MASTER´S IN DATA SCIENCE & BUSINESS ANALYTICS

"Real-Time Drowsiness Detection Using YOLOv11 and PyTorch”

Master's Thesis prepared by: Andrés Dos Reis Ramírez
Master's Thesis Supervisor: Juan Manuel Moreno Lamparero

- Madrid, September 2024 -

TABLE OF CONTENTS

SUMMARY

This project successfully developed a real-time drowsiness detection system using the YOLOv11 model, trained on a custom dataset, to distinguish between drowsy and alert driver states. The primary objective was to create a foundation for future integration into driver safety systems, aiming to enhance safety and potentially reduce accidents caused by driver fatigue.

The dataset included approximately 3,600 labeled images, with additional photos of 12 individuals (friends and relatives) in both drowsy and alert conditions to ensure diversity and robustness. Images were annotated via RoboFlow and split into training, validation, and test sets. The model was trained using GPU acceleration on Google Colab, with data augmentation techniques applied to enhance its generalization capabilities.

The YOLOv11 model underwent training for 30 epochs using the AdamW optimizer, with early stopping at epoch 28 to prevent overfitting. The training progress was monitored through loss curves, analyzing the accuracy of bounding box predictions and classifications for both drowsiness and alertness. The model consistently improved during training, with minimal signs of overfitting and strong generalization to unseen data.

Key performance metrics, including precision, recall, and mean Average Precision (mAP), were used to evaluate the model. By the final epoch, the model achieved a precision of 99.7%, meaning it accurately identified drowsy drivers in nearly all cases, and a recall of 99.8%, ensuring most instances of drowsiness were detected. The mAP@50 score of 99.4% indicated high detection accuracy, though the mAP@50-95 score of 77.8% showed room for improvement in precise object localization.

In real-world testing, the model was evaluated on an unseen test set with a confidence threshold of 0.5. The confusion matrix revealed that the model correctly classified 503 out of 526 total predictions, resulting in an accuracy of 95.63%. The model accurately identified 241 instances of drowsy individuals and 261 instances of alert individuals, with minimal false positives (1 false positive) and false negatives (16 false negatives), as well as 6 missed detections of drowsy individuals.

The final phase of the project involved integrating the model with OpenCV for live video feeds, allowing real-time drowsiness detection via webcam. If drowsiness was detected, the system triggered visual alerts and an audible alarm after 3 seconds of continuous drowsiness. This

integration achieved the project's goal of real-time monitoring and feedback, demonstrating the system's potential for driver safety applications**.**

Overall, the project succeeded in building a highly accurate and efficient drowsiness detection system. Future improvements could focus on refining performance at higher IoU thresholds and increasing robustness in diverse real-world scenarios.

INTRODUCTION AND BACKGROUND

Drowsiness behind the wheel is currently one of the leading causes of traffic accidents worldwide. Research indicates that fatigue and drowsiness are involved in a significant proportion of fatal accidents, highlighting the urgent need to develop effective systems for their detection and prevention. Technology can play a crucial role in addressing this issue, particularly through the use of computer vision systems that monitor the driver's condition in real-time.

Advancements in computer vision and deep neural networks have enabled the creation of models capable of analyzing images in real-time with high accuracy. In this context, YOLO (You Only Look Once) models have stood out for their ability to perform fast and precise detections, making them ideal for critical applications like drowsiness detection.

The primary goal of this project is to develop a real-time drowsiness detection system using advanced computer vision and artificial intelligence techniques, specifically employing the YOLOv11 model implemented in PyTorch. This system is designed to determine whether a person is drowsy or awake based on images captured by a webcam, with the aim of providing an effective tool that canal be integrated into driver safety systems.

A custom dataset was built to train a YOLO model for drowsiness detection. The dataset included around 3,600 images sourced from the internet for training and 1,032 images for validation. Additionally, 12 individuals were personally photographed in both drowsy and awake states. These images were carefully annotated using the Roboflow platform, with bounding boxes highlighting relevant facial regions. The test set, which included 526 images from these personal captures, was used exclusively to evaluate the model. No individuals appeared in more than one subset, ensuring the model's performance was tested on new, unseen faces for proper generalization.

The YOLOv11 model was trained on Google Colab, leveraging cloud GPU processing power to handle the large volume of data and optimize the model's performance. Once trained, the model was able to accurately detect whether a person was drowsy or awake in the images. Subsequently, this model was implemented in a real-time application using OpenCV, enabling continuous analysis of the individual's state via a webcam.

PROJECT OBJECTIVES

As established before, the primary goal of this project is to develop a real-time drowsiness detection system using advanced computer vision and artificial intelligence techniques, specifically utilizing the YOLOv11 model implemented in PyTorch. To achieve this, the following specific objectives have been defined:

1. **Efficiently Generate a Custom and Diverse Dataset:**
    - Utilize an existing dataset of around 3,000 images sourced from the internet for training and approximately 900 images for validation. These images represent a diverse range of individuals in both drowsy and alert states.
    - To further enhance the dataset, images of 12 different individuals (friends and family) were captured, each photographed in both drowsy and alert states. These personal images were then distributed across the sets: 4 individuals were added to the training set, 2 to the validation set, and 6 to the test set. The test set consists entirely of these personally captured images.
    - Ensure the dataset contains a balanced representation of both states (drowsy and awake), to improve the effectiveness of the model during training and evaluation.
2. **Annotate and Prepare the Dataset for Model Training:**
    - Use the RoboFlow platform to annotate the captured images with bounding boxes identifying the face region as a drowsy or alert state.
    - Split the dataset into training, validation, and test subsets, ensuring a balanced distribution of images to optimize the model's performance during the training process.
3. **Train a Drowsiness Detection Model Using YOLOv11 and PyTorch:**
    - Export to Google Colab our custom dataset created in Roboflow platform.
    - Implement and train the YOLOv11 model on Google Colab, leveraging GPU processing power to handle the data volume and improve training efficiency.
    - Ensure that the trained model can accurately identify whether a person is drowsy or awake in the test images.
4. **Develop a Real-Time Detection Application:**
    - Load the trained model in a jupyter notebook to enable webcam access through OpenCV.

- With our model loaded, start performing real-time detections via a webcam.
- Ensure that the application can operate continuously, detecting a person's drowsiness or alertness quickly and accurately in real-time.

5. **Evaluate and Optimize System Performance:**
    - Conduct testing of the system under real-world conditions to assess its accuracy, detection speed, and robustness.
    - Identify potential improvements in the model or implementation to enhance the system's effectiveness in drowsiness detection and reduce possible false positives or negatives.
6. **Document and Analyze the Results:**
    - Meticulously document all procedures used, including dataset creation, model training, and real-time detection implementation.
    - Analyze the results obtained, evaluating the model's performance and proposing potential improvements or future research directions in the field of drowsiness detection using computer vision.

MATERIALS AND METHODS

This section describes the materials used and the methods implemented to develop the real-time drowsiness detection system. The process covers everything from dataset creation to the implementation and testing of the trained model.

