

**Business Intelligence II**

Project Work

[**UNDEF-03**]

[Real Time Object Detection System for Cocktail Bar Logging System]

Author: Horacio Serrano Leon

Matriculation no.: 3821045

Submission date: 27.06.2025

Examiner: Prof. Dr. Bernd Knobloch

# Project Profile

[Please fill in the form to profile your project. – **ONE PAGE ONLY!!**]

|  |  |  |
| --- | --- | --- |
| Project Category |  | Hands-on |
| USE CASE |  | **Real-time detection of popular cocktails.** |
| (cf. User Story) |  | As a **data analyst**  I want to **deploy a real-time object detection system** to **create a record of popular cocktails in a bar, along with the bottles used to create them.** |
| Solution |  | Develop a deep learning model to execute out Real-Time Object Detection, using low-level assembly code and simple hardware. |
| (cf. Task Model) |  | Description of Task: Train a deep learning model using YOLOv8 and load the best weights into the code developed for the application. |
|  |  | **Object of study** [What you designed, processed, optimised, analysed...] |
|  |  | Most popular drinks and alcoholic products are used as ingredients. |
|  |  | **Objectives:** [What you wanted to achieve about the object of study.] |
|  |  | ***Functional Objectives:*** *The system must be able to detect and objects with minimal error*  ***Non-functional Objectives:*** *Analyze the correlation between training metrics and detection results.* |
|  |  | **Pre-events / Triggers:** [What initiated or motivated task execution] |
|  |  | The detection is triggered by an object entering the ROI |
|  |  | **Method / Approach:** [What procedures/techniques etc. you used] |
|  |  | Training with pre-labelled datasets and deployment with the Model. |
|  |  | **Resources:** [What was required to execute the task, e.g. tools etc.] |
|  |  | Computer Desktop, microcomputer, USB Camera, datasets, bottles, and cocktail images. |
|  |  | **Post-events / Consequences:** [What happened after executing the task] |
|  |  | Once the object is detected, the system classifies it and records the information. |

Contents

[List of Figures I](#_Toc201872354)

[List of Tables III](#_Toc201872355)

[List of Submitted Items IV](#_Toc201872356)

[1 Context and Motivation 1](#_Toc201872357)

[2 Task 4](#_Toc201872358)

[3 Solution Approach 5](#_Toc201872359)

[4 Resources 13](#_Toc201872360)

[5 Prerequisites 26](#_Toc201872361)

[6 Evaluation 27](#_Toc201872362)

[7 Conclusion 48](#_Toc201872363)

[Appendix 49](#_Toc201872364)

[Bibliography 53](#_Toc201872365)

[Affidavit 55](#_Toc201872366)

# List of Figures

[Figure 1 Specific Objects vs Generic Objects 1](#_Toc201784243)

[Figure 2 Different Types of Object Classification 2](#_Toc201784244)

[Figure 3 CNN Architecture 2](#_Toc201784245)

[Figure 4 YOLO Architecture 3](#_Toc201784246)

[Figure 5 Different types of bottles 8](#_Toc201784247)

[Figure 6 Captain Morgan Rum Bottle 8](#_Toc201784248)

[Figure 7 Cap Types 8](#_Toc201784249)

[Figure 8 Empty Black Label and Full SKYY bottle 8](#_Toc201784250)

[Figure 9 Cocktail Glasses 9](#_Toc201784251)

[Figure 10 Cocktail Garnishes 9](#_Toc201784252)

[Figure 11 Dataset I Structure 10](#_Toc201784253)

[Figure 12 Dataset II Structure 10](#_Toc201784254)

[Figure 13 Labelling with Bounding Box example 1 11](#_Toc201784255)

[Figure 14 Labelling with Bounding Box example 2 11](#_Toc201784256)

[Figure 15 Dataset I data.yaml 11](#_Toc201784257)

[Figure 16 Dataset II data.yaml 11](#_Toc201784258)

[Figure 17 Windows PowerShell framework for Windows 15](#_Toc201784259)

[Figure 18 System Flowchart 16](#_Toc201784260)

[Figure 19 Example of detection with ROI 18](#_Toc201784261)

[Figure 20 Desktop Computer Use for Training Process 20](#_Toc201784262)

[Figure 21 NVIDIA Jetson Orin Nano 21](#_Toc201784263)

[Figure 22 USB Camera 22](#_Toc201784264)

[Figure 23 Basic Team Structure 24](#_Toc201784265)

[Figure 24 Example of real real-size object 25](#_Toc201784266)

[Figure 25 Example of printed image 25](#_Toc201784267)

[Figure 26 Developing Pipeline 26](#_Toc201784268)

[Figure 27 Training Results 30](#_Toc201784269)

[Figure 28 Training Results 32](#_Toc201784270)

[Figure 29 Model I Confusion Matrix 37](#_Toc201784271)

[Figure 30 TP with limitations 38](#_Toc201784272)

[Figure 31 Example of FP (Left) and FN (Right) 39](#_Toc201784273)

[Figure 32 Example of multiple TP (Right) with instance of FN (Left) 40](#_Toc201784274)

[Figure 33 Model II Confusion Matrix 41](#_Toc201784275)

[Figure 34 Experimental set-up for simple test 43](#_Toc201784276)

[Figure 35 Example of detection with upward position 43](#_Toc201784277)

[Figure 36 Example of simple test detection 44](#_Toc201784278)

[Figure 37 Experimental set-up for advanced test 46](#_Toc201784279)

[Figure 38 Example of advanced test with cocktail recipes 43](#_Toc201784280)

[Figure 39 Source code I 49](#_Toc201784281)

[Figure 40 Source code II 50](#_Toc201784282)

[Figure 41 Source code III 51](#_Toc201784283)

[Figure 42 Source code IV 52](#_Toc201784284)

# List of Tables

[Table 1 System Description 16](#_Toc201564957)

[Table 2 Training Parameters 27](#_Toc201564958)

[Table 3 Model I vs Model II Comparison 31](#_Toc201564959)

[Table 4 Validation Parameters 32](#_Toc201564960)

[Table 5 Detection Parameters 33](#_Toc201564961)

[Table 6 Exporting Parameters 33](#_Toc201564962)

Table 7 Model I Validation Metrics……………………………………………………………………………………………………34[Please list all tables, figures, charts, screenshots, etc. included in this report.]

Table 8 Model I Class Detection………………………………………………………………………………………………………...36

Table 9 Model II Validation Metrics……………………………………………………………………………………………………37

Table 10 Modell I Class Detection. ……………………………………………………………………………………………………39

Table 11 Model I vs. Model II Comparison…………………………………………………………………………………………39

# List of Submitted Items

1. [Please list all items/documents/materials/files etc. submitted together with this report.]

1. Project GitHub available at: https://github.com/hbo-bomb/Cocktail-Log-BI\_II

2. Model I Dataset available at: https://universe.roboflow.com/hailid3111-gmail-com/www-nthwg

3. Model II Dataset available at: https://universe.roboflow.com/new-workspace-gfhju/geonui

4. NVIDIA Jetson Orin Nano documentation available at: https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit & https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2

[If you have set up a tooling / platform / architecture, please also submit a description on how to set it up – like a recipe for installation, so that your work can be reproduced by others.]

# Context and Motivation

Cocktail preparation requires a substantial amount of alcohol with varying quality levels and price ranges. The procurement of these items as ingredients for preparing alcoholic beverages is essential for both small and large bar owners, who must balance the acquisition of the best liquor brands available by keeping costs relatively stable. Therefore, it is vital to have an inventory that reflects the customer’s tendencies towards the most consumed and demanded drinks. This project aims to utilize Real-Time Object Detection with Deep Learning to create a current record of the most frequently used bottles and consumed drinks [1], [2], [3]. This information is used to update and adjust inventories according to popular demand and shift dynamics, ensuring the most efficient experience for both staff and customers alike [1], [2], [3].

Before continuing the task description, some key concepts must be explained:

**Deep Learning:** A subfield of Machine Learning and Artificial Intelligence (AI) that uses multi-layered neural networks to learn patterns from data automatically [4],[5]. In image processing, deep learning enables computers to extract features and make decisions autonomously, making it ideal for tasks such as object recognition [4],[5].

**Real-Time Object Detection:** A computer vision technique used to identify and localize objects in images or video frames as they are being captured [1], [6]. It enables immediate recognition, which is essential for dynamic environments such as surveillance, autonomous driving, and monitoring purposes, including project proposals and drink preparation in a bar [1], [6]. Figure 1 displays how this technique distinguishes between objects, and Figure 2 shows how it classifies them.

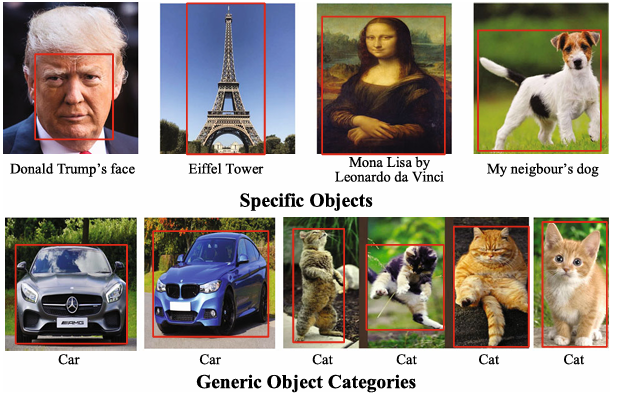


Figure 1 Specific Objects vs Generic Objects. Adapted from [2].

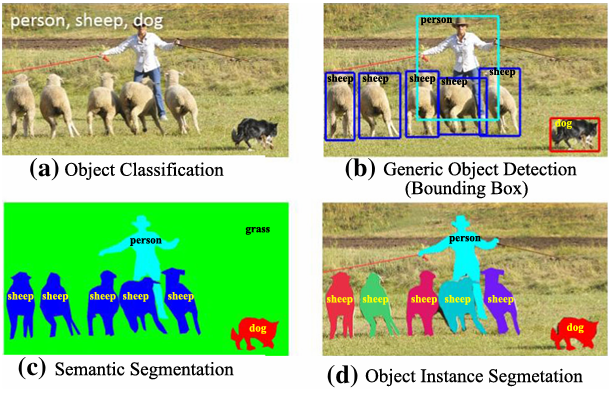


Figure 2 Different Types of Object Classification. Adapted from [7].

**Convolutional Neural Networks (CNNs):** A class of deep neural networks particularly effective at analyzing visual data [8], [9]. CNNs automatically detect features like edges, shapes, and patterns by applying filters (also known as kernels) through multiple layers [8], [9]. Figure 3 shows a typical example of a CNN architecture. They form the core architecture behind YOLO and most other image-related deep learning models [8], [9].

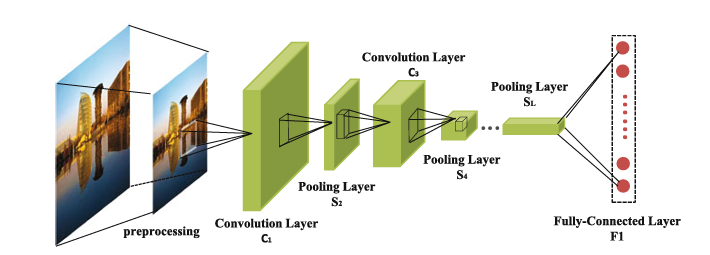


Figure 3 CNN Architecture. Adapted from [8].

