

# Machine Learning Project

**ESILV A4 - DIA2**

***Élèves :***

Nour AFFES  
Lucas BLANCHET  
Hugo BONNELL  
Rayan HAMADEH

***Enseignants :***

Nédra MELLOULI  
Zachary FAKIR

20 novembre 2024



## Table des matières

<b>1</b>	<b>Pre-Project</b>	<b>2</b>
1.1	Problem Definition . . . . .	2
1.1.1	Existing technologies . . . . .	2
1.1.2	Key business objectives . . . . .	2
1.2	Dataset Description . . . . .	3
1.3	Scope . . . . .	3
1.4	Organization . . . . .	3
<b>2</b>	<b>Stage 1</b>	<b>3</b>
2.1	Problem Formalization . . . . .	3
2.1.1	Problem Statement . . . . .	3
2.1.2	Understanding the Dataset . . . . .	4
2.2	Refining the Dataset . . . . .	4
2.3	Evaluation Metrics . . . . .	5
2.4	Data cleaning and pre-processing . . . . .	5

# 1 Pre-Project

## 1.1 Problem Definition

Drowsy driving, a serious hazard often resulting from insufficient rest, causes drivers to lose focus, react more slowly, and even experience brief micro-sleeps. This issue has devastating impacts, contributing to approximately 328,000 accidents, 109,000 injuries, and 6,400 fatalities annually (Bankrate [3]). Unlike the effects of alcohol or drugs, drowsiness is harder to detect, creating a challenge for preventive measures. A 2019 AAA Foundation [1] study reveals that while 96% of drivers recognize the extreme risk of drowsy driving, only 29% feel that they are at risk of getting pulled over by law enforcement.

### 1.1.1 Existing technologies

- **Mercedes-Benz ATTENTION ASSIST** [5] : analyzes driving pattern in the first few minutes of the ride, and then detects patterns that might be due to fatigue, and alerts by a coffee cup symbol on the dashboard.
  - **Ford Driver Alert System** [4] : monitor level of alertness based on driving behavior, an alert is displayed on the information display, that will be automatically cleared after a while and can also be cleared by pressing ok on the steering wheel. This feature can be turned off and will stay off even if you turn the ignition off and back on. This method can be impacted by windshield conditions such as bird droppings, ice, or snow. System is only available at speeds below activation speed (64 km/h)
- Proposed Solution and Business Objectives.

Our project aims to develop a machine-learning-powered system that monitors driver fatigue in real-time. This solution will leverage a camera-based approach, ideally suited to modern vehicles already equipped with interior cameras. Unlike current technology, this model will focus directly on the driver's condition by analyzing eye and mouth states, reducing dependence on external driving conditions.

### 1.1.2 Key business objectives

- **Increase Road Safety** : By alerting drivers when drowsiness indicators are detected, the system can help prevent accidents.
- **Enhance Detection Accuracy** : Monitoring eye and mouth states ensures fatigue detection remains consistent across different road, traffic, and weather conditions.
- **Offer a Scalable Solution** : Using a straightforward dataset and CNN model, the system is designed to integrate smoothly into existing vehicle technology, avoiding significant production costs.
- **Foundation for Advanced Safety Features** : In future applications, the model could work with autonomous driving technology to execute actions like parking assistance or alerting emergency services if the driver fails to respond.

## 1.2 Dataset Description

The dataset [2] for this project contains labeled images indicating whether a driver's eyes are "open" or "closed" and mouth states as "yawn" or "no yawn." This binary classification structure aligns well with a CNN model, allowing us to detect drowsiness through simple yet reliable features, focusing on real-time driver monitoring. By building a robust CNN on this well-labeled dataset, we emphasize a driver-centric approach, which addresses existing technology gaps in driver fatigue detection.

## 1.3 Scope

The scope of this project covers the development, testing, and evaluation of a computer vision-based drowsiness detection model. Key components include :

- **Model Development** : Implementing a Convolutional Neural Network (CNN) to accurately classify eye and mouth states from real-time image data. These classifications will enable reliable identification of closed eyes or yawning—critical indicators of drowsiness.
- **Real-Time Deployment Potential** : Optimizing the model architecture to minimize latency, with potential for integration into embedded vehicle systems.
- **Driver-Centric Focus** : The model will monitor the driver's physical state, making it adaptable to various driving environments, road types, and vehicle models.
- **Alert System** : Although primary development focuses on detection accuracy, we will outline alert system recommendations to ensure the notifications are clear and hard to ignore for drowsy drivers.

## 1.4 Organization

A structured Gantt chart outlines each phase of this project, helping to ensure timely and organized progression. The tasks are divided into four major phases : Implementing a standard solution, improving it for stage 2, further improvements for stage 3 and finally, Report and Presentation Preparation. Each task is designed with inter-dependencies in mind, ensuring a streamlined workflow and timely completion of project milestones.

# 2 Stage 1

## 2.1 Problem Formalization

### 2.1.1 Problem Statement

The primary objective here is to develop a machine-learning-powered system that monitors driver fatigue in real-time. This model will focus directly on the driver's condition by analyzing eye and mouth states and deduce if the driver is in a tired state accurately.