1. Development Environment

Hardware:

- **CPU:** A personal computer equipped with a standard CPU was used for image capture.
- **Webcam:** An integrated webcam on the computer was used to capture the images that make up the dataset.
- **GPU:** Model training was conducted on Google Colab, utilizing the available GPU to accelerate the training process.
- **Inference Hardware**: The model was deployed on an Alienware X17 laptop with a NVIDIA's GeForce RTX 30 GPU to facilitate faster and more efficient real-time detections during inference.

Software:

- **Python:** The programming language used throughout the project's development.
- **Jupyter Notebooks:** An interactive development environment used for writing and executing Python code, including image capture.
- **OpenCV:** A library used for capturing images from the webcam, processing the images, and implementing real-time detection.
- **RoboFlow:** A platform used for annotating images and managing the dataset, including splitting it into training, validation, and test subsets.
- **PyTorch:** A deep learning framework used to implement and train the YOLOv11 model.
- **Google Colab:** A cloud-based computing environment used to train the model, leveraging the GPU's processing power.
- **YOLOv11 (Ultralytics)**: The object detection model used in the project.

2. Dataset Creation

Creating a custom dataset was a crucial step in this project, providing the necessary data to train a real-time drowsiness detection model. The dataset was generated using a combination of pre-existing internet-sourced images and a Python script in Jupyter Notebooks, enabling efficient image capture via a webcam. Below is a detailed explanation of the creation process and the specific characteristics of the captured images.

2.1 Image Collection and Subject Selection:

For the custom dataset, the images were collected from two main sources: approximately 3,900 high-quality, internet-sourced images, and 1,200 custom-captured images from 12 diverse individuals (friends and family). These individuals were carefully selected to ensure diversity across gender, age, and race, which is crucial for creating a representative and generalized dataset.

Diversity in the dataset is important because visual cues for drowsiness can vary widely due to differences in facial structure, skin tone, and expression. Including such variation helps the model learn to detect drowsiness across a broad range of facial characteristics, improving its robustness and real-world applicability.

While the majority of the images from the internet were of high quality, the custom-captured images added a layer of realism necessary for real-time detection, simulating more "everyday" conditions. This approach, commonly referred to as the *domain gap* problem in data science, addresses the challenge of relying solely on ideal, curated images. This problem arises when a model is trained on one type of data (e.g., high-quality, curated images) but is expected to perform on a different set of data (e.g., real-world, "normal" images). This gap can lead to reduced model performance because the distribution of features in the training set doesn't fully align with the distribution in the testing or application environment. By incorporating real-world, "normal" photos, the model is better equipped to generalize effectively during real-time detection.

2.2 Captured States: Drowsy and Awake

For this project, it was essential to capture images in two clearly differentiated states: drowsy and awake. These states were defined as follows:

- **Drowsy State:** Characterized by physical signs indicating tiredness or drowsiness. Facial features sought in this state included:
    - **Half-closed or fully closed eyes:** One of the most obvious indicators of drowsiness is difficulty keeping the eyes open. Subjects were instructed to keep their eyes half-closed or fully closed during image capture.
    - **Head tilted or dropped:** Another common sign of drowsiness is the inability to keep the head upright. Subjects were photographed while tilting their heads forward or sideways, simulating the loss of muscle control associated with drowsiness.
    - **Relaxed facial expression or yawning:** Drowsiness is often accompanied by relaxed facial muscles and yawning. Subjects were photographed while yawning or maintaining a relaxed facial expression.

- **Awake State:** Represents an optimal level of attention and wakefulness. Facial features defining this state included:
    - **Fully open eyes:** In contrast to the drowsy state, subjects kept their eyes fully open, reflecting a high level of alertness and attention.
    - **Upright head:** The neck and head posture is firm, with no tilting. Subjects kept their heads in a neutral or slightly elevated position, showing complete muscle control and no signs of fatigue.
    - **Active facial expression:** In this state, facial muscles are slightly tense, and subjects were asked to maintain an active expression, such as a slight smile or focused look, to simulate a natural state of alertness.

2.3 Custom Capture Procedure:

The Python script, executed in a Jupyter Notebooks environment, allowed for automatic image capture. The following steps were followed to capture each state:

- **Webcam Setup:** Image capture was done using the OpenCV library, which initiated the webcam, captured video frames in real-time, and enabled image visualization during capture. This real-time visualization was crucial to ensure that the captured images correctly reflected the desired state (drowsy or alert).
- **Automatic Image Capture:** The script was configured to capture images for each state. During the capture session, subjects were instructed to maintain the drowsy or alert state for a period of time while the script automatically captured images at short intervals.

2.4 Challenges and Solutions in Image Capture:

- **Natural Drowsy Expression vs. Simulated Expression:** Capturing authentic drowsy expressions posed a challenge, as subjects might have difficulty simulating real drowsiness. Simulated expressions could lack the subtle physiological signs of actual drowsiness, leading to less reliable data**.**
- **False Drowsy Indicators Due to Eye Closure**: During image capture, participants would occasionally blink or close their eyes for brief moments, which could mistakenly appear as a drowsy state in the dataset. This introduced noise into the dataset, as some "alert" images might inaccurately resemble drowsiness due to these unintended eye closures. Affected images were discarded and replaced with new ones from subsequent sessions

3. Selection and Explanation of the YOLO Model

3.1 What is YOLO?

YOLO, short for You Only Look Once, is a state-of-the-art object detection system that stands out for its ability to perform real-time detection. Unlike traditional object detection methods, which break the task into multiple stages or processes, YOLO tackles the problem in a single step, as the name suggests — it "looks" at the image once and makes all its predictions in one pass.

The YOLO approach revolutionized the field of object detection by framing it as a regression problem rather than a classification problem with additional steps. This enables it to be both fast and accurate, which makes it particularly suited for applications where speed is crucial, such as in self-driving cars, surveillance, and even gaming.

3.2 How YOLO Works in Detail

To understand YOLO, we need to break down how it processes an image and makes predictions.

3.2.1 Image Division into a Grid:

YOLO begins by dividing the input image into a grid. For example, in the original YOLO algorithm, the image is divided into a S×S grid (for example, a 7x7 grid). Each grid cell is then responsible for detecting objects whose center lies within that cell. This grid system allows YOLO to consider all parts of the image at once, avoiding the need to look at smaller regions in isolation. Let's say the image size is 448x448 pixels and we divide it into a 7x7 grid. Each grid cell is now responsible for a region of 64x64 pixels.

3.2.2 Bounding Boxes Prediction:

Each grid cell predicts a set number of **bounding boxes**, usually 2 or more. These bounding boxes enclose the objects in the image and are defined by their parameters:

- **Center coordinates (x, y)** of the bounding box relative to the grid cell.
- **Width and height** of the box, also relative to the image dimensions.
- **Confidence score**: This is a crucial component of YOLO. It reflects two things:
    1. The **confidence** that the box actually contains an object.
    2. The **accuracy** with which the box is likely to fit around the object.

So, each bounding box includes both location information and a confidence score.

