. Syst., vol. 12, no. 2, pp. 596–614, Jun. 2011.

[7] S. Kaplan, M. A. Guvensan, A. G. Yavuz, and Y. Karalurt, “Driver behavior analysis for safe driving: A survey,” IEEE Trans. Intell. Transp. Syst., vol. 16, no. 6, pp. 3017–3032, Dec. 2015.

[8] V. Saini and R. Saini, “Driver drowsiness detection system and techniques: A review,” Int. J. Comput. Sci. Inf. Technol., vol. 5, no. 3, pp. 4245–4249, 2014. Page | 23

[9] A. Sahayadhas, K. Sundaraj, and M. Murugappan, “Detecting driver drowsiness based on sensors: A review,” Sensors, vol. 12, no. 12, pp. 16937–16953, 2012.

[10] Q. Wang, J. Yang, M. Ren, and Y. Zheng, “Driver fatigue detection: A survey,” in Proc. IEEE 6th World Congr. Intell. Control Autom. (WCICA), Dalian, China, vol. 2, Jun. 2006, pp. 8587–8591.

[11] (2011). Road Safety in Canada. Accessed: Mar. 24, 2017. [Online]. Available: https://www.tc.gc.ca/

[12] K. Azam, A. Shakoor, R. A. Shah, A. Khan, S. A. Shah, and M. S. Khalil, “Comparison of fatigue related road traffic crashes on the national highways and motorways in Pakistan,” J. Eng. Appl. Sci., vol. 33, no. 2, pp. 47–54, 2014.

[13] Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117. https://doi.org/10.1016/j.neu net.2014.09.003

[14] Silberman N, Guadarrama S (2016) TensorFlow-Slim image classification model library.

https://github.com/tensorflow/mo dels/tree/master/research/slim

[15] Simonyan K, Zisserman A (2014) Two-stream convolutional networks for action recognition in videos. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds)

[16] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC (2018) MobileNetV2: inverted residuals and linear bottlenecks. 2018 IEEE conference on computer vision and pattern recognition (CVPR). IEEE, Salt Lake City, UT, pp 4510–4520

[17] Advances in neural information processing systems 27. Curran Associates Inc, Montreal, pp 568–576. http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.pdf Soomro K, Zamir AR, Shah M (2012) UCF101: a dataset of 101 human actions classes from videos in the wild. Tech. Rep. CRCV-TR-12-01, Center for Research in Computer Vision, University of Central Florida, Orlando, FL

[18] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR), IEEE, Boston, MA,m pp 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[19] Carreira J, Zisserman A (2017) Quo vadis, action recognition? A new model and the Kinetics dataset. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR), IEEE, Honolulu, HI, pp 4724–4733. https://doi.org/10.1109/CVPR.2017.502

[20] Wijnands, J.S., Thompson, J., Nice, K.A. et al. Real-time monitoring of driver drowsiness on mobile platforms using 3D neural networks. Neural Comput & Applic 32, 9731–9743 (2020). https://doi.org/10.1007/s00521-019-04506-0

[21] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A Page | 24large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

[22] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.155