Machine Learning Project

ESILV A4 - DIA2

Élèves :
Nour AFFES
Lucas BLANCHET
Hugo BONNELL
Rayan HAMADEH

Enseignants :
Nédra MELLOULI
Zachary FAKIR

29 novembre 2024

Table des matières

1 Problem Overview

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 [1]). Unlike the effects of alcohol or drugs, drowsiness is harder to detect, creating a challenge for preventive measures. A 2019 AAA Foundation [2] 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** [3] : 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 [5] 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 Preparing the Data and the Analysis

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.
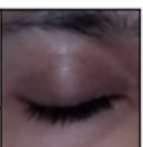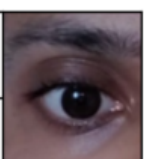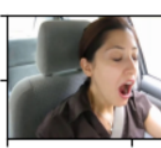
Dataset Overview :
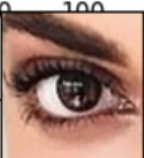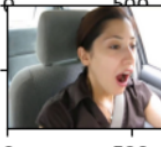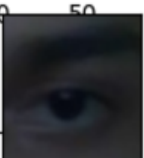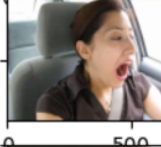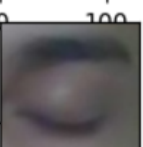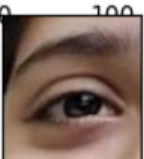
TABLE 1 – Examples of images per label.

Data Characteristics :

- Our dataset is made of images. Each image can be represented as a matrix. Each value in this matrix is an array with the RGB values of the corresponding pixel.

- The images in our dataset don't share the same dimensions. This may yield unexpected results after training models on this data. Thus, we'll have to handle image formatting before doing any train.

Model goals :

- **Primary objective :**
  - Being able to tell if a driver is showing tireness symptoms.

- **Secondary objectives :**
  - Ensure our model is robust.
  - Ensure the model can handle a variety of data inputs by not over-fitting.

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.

- We'll also use **Confusion Matrices** as a mean of better understanding the results of the models we're testing.

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`

```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 corrputed file in the dataset. Let's check for duplicates :

Code 3 : Checking for duplicate data Python

```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")


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 in Code 4, 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

```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 creates 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

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

3 Analysis

In this section, we evaluate and compare the performance of three machine learning classifiers—Support Vector Machine (SVM), Random Forest, and Logistic Regression—on the task of recognizing and classifying images into one of four categories : "eyes open," "eyes closed," "yawn," and "no yawn." Each classifier uses the same dataset of raw pixel features, enabling a direct comparison of their strengths and weaknesses. By examining key metrics such as precision, recall, F1-score, and overall accuracy, we aim to identify the most effective model for this task and highlight areas for potential improvement in future iterations. Dataset is split as : 20% for testing and 80% for training.

The hyperparameters retained in this document are the best we found after fine tuning.

3.1 SVM Classifier

Description

- **Objective :** The SVM aims to find the optimal hyperplane that separates the classes ("eyes opened", "eyes closed", "yawn", "no yawn") with the maximum margin in a high-dimensional feature space.
- **Input :** The SVM takes the image features as input. These features are the raw pixel values.
- **Output :** The classifier outputs a binary label for each input image.
- **Hyperparameters :**
  - Kernel : Specifies the kernel type to be used in the SVM algorithm. Here : 'linear', which means the model attempts to separate data using a linear hyperplane.
  - Random state : Fixed seed for the random number generator. We used 42.

Code and outputs

```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Train SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
# Predict on test data
y_pred_svm = svm_model.predict(X_test)