3.2.3 Class Prediction:

In addition to predicting bounding boxes, each grid cell also predicts the **probability distribution of the classes** that the object may belong to (e.g., car, person, dog, etc.). This means that for each object detected, YOLO predicts what class it belongs to. If a grid cell contains an object, it assigns a probability score for each possible class based on the likelihood of that object being present.

For example, if YOLO is trained on the COCO dataset, which has 80 object classes, each grid cell might predict probabilities for each of these 80 classes.

3.2.4 Combining Predictions:

The next step is to combine the predictions from all the grid cells. Each grid cell makes predictions for multiple bounding boxes and class probabilities. YOLO gathers all these predictions and applies some filtering to generate the final output:

- **Non-Maximum Suppression (NMS)**: This step is essential in YOLO. Since multiple bounding boxes often overlap, YOLO applies a filtering technique called **Non-Maximum Suppression** to eliminate redundant boxes. It keeps only the boxes with the highest confidence scores and discards boxes with a high overlap (as measured by **IoU** — Intersection over Union).

By the end of this process, YOLO outputs a final set of bounding boxes and corresponding class labels that indicate what objects are detected in the image and where they are located.

3.2.5 Loss Function:

YOLO's training uses a **custom loss function** that ensures the model learns to predict both accurate bounding boxes and correct classes. This loss function is composed of three parts:

- **Localization loss**: Penalizes inaccuracies in predicting the location (coordinates) of the bounding boxes.
- **Confidence loss**: Penalizes wrong predictions about whether or not an object is present in a grid cell.
- **Classification loss**: Penalizes incorrect class predictions for detected objects.

This multi-part loss function allows YOLO to learn from both the spatial location and the type of objects present in the image.

3.3 Why YOLO is Fast and Efficient

One of YOLO's greatest strengths is its **speed**. Here's why:

1. **Unified Model**: Unlike traditional methods (e.g., Region-based Convolutional Neural Networks or R-CNN), which require multiple passes over the image (first generating region proposals, then classifying each region), YOLO combines these steps into one single operation. It processes the entire image in one forward pass of the neural network, thus requiring significantly less computation time.
2. **Global Context**: YOLO looks at the entire image at once, considering global information rather than just focusing on local patches. This enables it to understand the context of the image and often leads to better detection of larger objects or objects in complex backgrounds.
3. **Single Pass**: YOLO's architecture is based on a single feed-forward pass through the network, meaning that it doesn't need to repeatedly process different regions or parts of the image. This results in real-time object detection, where YOLO can process video frames at over 30 FPS (frames per second).

4. Dataset Annotation and Preparation in RoboFlow

4.1. Introduction to RoboFlow

RoboFlow is a comprehensive platform designed to facilitate the management and preparation of datasets for computer vision projects. This tool is widely used for image annotation, dataset management, and preparation for use in deep learning models such as YOLOv11. One of the key reasons for choosing RoboFlow for this project is that it is the recommended tool by Ultralytics, the company that developed YOLOv11, due to its direct and efficient integration with YOLO models and their training environment.

4.2. Image Annotation Process in RoboFlow

With the captured dataset, the next crucial step was image annotation, which involves identifying and marking the areas of interest in each image where signs of drowsiness or alertness are present. The annotation process in RoboFlow was carried out as follows:

- **Uploading the Dataset:** The set of images was uploaded to the RoboFlow platform. The RoboFlow interface allowed for easy management and visualization of all images at once.
- **Creating Annotation Labels:** Two main labels were created for annotation: "Drowsy" and "Awake". These labels were assigned based on the overall facial structure observed in each image, assessing the general state of the individual, such as whether they appeared **drowsy** or **awake**.
- **Manual Annotation:** Although RoboFlow offers AI-assisted automatic annotation tools, manual annotation was chosen for this project to ensure maximum accuracy. Each image was carefully reviewed and annotated by drawing bounding boxes around the face area that indicated drowsiness or alertness.
- **Review and Correction:** After the initial annotation, all images were reviewed to ensure that labels were applied correctly and that there were no errors in the placement of bounding boxes.

4.3. Preparing the Dataset for Training

Once annotation was completed, the dataset was prepared for training the YOLOv11 model:

- **Dataset Division:** RoboFlow allows for dividing the dataset into training, validation, and test subsets with just a few clicks. For this project, a split of 82% of images for training, 12% for validation, and 6% for testing was used. This division was essential to ensure that the YOLOv11 model could generalize well to new, unseen images during training.
- **Data Augmentation:** RoboFlow also provides tools for data augmentation, a process in which new images are generated from existing ones through transformations such as rotations, scaling, and brightness adjustments. These techniques help improve the model's robustness, allowing it to adapt to variations in image capture conditions. This data augmentation technique was applied to the dataset extending it to 7316 images in total for training. The data augmentation tools used where:
    - Saturation: Between -14% and +14%
    - Brightness: Between -15% and +15%

- **Dataset Export:** Finally, the annotated, divided and augmentated dataset was exported in a format compatible with YOLOv11. RoboFlow facilitates this process, allowing direct export in formats such as YOLO, Pascal VOC, COCO, among others. For this case the “download dataset” option was used. This works as a code snippet that is runned on Colab to download and process the dataset directly.
- For this case the “download dataset” option was used to then use it on Google Colab. This works as a code snippet that is runned on Colab to download and the dataset directly.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("drowsyproject").project("drowsiness_project")
version = project.version(4)
dataset = version.download("yolov11")
```

4.4. Advantages of Using RoboFlow

RoboFlow proved to be an extremely useful tool in this project for several reasons:

- **Direct Integration with YOLOv11:** As the tool recommended by Ultralytics, RoboFlow integrates perfectly with YOLOv11, facilitating both annotation and dataset export in a format ready for training.
- **Ease of Use:** RoboFlow’s intuitive interface allows users to upload, annotate, and manage large datasets without the need for additional tools or advanced programming knowledge.
- **Data Augmentation and Management:** The data augmentation capabilities and ease of dividing the dataset into balanced subsets ensure that the trained model is more robust and accurate.
- **Efficiency and Precision:** Combining manual annotation with AI-assisted tools provides complete control over dataset quality, reducing the time required for preparation and improving annotation accuracy.
-

5. Model Training

In this section, we describe the process of training our YOLOv11-based object detection model for detecting drowsiness and alertness. The goal was to optimize the model for high accuracy in real-time inference using our custom dataset of images. We trained the model using Google Colab with GPU acceleration, which allowed for efficient processing of the large dataset.

5.1 Data Preparation

Our training dataset consisted of a total of 3658 labeled images, evenly divided into two categories: drowsy and alert. These images were collected under various conditions to ensure diversity, incorporating factors such as different lighting environments, facial expressions, and head positions to simulate real-world variability.

After applying data augmentation techniques to enhance the dataset, the images were divided into training and validation sets as follows:

- **Training set:** 7316 images (82%)
- **Validation set:** 1032 images (12%)

5.2 Model Setup

A "*data.yaml*" file was created to specify the paths to our training, validation, and test sets. This file contained essential metadata, including the class labels (drowsy and alert) and the directory paths to the images and annotations. The following command was used to start the training process:

```
# Training
!yolo task=detect mode=train model=yolo11n.pt data={dataset.location}/data.yaml epochs=30 imgsz=640 patience=5
```

- **Task**: We set the task to 'detect' as our goal was object detection (drowsy or alert face detection).
- **Mode**: The mode is set to 'train', which indicates that the goal is to train the YOLO model on the dataset provided.
- **Model**: We used the yolov11n.pt weights to initialize the model.
- **Data**: The dataset was specified using the "data.yaml" file, which contained the paths to both training and validation datasets.

- **Epochs**: The model was trained for 30 epochs, balancing between training time and model performance.
- **Image Size**: We resized the input images to 640x640 pixels to maintain consistency during training and to optimize for both accuracy and speed.
- **Patience**: A patience of 5 was set to allow early stopping if the validation loss did not improve within 5 epochs.

For optimization, the AdamW optimizer was automatically selected by YOLOv11. Through it we guide the model in finding the best set of weights that minimize the error between the predicted and actual values, allowing the model to make better predictions over time. This optimizer provided adaptive learning rates, improving the convergence rate during training. For our training, the optimizer was initialized with a learning rate of 0.001667, which was selected automatically as well. This learning rate strikes a balance between making sufficiently large updates for faster convergence while preventing instability in the training process.

In general terms, YOLOv11 automatically chose the best optimizer, learning rate, and momentum based on the characteristics of the dataset and model performance. This automation allowed the optimizer to fine-tune these hyperparameters dynamically, leading to better convergence without manual tuning.

5.3 Training Process Overview

- **Number of Epochs:** The model was initially set to train for 30 epochs. An epoch represents one complete pass through the entire training dataset. However, a **patience** parameter of 5 was set, meaning that if the model showed no improvement for 5 consecutive epochs, the training would stop early to prevent overfitting. In this case, **EarlyStopping** triggered after 28 epochs, with the best model saved at epoch 23. This ensured the training process stopped once the model reached its optimal performance.
- **Training Time:** The total training process for 28 epochs took approximately **1.283 hours**. The use of a Tesla T4 GPU significantly accelerated the training compared to using a CPU, allowing the model to learn effectively within this time frame.

- **Loss Function**

  YOLOv11 utilizes a combination of loss functions to optimize the detection and classification task:

- **Bounding Box Loss** *("box_loss")* : This measures how accurate the predicted bounding box is in relation to the actual object location.

- **Classification Loss(***"cls_loss")*: This checks how well the model classifies the detected objects, in this case, identifying "drowsy" or "alert" states.

5.4 Training Loss Curves Analysis

The training process of the object detection model was monitored through the visualization of loss curves, as illustrated in ***Figure 1.*** These curves provide critical insight into the model's optimization behavior and its ability to generalize to unseen data. The training and validation losses, including the box loss, class loss, are plotted across the training epochs. A detailed analysis of each loss component is provided below.

