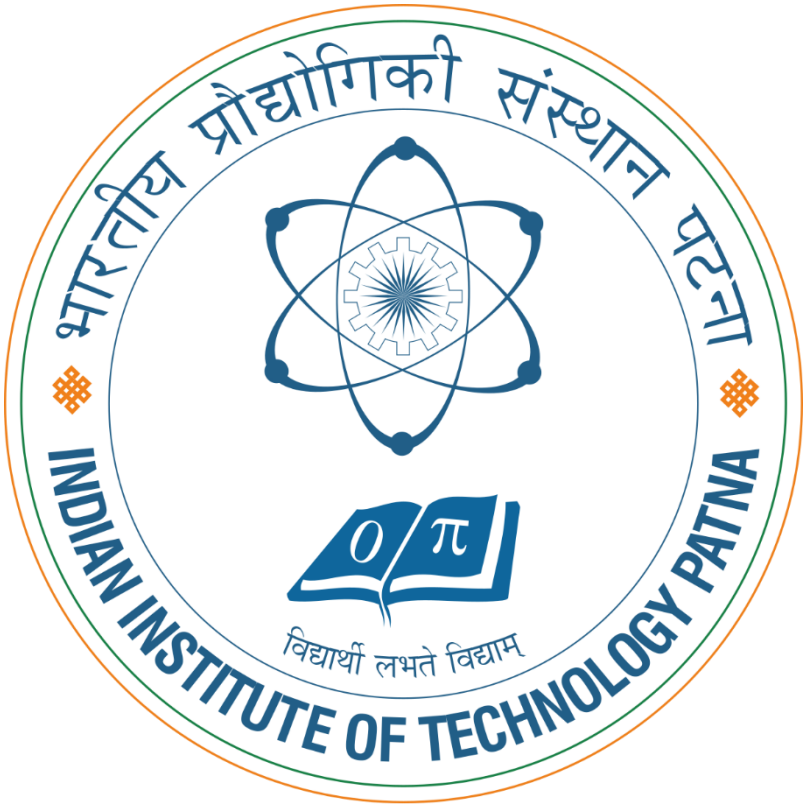
REPORT ON

DRIVER DROWSINESS DETECTION SYSTEM USING DEEP LEARNING



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PATNA

SUBMITTED BY:

NAMES	ROLL NUMBER
AVIJIT DATTA	2511AI33
ASHWANI KUMAR	2511CS18
JADHAV PRAFUL NAYAK	2511CS03
BISWARAJ BHATTACHARYYA	2511AI43
JOSIGA	2511AI10
SHREYA SINGHAL	2511AI21
HIMANSHU MEHRA	2511CS19
AVNI VERMA	2511AI08
ARYAN	2511AI29
KEDHARNATH REDDY	2511AI36

INDEX

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

It is without dispute that sleep deprivation increases the risk of workplace accidents. A heightened risk of job injuries can also result from long workdays and inadequate sleep. According to police personnel monitoring the motorways and major roads here, sleep-deprived drivers continue to account for 40% of traffic accidents. When examining the causes of accidents on the Agra-Lucknow Expressway, researchers discovered that 40% of the events were caused by drivers falling asleep behind the wheel. Nitin Gadkari, minister of roads and highways, just announced the most recent statistics, which show that 1,47,913 people died in traffic accidents in 2022. Uttar Pradesh reported the highest number of fatalities, with 20,124, followed by Tamil Nadu with 16,157.

Driver drowsiness is primarily identified with the help of three different types of measurements: a) physiological (using EEG, ECG, EOG signals), b) vehicle-based (concentrating on lane departures, steering wheel movement, and pressure on the accelerator pedal), c) behavioral (including yawning, eye closure, head pose, etc.).

A cross-platform package called OpenCV allows us to create real-time computer vision applications. The primary areas of focus are image processing, video capture, and analysis, which includes tools for object and face detection. In this project, OpenCV Haar Cascade Classifier is used for video capturing. There are numerous image-processing algorithms for extracting features in this regard. A variety of architectures enable the recognition of visual patterns directly from pixel pictures without much to no preprocessing. For instance, AlexNet, MobileNetV2, ResNet, VGGNet, and YOLO have varied numbers of convolutional layers, pooling layers, and fully connected layers. With the help of trained ResNet50, VGG16, and MobileNetV2 features, facial movements like blinking, yawning, and swaying can be detected with 78% accuracy.

Recurrent neural networks (RNN), a different class of neural network, utilize knowledge from the past to carry out current tasks. Although video sequence detection and image captioning

can also be taken into consideration, language modelling is where they are most frequently applied.

PROBLEM STATEMENT

Road accidents cause a great number of significant injuries in addition to casualties each year. Lack of sleep is one of many causes of accidents. The analysis on tired drivers has highlighted the need to inform highway drivers about the need of taking regular breaks and getting enough sleep for safety in a nation where road accidents take roughly three lives every minute. Hence, a drowsiness detection model is very crucial and high in demand.

The goal of this initiative is to reduce the number of accidents on the road that are brought on by tiredness. The driver's photos are taken by the drowsiness detector, which alerts them if it detects any signs of impairment. As a result, there would be fewer traffic accidents, making the world a safer place to live.

1.2 FACIAL FEATURES

To gather the data required for driver sleepiness detection, facial features are essential. The following prominent facial characteristics are frequently employed in driver sleepiness detection systems:

(a)EYE CLOSURE: Monitoring the state of the driver's eyes is one of the primary indicators of drowsiness. Features such as eye closure duration, blink rate, and eye openness can be extracted. For example, tracking the percentage of eye closure or the duration of eye closure over time can provide insights into the driver's drowsiness level.

(b)EYE LANDMARKS: Identifying specific points on the driver's eyes, such as the corners, iris, and pupil, can provide valuable information about eye movements and gaze direction. Techniques like facial landmark detection or eye tracking can be used to extract these features.

(c)EYE MOVEMENT: Analysing eye movement patterns, including saccades (rapid eye movements between fixation points) and smooth pursuit (smooth tracking of moving objects), can provide additional information about the driver's alertness. By tracking the trajectory and speed of eye movements, drowsiness-related changes can be detected.

(d)EYE ASPECT RATIO (EAR): EAR is a measure of eye openness or closure based on the ratio of the distances between certain eye landmarks. By calculating the EAR over time, changes in eye openness can be used to infer drowsiness.

(e)HEAD POSE: Monitoring the driver's head pose, including pitch, yaw, and roll angles, can help detect signs of drowsiness. Sudden or prolonged head drooping, tilting, or nodding can indicate fatigue.

CHAPTER 2

LITERATURE SURVEY

Danisman et al. [1] proposed a method to detect a drowsiness depended on the changes in eye blink rate. Here, the face region from the photos was found using the Viola-Jones detection technique. After that, a neural network-based eye detector was employed to determine where the pupils were. Then the number of blinks per minute is calculated. A rise in blinks indicates that the driver becomes drowsy.

Abtahi et al. [2] proposed a method for detection of drowsiness by yawning. This approach first identified facial features before identifying mouth and eye locations. They computed a hole in the mouth due to the broad mouth opening. The face with the larger hole indicates a yawning mouth.

Hayawi and Waleed [3] proposed a method to detect drowsiness through a signal of Heart Rate Variability (HRV), which is obtained by using EEG (Electroencephalogram) sensors.

Mutya et al. [4] proposed a method to detect drowsiness that uses the steering wheel algorithm. It is primarily based on image-formed steering movements or pictorial-based steering movements and the CNN algorithm for accurate drowsiness classification, which can also lower the rate of false drowsy detection.

Naurois et al. [5] applied MATLAB neural network toolbox to detect and predict the drowsiness of the driver. Through the simultaneous use of driver behavior, psychological, and vehicle metrics in this investigation, they were able to forecast a time of 4.18 minutes.

Chirra et al. [6] proposed a method to detect drowsiness that uses the Viola-jones Algorithm. This algorithm is used for detecting the face and eyes. As a symbol of drowsiness, a four-layer convolutional classifier is trained in order to detect closed eyes.

CHAPTER 3

ALGORITHMS USED

3.2 MOBILENET V2:

MobileNetV2 is a state-of-the-art convolutional neural network (CNN) architecture designed for efficient image classification and recognition tasks on mobile and embedded vision applications. Developed by Google, it is an improvement over the original MobileNetV1, providing a good balance between **accuracy** and **computational efficiency**.