**YOLO (You Only Look Once):** A fast and efficient deep learning algorithm for real-time object detection [2], [10]. YOLO divides an input image into a grid and predicts bounding boxes and class labels in one pass through the network [2], [10]. Its single-shot design makes it significantly faster than traditional two-stage detectors [2], [10]. In this project, YOLOv8 is utilized for enhanced accuracy, improved small-object detection, and simplified deployment. Figure 4 shows a simple example of YOLO architecture.

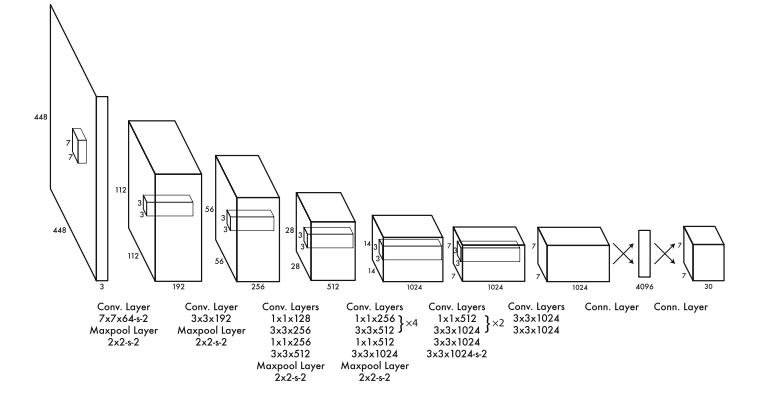


Figure 4 YOLO Architecture. Adapted from [10].

Although newer versions, such as YOLOv9 and YOLOv10, have emerged, YOLOv8 remains the optimal choice for this project due to its superior performance, usability, and compatibility [2], [11], [12], [13]. Other reasons include:

* Balanced Performance: YOLOv8 strikes a substantial trade-off between accuracy and inference speed, making it ideal for real-time applications such as live drink logging in a bar [2], [11], [12].
* Anchor-Free Architecture: It eliminates the need for predefined anchor boxes, thereby improving performance in detecting small and overlapping objects, which are common in cluttered bar environments [14], [15].
* Decoupled Detection Heads: Enhances detection accuracy by separating object classification and localization processes [16].
* Flexible Deployment: Supports ONNX and PyTorch formats, enabling use across various platforms, including Windows systems and edge devices, such as the NVIDIA Jetson Orin Nano [17], [18], [19].
* Developer-Friendly Tools: Comes with an intuitive CLI and Python API, accelerating prototyping and deployment [20], [21].
* Proven Track Record: Widely adopted and maintained by Ultralytics, with strong documentation and community support [2].

# Task

The objective of this project is to develop a functional, upgradable, user-friendly dynamic system that utilizes low-level assembly code and affordable devices to detect and classify liquor bottles, creating records that aid in inventory management for future procurement and the preparation of cocktails derived from these items. The determination of the cocktail´s popularity will be an additional objective of the system's outcome, as analysed by the current record produced.

## Object of study

The cocktail´s popularity and ingredients used in the mixing process are determined by a system that utilizes a low-assemble, single Python script to load the best-performing YOLOv8 model trained on a dataset comprising images of alcohol bottles. The system´s interface will be a USB connection to a device where the script is being run, allowing it to process the images in real-time and compare them with the results obtained from the training. Recurrent matches are interpreted as instances by the system and counted to provide a total of bottles used and what cocktails were prepared. The system itself serves as a platform tool that enables automated data collection in a dynamic environment, combining key aspects of deep learning, image recognition, and low-cost deployment.

## The Functional Objective

The system must detect the bottles and correctly classify them with the highest possible confidence threshold, while maintaining a low margin of error. This is essential to avoid false positives, such as a bottle being incorrectly logged under the wrong class, and false negatives, where the bottle is not counted despite being presented.

To achieve this goal, the YOLOv8 model is trained using labeled datasets of alcohol bottles and integrated into a Python script capable of processing real-time video from a USB-connected camera. The script includes recipes for the most popular cocktails, to serve as guidance for arranging the given sequences of bottles. These sequences will allow the function for producing records of the drinks being prepared. Additionally, the script also handles session control, ensuring that object detections within the region of interest (ROI) are counted only when relevant, thereby contributing to the overall accuracy and usability of the system in a real-world setting.

## Non-Functional Objective

The system must perform in real-time with optimal performance and low latency, both on advanced desktop PCs and on devices such as microcomputers. It should also be able to run in various environments, including Windows and Linux. Additionally, the system's requirements should be balanced to allow low resource consumption and remain stable under different conditions found in realistic scenarios.

Detection must be reliable enough to reduce the number of false positives and false negatives, even if bottles are moved quickly or partially hidden. The results, including logs and cocktail counts, should be clearly written and easy to understand so that they can be used later for analysis or inventory decisions.

Another important goal is to analyze how the training metrics, as described in Section 6, relate to what the system detects in real-time. This assessment will enable further improvements to the system.

# Solution Approach

The YOLOv8 model is trained using pre-labelled datasets procured from computer vision platforms. This training is carried out in a Windows environment. Based on the resulting metrics, the model's effectiveness is assessed and subsequently tested in two ways: first, using digital images on a desktop PC, and second, with real photographs and physical bottles. Once the model achieves reliable results, it’s converted into a format compatible with other platforms, allowing it to be deployed on a microcomputer running Linux.

## Concept of Solution

To achieve the project's goal, it is necessary to obtain several images that are correctly labeled for YOLOv8 training. These images are sourced from Roboflow, a widely used platform for computer vision and real-time object detection projects. The images are then used to train the model, which is subsequently deployed in the Python script developed to streamline the entire process.

The reason for using pre-labeled datasets is due to the project's constraints. Creating large datasets with high image quality and proper labeling requires significant effort, with labeling being especially time-consuming. However, if training metrics and testing results are not satisfactory, creating a smaller, custom dataset is considered to improve the model’s performance before deployment.

Once the model is trained, it can be tested across different platforms and environments to evaluate how well the training metrics align with real-world performance.

The motivation for creating this type of real-time logging system is to support staff in bars or clubs by identifying trends that need to be quantified, enabling them to manage inventory and pricing more effectively. The dynamic and often chaotic nature of these environments leaves little time for owners or bartenders to accurately track consumption. Relying on manual counting in such conditions can lead to costly errors over time.

By utilizing deep learning, the system can provide quick and reliable feedback on customer preferences, helping to optimize inventory planning. This solution is designed for real scenarios where bottles are constantly in use and fast service is essential. It could also be adapted to other contexts where automatic detection and object logging are proper.

## Analytical Problem

The detections must follow a particular criterion that allows the system to focus on the most essential features of the images in the dataset, so that the results presented after the YOLOv8 Model´s deployment can coincide with the actual events that are happening in the real environment. Additionally, the training metrics must give a favourable forecast of what the test results can be, even before an actual deployment of the Model takes place. Therefore, it is crucial to realize which features must be prioritized when creating the dataset.

### Business Problem (Purpose of analysis)

Given that the software used for the system´s applications is open source and can be adapted to operate on multiple platforms and environments, the cost problems are more related to the data itself. Users don´t have time to learn how to train deep learning models and the datasets necessary for this task. Hence, the service provided with this tool should specialize in data customization at a price that fits the customer's requirements. This price must be set in accordance with the amount of necessary data, as well as the subsequent tasks related to the training (gathering, splitting, labeling, etc.), and the size of the operation whether is for a big establishment that offers expensive products or a smaller bar/club with a relative lesser demand. This last factor is especially important to assess a more fitting training process that will eventually be based on the data being sold.

Furthermore, it is also essential to determine how to market the system without giving the false impression of a fully automated inventory tool; instead, to emphasize that the software is a smart assistance tool that requires the user to familiarize themselves with its inner workings to get the most out of it in an efficient way.

### Analytical Question(s) (What do you need to know to solve the business problem?)

To create datasets that can fulfill the previously mentioned requirements in Section 3.2.1, the most representative object features must be prioritized. The selection criteria should emphasize attributes that can be used to optimize the training process and simplify real-time detections. Additionally, these attributes must also facilitate object classification.

While liquor bottles are a ubiquitous everyday object found in places like supermarkets, convenience stores, and, for our study, bars and similar establishments, they possess a lot of attributes that can lead to many possible classifications, such as:

* Shape
* Size
* Material
* Alcohol type
* Liquor brand (label)
* Bottle cap type
* Fill level, whether it is complete, half empty, or empty
* Contents: for example, whisky, vodka, gin, etc

Figures 5, 6, 7, and 8 display examples of these different attributes.

Given that the system´s goal is detection with the highest possible accuracy, focusing on each of these attributes would be counterproductive, as the training time would increase for every feature being considered. The label or brand printed on the bottle was selected as the primary attribute for classification. This feature is visually distinctive, consistently placed on the object, and directly relevant to the system’s functional objective, determining which bottles were used and, by extension, which cocktails were made. Focusing on this single attribute simplifies training and reduces false positives, while still allowing the system to deliver valuable, actionable data.



Figure 5 Different types of bottles.

|  |  |  |
| --- | --- | --- |
| Figure 6 Captain Morgan Rum Bottle. | Figure Cap Types. | Figure 8 Empty Black Label and Full SKYY bottle. |

As for the cocktails, the features that can be attributed include:

* Cocktail content, or type of mix
* Cocktail state
* Type of glass being mixed as seen in Figure 9
* Garnishes such as the ones found in Figure 10



Figure 9 Cocktail Glasses. Adapted from [22].



Figure 10 Cocktail Garnishes. Adapted from [23].

These attributes further complicate the detection and classification process compared to those found in the alcohol bottles, making it an impractical task to develop a separate framework solely for detecting and classifying drinks. This is why the drinks are considered byproducts of the bottle sequences rather than a separate category, and most of their attributes are not directly considered when designing the system.

### Analytical Data

The dataset used consists of labeled images of liquor bottles, similar to those found with bounding boxes in Figures 13 and 14, where each class corresponds to a specific brand or label (e.g., “Baileys”, “Smirnoff”, “Jose Cuervo”). This attribute was chosen for classification because:

* It is visually distinctive and easily captured in training images,
* It provides a direct link to cocktail recipes, and
* It allows the model to differentiate between objects with high reliability in real-time conditions.

Other attributes, such as shape, color, or fill level, were excluded due to their overlap across different classes, visual ambiguity, or limited relevance to the project's goal.

The datasets containing the training images were procured and are publicly available, hosted on the computer vision platform Roboflow. For this study, two datasets were primarily used that classify the images according to the label on the bottle, which corresponds to the alcohol brand. The first dataset is the Geonui dataset (Roboflow, 2022), which comprises 9,980 labeled images of various alcohol bottles and is licensed under the MIT license [24]. The second dataset is the www-nthwg dataset (Roboflow, 2022), which contains 6,448 labeled images and was published by user hailid3111-gmail-com under the CC BY 4.0 license [25]. Both datasets are labeled in the YOLO format for training with YOLOv8 and were sourced from the Roboflow Universe, having been chosen based on their relevance to bottle brand detection and compatibility with object detection tasks.