5.4.1 Training Losses

The training losses represent how well the model fits the training data. They include both box loss (for bounding box accuracy) and class loss (for object classification accuracy):

- **Box Loss (train/box_loss):** The box loss, which indicates the model's performance in predicting object bounding boxes, steadily decreases from an initial value of approximately **1.1** to **0.71** by the 28th epoch. This consistent decline reflects the model's increasing proficiency in accurately locating objects in the training images. The smooth reduction over time shows the stability of the learning process and the optimization of bounding box parameters.
- **Class Loss (train/cls_loss):** The class loss, which measures how well the model is classifying objects within the bounding boxes, also shows a consistent decrease from around **1.62** in the first epoch to **0.27** by the final epoch. This clear downward trend demonstrates that the model is becoming increasingly adept at correctly classifying objects as either "Drowsy" or "Alert" as training progresses. The absence of major fluctuations suggests that the model did not encounter significant instability or overfitting during the training process.

5.4.2 Validation Losses

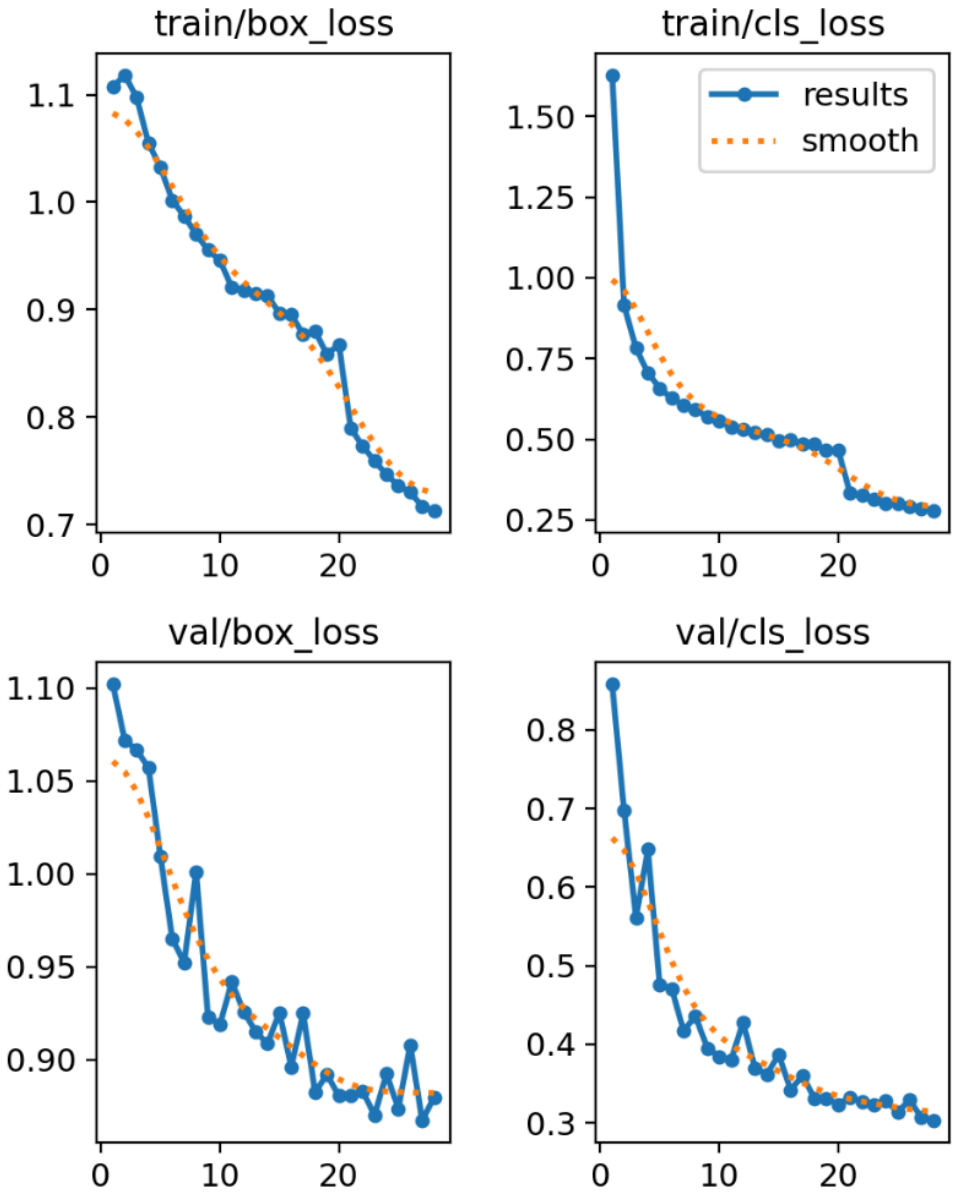
The validation losses assess the model's performance on unseen data and reflect how well it generalizes. Similar to the training losses, the validation losses include box and class loss:

- **Box Loss (val/box_loss):** The validation box loss, which tracks the accuracy of bounding box predictions on the validation set, decreases from **1.10** in the first epoch to **0.88** by the 28th epoch. This gradual reduction mirrors the trend in training box loss, suggesting that the model is effectively generalizing in terms of object localization. The

convergence between the training and validation box losses indicates that overfitting was successfully avoided during the training process.

- **Class Loss (val/cls_loss):** The class loss for the validation data starts at **0.86** and steadily drops to **0.30** by the final epoch. After the initial steep decline, the validation class loss stabilizes, indicating that the model has learned to generalize its classification performance effectively. Despite some minor fluctuations early in the training, the validation class loss aligns closely with the training class loss by the end, confirming that the model is not only learning effectively but also generalizing well on unseen data.

Figure 1



5.5 Performance Metrics Analysis in Training/Validation

5.5.1 Analysis of Model Performance in Drowsy Driver Detection

The evaluation of the drowsy driver detection model was conducted using several key performance metrics, specifically **precision**, **recall**, and **mean Average Precision (mAP)**. These metrics provide a comprehensive understanding of the model's ability to accurately identify drowsy and alert states in drivers.

- **Precision**
  This measures the accuracy of the model's positive predictions, specifically the proportion of true positive predictions relative to the total predicted positives. In the context of the drowsy driver detection model, precision indicates how many of the drivers identified as drowsy are actually drowsy.