MobileNetV2 introduces a novel layer module called the **inverted residual block with linear bottleneck**, which significantly reduces computation and memory usage while maintaining high accuracy. The model architecture primarily focuses on using **depth wise separable convolutions**, which break down a standard convolution into two smaller and more efficient operations — a **depth wise convolution** and a **pointwise convolution**.

The main advantages of MobileNetV2 are:

- It achieves **high accuracy with minimal computational resources**.
- It is optimized for **real-time applications** on mobile and embedded systems.
- It offers **fast inference speed** while maintaining performance close to larger models.
- It uses **inverted residuals** and **linear bottlenecks** for better feature representation.

3.2.1 THE FINAL MOBILENET V2 MODEL

The complete MobileNetV2 model is composed of **53 layers**, including convolutional, depth wise, and pointwise layers. It is a deeper and more refined version compared to MobileNetV1, but it remains lightweight and fast. The design allows it to perform efficiently even on low-power devices.

After implementing all the architectural innovations, such as:

1. **Inverted Residuals with Linear Bottlenecks** – These blocks reduce the number of parameters and improve feature reuse across layers.
2. **Depth wise Separable Convolutions** – Break down convolutions into smaller, more efficient operations to reduce computation.

3. **Shortcut Connections** – Enable better gradient flow and network stability during training.
4. **Efficient Feature Extraction** – Extracts essential spatial features with minimal redundancy.

The final MobileNetV2 model can be represented as a **highly efficient, compact, and scalable neural network**, ideal for **real-time driver drowsiness detection** and other computer vision tasks.

Key Characteristics:

- **Lightweight and efficient design** suitable for mobile and embedded systems.
- **Lower computational cost** compared to traditional CNNs.
- **High accuracy-to-complexity ratio**.
- **Excellent trade-off** between performance and model size.

3.2 IMAGENET DATASET

- Image preprocessing is a crucial part of the system and can influence the maximum accuracy that the model attains during training. At a minimum, images need to be decoded and resized to fit the model. For Inception, images need to be 299x299x3 pixels. The image annotations were crowdsourced. This actually made the testbed of computer vision tasks really very robust, large, and expensive. Based on ImageNet a 1000 class classification challenge started with the name ImageNet Large Scale Visual Recognition Challenge (ILSVRC).This competition is responsible for the birth of most of the prominent CNN models.

- However, simply decoding and resizing are not enough to get good accuracy. The ImageNet training dataset contains 2500 images. One pass over the set of training images is referred to as an epoch. During training, the model requires several passes through the training dataset to improve its image recognition capabilities. To train Inception v3 to sufficient accuracy, use between 20 to 30 epochs depending on the global batch size.

- It is useful to continuously alter the images before feeding them to the model so that a particular image is slightly different at every epoch. How to best do this preprocessing of images is as much art as it is science. A well-designed preprocessing stage can significantly boost the recognition capabilities of a model. Too simple a preprocessing stage may create an artificial ceiling on the accuracy that the same model can attain during training

3.3 SOFTMAX ACTIVATION FUNCTION

Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

Mathematically:

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e$-24 (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Fig 3.2 Mathematical representation of softmax function.

3.4 ADAM OPTIMIZER

The **Adam optimizer** is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

3.4.1 BENEFITS OF USING ADAM OPTIMIZER

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

3.4.2 WORKING OF ADAM OPTIMIZER

Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the

gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

3.4.3 PARAMETERS OF ADAM OPTIMIZER

1. **alpha**. Also referred to as the learning rate or step size. The proportion that weights are updated. Larger values results in faster initial learning before the rate is updated. Smaller values slow learning right down during training
2. **beta1**. The exponential decay rate for the first moment estimates
3. **beta2**. The exponential decay rate for the second-moment estimates. This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
4. **epsilon**. Is a very small number to prevent any division by zero in the implementation.

CHAPTER 4

HARDWARE AND SOFTWARE COMPONENTS

4.1 HARDWARE REQUIREMENTS

- Processor: i5 and above
- Ram: 8GB and above
- System:32 or 64-bit Windows 10

4.2 SOFTWARE REQUIREMENTS

- **Python:** Ensure Python is installed on the system. Link to download and install the latest version of Python from the official Python website is https://www.python.org/
- **TensorFlow:** Install TensorFlow, an open-source machine learning framework, which includes various deep learning functionalities. It can be installed using the pip package manager by running the command: **pip install tensorflow** in command prompt.
- **Keras:** Keras is a high-level neural networks API that runs on top of TensorFlow. It provides a user-friendly interface for building and training deep learning models. Install Keras using the pip package manager by running the command: **pip install keras**
- OpenCV: OpenCV (Open Source Computer Vision Library) is a popular library for computer vision tasks, such as working with images and videos. Install OpenCV using the pip package manager by running the command: **pip install opencv-python**
- Jupyter Notebook (optional): Jupyter Notebook is an interactive development environment that allows to create and run Python code in a browser-based interface. It's useful for experimenting and prototyping the code. Install Jupyter Notebook using the pip package manager by running the command**: pip install jupyter**

CHAPTER 5

PROPOSED SYSTEM

5.1 SYSTEM ARCHITECTURE

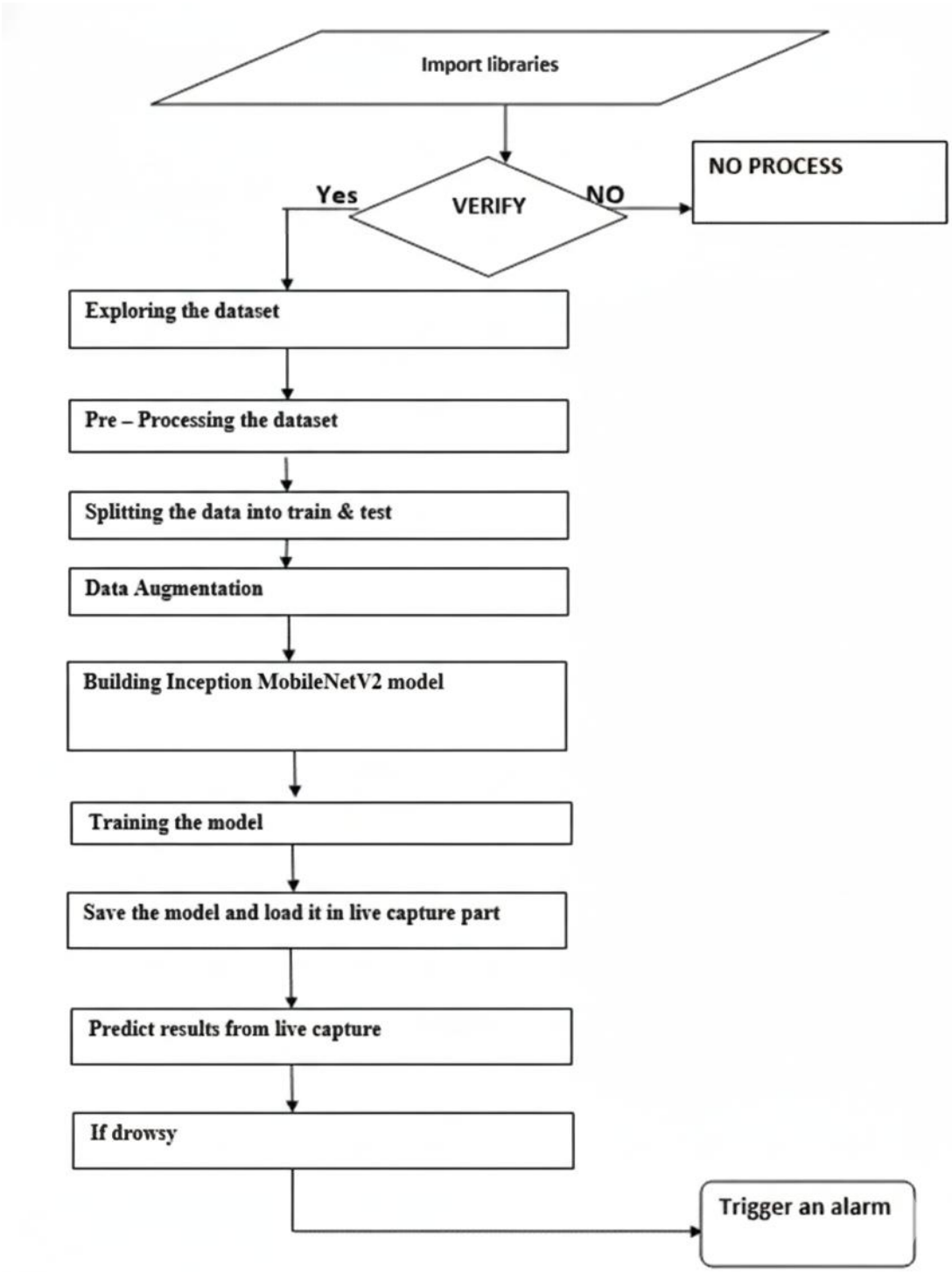


Fig 5.1 Flowchart of the proposed model

IMPORT LIBRARIES: All the required libraries are imported initially.