### Dataset Structuring

The images in the dataset are organized into three different folders: train, val, and test, with a split of 70% for training, 20% for validation, and 10% for testing, as shown in Figures 11 and 12, respectively. This is the standard split for this type of project, as shown in the figure. Each folder contains the images and their corresponding labels, separated. For each image file, there is a respective label .txt file with the following format:

<class\_id> <x\_center> <y\_center> <width> <height>

An example from the first dataset would be:

11 0.33653846153846156 0.49278846153846156 0.140625 0.15144230769230768

Where 11 is the class\_id, and the rest are the x, y center coordinates, followed by the width and length.

Examples of labelled images with bounding boxes can be seen in Figure.

|  |  |
| --- | --- |
| Figure 11 Dataset I Structure | Figure Dataset II Structure |

|  |  |
| --- | --- |
| Figure 13 Labelling with Bounding Box example 1. Adapted from [25]. | Figure 14 Labelling with Bounding Box example 2. Adapted from [25]. |

Aside from these folders, each dataset has its data.yaml file, which contains:

* Path for each folder.
* Number of classes.
* Class names.

Both examples are displayed in Figure 15 for the first dataset and Figure 16 for the second one.

train: C:/yolov5/Dataset\_bottles\_I/train/images

val: C:/yolov5/Dataset\_bottles\_I/valid/images

test: C:/yolov5/Dataset\_bottles\_I/test/images

nc: 16

names: ['10', '11', '12', '13', '14', '15', 'Absolut', 'Barcadi', 'Beefeater', 'Bombay-Sapphire', 'Captain-Morgan', 'Gordons', 'Havana-Club', 'Jose-Cuervo', 'Skyy', 'Smirnoff']

Figure 15 Dataset I data.yaml. Adapted from [24].

train: C:/yolov5/Dataset\_bottles\_II/train/images

val: C:/yolov5/Dataset\_bottles\_II/valid/images

test: C:/yolov5/Dataset\_bottles\_II/test/images

nc: 16

names: ['Absolut', 'Baileys', 'Ballantines', 'Beefeater', 'Bileys', 'Havana\_Club', 'Jack-Daniels', 'Jagermeister', 'Jim-Beam', 'Johnie-Walker', 'Jose-Cuervo', 'Kahlua', 'Malibu', 'Martini', 'Scotch-Blue', 'Smirnoff']

Figure 16 Dataset II data.yaml. Adapted from [25].

## Methods and Techniques

The Pipeline for creating the YOLOv8 Model was streamlined with the following typical steps for training with YOLOv8 [26]:

1. Training

Using Windows PowerShell, the standard command for training with the YOLOv8 CLI structure is executed. The training process is further explained in Sections t5 and 6.

1. Command Testing

To validate the training metrics, a simple detection test using the Windows PowerShell command for the YOLOv8 CLI structure is employed with a predetermined Confidence Threshold.

1. Coding

Once the weights from the previous training that contain the best results are obtained. The best.pt results (used for the remainder of the experiment) are obtained and loaded into a simple, low-level Python script that incorporates all the main and sub-tasks. Section 4.1 provides a more detailed explanation of these features.

1. Simple Testing

The script is deployed on a Windows Environment with a USB camera to test the software's ability to detect objects in real time.

1. Formatting

Once tested, the best.PT results are converted to a suitable format for use in a Linux Environment. The choose format is ONNX. This conversion is also done with the respective CLI YOLOv8 command.

1. Final Deployment

The format is best.onnx results are loaded in the same script on an NVIDIA Jetson Nano Orin and carry out more tests, similar to those in step 4, to observe the model´s performance across platforms and environments.

## Rationale for Choosing this Solution

Using pre-labelled datasets comes as a necessity given the project´s time constraints. Manually or semi-automatically labelling can be a lengthy process, and even automatic labelling presents its challenges, due to possible inaccuracies that can remain unchecked without a meticulously curated process. Considering these difficulties, creating a dataset with large quantities of images becomes an arduous process that can be avoided by obtaining images with labels that can be outsourced.

Figure 19 illustrates a typical example of how labeling can be performed using available programs, such as LabelMe, where images are often labeled with bounding boxes and assigned their respective class designations.

The Pipeline is streamlined to allow for simplicity and create synergy between each step. This approach allows for continuous feedback and iterations of process, such as training and testing based on the obtained results, while at the same time being able to correct, debug or modify problems on the Python script in real time, creating a more iterable system that can be modify which every training and testing run. Following these criteria, the Python script itself is open source, allowing users to modify it using the instructions found in Section 4.1.

While this is the desire goal with this approach, the study presented in this report shows a simplify version of this process, which follows the steps previously outlined in Section 3.3, without any other iterations, due to the present time and material conditions during the development of this project.

During the evaluation of the final training metric and the training results presented in Section 6, the proper insights were taken into consideration to recommend further development of all the pipeline´s main steps and improve previous iterations of this experiment.

# Resources

Following the project´s requirements, both the software developed and the hardware used for the system´s deployment need to be affordable, versatile, and accessible to all user levels. Therefore, the software is primarily comprised of the single Python Script mentioned in Section 3, along with the addition of the YOLOv8 Model, the datasets, and all the training data. As for the hardware, except for the advanced Desktop PC, the rest of the components are downscale for easier deployment.

## Software

The project uses a Python-based software solution designed for real-time object detection and logging of alcoholic beverages. The core of the system is a custom script that integrates a trained YOLOv8 model, manages detection logic, and handles user interaction and data logging. The software is optimized to run on both Windows and Linux environments, utilizing prompt commands in frameworks such as PowerShell, as shown in Figure 17, making it suitable for deployment on desktop PCs as well as low-resource devices, such as the NVIDIA Jetson Orin Nano. Additional tools such as OpenCV, NumPy, and Ultralytics' YOLO API are used to support image processing and model inference.

### Software Architecture

The programming specifications are the following:

* CPython Architecture
* Bytecode Compilation (.pyc)
* Standard UTF-8 Encoding
* Automatic memory management via reference counting & cyclic garbage collector
* Global Interpreter Lock (GIL)
* Single-threaded process execution (parallelized via PyTorch if GPU acceleration is enabled)
* Dynamic Memory Allocation, Stack & Heap Segmentation
* NumPy/PyTorch tensors for image and model data
* CUDA acceleration via PyTorch
* OpenCV for real-time video input and output
* Ultralytics YOLOv8 detection model
* Cross-platform compatibility (Windows/Linux)
* Standard Python file I/O for summary logging

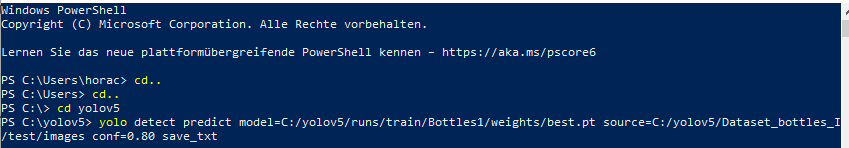


Figure 17 Windows PowerShell framework for Windows

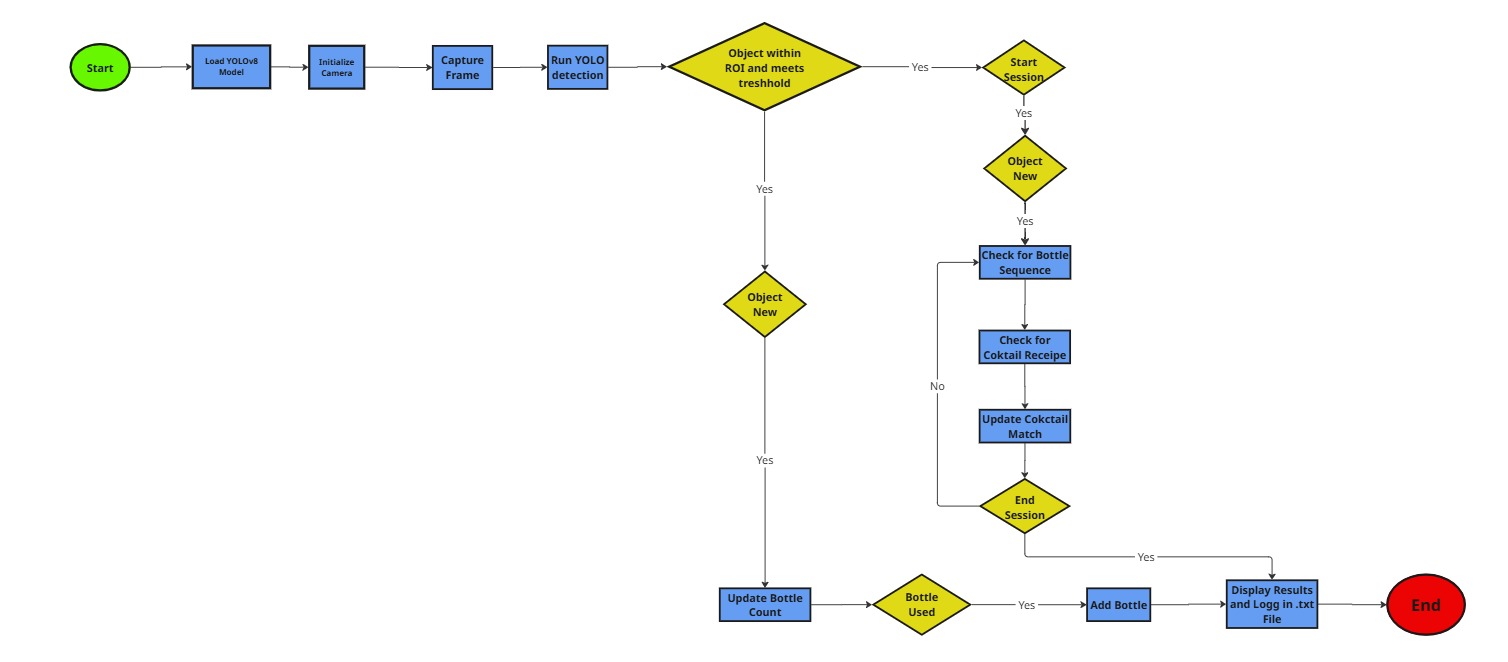


Figure System Flowchart

### Software Products

As shown in the flowchart in Figure 18, the steps the system follows are:

| ***Module / Section*** | ***Functionality Description*** |
| --- | --- |
| *Initialization & Configuration* | *Load YOLOv8 model, initialize camera, define ROI, set thresholds, and variables.* |
| *Image Capture & Processing* | *Capture frames from the camera and prepare them for object detection and recognition.* |
| *Object Detection* | *Run YOLOv8 inference, apply confidence and ROI filtering, and extract class labels.* |
| *Detection Filtering & Counting* | *Check if the object is new using cooldown logic and update the bottle detection counts.* |
| *Session Control* | *Handle user input (S, E, Q) to start, end, or exit detection sessions.* |
| *Cocktail Matching* | *Compare used bottle combinations to known recipes and increment the cocktail counters.* |
| *Logging & Summary Output* | *Write final session summary (total bottles + cocktails) to a .txt log file.* |
| *Visualization* | *Display bounding boxes, class names, and real-time counters on the video feed.* |
| *Cleanup & Exit* | *Release hardware resources, close files, and terminate the script safely to ensure a clean shutdown.* |

Table 1 System Description

The complete Python script can be found on the project´s GitHub, which includes instructions on how to use and modify, as well as a guide for creating and training new datasets with YOLOv8. Specific sections of the script can also be found in the Annex Section. Corresponding with Table 1, the key sections to understand this source code's functioning and modification for future iterations are the following:

1. Libraries and Frameworks:

The necessary libraries and modules are initialized for essential functions such as:

* Real-time video (OpenCV)
* Deep learning (YOLOv8 via Ultralytics)
* Time management and logging
* Counting occurrences (e.g., how many times each bottle appears)

1. Model Loading and Configuration:

Loads the pre-trained YOLOv8 model weights best.pt results (other formats such as ONNX or TensorRT can also work in this framework) obtained from the training with YOLOv8.