  Results: The precision of the model improved over the course of training and achieved a high score of approximately **0.99685** by the final epoch. This signifies that the model correctly identified drowsy drivers nearly **99.7%** of the time, which indicates a low occurrence of false positives. This high precision rate is critical for minimizing unnecessary or incorrect alerts, fostering greater user trust in the detection system.
- **Recall**
  Also known as sensitivity, recall asseses the model's ability to identify all actual positive cases, reflecting the proportion of true positives relative to the total actual positives. In this context, recall reveals how effectively the model detects drowsy drivers.
  Results: The recall score reached approximately **0.99762** by the last epoch, demonstrating that the model successfully detected nearly **99.8%** of drowsy drivers. This high recall rate is crucial for the effectiveness of the system, as it ensures that almost all instances of drowsiness are detected, helping to reduce the risk of accidents caused by undetected drowsiness.

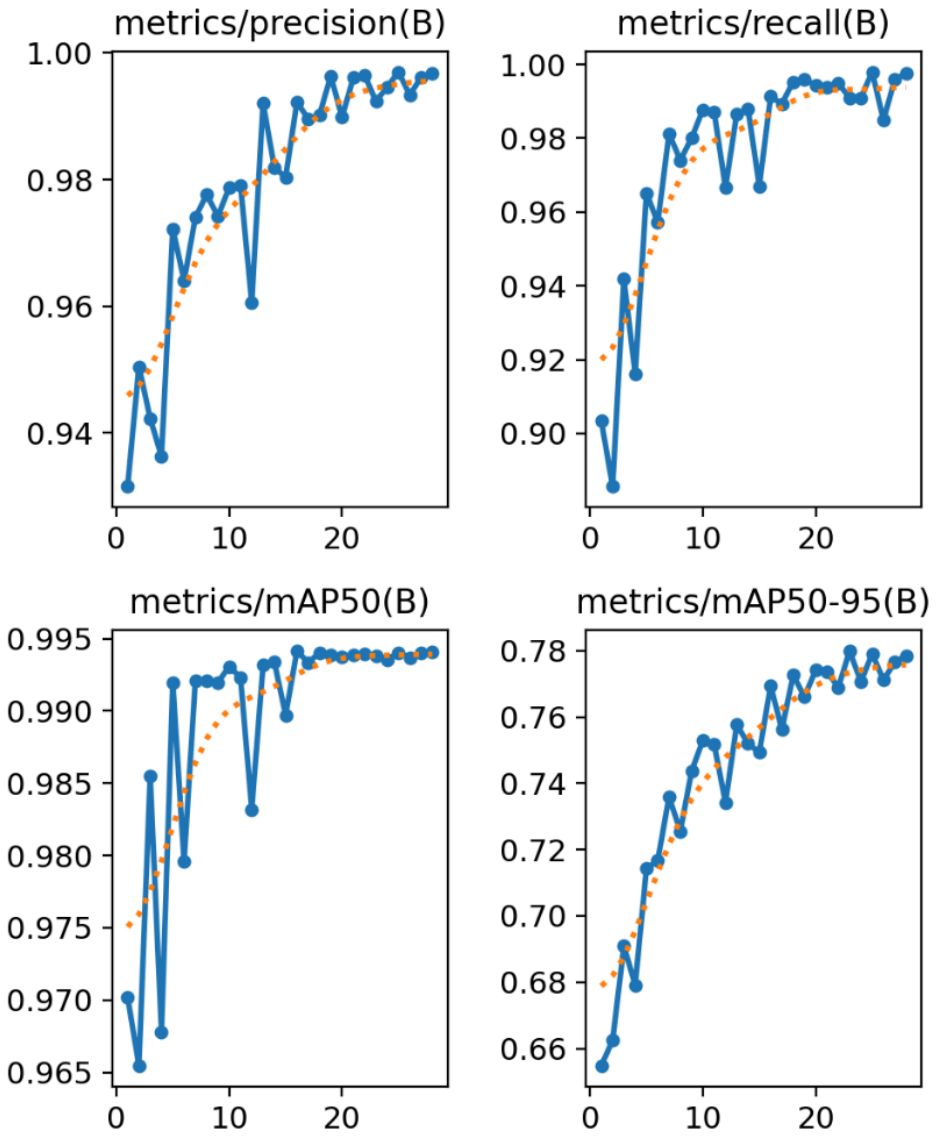
In conclusion, precision helps us understand how reliable the model is when it predicts drowsiness, while recall tells us how good the model is at not missing any drowsy drivers.

Mean Average Precision (mAP)

Mean Average Precision is a robust metric that provides a comprehensive evaluation of the model's performance across different levels of intersection over union (IoU) thresholds. Two specific mAP values were analyzed: **mAP@50** and **mAP@50-95**.

- **mAP@50:**
  The model achieved a **mAP@50** score of approximately **0.99405** by the final epoch. This indicates that the model is capable of maintaining high precision and recall when evaluated with an IoU threshold of 0.50, where detections with at least 50% overlap are considered correct. A score of **99.4%** suggests that the model is highly effective at identifying true positives with minimal errors, under moderate IoU thresholds.

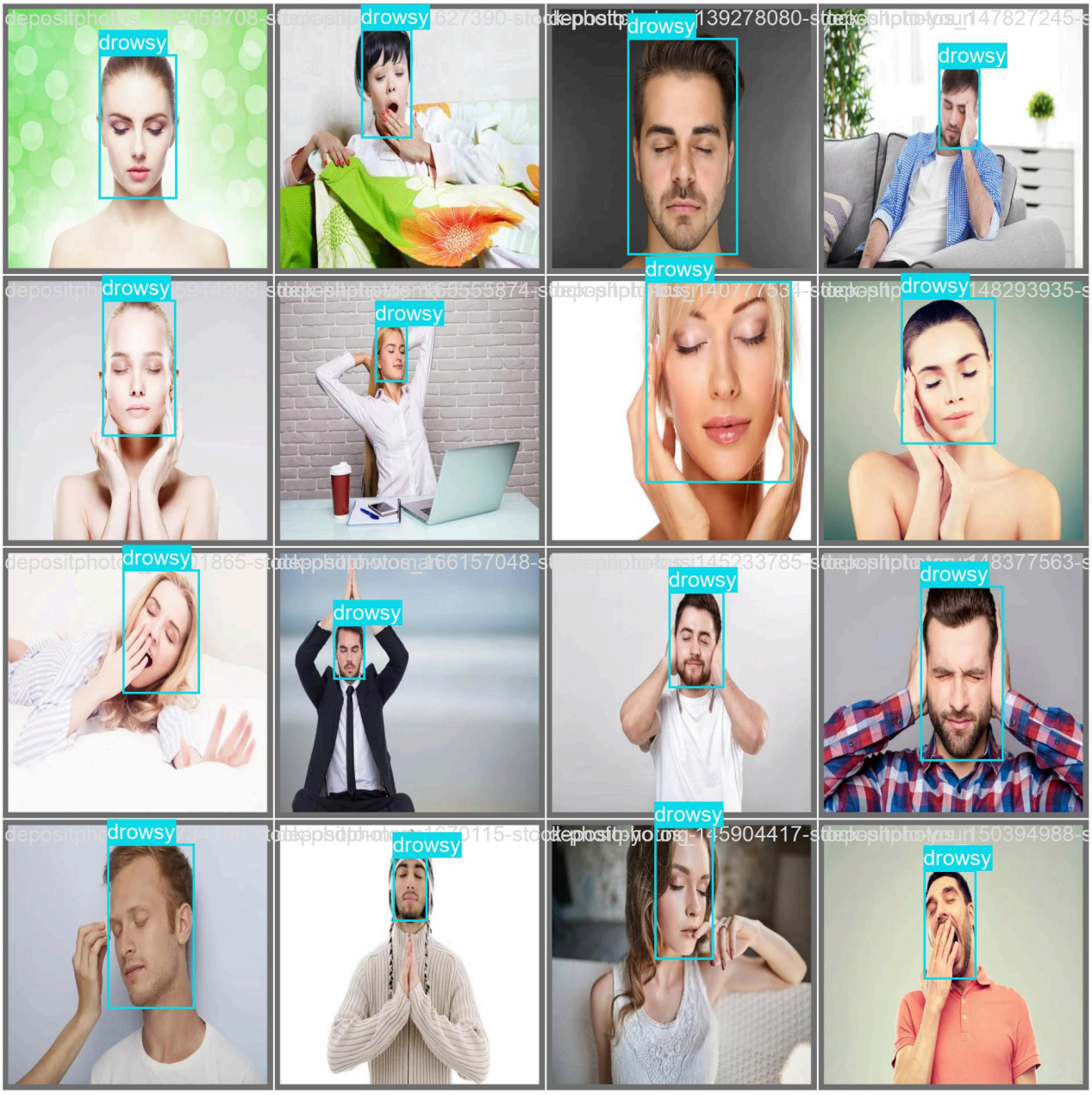
- **mAP@50-95:**
  The **mAP@50-95** score, which evaluates the model's performance across a wider range of IoU thresholds (from 0.50 to 0.95), was recorded at approximately **0.77852**. This reflects the model's ability to handle more challenging detection scenarios that require greater overlap (up to 95%) between predicted and actual bounding boxes. While the performance is still strong, the lower mAP@50-95 score compared to mAP@50 suggests that the model may face some difficulties with more refined detections at higher IoU thresholds. This provides an opportunity for future enhancements, such as optimizing the model's bounding box predictions.



5.6 Sample Detections

Figure 2

Figure 3


drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy
drowsy

6. Model Predictions and Evaluation on Test Set

In this section, we outline the process of using the trained YOLOv11 model to make predictions on unseen data (the test set). The goal is to evaluate the model's ability to generalize and accurately detect drowsiness and alertness in images it has never encountered before.

6.1 Prediction Code

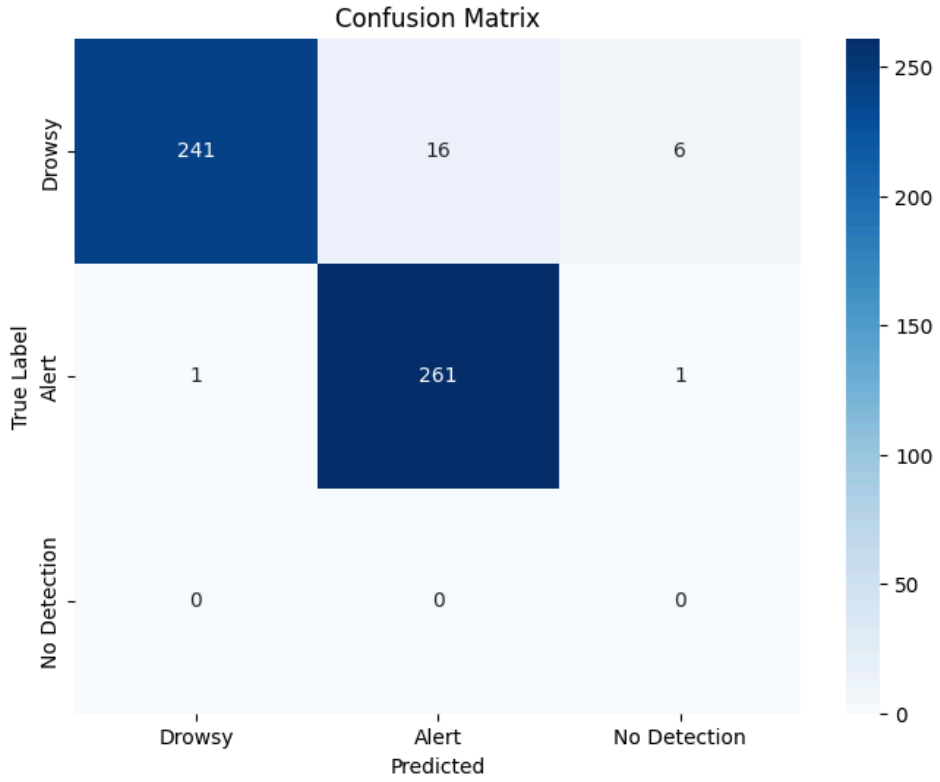
To run the trained model on the test images, we used the following command in Google Colab:

```
!yolo task=detect mode=predict model=/content/runs/detect/train/weights/best.pt conf=0.5
source={dataset.location}/test/images save=True save_txt=True
```

Here's a breakdown of each part of the command:

- **Task**: We set the task to 'detect', as the goal was object detection—specifically to detect faces categorized as either drowsy or alert.
- **Mode**: The mode is set to 'predict', which indicates that the trained YOLO model will be used to make predictions on a new, unseen test dataset.
- **Model**: We used the weights file *best.pt* , which was obtained after training the model. These weights represent the best-performing version of the model, which was saved at the epoch with the highest validation performance.
- **Source**: The source is specified as the directory containing the test images. These are the images on which we want to make predictions. The model reads the images from *{dataset.location}/test/images,* where the test set is stored.
- **Confidence Threshold**: The confidence threshold (*conf*) was set to 0.5. This parameter filters out predictions with a confidence level below 50%, which ensures that only the predictions the model is relatively confident about are considered.
- **Save**: The save parameter is set to *True*, which instructs the model to save the predicted bounding boxes, class labels (drowsy or alert), and confidence scores on the test images. The results are saved in a newly created directory.
- **Save_txt**: The *save_txt=True* option in YOLO saves the detection results (like bounding box coordinates, class labels, and confidence scores) in a *.txt* file for each image. This allows us to have the prediction details in a text format for further analysis or processing.

Figure 4



6.2 Analysis of the Confusion Matrix

The confusion matrix ***(Figure 4)*** provides valuable insight into how the YOLOv11 model performed in classifying images into three categories: "Drowsy," "Alert," and "No Detection." By understanding this matrix, we can assess how well the model is at correctly identifying the drowsiness state of a subject, where it excels, and where there may be room for improvement.

6.2.1 Overall Interpretation

The model's main task was to differentiate between individuals who were drowsy and those who were alert, and also account for situations where the model could not make a prediction (labeled as "No Detection"). The majority of the data falls into the two key classes, "Drowsy" and "Alert," which are critical for our application.

The matrix can be analyzed in terms of the number of correct predictions (where the model accurately detects the class) and incorrect predictions (where the model misclassifies or fails to detect a state). Let's break this down.

6.2.2 Correct Classifications (Drowsy and Alert)

- The model correctly identified **241** instances of drowsy individuals, meaning it was able to accurately detect when a person showed signs of drowsiness. This suggests the model performs well when individuals exhibit typical characteristics associated with drowsiness, such as closed eyes or slouched posture.
- Similarly, the model correctly detected **261** instances of alert individuals, successfully recognizing when a person was fully awake and engaged. This highlights the model's strong ability to detect alert states, which is just as crucial as identifying drowsiness in a driver monitoring system.

6.2.3 Misclassifications (False Positives and False Negatives)

The errors in the model's predictions are as important as its successes, as they reveal potential blind spots in its detection capabilities:

- **Drowsy Misclassified as Alert**: In **16** instances, the model misclassified a drowsy person as alert. This type of error, known as a false negative for drowsiness, is critical in our application because it represents missed detections of drowsiness. Such errors could occur when the signs of drowsiness are subtle, for example, when a person is on the verge of falling asleep but still exhibits some alert behavior, like slight eye movements or minor posture shifts.
- **Alert Misclassified as Drowsy**: In **1** case, the model misclassified an alert individual as drowsy. This type of error, though less common, is less detrimental than missing drowsiness, as a false positive for drowsiness (labeling an alert person as drowsy) might trigger unnecessary warnings but doesn't pose the same safety risk as missing a drowsy driver.

6.2.4 Missed Detections (No Detection)

Interestingly, **6** instances of drowsy individuals resulted in no detection at all, meaning the model couldn't make any prediction. This suggests that certain images may present challenging conditions, such as poor lighting, facial occlusion, or less distinct facial expressions that prevented the model from confidently identifying the state. No such missed detections were present for the "Alert" class, indicating the model is more consistent in detecting alert individuals.

The confusion matrix offers a comprehensive overview of how well the YOLOv8 model performed in classifying subjects as "Drowsy," "Alert," or "No Detection." By interpreting the key metrics—accuracy, precision, and recall—we can gain deeper insights into the strengths and potential areas of improvement for this model in a real-world application, such as driver monitoring systems.

6.2.5 Accuracy

Accuracy measures the overall proportion of correct predictions made by the model. Out of 526 total predictions, the model correctly classified 503 instances, resulting in an accuracy of **95.63%**. This high level of accuracy demonstrates that the model is effective at distinguishing between drowsy and alert states. It successfully identifies the correct class for the vast majority of images, suggesting that the model's general understanding of what characterizes a drowsy versus an alert state is robust. This strong overall performance is promising for applications where timely and accurate detection of drowsiness is critical for safety.