EXPLORING THE DATASET: The existing yawn_eye_dataset_new [7] is used as a dataset which contains 4 different directories namely Closed, Open, no_yawn, yawn. A customized dataset is created by using opencv under different conditions such as with or without spectacles, with or without lighting.

PRE-PROCESSING THE DATASET: In each image, faces are identified using Haarcascade classifier to quickly discard non-faces and avoid wasting precious time and computations.

SPLITTING THE DATA INTO TRAIN AND TEST: Dataset is splitted into 2 sets namely training and testing sets. Training set is of 70% and testing set is of 30% from the overall dataset.

DATA AUGMENTATION: Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points. In the case of image augmentation, we make geometric and color space transformations (flipping, zooming, rotating) to increase the size and diversity of the training set.

BUILDING INCEPTION V3: Pretrained Inception v3 is loaded and fine-tuned for the task. The output features from Inception v3 are used for classification in the drowsiness detection system. This approach is also called transfer learning.

TRAINING THE MODEL: After the model is defined, it is compiled and trained using the training dataset.

SAVE THE MODEL AND LOAD IT IN LIVE CAPTURE PART: After training, the model is saved and later loaded for live prediction.

PREDICT RESULTS FROM LIVE CAPTURE: OpenCV is used for capturing live facial input, and the trained model detects drowsiness. When the eyes are open, it displays ‘Eyes Open’; when the eyes are closed, it shows ‘Eyes Closed’. To avoid false positives due to blinking, a counter is maintained to track the duration of eye closure. The alarm is triggered only when this counter exceeds a predefined threshold, ensuring accurate detection of drowsiness.

5.2 ADVANTAGES OF PROPOSED SYSTEM

MobileNetV2 is a highly efficient deep learning architecture that offers several advantages for driver drowsiness detection. It effectively extracts high-level visual features from images with significantly reduced computational cost, achieving high accuracy and generalizing well to different lighting and facial conditions, making it suitable for real-time driver monitoring applications on edge devices.

5.2.1 ADVANTAGES OF MOBILENETV2:

1. High accuracy with efficiency: MobileNetV2 is a lightweight convolutional neural network (CNN) architecture designed for mobile and embedded vision applications. It achieves excellent performance in image classification tasks, comparable to larger models, while being significantly more computationally efficient. This makes it ideal for real-time driver drowsiness detection where resources are limited.
2. Efficient feature extraction: MobileNetV2 utilizes "inverted residuals" and "linear bottlenecks," which allow it to extract rich visual features efficiently. The inverted residuals help in preserving information flow, while linear bottlenecks prevent non-linearities from destroying information in low-dimensional spaces, capturing both fine-grained and high-level information from input images effectively.
3. Reduced computational complexity: The architectural design of MobileNetV2, particularly its use of depth wise separable convolutions and linear bottlenecks, drastically reduces the number of parameters and computational operations compared to traditional CNN architectures. This makes it highly efficient for real-time applications, like driver drowsiness detection, where processing speed and low power consumption are crucial, especially on edge devices.
4. Transfer learning: MobileNetV2's pretraining on ImageNet provides a valuable starting point for transfer learning. By leveraging the pretrained weights, the model can be fine-tuned on a smaller dataset specific to driver drowsiness detection, even if the available dataset is limited. This can help overcome data scarcity challenges and improve performance.

5.2.3 ADVANTAGES OF TRANSFER LEARNING:

1. Reduced training time and resource requirements: Pretraining deep neural networks on large-scale datasets, such as ImageNet, requires significant computational resources and time. Transfer learning allows us to leverage the knowledge and learned representations from these pretrained models, significantly reducing the training time and computational requirements for a specific task. Instead of training a model from scratch, we can start with a pretrained model and fine-tune it on a smaller dataset specific to the target task, resulting in faster development and deployment.

2. Overcoming data scarcity: In many practical scenarios, obtaining a large labelled dataset for training deep learning models can be challenging. Transfer learning addresses the issue of limited data availability by utilizing pretrained models trained on massive datasets. By leveraging the learned representations from these datasets, we can achieve good performance even with a smaller dataset for the target task. The pretrained model acts as a knowledge transfer mechanism, providing a head start for training on the specific task with limited data.

3. Improved generalization: Pretrained models have typically been trained on diverse and extensive datasets, capturing a wide range of visual patterns and features. As a result, they can effectively generalize to new, unseen data. Transfer learning allows us to benefit from this generalization capability, as the pretrained models have already learned relevant features and high-level representations that can be useful for the target task. By transferring this knowledge, the model can generalize better, even when the target task has a different distribution or limited training data.

4. Better convergence and performance: Transfer learning provides a better initialization for the model parameters compared to random initialization. This initialization with pretrained weights brings the model closer to the optimal solution for the target task, allowing it to converge faster and achieve better performance. The pretrained model has learned low-level features that are applicable across different tasks, and by fine-tuning these features on the target task, the model can adapt more effectively and potentially outperform models trained from scratch.

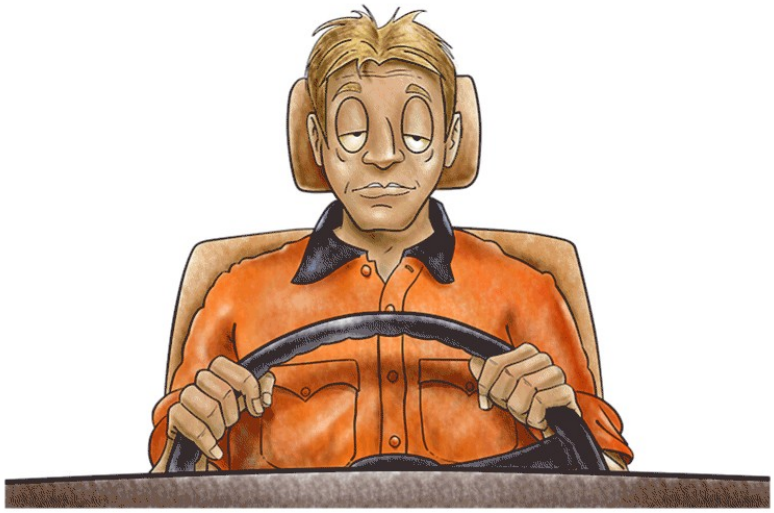
5. Domain adaptation: Transfer learning is particularly beneficial when dealing with tasks where the source and target domains are different but related. Pretraining on a source domain with abundant labelled data helps the model learn generic features that

are transferrable to the target domain. The model can then be fine-tuned on a smaller labelled dataset from the target domain, effectively adapting its knowledge to the specific domain. This is especially useful in scenarios where collecting labeled data from the target domain is expensive or impractical.

Overall, transfer learning enables the reuse of learned representations, accelerates training, improves generalization, and helps overcome data limitations. It is a powerful technique that enhances the efficiency and effectiveness of deep learning models, making them more accessible and practical in various real-world applications. MobileNetV2 excels at extracting rich visual features from facial images with high efficiency, making it highly effective for driver drowsiness detection, especially on resource-constrained devices. By leveraging its deep feature extraction capability and computational efficiency, the system can achieve improved performance and robustness in recognizing drowsiness patterns, leading to accurate and timely alerts.