1. Cocktail Recipes Definition

This part acts as a dictionary, where each key is a cocktail name and its corresponding value is a list of bottle names and sequences, which serve as the determining classes in the model.

1. Initialize Camera and ROI

This function is responsible for initializing the camera interface and setting the resolution. Additionally, it defines the Region of Interest (ROI), an area where the user must place objects to trigger detection. Objects outside the ROI won´t be detected, as seen in Figure 19.

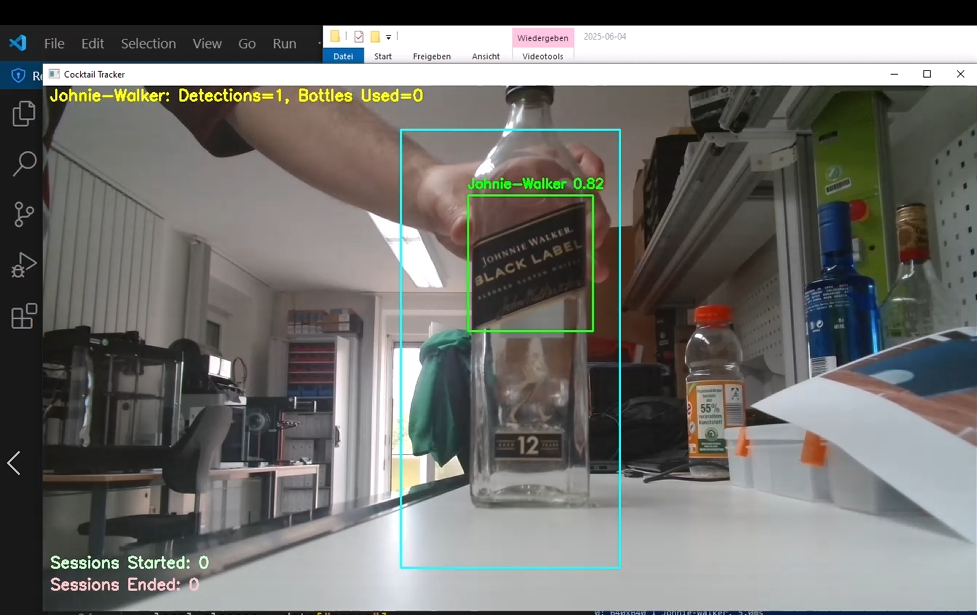


Figure 19 Example of detection with ROI.

1. Detection Settings and Counters

Essential values for parameters such as Confidence Threshold and Cooldown Time are set, both of which serve to detect and avoid repeated object counts.

1. Logging Setup

The log file is created to store all detections and cocktails, along with their timestamps.

1. Main Loop

This section is the core of the script and executes the following sequence for processing the images provided by the USB camera, to be compared with the loaded model:

* 1. Capture a video frame
  2. Resize for the model
  3. Detect bottles using YOLO
  4. If the bottle is in ROI and above the confidence threshold, it is logged in
  5. Display bounding box and name on screen

1. Detection & Cooldown Logic

The purpose of this code section is to:

* Filters out weak detections
* Ensures a bottle is not counted again too soon
* Updates the count and writes it to the log

It is in this section that the number of bottles is also counted, following the Cooldown Time instances. For this experiment, the Cooldown Time is set to 3 seconds, which requires approximately six appearances or 18 seconds to count a bottle as used. The user can adjust this factor to better reflect the specific conditions in which the system is being deployed.

1. Session Control and Cocktail Matching Logic

To enable the code for counting the number of cocktails prepared based on the bottle sequence, the user must press *s* to start the session. If the bottles are detected in the correct sequence, the session can end by pressing 'e', and the prepared cocktail can be logged. The user can change these keys to suit their convenience.

1. Visualization & On-Screen Text

Given that the user wants to supervise the detections in real time, the script offers an interface that shows:

* Brands or classes being detected with respective instances and total bottles used
* Number of cocktails prepared
* Number of sessions executed

1. End-of-Script Summary Output

When the program ends, a summary report is saved with the total:

* Sessions
* Cocktails prepared
* Bottles used

1. Cleanup and Exit

The user can stop and close the program by pressing *the 'q' key*. This key can also be changed for convenience.

## Hardware

The hardware use varies in capacity and performance depending on the tasks and deployment performed. Overall, the devices can be used in both Professional and DIY data analysis and computer vision projects.

### Hardware Architecture

Given that most processes are executed in a virtual environment, the hardware architecture doesn´t consist of complex networks, but relatively simple connections between a visual interface and a device.

### Hardware Devices

**Desktop PC** **AMD Ryzen 9 5950X**



Figure 20 Desktop Computer Use for Training Process

A high-performance Windows workstation, as shown in Figure 20, was utilized for the development and training phases of the deep learning model used in this project. Equipped with an AMD Ryzen 9 5950X 16-core processor, an NVIDIA GeForce RTX 3080 Ti GPU, and 64 GB of RAM, this machine provided the computational capacity required for intensive training tasks and model optimization. The RTX 3080 Ti’s CUDA compatibility (CUDA 11.8) enabled accelerated parallel processing and efficient handling of large datasets using frameworks such as PyTorch and YOLOv8. Python 3.11 was used to leverage recent language improvements and ensure compatibility with modern AI libraries. This workstation was responsible for generating and testing the best.pt weights file, which was later deployed on the NVIDIA Jetson Orin Nano for real-time inference. Its combination of high-end CPU and GPU performance made it ideal for model iteration, hyperparameter tuning, and validation prior to deployment in a low-power embedded environment.

**NVIDIA Jetson Orin Nano**

The NVIDIA Jetson Orin Nano was selected for this project due to its strong AI performance in a compact, energy-efficient form factor. With up to 67 TOPS of processing power and configurable consumption ranging from 7W to 25W, it supports real-time object detection while remaining suitable for low-power environments, such as solar-powered systems. It achieves approximately 10–12 FPS when running YOLO models at 1080p, making it well-suited for accurate, real-time detection. In this project, the Jetson Orin Nano is explicitly used for deploying the trained YOLOv8 model after testing. Figure 21 shows the distinctive feature of this device.

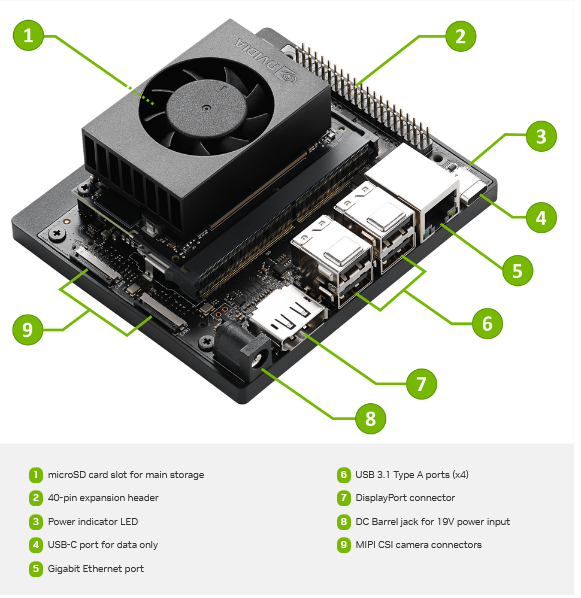


Figure 21 NVIDIA Jetson Orin Nano

According to the Official Jetson Orin Nano Super Developer Kit Datasheet, the specifications of this device are [18]:

* Jetson Orin Nano 8GB Module
* GPU: NVIDIA Ampere architecture with 1024 NVIDIA® CUDA® cores and 32 Tensor cores
* CPU: 6-core Arm Cortex-A78AE v8.2 64-bit CPU 1.5MB L2 + 4MB L3
* Memory: 8GB 128-bit LPDDR5 68 GB/s
* Storage: external via microSD slot, external NVMe via M.2 KeyM
* Power: 7W to 15W

The NVIDIA Jetson Orin Nano was configured according to the instructions in the Developer Kit Getting Started Guide from NVIDIA´s official site [17].

**Logi HD 1080p USB-Camera**

****

Figure 22 USB Camera

A Logitech HD 1080p USB webcam, as shown in Figure 22, was used in this project to capture real-time video input during the development and testing of the object detection system. This webcam, likely a Logitech C920 or similar variant, offers Full HD resolution at 30 frames per second with a 78-degree field of view and built-in autofocus [27]. Its high-quality glass lens and automatic light correction ensure clear image capture in varied lighting conditions, making it suitable for indoor laboratory setups [27]. The webcam was connected via USB and used as the primary image source during the review of training data, live testing, and validation of YOLO-based models on both the development workstation and the deployed NVIDIA Jetson Orin [27]. Nano system. Its reliable performance and plug-and-play compatibility with Python and OpenCV libraries made it an effective and practical choice for this computer vision application [27].

## People

While this project is the work of a single person, usually an endeavour of this sort requires a team divided into categories that relate to the specific main tasks previously described in this report. Therefore, there should be two teams: One for the Deep Learning training and another for testing the YOLOv8 Model deployment [28],[29].

Each team should also be divided according to specific subtasks [28],[30].

The Deep Learning Team handles the curation of images for creating datasets, which includes:

* Image collection (in the case of own or customized dataset)
* Labelling of images
* Quality control (in case of outsourced material, check if images and labels correspond to each folder, e.g., train/val/test)
* Training execution
* Validation of training results

The Deployment Team tests the Model performance by [28],[30]:

* Adjusting the software (modify the script parameters and update the cocktail recipe)
* Run tests in diverse conditions and scenarios, with different devices, platforms, and environments
* Validate the test results

The team's size depends on the project´s scale, complexity, and professional level. For a project similar in scope and complexity to the one outlined in this report, a team of approximately 4 to 10 individuals should be sufficient [31], [32]. This team should be evenly split between the Deep Learning and Deployment Teams to ensure a balanced workload distribution, assuming similar timeframes, available resources, and task requirements [28],[29]. Each team member would likely take on multiple roles, especially in smaller-scale operations, requiring flexibility and a cross-functional knowledge base [28],[29]. In the case of a semi-professional project, one that benefits from moderate financial backing, access to relatively advanced equipment, and a clear target for commercial or business application, the scale of operations expands considerably. Under such conditions, each team would ideally consist of 20 to 40 professionals [28],[29]. Finally, for a thoroughly professional and commercially viable project aiming to launch a comprehensive, marketable product, which includes integrated platforms, customer support systems, specialized software with pre-labeled datasets, and potentially customized hardware solutions, a larger and more structured team is necessary [31], [32].In such cases, a staff of up to 100 people or more may be required [31], [32].