6.2.6 Precision (Drowsy)

Precision is an important metric when considering how often the model was correct when predicting a particular class—in this case, drowsiness. The precision for detecting drowsiness is **93.79%**, which means that nearly 94% of the time when the model classified a person as drowsy, the prediction was accurate. This high precision indicates that the model makes few false positives for drowsiness, i.e., it rarely labels an alert individual as drowsy.

This level of precision is particularly important in a driver monitoring system because an incorrect classification of an alert person as drowsy could lead to unnecessary interventions, such as triggering alarms or safety systems. The high precision suggests that when the model detects drowsiness, users can be confident that the detected state is indeed accurate, minimizing false alarms and unnecessary distractions.

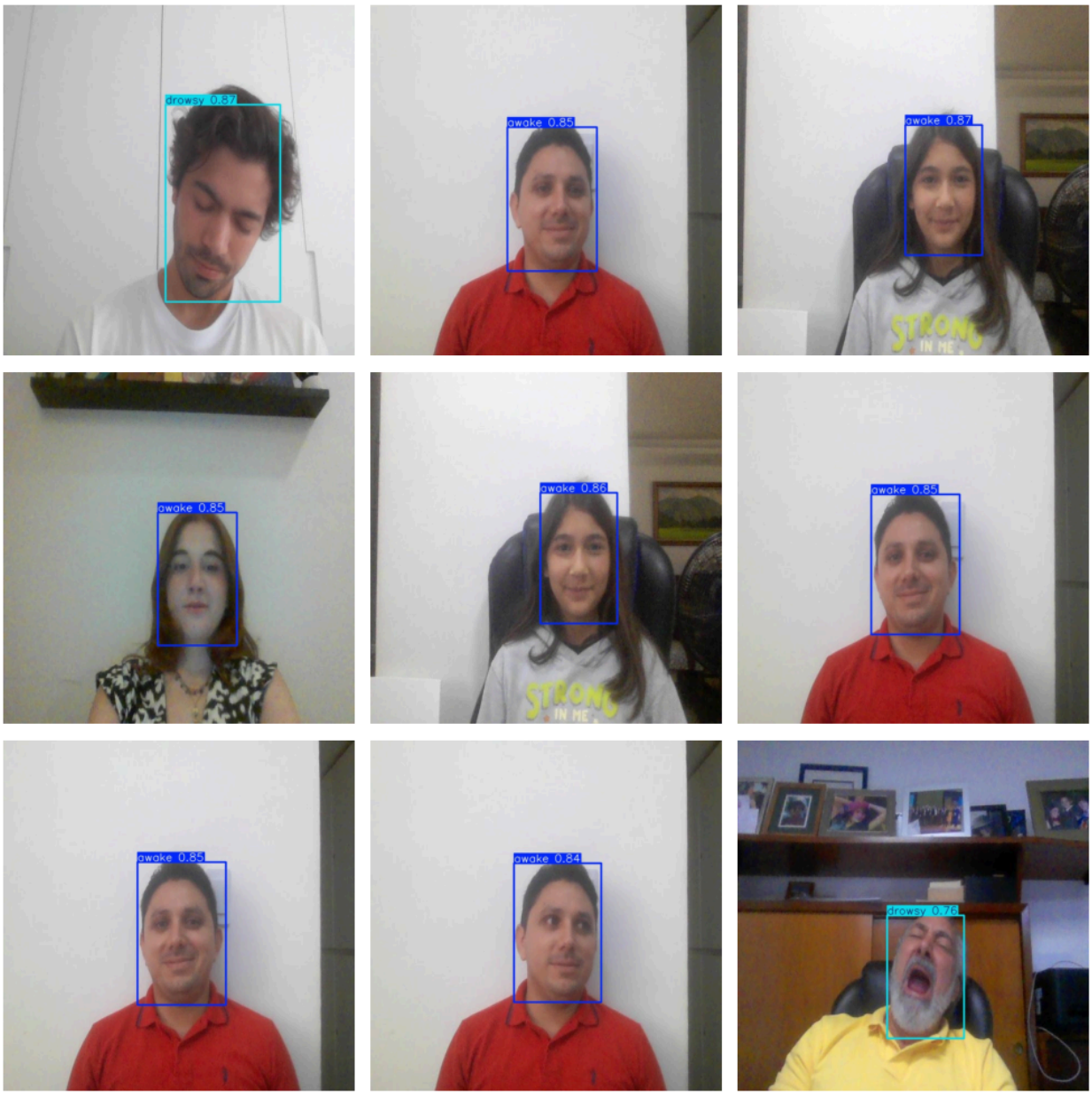
6.2.7 Recall (Drowsy)

Recall, also known as sensitivity, measures the ability of the model to detect all instances of a specific class—in this case, drowsiness. The recall for detecting drowsiness is **91.63%**, meaning that the model successfully identified about 92% of all actual drowsy cases. This is crucial for a system designed to ensure safety, as missing drowsy individuals (false negatives) can lead to critical situations, particularly in environments like driving where drowsiness detection is intended to prevent accidents.

The 92% recall shows that the model is adept at identifying drowsy individuals, but it also highlights that around 8% of actual drowsy cases went undetected. These missed detections could occur in cases where the visual signs of drowsiness were subtle or ambiguous, such as minimal eye closure or slight head tilts. Improving recall would involve enhancing the model's sensitivity to subtle signs of drowsiness, ensuring that even less obvious cases are detected. In safety-critical applications, even a small improvement in recall can have a significant impact.

6.2.8 Prediction Results Sample

Figure 5



7. Real-Time Implementation

The real-time drowsy driver detection system was developed by integrating the trained YOLOv11 model with OpenCV to run live inference on webcam video feeds. This allows for continuous monitoring of the driver's alertness and provides real-time feedback in case drowsiness is detected, finally showcasing the project´s main goal. The process is as follows:

- **Model Loading**: The fine-tuned YOLOv11 model (*best.pt*) was downloaded from google colab and loaded on Jupyter Notebooks to perform inference on the webcam feed. The model was set up to predict whether a person was in a drowsy or alert state based on facial and eye movements captured in real-time.
- **Webcam Integration**: The video feed from the webcam is processed frame by frame using OpenCV. Each frame is passed through the YOLOv11 model, which detects whether the driver is in an 'alert' or 'drowsy' state. The model runs inference in real-time to ensure timely detection of drowsiness.
- **Drowsiness Detection**: If the model identifies the driver as drowsy in any frame, an on-screen alert in the form of a blinking "Wake Up" message is displayed. This immediate feedback is designed to catch the driver’s attention before their drowsiness worsens.
- **Alarm System**: To prevent prolonged drowsiness from going unnoticed, an additional safety mechanism was implemented. If the system detects a drowsy state for a continuous period of more than 3 seconds, an alarm sound is triggered. This audio cue serves as a stronger warning to wake the driver and prevent potential accidents.
- **Continuous Monitoring**: The system constantly monitors the driver's state by analyzing each frame from the video feed. By combining both visual (on-screen alerts) and auditory (alarm sound) feedback, the system provides a robust solution to real-time drowsiness detection, ensuring that drowsy behavior is identified and addressed promptly.

REFERENCES

Statlect. (n.d.). *Domain shift*. Statlect. https://www.statlect.com/machine-learning/domain-shift

Sürüş Güvenliği. (2024, October). Sürüş Güvenliği Dataset [Open source dataset]. Roboflow. https://universe.roboflow.com/test-dfgra/surus-guvenligi

Ultralytics. (n.d.). *YOLOv8 documentation*. Retrieved September 3, 2024, from https://docs.ultralytics.com/

RoboFlow. (n.d.). *RoboFlow: The platform for computer vision projects*. Retrieved September 3, 2024, from https://roboflow.com/

Rajeev, A. (2023, March 5). *Driver drowsiness detection using OpenCV and YOLOv5*. Medium. https://medium.com/@aishwaryarajeev_68039/driver-drowsiness-detection-using-opencv-and-yolov5-5e4180b36744

OpenCV. (n.d.). *OpenCV: Open source computer vision*. Retrieved September 3, 2024, from https://opencv.org/

Ultralytics. (n.d.). *Ultralytics GitHub repository*. Retrieved September 3, 2024, from https://github.com/ultralytics/ultralytics

PyTorch. (n.d.). *PyTorch*. Retrieved September 3, 2024, from https://pytorch.org/

Nochnack, N. (2023). Drowsiness detection tutorial. GitHub. https://github.com/nicknochnack/YOLO-Drowsiness-Detection/blob/main/Drowsiness%20Detection%20Tutorial.ipynb

Vartak, P., Patil, R., & Jha, S. (2024). *Driver drowsiness detection system using YOLO V5*. EasyChair. https://easychair.org/smart-slide/slide/GLGd

Jain, A. (2021, May 20). *Data augmentation*. Medium. https://medium.com/@abhishekjainindore24/data-augmentation-00c72f5f4c54

PyTorch. (n.d.). *Transforms*. PyTorch. https://pytorch.org/vision/stable/transforms.html

Suhedaras. (2022, August 10). *Drowsiness detection with YOLOv5*. GitHub. https://github.com/suhedaras/Drowsiness-Detection-with-YoloV5