CHAPTER 6

IMPLEMENTATION



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

```
import sys
print(sys.executable)
print(sys.version)
```

```
/opt/anaconda3/envs/drowsy/bin/python
3.10.19 (main, Oct 21 2025, 16:37:10) [Clang 20.1.8 ]
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import cv2
```

labels

```
labels = os.listdir("./train")

labels
```

```
['Closed', 'no_yawn', 'yawn', 'Open']
```

visualize random 1 image

```
from pathlib import Path
import random
import matplotlib.pyplot as plt

TRAIN_DIR = Path("train")

# pick a random class folder
label = random.choice([d.name for d in TRAIN_DIR.iterdir() if
d.is_dir()])

# pick a random image from that class folder
image_file = random.choice(list((TRAIN_DIR / label).glob("*.jpg")))

# display image
img = plt.imread(image_file)
plt.imshow(img)
plt.title(label)
plt.axis("off")
plt.show()
```

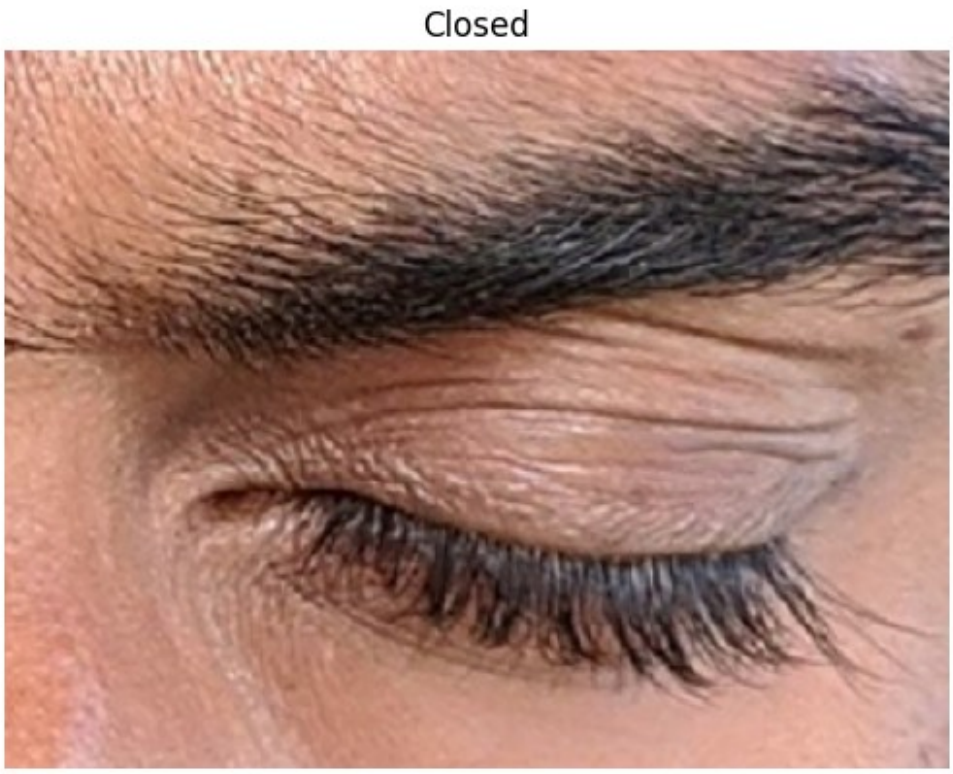


image array

```
from pathlib import Path
import random
import matplotlib.pyplot as plt

DATASET_DIR = Path("train")

label = random.choice([d.name for d in DATASET_DIR.iterdir() if
d.is_dir()])
image_path = random.choice(list((DATASET_DIR / label).glob("*.jpg")))

a = plt.imread(image_path)

plt.imshow(a)
plt.title(label)
plt.axis("off")
```

```
(-0.5, 253.5, 246.5, -0.5)
```

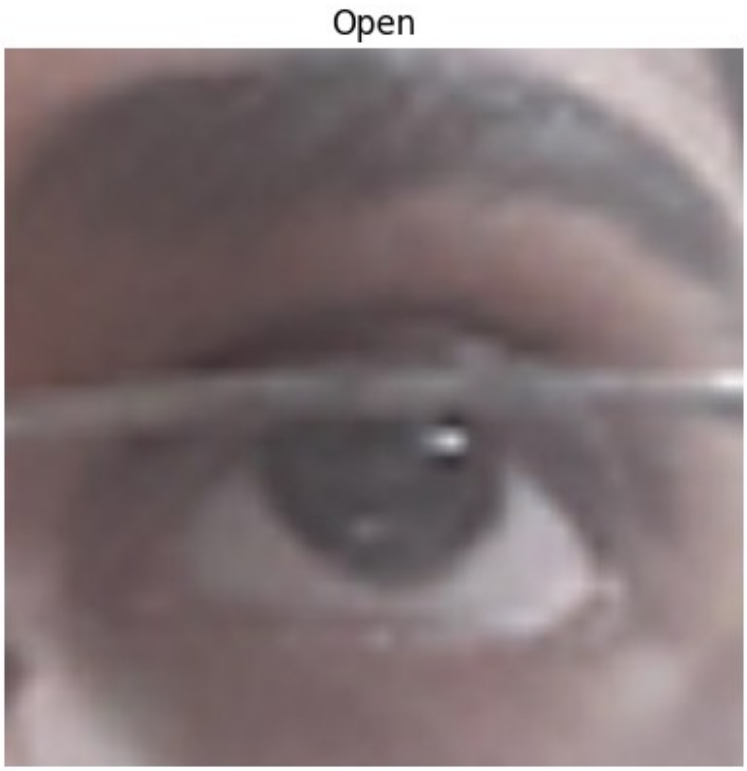


image shape

```
a.shape
```

```
(247, 254, 3)
```

visualize yawn image.

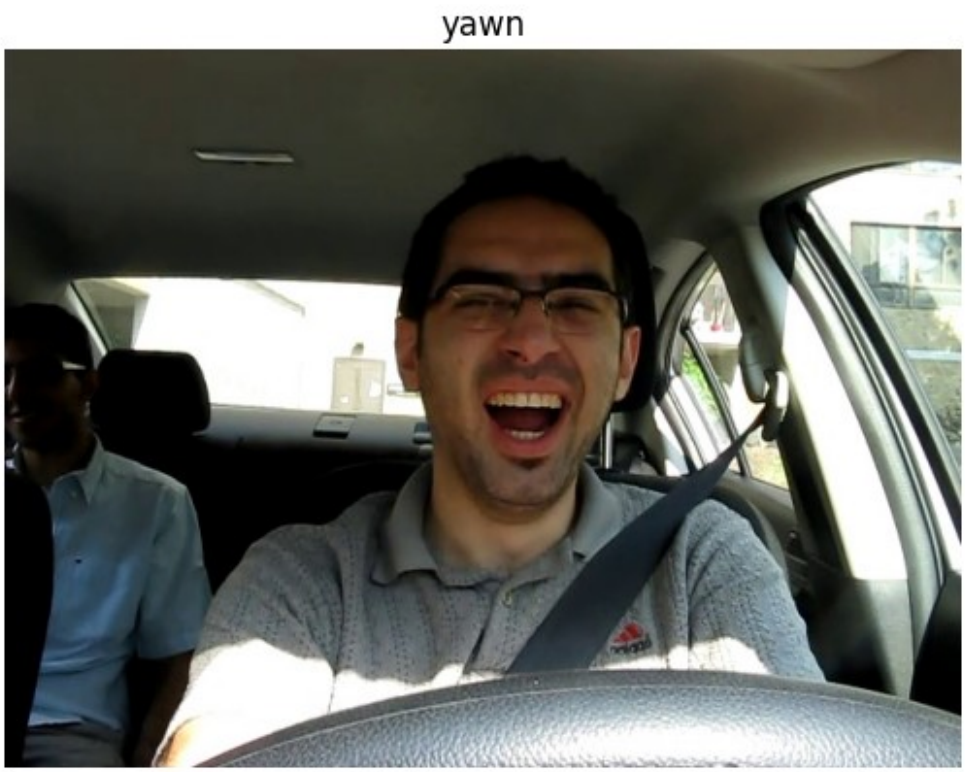
Here background is unnecessary. we need only face image array

```
from pathlib import Path
import random
import matplotlib.pyplot as plt

DATASET_DIR = Path("train")
label = "yawn"   # or Closed / Open / no_yawn
image_path = random.choice(list((DATASET_DIR / label).glob("*.jpg")))

img = plt.imread(image_path)
plt.imshow(img)
plt.title(label)
plt.axis("off")
```

```
(-0.5, 639.5, 479.5, -0.5)
```



for yawn and not_yawn. Take only face

```
from pathlib import Path
import cv2
import os

def face_for_yawn(direc="train",
face_cas_path="haarcascade_frontalface_default.xml"):
    yawn_no = []
    IMG_SIZE = 145
    categories = ["yawn", "no_yawn"]

    direc = Path(direc)

    for category in categories:
        path_link = direc / category
        class_num = categories.index(category)
        print("Processing:", category)

        for image in os.listdir(path_link):
            img_path = path_link / image
            image_array = cv2.imread(str(img_path), cv2.IMREAD_COLOR)
            if image_array is None:
                continue

            face_cascade = cv2.CascadeClassifier(face_cas_path)
            faces = face_cascade.detectMultiScale(image_array, 1.3, 5)

            for (x, y, w, h) in faces:
                roi_color = image_array[y:y+h, x:x+w]
                resized_array = cv2.resize(roi_color, (IMG_SIZE,
IMG_SIZE))
                yawn_no.append([resized_array, class_num])

    return yawn_no

yawn_no_yawn = face_for_yawn()
```

```
Processing: yawn
Processing: no_yawn
```