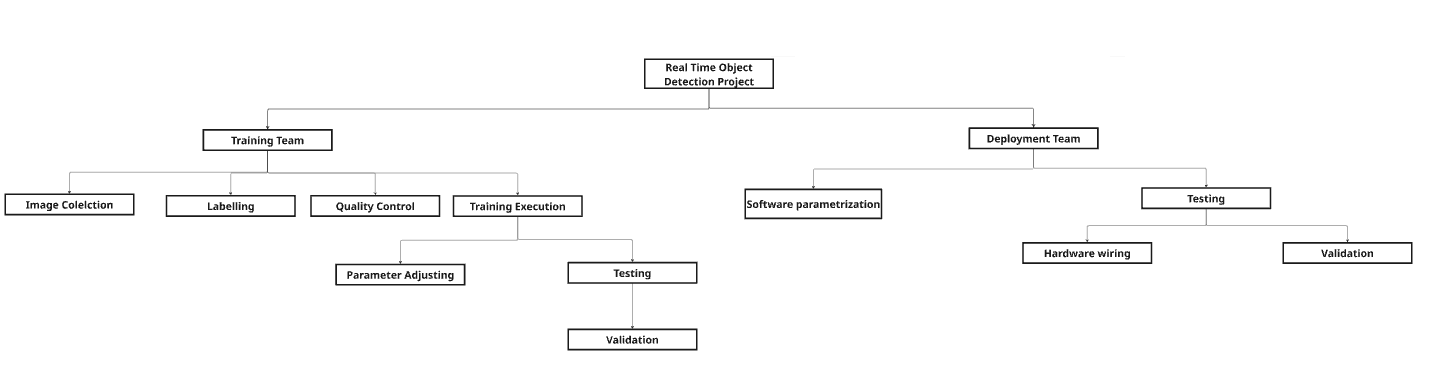


Figure 23 Basic Team Structure

Figure 23 illustrates a general organization that could serve as a potential scale for various projects and potentially be expanded into new branches.

Depending on the task requirements, team members may distribute the necessary work evenly or unevenly [28],[29]. The organization should also work with sufficient synergy, so that the training results can inform testing deployment and be adjusted with as few iterations as possible to achieve the desired results [28],[29]. For example, on the training team the member(s) that execute the training should apply quality control to both the data their working on or the results they are validating through testing or in the case of being other members that do the task separately should be informed about problems with the training results so they can correct the labelling and the image quality in real time. The same principle should apply to the deployment team, where software adjustments should occur after continuous testing, creating a feedback loop for the individuals involved. In short, there should be continuous improvement along the pipeline by integrating each team and its subdivisions, so that training results inform testing and deployment, and vice versa.

## Miscellaneous Resources

Aside from the equipment mentioned in Section 4.2, Deep Learning Training requires a clean and organized environment that allows for numerous testing variations, a requirement typically found in an IoT or computer Lab. Testing materials should be available, such as photographs and real-size objects. Figures 24 and 25 show examples of both testing material types used for this project. An above-average and organized network is necessary for providing services such as the internet and cloud storage. Devices that could replace the existing one in other variations should also be present in case of malfunction due to constant work.

|  |  |
| --- | --- |
| Figure 24 Example of real real-size object. | Figure 25 Example of printed image. Adapted from [25]. |

# Prerequisites

The pipeline follows the sequence shown in the Figure for the system´s execution:



Figure 26 Developing Pipeline

The data must be collected and organized for training and validation purposes [26]. These categories are always required for training [26]. Optionally, data can also be prepared for testing, but it is not necessary [26]. Once the data is organized, the labelling process for all of the images leads to the complete creation of the dataset [33]. Afterwards, training can begin [26]. Once the training metrics are verified as optimal and the user´s criteria are satisfied, deployment tests of the Model Are best.pt weights can be executed [34]. If these test results are successful, the weights can be imported into another format for use on other devices and in various environments [34]. In summary, the prerequisites for conducting a successful Deep Learning Training with YOLOv8 are that the data is correctly ordered and labelled for training and validation folders, with each image having a corresponding label [34]. Otherwise, the training process cannot be completed, or at best, the results would be unsatisfactory. After training, the deployment testing results should serve as verification is the Model is ready to be exported into other devices for further use and application [34]. The complete Pipeline is shown in Figure 26.

# Evaluation

The Model´s performance was evaluated using the YOLOv8 training metrics and the detection rate. The training metrics indicate the effectiveness of the training process described in Section 3.3. This process is conducted in a single stage using the YOLOv8 CLI command structure. The detection rate is first tested using the CLI command structure with digital images of the objects, and later in a real-life setting on a Windows environment with a small, printed sample of these images. Finally, a deployment test is executed, using both real-life and printed images on a Linux environment, to test the model´s capability after being converted into another format and cross-platform for other devices with different operating systems

## Outcomes

To understand the evaluation outcomes, it's vital to understand the business impact of a potential product and its performance measures, the result of all the previously listed tests. These outcomes help to understand this project both as a commercial endeavour with several technical requirements that need to be fulfilled in order to offer the best service possible.

### Business Impact (in case of data analysis)

The intended business impact of this system lies in its ability to support inventory control, optimize procurement, and provide real-time insights into customer preferences at bars or similar establishments. By developing a comprehensive system that assists users with either complete or partial automation of tasks such as bottle detection and logging, as well as cocktail preparation, the system aims to reduce manual errors, improve stock management, and inform data-driven decisions.

If the model performs with sufficient accuracy and speed, it could allow small and medium-sized venues to adopt affordable AI-powered tools that were previously out of reach due to cost or complexity

### Performance Measures (in case of data analysis)

According to the developing pipeline mentioned in Section 5, to assess the system performance, the training metrics and the detection rates of Model I and Model II, respectively, were analyzed to conclude the suitability of both models for deployment tasks.

**Training Metrics**

The training process with the two datasets mentioned in Section 3.3 produced two different Models, I and II. The metrics analyzed to determine the result´s quality were:

**Precision:** Indicates the proportion of predicted detections that are correct. A high precision value means the model rarely produces false positives, i.e., it rarely identifies an object when there is none [35], [36], [37].

**Recall:** Measures the proportion of actual objects that the model successfully detects. A high recall means the model rarely misses a bottle that appears in the frame. This is particularly important in dynamic scenarios where objects may only be visible for a brief moment [35], [36], [37].

**F1 Score:** Is the harmonic mean of precision and recall. It provides a balanced single metric when both false positives and false negatives must be minimized [35], [36], [37].

**mAP@0.5 (Mean Average Precision at IoU = 0.5):** This metric averages the precision across all classes at an Intersection over Union (IoU) threshold of 0.5. In practical terms, it measures how well the predicted bounding boxes align with ground-truth annotations when a moderate level of overlap is acceptable. It is the most used benchmark for object detection [35], [36], [37].

**mAP@0.5:0.95:** A more stringent metric, mAP@0.5:0.95, averages the precision over multiple IoU thresholds from 0.5 to 0.95. It evaluates not only whether objects are detected, but also how precisely their positions are estimated. A high value in this metric reflects fine-grained localization performance [35], [36], [37].

**Loss Values:** During training, three primary loss components are monitored [35], [36], [37]:

**Box loss:** Indicates the distance between the predicted bounding box and the correct location. [35], [36], [37]

**Classification loss:** Reflects errors in labeling the detected object [35], [36], [37].

**DFL (Distribution Focal Loss):** A refined loss used in YOLOv8 to improve bounding box prediction by modelling edge distributions [38].

Lower loss values across all three categories indicate a well-trained model [38], [39], [40].

The following CLI commands [41] were used to execute the training on Model I & II, respectively:

## Model I Training

* yolo detect train model=yolov8s.pt data=C:/yolov5/Dataset\_bottles\_I/data.yaml epochs=300 imgsz=640 batch=16 name= Bottle1

## Model II Training

* yolo detect train model=yolov8s.pt data= data=C:/yolov5/Dataset\_bottles\_II/data.yaml epochs=300 imgsz=640 batch=16 name=Bottle2

## YOLOv8 Training Command Parameters [41]

|  |  |
| --- | --- |
| *Parameter* | *Description* |
| *model* | *The base* |
| *data* | *Path to data.yaml file* |
| *epochs* | *Number of training iterations (epochs)* |
| *imgsz* | *Input image resolution (e.g., 640 for 640x640)* |
| *batch* | *Batch size per training step* |
| *name* | *Name for the training run* |

Table 2 Training Parameters

**Model I Metrics**

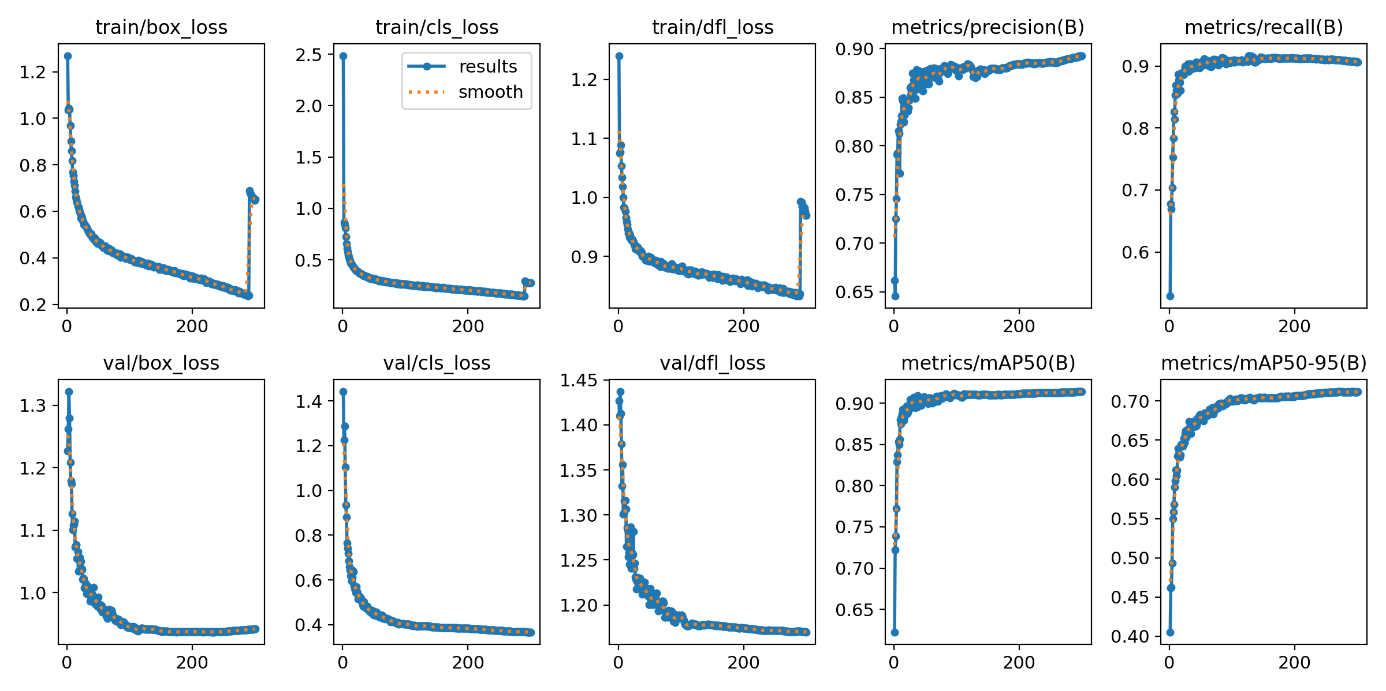


Figure 27 Training Results

The training results shown in Figure 27 of Model I exhibit ideal metrics and optimal behavior across all evaluated dimensions, indicating a highly stable and effective learning process.