# Evaluate SVM performance
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm))
print(f"SVM Accuracy: {accuracy_score(y_test, y_pred_svm):.2f}")
```
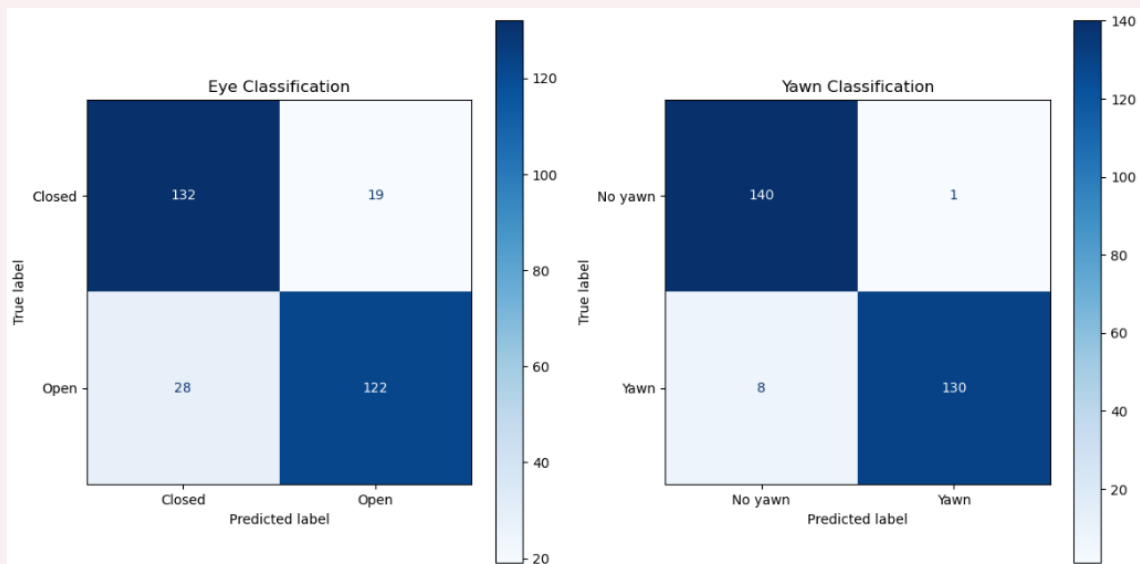
Code 7 : SVM Classifier Python

Code 8 : SVM Classification Report and Confusion Matrices

```
SVM Classification Report:
              precision    recall  f1-score   support

      Closed       0.82      0.87      0.85       151
        Open       0.87      0.81      0.84       150
     no_yawn       0.95      0.99      0.97       141
        yawn       0.99      0.94      0.97       138

    accuracy                          0.90       580
   macro avg       0.91      0.91      0.91       580
weighted avg       0.90      0.90      0.90       580

SVM Accuracy: 0.90
```


Interpretation

Overall, the model excels at distinguishing between "yawn" and "no yawn," achieving high precision, recall, and F1-Score. It demonstrates high accuracy and balanced performance across most classes. However, some challenges remain in the eye-related classes, with precision for "closed" at 82% and recall for "open" at 81%. These issues should be monitored in future iterations to determine whether they represent persistent difficulties.

The macro-average metrics—91% for precision, recall, and F1-Score—indicate that the model performs well across all classes without bias. Similarly, the weighted average confirms strong performance across the dataset, reflecting the balanced class distribution achieved during preprocessing.

3.2 Random Forest Classifier

Description

- **Objective :** The Random Forest classifier works by creating an ensemble of decision trees to classify images as "eyes opened" or "eyes closed." It combines predictions from multiple decision trees to improve accuracy and reduce the risk of overfitting. Each decision tree is trained on a random subset of the data and features (bagging and feature sampling) to ensure diversity.
- **Input :** The Random Forest takes the image features as input. These features are the raw pixel values. Each feature is used to make binary decisions at the nodes of the decision trees.
- **Output :** The classifier outputs a probability score or a class label :
  - **Probability Score :** For each image, the Random Forest outputs the proportion of trees voting for each class.
  - **Class Label :** The final prediction is the class that receives the majority vote from all decision trees.
- **Hyperparameters :**
  - n estimators : The number of decision trees in the forest. Here, it is set to 100.
  - Random state : Fixed seed for the random number generator. We used 42.
  - Max depth : By default set to None, tree expands until all leaves contain only one class or until the number of samples in a node is less than min sample split (by default 2).

Code and outputs

Code 9 : Random Forest Classifier Python