for closed and open eye

```
from pathlib import Path
import cv2
import os

def get_data(dir_path="train"):
    data = []
```

```
    IMG_SIZE = 145
    labels = ["Closed", "Open"]

    dir_path = Path(dir_path)

    for label in labels:
        path = dir_path / label
        class_num = labels.index(label)

        print(f"Loading: {label} ({class_num})")

        for img in os.listdir(path):
            try:
                img_array = cv2.imread(str(path / img),
cv2.IMREAD_COLOR)
                img_array = cv2.resize(img_array, (IMG_SIZE,
IMG_SIZE))
                data.append([img_array, class_num])
            except Exception as e:
                continue

    return data

data_train = get_data()
len(data_train)
```

```
Loading: Closed (0)
Loading: Open (1)

1452
```

extend data and convert array

```
def append_data():
    yawn_no = face_for_yawn()   # returns list
    eye_data = get_data()       # returns list
    combined = yawn_no + eye_data

    X = []
    y = []

    for features, label in combined:
        X.append(features)
        y.append(label)

    X = np.array(X).reshape(-1, 145, 145, 3)
    y = np.array(y)

    return X, y
```

new variable to store

```
X, y = append_data()
X.shape, y.shape

Processing: yawn
Processing: no_yawn
Loading: Closed (0)
Loading: Open (1)

((1925, 145, 145, 3), (1925,))
```

separate label and features

```
def append_data():
    yawn_no = face_for_yawn()      # list of [image, label]
    eye_data = get_data()          # list of [image, label]

    combined = yawn_no + eye_data

    X = []
    y = []

    for features, label in combined:
        X.append(features)
        y.append(label)

    X = np.array(X).reshape(-1, 145, 145, 3)
    y = np.array(y)

    return X, y
```

reshape the array

```
X, y = append_data()
print(X.shape, y.shape)

Processing: yawn
Processing: no_yawn
Loading: Closed (0)
Loading: Open (1)
(1925, 145, 145, 3) (1925,)
```

LabelBinarizer

```
import sys
print(sys.executable)
```

```
/opt/anaconda3/envs/drowsy/bin/python
```

```
import sklearn
print(sklearn.__version__)
```

```
1.7.2
```

```
import tensorflow as tf
import cv2
import mediapipe as mp
import numpy as np

print("TensorFlow:", tf.__version__)
print("NumPy:", np.__version__)
print("OpenCV:", cv2.__version__)
print("Mediapipe:", mp.__version__)
print("GPU:", tf.config.list_physical_devices("GPU"))
```

```
objc[64224]: Class CaptureDelegate is implemented in both
/opt/anaconda3/envs/drowsy/lib/python3.10/site-packages/cv2/cv2.abi3.s
o (0x16117a6b0) and /opt/anaconda3/envs/drowsy/lib/python3.10/site-
packages/mediapipe/.dylibs/libopencv_videoio.3.4.16.dylib
(0x300cc8860). This may cause spurious casting failures and mysterious
crashes. One of the duplicates must be removed or renamed.
objc[64224]: Class CVWindow is implemented in both
/opt/anaconda3/envs/drowsy/lib/python3.10/site-packages/cv2/cv2.abi3.s
o (0x16117a700) and /opt/anaconda3/envs/drowsy/lib/python3.10/site-
packages/mediapipe/.dylibs/libopencv_highgui.3.4.16.dylib
(0x17cae8a68). This may cause spurious casting failures and mysterious
crashes. One of the duplicates must be removed or renamed.
objc[64224]: Class CVView is implemented in both
/opt/anaconda3/envs/drowsy/lib/python3.10/site-packages/cv2/cv2.abi3.s
o (0x16117a728) and /opt/anaconda3/envs/drowsy/lib/python3.10/site-
packages/mediapipe/.dylibs/libopencv_highgui.3.4.16.dylib
(0x17cae8a90). This may cause spurious casting failures and mysterious
crashes. One of the duplicates must be removed or renamed.
objc[64224]: Class CVSlider is implemented in both
/opt/anaconda3/envs/drowsy/lib/python3.10/site-packages/cv2/cv2.abi3.s
o (0x16117a750) and /opt/anaconda3/envs/drowsy/lib/python3.10/site-
packages/mediapipe/.dylibs/libopencv_highgui.3.4.16.dylib
(0x17cae8ab8). This may cause spurious casting failures and mysterious
crashes. One of the duplicates must be removed or renamed.

TensorFlow: 2.15.0
NumPy: 1.23.5
OpenCV: 4.12.0
```

```
Mediapipe: 0.9.2.1
GPU: [PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
```

label array

```
import os
import cv2
import numpy as np

def load_dataset():
    base = "./train"  # your dataset folder
    classes = os.listdir(base)
    data = []

    for label_index, label_name in enumerate(classes):
        folder = os.path.join(base, label_name)
        print(f"Loading: {label_name}")
        for img_file in os.listdir(folder):
            img_path = os.path.join(folder, img_file)
            img = cv2.imread(img_path)
            img = cv2.resize(img, (145,145))
            data.append((img, label_index))

    return data

new_data = load_dataset()
print("Samples Loaded:", len(new_data))
```

```
Loading: Closed
Loading: no_yawn
Loading: yawn
Loading: Open
Samples Loaded: 2900
```

```
X = []
y = []

for feature, label in new_data:
    X.append(feature)
    y.append(label)

X = np.array(X)
y = np.array(y)

print("X shape:", X.shape)
print("y shape:", y.shape)
print("Unique labels:", np.unique(y))
```

```
X shape: (2900, 145, 145, 3)
y shape: (2900,)
Unique labels: [0 1 2 3]

X = X / 255.0   # Normalize
```

train test split+one hot encoding

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=True, random_state=42
)

# One-hot encode labels
y_train = to_categorical(y_train, num_classes=4)
y_test = to_categorical(y_test, num_classes=4)

print("Train data:", X_train.shape, y_train.shape)
print("Test data:", X_test.shape, y_test.shape)
```

```
Train data: (2320, 145, 145, 3) (2320, 4)
Test data: (580, 145, 145, 3) (580, 4)
```

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Convert y_train back to class labels (remove one-hot)
y_train_labels = np.argmax(y_train, axis=1)

# Compute class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_labels),
    y=y_train_labels
)

class_weights = dict(enumerate(class_weights))
print("Class Weights:", class_weights)
```

```
Class Weights: {0: 1.008695652173913, 1: 0.9931506849315068, 2:
1.0104529616724738, 3: 0.9880749574105622}
```

length of X_test

```
len(X_test)
```

```
580
```

```
print("X_train min:", X_train.min(), "max:", X_train.max())
print("y_train example:", y_train[:5])
```

```
X_train min: 0.0 max: 1.0
y_train example: [[0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]]
```

Not necessary, only use to matching with my pc version

```
# !pip install tensorflow==2.3.1
# !pip install keras==2.4.3
```

import some dependencies

```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,
Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
```

tensorflow version

```
tf.__version__
```

```
'2.15.0'
```

keras version

```
import keras
keras.__version__
```

```
'2.15.0'
```

Data Augmentation

```
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)
```

Model

```
model = Sequential()

model.add(Conv2D(256, (3, 3), activation="relu",
input_shape=X_train.shape[1:]))
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()
```

```
2025-11-11 15:05:12.135998: I
metal_plugin/src/device/metal_device.cc:1154] Metal device set to:
Apple M2
2025-11-11 15:05:12.151014: I
metal_plugin/src/device/metal_device.cc:296] systemMemory: 8.00 GB
2025-11-11 15:05:12.151021: I
```

```
metal_plugin/src/device/metal_device.cc:313] maxCacheSize: 2.67 GB
2025-11-11 15:05:12.152960: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_facto
ry.cc:306] Could not identify NUMA node of platform GPU ID 0,
defaulting to 0. Your kernel may not have been built with NUMA
support.
2025-11-11 15:05:12.154783: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_facto
ry.cc:272] Created TensorFlow device
(/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) ->
physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 143, 143, 256)     7168

 max_pooling2d (MaxPooling2  (None, 71, 71, 256)       0
 D)

 conv2d_1 (Conv2D)           (None, 69, 69, 128)       295040

 max_pooling2d_1 (MaxPoolin  (None, 34, 34, 128)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        73792

 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        18464

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 flatten (Flatten)           (None, 1568)              0

 dropout (Dropout)           (None, 1568)              0

 dense (Dense)               (None, 64)                100416

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 495140 (1.89 MB)
Trainable params: 495140 (1.89 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Convert one-hot encoded y_train back to class numbers
y_train_labels = np.argmax(y_train, axis=1)

# Compute class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_labels),
    y=y_train_labels
)

# Convert to dictionary form required by model.fit()
class_weights = dict(enumerate(class_weights))

print("Class Weights:", class_weights)
```

```
Class Weights: {0: 1.008695652173913, 1: 0.9931506849315068, 2:
1.0104529616724738, 3: 0.9880749574105622}
```