**Box Loss (train/val):** The box loss decreases steadily from an initial value above 1.2 to below 0.25, both in training and validation. This indicates that the model is learning to predict bounding box coordinates with minimal error accurately [35], [36], [37].

**Classification Loss (train/val):** This loss begins relatively high but quickly stabilizes below 0.25. The model shows strong convergence in classification learning, suggesting that it reliably distinguishes between object classes [35], [36], [37].

**DFL Loss (train/val):** Distribution Focal Loss decreases consistently and smoothly, indicating that the model is improving in terms of bounding box edge precision. Final values remain low (~0.85), aligning with the expectations of a well-calibrated YOLOv8 model [38].

**Precision:** Precision increases early in training and stabilizes at around 0.93, indicating that most detections are correct and the model effectively avoids false positives [35], [36], [37].

**Recall:** Recall stabilizes above 0.91, indicating strong coverage of all relevant objects in the dataset and very few false negatives.

**mAP@0.5**: This value peaks at 0.914, reflecting the model’s strong overall detection accuracy when a moderate overlap (IoU ≥ 0.5) is required between predicted and ground truth boxes [35], [36], [37].

**mAP@0.5:0.95**: The model reaches approximately 0.71 at this stricter IoU range, confirming that it also performs well under higher precision demands [35], [36], [37].

Model I demonstrates efficient and balanced learning across all metrics, making it a strong candidate for real-time object detection in production environments [35], [36], [37],[38].

**Model II Metrics**

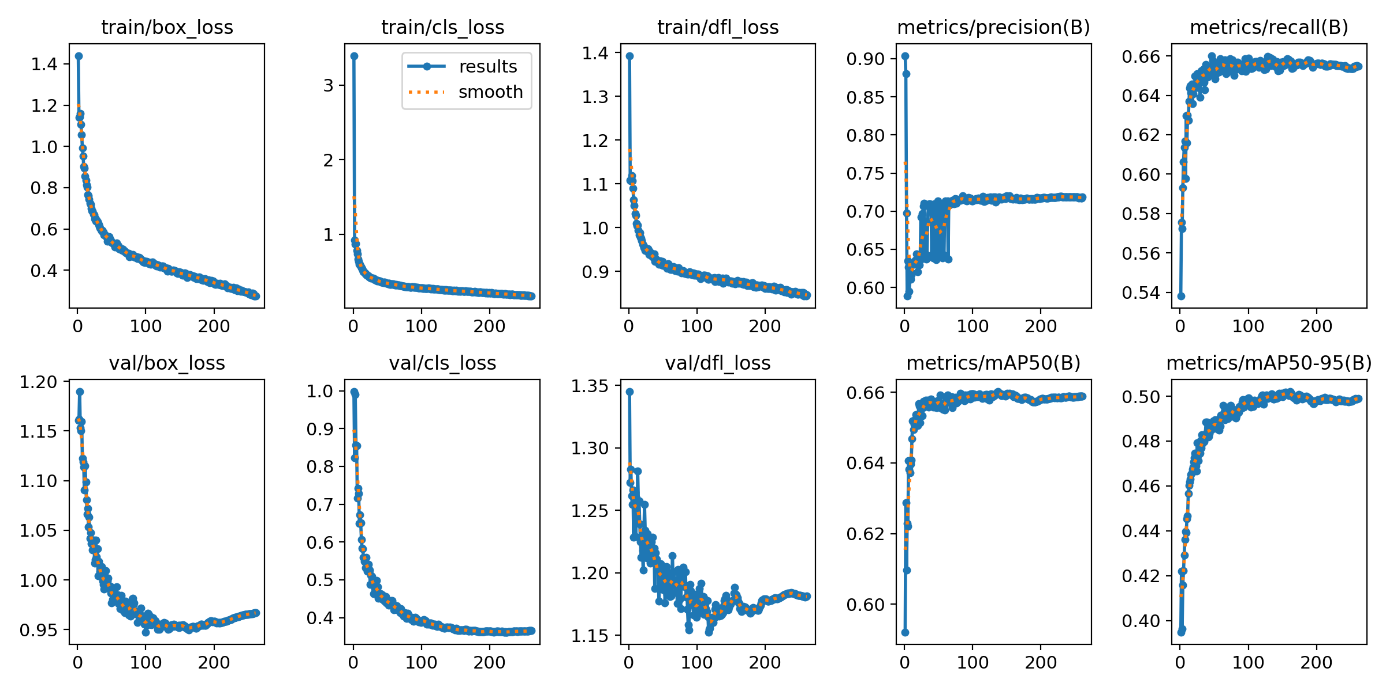
****

Figure 28 Training Results

The Model II training results, displayed in Figure 28, show acceptable but clearly inferior performance across all training metrics, with signs of limited generalization and potential issues related to the dataset or annotations.

**Box Loss (train/val):** The loss decreases from ~1.4 to ~0.30 but does not reach the same low levels as Model I. The higher floor suggests that bounding box predictions are less accurate [35], [36], [37].

**Classification Loss (train/val):** Starting above **3.0**, the classification loss takes longer to stabilize and settles around ~0.35–0.40. This indicates that the model struggles more with distinguishing between object classes [35], [36], [37].

**DFL Loss (train/val):** The final DFL loss remains higher and fluctuates more than in Model I, showing weaker bounding box edge accuracy [38].

**Precision:** While increasing over time, precision remains unstable, reaching only 0.73, with clear signs of fluctuations throughout training. This suggests inconsistent performance across batches [35], [36], [37].

**Recall:** The final recall stabilizes at around 0.66, indicating that the model is missing a significant number of actual objects during detection [35], [36], [37].

**mAP@0.5:** The model reaches a peak of 0.659, which is notably lower than Model I, suggesting reduced detection accuracy [35], [36], [37].

**mAP@0.5:0.95:** Model II reaches about 0.50, reflecting suboptimal localization and detection quality across varying IoU thresholds [35], [36], [37].

Model II performs adequately but suffers from low recall, higher loss values, and detection failures in several classes, making it less suitable for deployment in real-world object detection tasks [35], [36], [37], [38].

## YOLOv8 Model I & Model II Metrics Comparison

|  |  |  |
| --- | --- | --- |
| *Metric* | *Model I (High Performance)* | *Model II (Lower Performance)* |
| *Final Box Loss (val)* | *~0.25* | *~0.30–0.32* |
| *Final Cls Loss (val)* | *~0.25* | *~0.35–0.40* |
| *Final DFL Loss (val)* | *~0.85* | *~0.90–0.95* |
| *Precision* | *0.93* | *0.73* |
| *Recall* | *0.91* | *0.66* |
| *mAP@0.5* | *0.914* | *0.659* |
| *mAP@0.5:0.95* | *~0.71* | *~0.50* |
| *Loss Curve Stability* | *Smooth* | *Fluctuating* |
| *Class Failures (0 mAP)* | *1 class* | *≥ 5 classes* |

Table 3 Model I vs Model II comparison

**Detection Rate**

For the first type of test to test the detection rate with Model I & II, respectively, the following CLI commands [41] were used:

## Model I

* yolo detect val model=C:/yolov5/Dataset\_bottles\_I/weights/best.pt data= C:/yolov5/Dataset\_bottles\_I/data.yaml conf=0.80
* yolo detect predict model= C:/yolov5/Dataset\_bottles\_I/weights/best.pt source= C:/yolov5/Dataset\_bottles\_I/test/images conf=0.80 save\_txt save\_conf

## Model II

* yolo detect val model=C:/yolov5/Dataset\_bottles\_II/weights/best.pt data= C:/yolov5/Dataset\_bottles\_II/data.yaml conf=0.80
* yolo detect predict model= C:/yolov5/Dataset\_bottles\_II/weights/best.pt source= C:/yolov5/Dataset\_bottles\_II/test/images conf=0.80 save\_txt save\_conf

## YOLOv8 Validation Command Parameters [41]

|  |  |
| --- | --- |
| *Parameter* | *Description* |
| *model* | *Path to the trained YOLOv8* |
| *data* | *Path to data.yaml containing test/val dataset and class info* |
| *conf* | *Confidence threshold for* |
| *save\_txt* | *Saves predictions in YOLO text format* |
| *save\_conf* | *Saves the confidence score with each prediction* |
| *save\_hybrid* | *Combines predictions and ground truth for easier analysis* |

Table 4 Validation Parameters

## YOLOv8 Detection Command Parameters [41]

|  |  |
| --- | --- |
| *Parameter* | *Description* |
| *model* | *Path to the trained YOLOv8 model* |
| *source* | *Path to input images or video for detection* |
| *conf* | *Confidence threshold to filter out low-confidence detections* |
| *save\_txt* | *Saves detection outputs in YOLO text format* |
| *save\_conf* | *Appends confidence scores to detection outputs* |
| *save\_crop* | *Saves cropped images of detected objects* |
| *project* | *Custom output directory for results* |
| *name* | *Name inside the project.* |

Table 5 Detection Parameters

As for exporting the model, both Model I and Model II weights are best.pt results, respectively, the following commands [41] were used:

## Model I

## yolo export model=C:/yolov5/Dataset\_bottles\_I/weights/best.pt format=onnx dynamic=True simplify=True opset=12 half=True

## Model II

## yolo export model=C:/yolov5/Dataset\_bottles\_II/weights/best.pt format=onnx dynamic=True simplify=True opset=12 half=True

## YOLOv8 Exporting Command Parameters [41]

|  |  |
| --- | --- |
| *Parameter* | *Description* |
| *model* | *Path to the trained YOLOv8 model file* |
| *format* | *Export format for deployment* |
| *dynamic* | *Exports a dynamic ONNX model for variable input sizes (optional)* |
| *simplify* | *Simplifies ONNX model to optimize the graph structure (optional)* |
| *opset* | *Specifies the ONNX opset version (optional, default is 12)* |
| *half* | *Converts model weights to FP16 (half precision) where supported (optional)* |

Table 6 Exporting Parameters

## Command Testing

For these tests, the two sets of commands previously mentioned were used to validate the accuracy and precision of the detection rate. This refers to the number of objects that were properly detected and correctly classified with a Confidence Threshold equal to or greater than the one used in the command (0.80).

## *Model I*

## YOLOv8 Validation Summary – Global Metrics

|  |  |  |
| --- | --- | --- |
| *Metric* | *Value* | *Meaning* |
| *Images* | *2006* | *Total number of test images evaluated* |
| *Instances* | *2367* | *Total number of labeled objects in the test set* |
| *Box(P)* | *0.911* | *Precision: 91.1% of the model’s detections were correct* |
| *R* | *0.828* | *Recall: 82.8% of the actual objects were detected by the model* |
| *mAP50* | *0.871* | *Mean Average Precision at IoU 0.5 (strong general performance)* |
| *mAP50-95* | *0.692* | *Average mAP across IoUs from 0.5 to 0.95 (more strict, very solid)* |

Table 7 Model I Validation Metrics

## YOLOv8 Detection Summary

With the detection command mentioned previously, Model I was able to detect 910 out of a test batch of 1004 digital images, resulting in an approximate detection rate of 91% with a Confidence Threshold of 0.80, which confirms the precision metric in Table 7.