### 2.1.2 Understanding the Dataset

As we said before, our dataset consists of labeled images indicating whether a driver's eyes are "open" or "closed" and mouth states as "yawn" or "no yawn." The data is balanced : we have the same distribution between classes (726 for both closed and open and 723/725 for yawn and no yawn respectively) so we won't have an issue with biased model prediction or poor performance on minority class.

#### Dataset Overview :

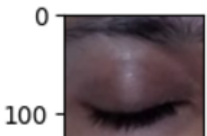
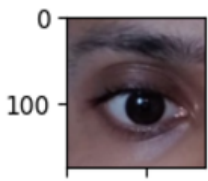
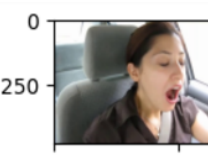

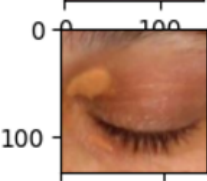
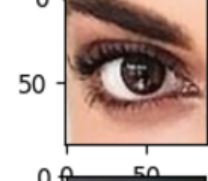
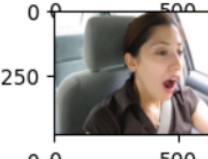



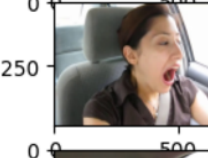

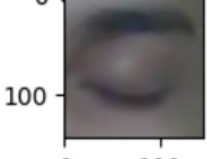
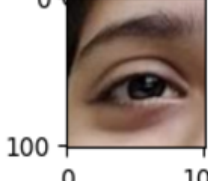


Eye Closed	Eye Opened	Mouth Yawn	Mouth no Yawn
			
			
			
			

TABLE 1 – Un tableau simple.

We notice that the images don't share the same format, we'll thus be tackling this issue.

## 2.2 Refining the Dataset

Our initial task involved meticulously cleaning and pre-processing the data. This process was crucial to ensure the integrity and usability of the dataset for machine learning models. Key steps included :

- To handle missing or corrupted values.

- To remove duplicates if any.
- To make sure that all pictures have the same size.

## 2.3 Evaluation Metrics

In addition to **accuracy**, we considered other metrics like **precision**, **recall**, and **F1 score** for a comprehensive assessment of our models. These metrics provided a more nuanced view of the model's performance because we need to be as accurate as possible.

## 2.4 Data cleaning and pre-processing

In the process of preparing our data for our machine learning project, our group followed a set of essential steps.

### Code 1 : Checking for missing or corrupted data Python

```
from PIL import Image
import os

def count_corrupted_images(dataset_path):
    corrupted_count = 0
    total_images = 0

    for root, _, files in os.walk(dataset_path):
        for file in files:
            file_path = os.path.join(root, file)
            total_images += 1
            try:
                with Image.open(file_path) as img:
                    img.verify()
            except Exception as e:
                corrupted_count += 1
                print(f"Corrupted: {file_path} - Error: {e}")

    print(f"\nTotal Images Checked: {total_images}")
    print(f"Total Corrupted Images: {corrupted_count}")
    return corrupted_count

count_corrupted_images(path+"/train/Closed")
count_corrupted_images(path+"/train/Open")
count_corrupted_images(path+"/train/yawn")
count_corrupted_images(path+"/train/no_yawn")
```

### Code 2 : Output of Code 1

Total Images Checked : 726

Total Corrupted Images : 0

Total Images Checked : 726

Total Corrupted Images : 0

Total Images Checked : 723

Total Corrupted Images : 0

Total Images Checked : 725

Total Corrupted Images : 0

We don't have any corrupted file in the dataset. Let's check for duplicates :

### Code 3 : Checking for duplicate data Python

```
import hashlib
import cv2
import os

def remove_duplicates(dataset_path):
    image_hashes = {}
    unique_images = []

    for root, _, files in os.walk(dataset_path):
        for file in files:
            file_path = os.path.join(root, file)

            img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

            img_bytes = img.tobytes()

            img_hash = hashlib.md5(img_bytes).hexdigest()

            if img_hash not in image_hashes:
                image_hashes[img_hash] = file_path
                unique_images.append(file_path)
            else:
                return(f"Duplicate found: {file_path}")

    return ("There are no duplicate images")

# Example usage:
# Assuming 'path' is your base dataset path
remove_duplicates(path + "/train/Closed")
remove_duplicates(path + "/train/Open")
remove_duplicates(path + "/train/yawn")
remove_duplicates(path + "/train/no_yawn")
```

### Code 4 : Output of Code 3

```
'There are no duplicate images'
```

As stated, the pictures are all unique, we can use all of them.  
Finally, we will use some pre-processing techniques such as :

- **Image Transformation** to make all the image have the same size and increases the accuracy.



#### Code 5 : Image transformation - resize images Python

```
def load_images(dataset_path, target_size=(64, 64)):
    images = []
    labels = []
    for label in os.listdir(dataset_path):
        class_path = os.path.join(dataset_path, label)
        if not os.path.isdir(class_path):
            continue
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                img_resized = cv2.resize(img, target_size)
                images.append(img_resized.flatten())
                labels.append(label)
    return np.array(images), np.array(labels)

images, labels = load_images(path+'/train')
```

- **Normalization** to create a consistency in pixel value ranges for better training and numerical stability, to make every pixel of the image having a value between 0 and 1 instead of 0 and 255.

#### Code 6 : Normalize images Python

```
X, y = load_images(path+'/train')
X = X/255.0
```