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.models import Model

IMG_SIZE = 145

base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3),
include_top=False, weights='imagenet')
base_model.trainable = False  # Freeze pretrained weights

x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
output = Dense(4, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output)

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

model.summary()
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows`
is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224)
will be loaded as the default.
Model: "model"
```

```
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==========================================================================================
 input_1 (InputLayer)        [(None, 145, 145, 3)]     0           []


 Conv1 (Conv2D)              (None, 73, 73, 32)        864         ['input_1[0][0]']


 bn_Conv1 (BatchNormalizati  (None, 73, 73, 32)        128         ['Conv1[0][0]']
 on)


 Conv1_relu (ReLU)           (None, 73, 73, 32)        0           ['bn_Conv1[0][0]']


 expanded_conv_depthwise (D  (None, 73, 73, 32)        288         ['Conv1_relu[0][0]']
 epthwiseConv2D)


 expanded_conv_depthwise_BN  (None, 73, 73, 32)        128         ['expanded_conv_depthwise[0][0
                                                                   ]']
 (BatchNormalization)


 expanded_conv_depthwise_re  (None, 73, 73, 32)        0           ['expanded_conv_depthwise_BN[0
 lu (ReLU)                                                         ][0]']


 expanded_conv_project (Con  (None, 73, 73, 16)        512         ['expanded_conv_depthwise_relu
 v2D)                                                              [0][0]']


 expanded_conv_project_BN (  (None, 73, 73, 16)        64
```

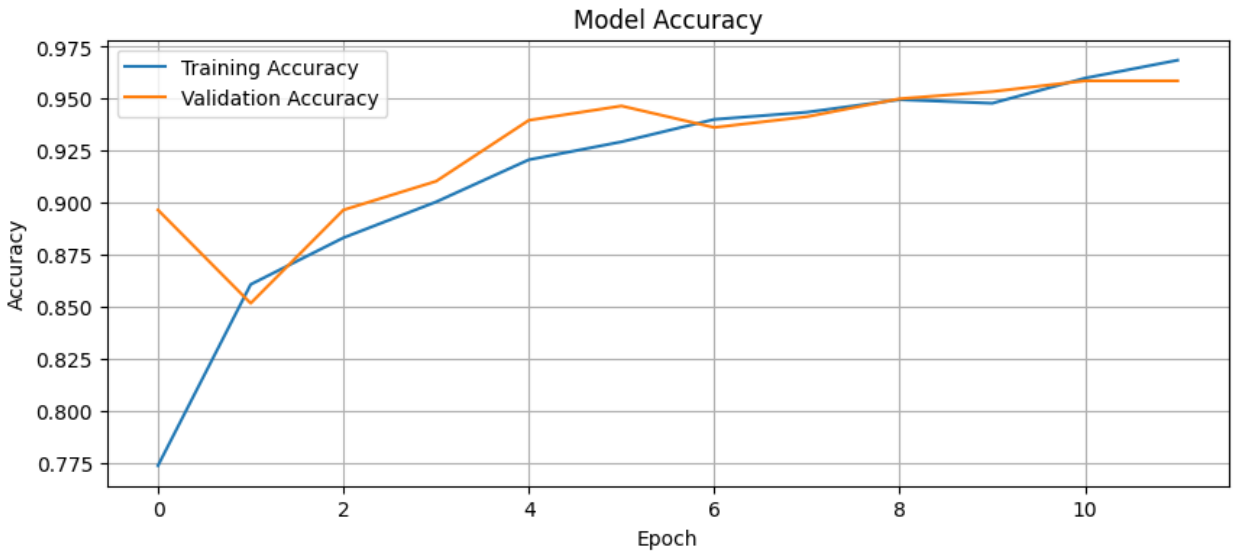
```
73/73 [==============================] - 12s 119ms/step - loss: 0.8077
- accuracy: 0.7737 - val_loss: 0.2879 - val_accuracy: 0.8966
Epoch 2/12
73/73 [==============================] - 4s 54ms/step - loss: 0.4020 -
accuracy: 0.8608 - val_loss: 0.3207 - val_accuracy: 0.8517
Epoch 3/12
73/73 [==============================] - 4s 54ms/step - loss: 0.3094 -
accuracy: 0.8832 - val_loss: 0.2382 - val_accuracy: 0.8966
Epoch 4/12
73/73 [==============================] - 4s 52ms/step - loss: 0.2281 -
accuracy: 0.9004 - val_loss: 0.2125 - val_accuracy: 0.9103
Epoch 5/12
73/73 [==============================] - 4s 54ms/step - loss: 0.1729 -
accuracy: 0.9207 - val_loss: 0.1532 - val_accuracy: 0.9397
Epoch 6/12
73/73 [==============================] - 4s 54ms/step - loss: 0.1659 -
accuracy: 0.9293 - val_loss: 0.1400 - val_accuracy: 0.9466
Epoch 7/12
73/73 [==============================] - 4s 55ms/step - loss: 0.1515 -
accuracy: 0.9401 - val_loss: 0.1685 - val_accuracy: 0.9362
Epoch 8/12
73/73 [==============================] - 4s 58ms/step - loss: 0.1353 -
accuracy: 0.9435 - val_loss: 0.1723 - val_accuracy: 0.9414
Epoch 9/12
73/73 [==============================] - 4s 61ms/step - loss: 0.1164 -
accuracy: 0.9496 - val_loss: 0.1390 - val_accuracy: 0.9500
Epoch 10/12
73/73 [==============================] - 4s 54ms/step - loss: 0.1234 -
accuracy: 0.9478 - val_loss: 0.1429 - val_accuracy: 0.9534
Epoch 11/12
73/73 [==============================] - 4s 57ms/step - loss: 0.0891 -
accuracy: 0.9599 - val_loss: 0.1358 - val_accuracy: 0.9586
Epoch 12/12
73/73 [==============================] - 4s 59ms/step - loss: 0.0766 -
accuracy: 0.9685 - val_loss: 0.1308 - val_accuracy: 0.9586
```

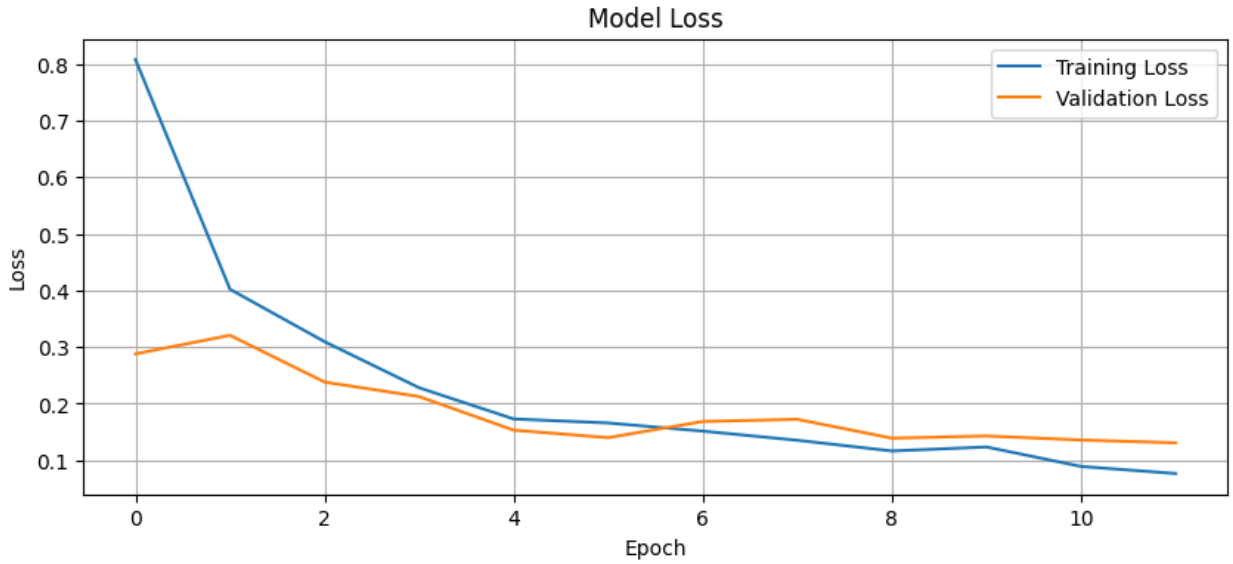
```python
import matplotlib.pyplot as plt