```python
from sklearn.ensemble import RandomForestClassifier
# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate Random Forest performance
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf):.2f}")
```
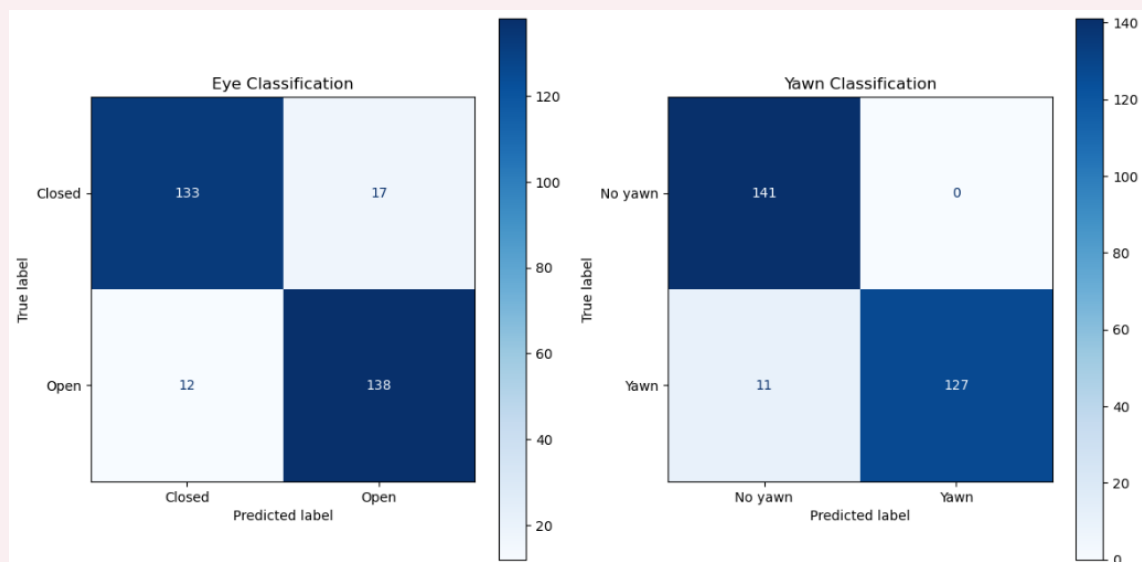
Code 10 : Random Forest Classification Report and Confusion Matrices

```
Random Forest Classification Report:
              precision    recall  f1-score   support

      Closed       0.92      0.88      0.90       151
        Open       0.89      0.92      0.90       150
     no_yawn       0.92      1.00      0.96       141
        yawn       1.00      0.92      0.96       138

    accuracy                           0.93       580
   macro avg       0.93      0.93      0.93       580
weighted avg       0.93      0.93      0.93       580

Random Forest Accuracy: 0.93
```


Interpretation

The accuracy of this model is better than the previous one. According to the confusion matrix, the number of errors in labelization has reduced significantly. The "open"/"closed" classes remain a bit difficult to labelize in comparison of the "yawn", but it had made some progress. Note that this model tend to label a little bit more of "yawn" even if it implies more false negatives for the "no yawn" part. F1 and recall scores underline that this model performs well across all classes without bias.

The Random Forest Classifier is slightly better for this dataset than the SVM with those hyperparameters. It has performed well for the eye with the "closed" and "open" class. But we need to also focus on the false negative result in the "yawn" class , and see if any particular features are causing confusion with "no yawn".

3.3 Logistic Regression

Description

- **Objective :** Model the probability that a given image belongs to one of the two classes. It uses a logistic (sigmoid) function to map the linear combination of input features to a probability score between 0 and 1, which can then be thresholded to assign a class label.

- **Input :** The Logisitc Regression takes the image features as input. These features are the raw pixel values.Regression takes a vector of these features and applies a linear transformation (i.e., weighted sum) to calculate a score.

- **Output :** Logistic Regression outputs a probability score between 0 and 1. This score represents the likelihood that the image belongs to a particular class (e.g., the probability that the image is "eyes opened").

- **Hyperparameters :**
  - max iter : Maximum number of iterations for the solver to converge. Here, it is set to 1000.
  - Random state : Fixed seed for the random number generator. We used 42.
  - Solver : lbfgs by default.

Code and outputs

Code 11 : Logistic Regression Python