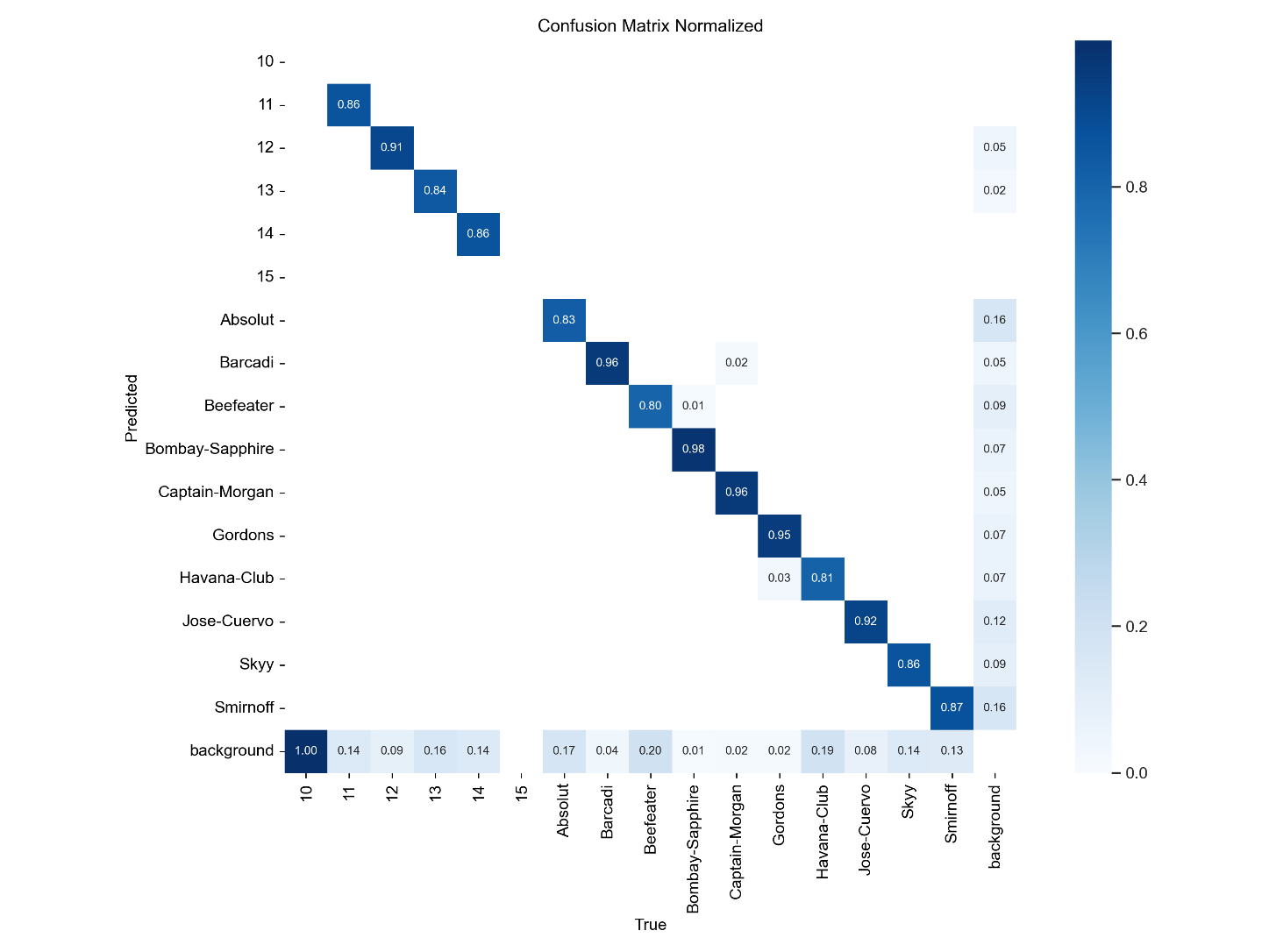
However, the confusion matrix in Figure 29 shows Model I’s performance across all classes, highlighting both correct and incorrect predictions. The diagonal values represent correct classifications (true positives), while off-diagonal values indicate misclassifications [42], [43],[44]. Classes such as Bacardi, Bombay Sapphire, and Captain Morgan exhibit strong performance with minimal errors. In contrast, others, such as Absolut, Beefeater, and Havana Club, display moderate confusion, with a notable portion of instances being misclassified or predicted as background noise. This visual representation sets the foundation for the following class-level error analysis, where FP (false positives) and FN (false negatives) are estimated using precision, recall, and instance counts from the validation results [42],[43],[44].

Figure 29 Model I Confusion Matrix

## Both values are calculated with the following calculations:

True Positives (TP):

* TP = Recall × Number of Instances [45],[46]

False Positives (FP):

* FP = (TP / Precision) – TP [45],[46]

False Negatives (FN):

* FN = (TP / Recall) – TP [45],[46]

Where:

- TP: True Positives (correct detections)

- FP: False Positives (incorrect detections)

- FN: False Negatives (missed detections)

- Precision and Recall are derived from the YOLOv8 validation output.

- Instances refer to the number of labeled objects for that class in the validation set.

Table 8 shows an estimation of FP and FN values for all classes:

## Model I – Estimated TP, FP, and FN per Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Class* | *Instances* | *True Positives (TP)* | *False Positives (FP)* | *False Negatives (FN)* |
| *Absolut* | *133* | *110* | *7* | *23* |
| *Barcadi* | *202* | *194* | *3* | *8* |
| *Beefeater* | *275* | *220* | *5* | *55* |
| *Bombay-Sapphire* | *117* | *115* | *3* | *2* |
| *Captain-Morgan* | *54* | *52* | *2* | *2* |
| *Gordons* | *61* | *58* | *3* | *3* |
| *Havana-Club* | *302* | *244* | *5* | *58* |
| *Jose-Cuervo* | *218* | *201* | *5* | *17* |
| *Skyy* | *277* | *239* | *4* | *38* |
| *Smirnoff* | *281* | *245* | *7* | *36* |

Table 8 Model I Class Detection



Figure 30 TP with limitation. Adapted from [25].



Figure 31 Example of FP (Left) and FN (Right). Adapted from [24].

## *Model II*

## YOLOv8 Validation Summary – Global Metrics

|  |  |  |
| --- | --- | --- |
| *Metric* | *Value* | *Meaning* |
| *Images* | *1280* | *Total number of test images evaluated* |
| *Instances* | *1556* | *Total number of labeled objects in the test set* |
| *Box(P)* | *0.659* | *Precision: 65.9% of the model’s detections were correct* |
| *R* | *0.622* | *Recall: 62.2% of the actual objects were detected by the model* |
| *mAP50* | *0.641* | *Mean Average Precision at IoU 0.5 (moderate detection accuracy)* |
| *mAP50-95* | *0.499* | *Average mAP across IoUs from 0.5 to 0.95 (lower precision under strict conditions)* |

Table 9 Model II Validation Metrics.

## YOLOv8 Detection Summary

As for Model II, out of 655,608 digital images, 610,800 were detected, resulting in an approximate detection rate of 93%, a notable contrast with the validation metrics presented in Table 9. The model´s confusion matrix, as shown in Figure 30, provides further insight into the accuracy of this detection test.

## 

Figure 32 Example of multiple TP (Right) with instance of FN (Left). Adapted from [25].

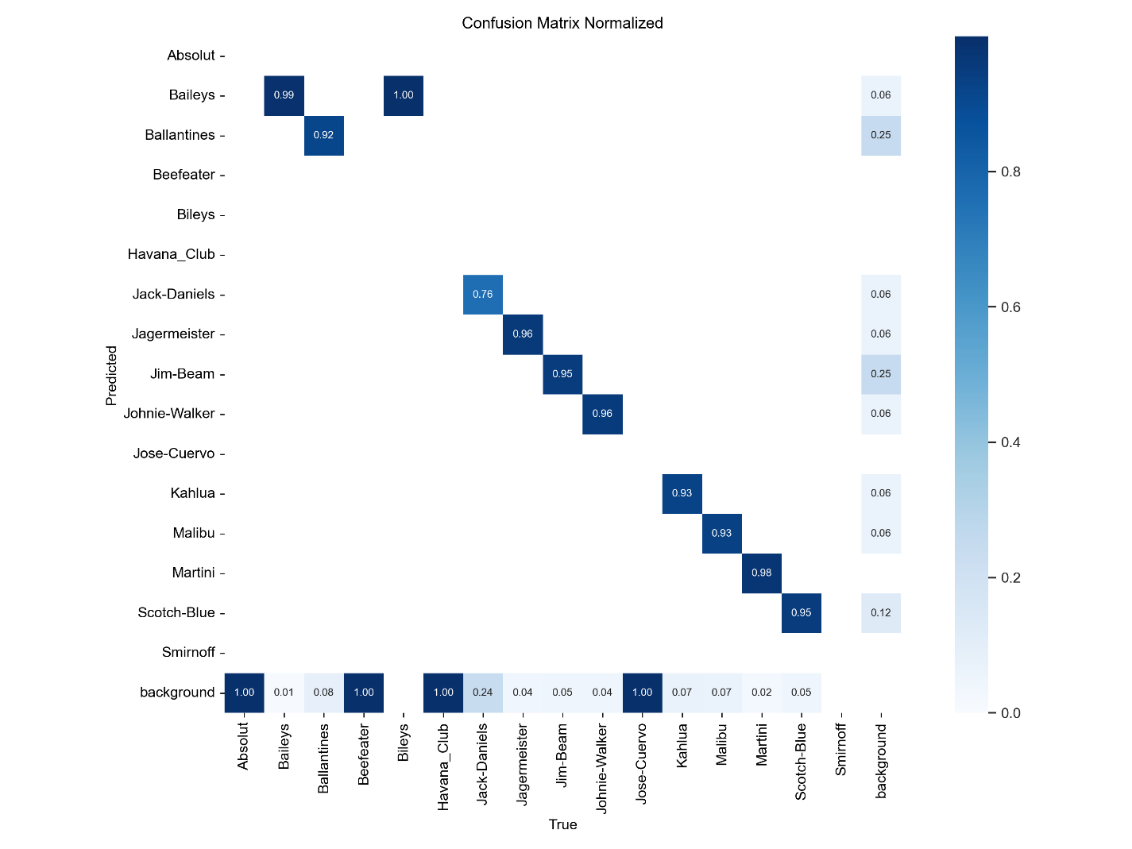


Figure 33 Model II Confusion Matrix.

Classes such as Baileys, Ballantine's, and Johnnie Walker demonstrate high detection accuracy with minimal false positives or negatives, supported by strong diagonal values in the confusion matrix. However, several classes, including Absolut, Beefeater, Baileys, and Havana Club, show either no successful detections or significant misclassification, with recall and precision values of zero.