# Accuracy Plot
plt.figure(figsize=(10,4))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Loss Plot
```

```
plt.figure(figsize=(10,4))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```






```
model.save("drowsiness_mobilenetv2.h5")
```

```
/opt/anaconda3/envs/drowsy/lib/python3.10/site-packages/keras/src/
engine/training.py:3103: UserWarning: You are saving your model as an
```

```
HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

Prediction

```
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)

y_true = np.argmax(y_test, axis=1)
```

```
19/19 [==============================] - 2s 56ms/step
```

Classification Report

```
from sklearn.metrics import classification_report, confusion_matrix

labels_new = ["yawn", "no_yawn", "Closed", "Open"]

print(classification_report(y_true, y_pred, target_names=labels_new))
```

```
              precision    recall  f1-score   support

        yawn       1.00      0.99      1.00       151
     no_yawn       0.91      0.93      0.92       141
      Closed       0.93      0.91      0.92       149
        Open       0.99      1.00      1.00       139

    accuracy                           0.96       580
   macro avg       0.96      0.96      0.96       580
weighted avg       0.96      0.96      0.96       580
```

predicting function

```
import os

for folder in ["yawn", "no_yawn", "Closed", "Open"]:
    path = os.path.join("train", folder)
    print(f"\n {folder} -> Sample files:")
    print(os.listdir(path)[:5])  # show first 5 files
```

```
 yawn -> Sample files:
['63.jpg', '189.jpg', '77.jpg', '638.jpg', '604.jpg']
```

```
 no_yawn -> Sample files:
['77.jpg', '837.jpg', '638.jpg', '176.jpg', '610.jpg']

 Closed -> Sample files:
['_80.jpg', '_94.jpg', '_569.jpg', '_541.jpg', '_227.jpg']

 Open -> Sample files:
['_80.jpg', '_94.jpg', '_569.jpg', '_541.jpg', '_227.jpg']
```

```python
from tensorflow.keras.models import load_model
import cv2
import numpy as np

# Load trained model
model = load_model("drowsiness_mobilenetv2.h5")

labels_new = ["yawn", "no_yawn", "Closed", "Open"]
IMG_SIZE = 145

def prepare(filepath):
    img = cv2.imread(filepath)
    if img is None:
        print(" ERROR: Wrong image path:", filepath)
        return None
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img.astype("float32") / 255.0
    img = np.expand_dims(img, axis=0)
    return img

def predict_image(filepath):
    img = prepare(filepath)
    if img is None:
        return
    prediction = model.predict(img)
    class_id = np.argmax(prediction)
    print(f" Predicted Class: {class_id} → {labels_new[class_id]}")

# Test Predictions
predict_image("train/yawn/63.jpg")
predict_image("train/no_yawn/77.jpg")
predict_image("train/Closed/_80.jpg")
predict_image("train/Open/_80.jpg")
```

```
1/1 [==============================] - 1s 688ms/step
 Predicted Class: 2 → Closed
1/1 [==============================] - 0s 27ms/step
 Predicted Class: 1 → no_yawn
1/1 [==============================] - 0s 20ms/step
 Predicted Class: 0 → yawn
```