```python
from sklearn.linear_model import LogisticRegression

# Train Logistic Regression model
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)

# Predict on test data
y_pred_lr = lr_model.predict(X_test)

# Evaluate Logistic Regression performance
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr))
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_lr):.2f}")
```
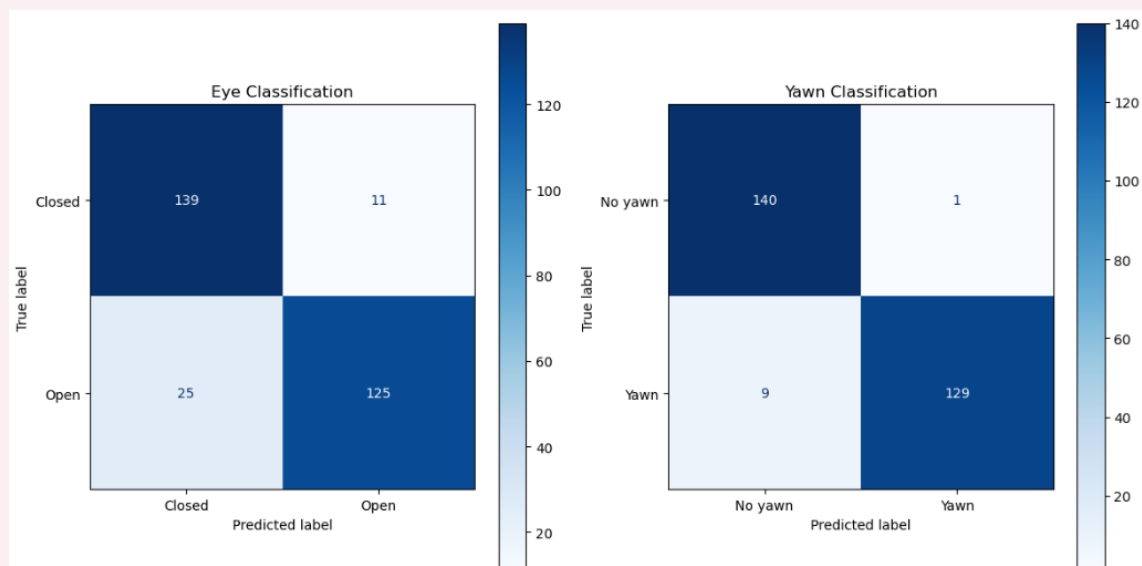
Code 12 : Logistic Regression Report and Confusion Matrices

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

      Closed       0.85      0.92      0.88       151
        Open       0.92      0.83      0.87       150
     no_yawn       0.93      0.99      0.96       141
        yawn       0.99      0.93      0.96       138

    accuracy                          0.92       580
   macro avg       0.92      0.92      0.92       580
weighted avg       0.92      0.92      0.92       580

Logistic Regression Accuracy: 0.92
```


Interpretation

The overall accuracy for Logistic Regression is high, matching the Random Forest model's performance and slightly better than the SVM model. We lost a lot of precision in the "close" class -almost the double of errors- but overall made some progress for the other class, gaining some precision on the "open" class and in the "no yawn". The macro average and the weighted average is almost the same as the other model. So, Logistic Regression offers competitive performance with some trade-offs compared to the other models.

4 Testing our models' Robustness

The problem with training on datasets found online, especially on Kaggle, is the similarity the data may exhibit. This can lead to models that overfit or underfit the data we provide.

To avoid this issue, we gathered what we referred to as "Real-Life Data." Specifically, we took pictures ourselves in the real world (as opposed to sourcing them online) to test our models. These images featured a variety of shapes, new faces, and different eye conditions for testing.

The results were promising out of 32 images we got :
- 97% of correctly labeled images by the SVM.
- 97% of correctly labeled images by the Random Forest.
- 100% of correctly labeled images by the Linear Regression.

Although this outcome can be partly attributed to the relatively small size of our Real-Life Dataset, it still highlights the robustness and consistency of our models.

Notes :
- We limited our tests to labeling images as "eyes open" or "eyes closed. This ensured the used pictures to be centered around the eyes of our volunteers, ensuring some kind of privacy regarding their identities.
- The "Real-Life Data" isn't available on the project's GitHub, or anywhere else, for privacy matters.

5 Going Deeper

(Convolutional) Neural Network ... Will be implemented in Stage 3!

6 Bibliographie

[1]   *Drowsy driving statistics and facts 2024.* URL : `https://www.bankrate.com/insurance/car/drowsy-driving-statistics/`.

[2]   *2019 Traffic-Safety-Culture-Index.* URL : `https://aaafoundation.org/wp-content/uploads/2020/06/2019-Traffic-Safety-Culture-Index.pdf`.

[3]   *What is Mercedes-Benz ATTENTION ASSIST ?* URL : `https://www.fjmercedes.com/mercedes-benz-attention-assist/`.

[4]   *How do I use the Driver Alert System* in my Ford ?* URL : `https://www.ford.com/support/how-tos/ford-technology/driver-assist-features/how-do-i-use-the-driver-alert-system/`.

[5]   *Dataset.* URL : `https://www.kaggle.com/code/adinishad/driver-drowsiness-using-keras/notebook`.