```
1/1 [==============================] - 0s 31ms/step
 Predicted Class: 3 → Open
```

CHAPTER 7

RESULTS

Fig 7.1 Picture of person with Alert and Safe

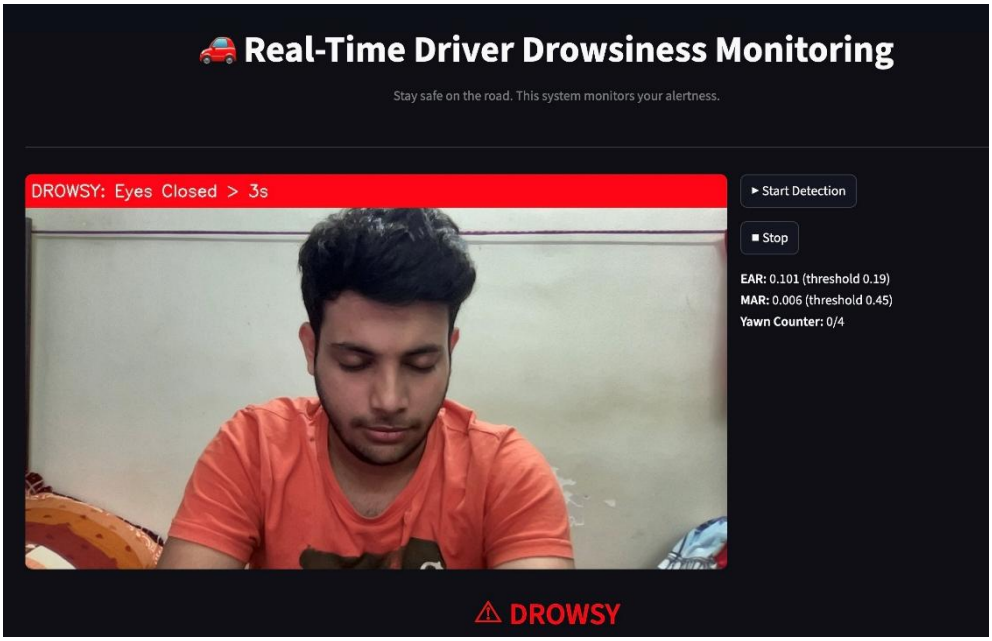


Fig 7.2 Picture of person with eyes closed

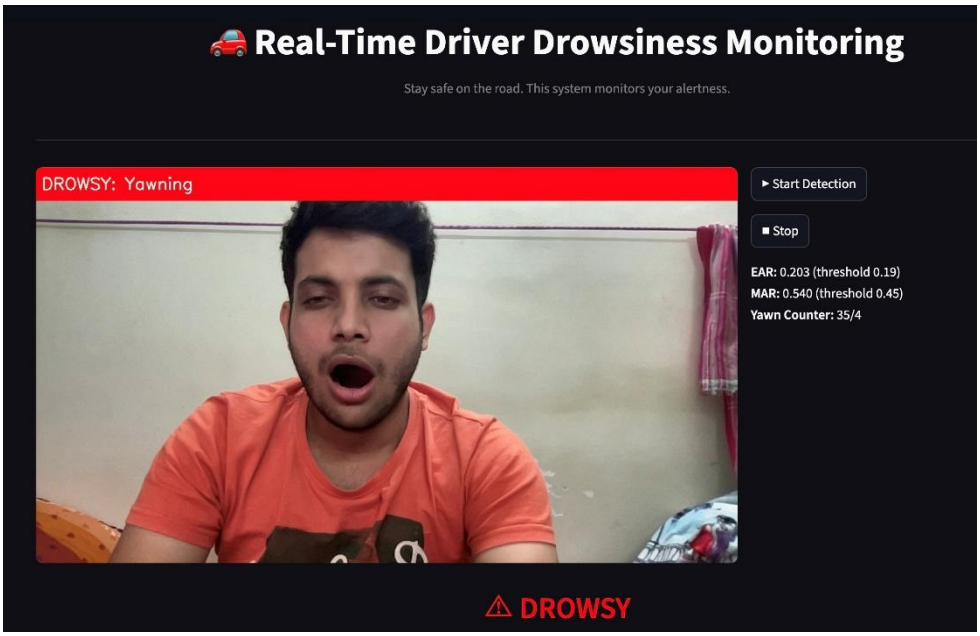


Fig 7.3 Picture of person yawning.

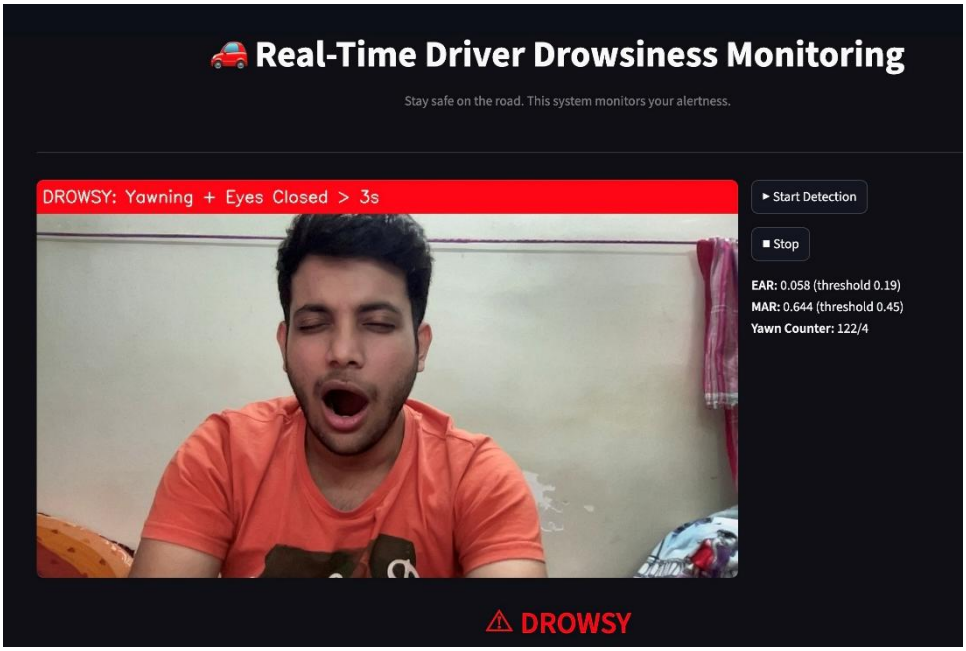


Fig 7.4 Picture of person with closed eyes + yawning.

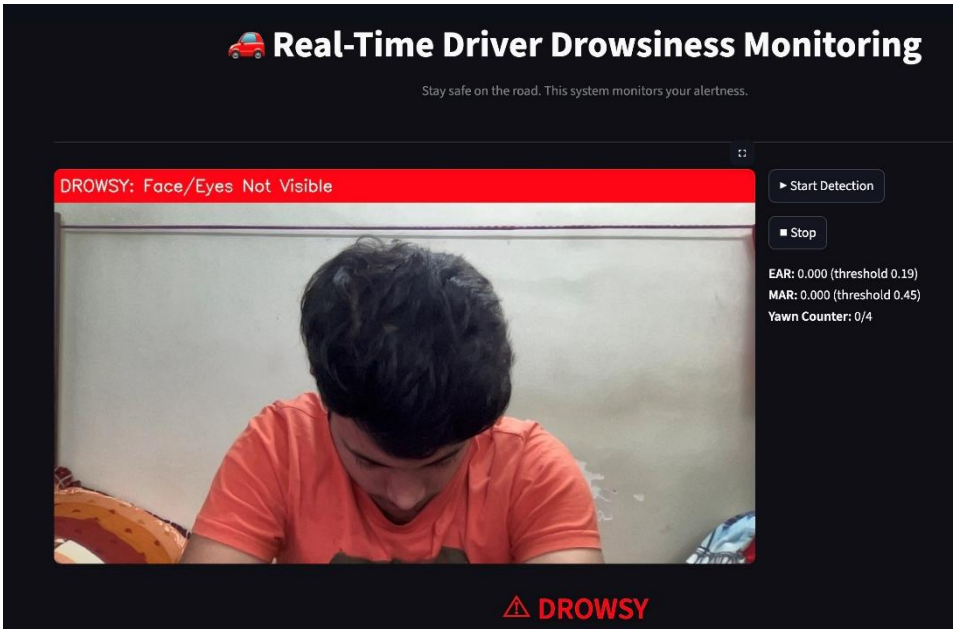


Fig 7.5 Picture of person with closed slept.

CHAPTER 8

FUTURE SCOPE

The future scope for the proposed MobileNetV2 + LSTM model for driver drowsiness detection holds several exciting possibilities. Here are some potential areas of development and advancements:

1. Multimodal fusion: In addition to visual facial features, incorporating other modalities such as audio or physiological signals (e.g., heart rate, electroencephalogram) can provide a more comprehensive understanding of drowsiness. The MobileNetV2

- LSTM model can be extended to fuse information from multiple modalities, enabling a more robust and accurate drowsiness detection system.

2. Real-world deployment and edge optimization: Further research and development efforts can focus on deploying the model in real-world settings, such as integrating it into in-car systems or wearable devices. Given MobileNetV2's inherent efficiency, optimizing the model for various resource-constrained environments and leveraging the power of edge computing will be crucial. This will enable the drowsiness

detection system to provide real-time alerts and interventions to drivers, significantly enhancing road safety with minimal latency and power consumption.

3. Transfer learning to specific driver profiles: Currently, transfer learning is typically employed from large-scale datasets like ImageNet. However, future advancements can explore transfer learning from driver-specific datasets. By training the MobileNetV2 + LSTM model on data from individual drivers, it can learn driver-specific patterns of drowsiness, resulting in personalized drowsiness detection and potentially higher accuracy, while maintaining its computational efficiency.

CHAPTER 9

CONCLUSION

The model has been trained using different deep learning approaches. The accuracy of a basic CNN was 89.02%, while the **MobileNetV2** model achieved an accuracy of 93.6%, outperforming the simpler architecture. The disadvantage that a basic CNN may struggle to capture complex spatial patterns or subtle variations in drowsiness-related features has been effectively overcome by the MobileNetV2 model. In conclusion, the implementation of the MobileNetV2 architecture for drowsiness detection presents a powerful and computationally efficient solution that combines advanced computer vision and feature extraction techniques. The MobileNetV2 network efficiently extracts relevant visual features from input images, capturing important facial cues and eye behaviour indicative of drowsiness, all while maintaining a low computational footprint. By utilizing this optimized architecture, a robust and efficient drowsiness detection system has been developed, capable of accurately identifying signs of fatigue and alerting drivers in real time. This model has the potential to significantly enhance road safety by mitigating the risks associated with drowsy driving and preventing potentially life-threatening accidents, particularly in resource-constrained environments.

REFERENCES

DATASET:

https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset?resource=download

[1] Danisman, T., Bilasco, I.M., Djeraba, C., Ihaddadene, N. (2014). Drowsy driver detection system using eye blink patterns. 2010 International Conference on Machine and Web Intelligence, Algiers, Algeria, pp. 230-233.

https://www.researchgate.net/publication/251970873_Drowsy_driver_detection_system_using_eye_blink_patterns

[2] Abtahi, S., Hariri, B., Shirmohammadi, S. (2011). Driver drowsiness monitoring based on yawning detection. 2011 IEEE International Instrumentation and Measurement Technology Conference, Binjiang, pp. 1-4. https://doi.org/10.1109/imtc.2011.5944101

https://www.researchgate.net/publication/224245285_Driver_drowsiness_monitoring_based_on_yawning_detection

[3] A. A. Hayawi and J. Waleed, "Driver's drowsiness monitoring and alarming auto-system based on EOG signals," in 2019 2nd International Conference on Engineering Technology and its Applications (IICETA), pp. 214–218, IEEE, 2019.

https://ieeexplore.ieee.org/document/9013000

[4] K. Mutya, J. Shah, A. D. McDonald, and J. Jefferson, "What are steering pictures are worth? using image-based steering features to detect drowsiness on rural roads," Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 63, no. 1, pp. 2041–2045, 2019.

https://journals.sagepub.com/doi/abs/10.1177/1071181319631220

[5] C. J. de Naurois, C. Bourdin, A. Stratulat, E. Diaz, and J.-L. Vercher, "Detection and prediction of driver drowsiness using artificial neural network models," Accident Analysis & Prevention, vol. 126, pp. 95– 104, 2019

https://www.sciencedirect.com/science/article/pii/S0001457517304347

NAMES	ROLL NUMBER	CONTRIBUTION
AVIJIT DATTA	2511AI33	10%
ASHWANI KUMAR	2511CS18	10%
JADHAV PRAFUL NAYAK	2511CS03	10%
BISWARAJ BHATTACHARYYA	2511AI43	10%
JOSIGA	2511AI10	10%
SHREYA SINGHAL	2511AI21	10%
HIMANSHU MEHRA	2511CS19	10%
AVNI VERMA	2511AI08	10%
ARYAN	2511AI29	10%
KEDHARNATH REDDY	2511AI36	10%