## Model II – Estimated TP, FP, and FN per Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Class* | *Instances* | *True Positives (TP)* | *False Positives (FP)* | *False Negatives (FN)* |
| *Absolut* | *1* | *0* | *—* | *—* |
| *Baileys* | *134* | *133* | *2* | *1* |
| *Ballantines* | *224* | *205* | *4* | *19* |
| *Beefeater* | *2* | *0* | *—* | *—* |
| *Bileys* | *1* | *0* | *—* | *—* |
| *Havana\_Club* | *3* | *0* | *—* | *—* |
| *Jack-Daniels* | *221* | *168* | *1* | *53* |
| *Jagermeister* | *139* | *134* | *1* | *5* |
| *Jim-Beam* | *202* | *192* | *4* | *10* |
| *Johnie-Walker* | *180* | *172* | *1* | *8* |
| *Jose-Cuervo* | *1* | *0* | *—* | *—* |
| *Kahlua* | *168* | *156* | *1* | *12* |
| *Malibu* | *150* | *139* | *2* | *11* |
| *Martini* | *50* | *49* | *0* | *1* |
| *Scotch-Blue* | *80* | *76* | *2* | *4* |
| *Smirnoff* | *61* | *58* | *3* | *3* |

Table 10 Model II Class Detection Values.

## Model I vs. Model I Validation and Detection Comparison

|  |  |  |
| --- | --- | --- |
| *Metric* | *Model I* | *Model II* |
| *Validation Precision* | *0.911* | *0.659* |
| *Validation Recall* | *0.828* | *0.622* |
| *mAP@0.5* | *0.871* | *0.641* |
| *mAP@0.5:0.95* | *0.692* | *0.499* |
| *Detection Test Rate* | *N/A (qualitative only)* | *93%* |
| *Estimated Avg FP per Class* | *8* | *6* |
| *Estimated Avg FN per Class* | *13* | *12* |

Table 11 Model I vs. Model II Detection Rate Comparison.

## Real Time Object Detection Test

## As previously mentioned, this type of test is divided into two stages. The first stage, called Simple Test, executed in Windows, focuses solely on detecting a small sample of printed images from the previous test batch used with the command test, to assess the system's overall performance, with the accuracy of the detection rate being the primary focus. The second stage, on the other hand, acts as the final system´s deployment, with the goal of testing all previously described features in Section 4.1. This deployment is executed on a Linux environment using the NVIDIA Jetson Orin Nano. Both tests use the same source code, with the distinction being the weights.pt result for each model, being used for the Simple Test, and the weights onnx.PT results are a formatted adaptation of the Advanced Test.

## Simple Test

A small subset sample from the original Model I and II batches, approximately 10%, was used for this test. A total of 165 printed images, as shown in Figure 34, were tested: 100 for Model I and 65 for Model II. In this experimental set-up, the Logi HD 1080p USB-Camera, was placed to detect on an upwards position as see in Figure 34, to simulate better the conditions of deployment, where a user places the bottle within the ROI for a determine time frame, to mix the liquors following a sequence according to the recipe list found in the script as displayed previously on Section 4.1.2.

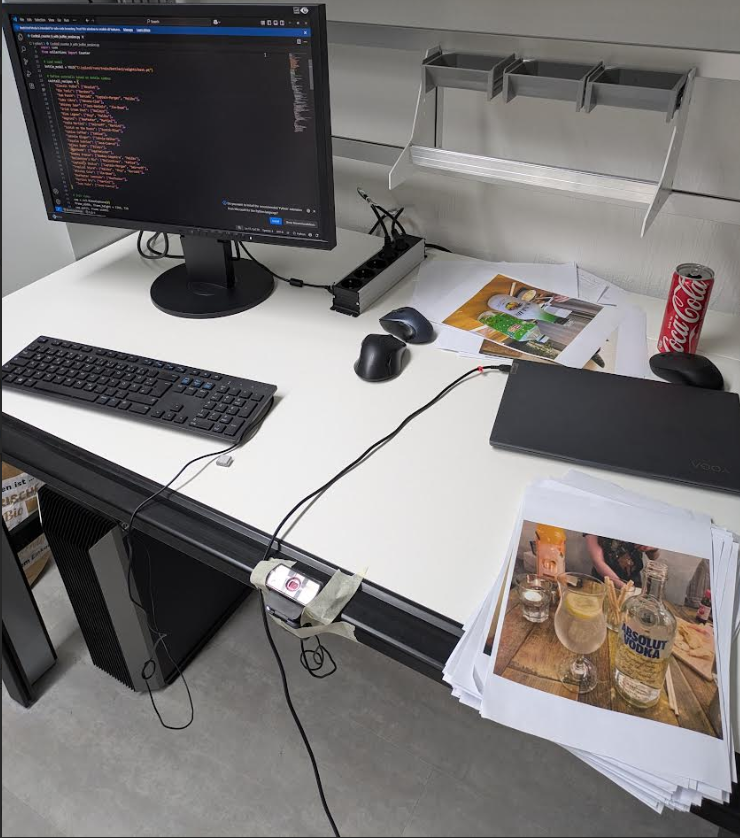


Figure 34 Experimental set-up for simple test.

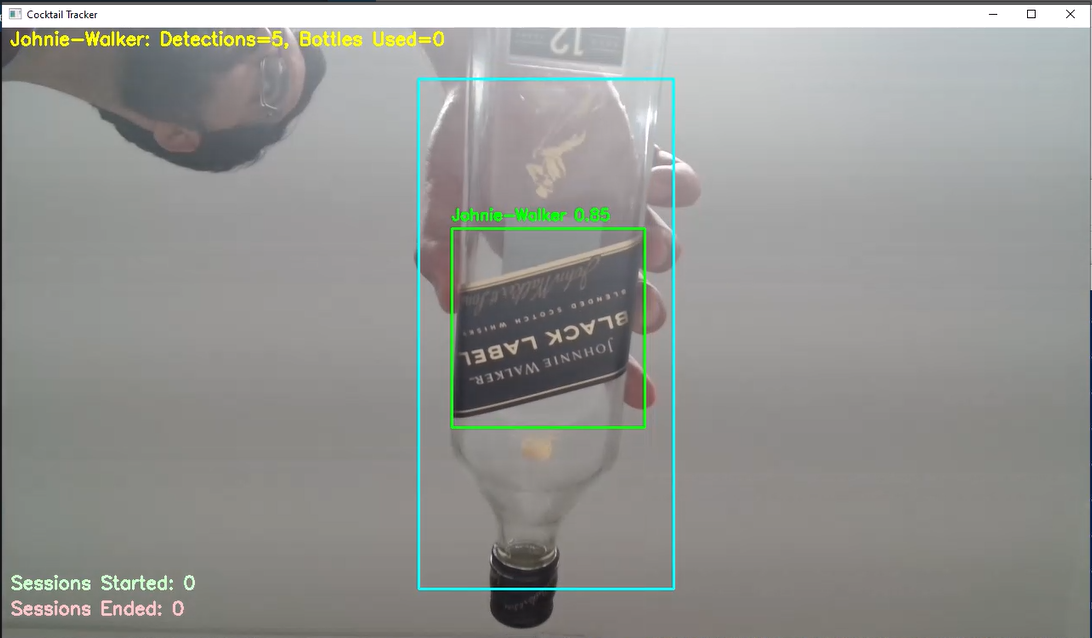


Figure 35 Example of detection with upward position

## Model I

From 100 printed images, Model I had a detection rate of approximately 90 percent with 70 percent accuracy, as with the previous tests with the digital version of the image batch, the classes Absolut, Beefeater, and Havana-Club show a higher tendency to be misclassified with brands that have similar attributes, such as font or label shape, maintaining the same rate of FP and FN as in the validation metrics. This underperforming tendency was also present in cocktail preparation, due to prevalent confusion between labels. Therefore, many recipes involving many bottles as ingredients were incorrect as a result of this misclassification. For simpler cocktails that required only one bottle as an ingredient, the results were more accurate, with the logging reflecting a 70 percent accuracy, consistent with the detection of individual bottles. Latency was also a bigger factor in this behavior, making this task more complicated to achieve.

## Model II

Model II, on the other hand, and contrary to the validation metrics of the previous detection task, managed to score a detection rate of up to 93 percent, with all detected printed images being correctly classified. Due to the fact that many of the bottles were primarily used for simple cocktails with a single ingredient, it was not possible to prepare multiple cocktails that required various bottles. Latency also affected Model II, and it impacted the preparation of the more elaborate cocktail.

In both tests, the session function that allows the recording of cocktails (start and end) functioned properly, without any latency, every time it was initialized. The function for estimating the number of bottles used after multiple detections with the Cooldown Time performs optimally. Still, its success rate was also linked to the amount of FP and FN, along with latency, as it missed recording instances when objects weren’t detected.

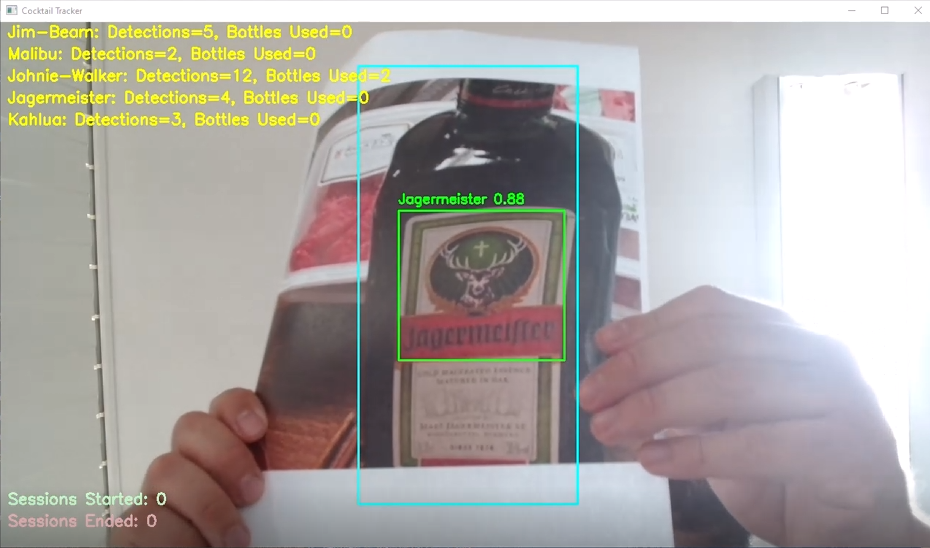


Figure 36 Example of simple test detection. Adapted from [25].

## Advanced Test

For the final deployment, the weights are best.pt results from Model I and Model II were converted to the best. onnx, using the CLI commands mentioned previously in this section, so that they can be used on the Linux Ubuntu environment. The same printed images from the previous experiment were also tested in this environment, with a higher focus on cocktail preparation. The models were also tested with a small number of real objects, in the form of bottles, to assess the behaviors of both models more accurately.

## Model I

Due to a greater latency presented in this environment, the model´s underperformance was even greater, with the detection rate dropping to 80 percent but maintaining the same accuracy rate, and the presence of inaccuracies with the same rate of FP and FN. Cocktail preparation was severely affected in this particular test, with many of the results yielding incorrect mixes and even the simplest ones being absent, due to the problems mentioned earlier related to latency. The model was tested with two real bottles of Skyy, with no detection; a bottle of Bombay Sapphire that produced a successful classification; and another bottle of Captain Morgan, which was successfully detected and classified.

## Model II

This model´s performance was also affected by latency. The detection rate also decreased slightly to around 89 percent, while maintaining the same level of detection. The logging of simple cocktails was successful, but the same behaviour seen in the previous test for more complicated mixes was also present in this experiment. Additionally, the model was tested to detect a bottle of Johnnie Walker, specifically Black Label, as shown in Figure 37, demonstrating good performance with accurate detection and classification. The same behaviour was present when detecting two bottles of Jägermeister. With a bottle of Absolut Vodka, there was no detection present.

As in the previous set of tests, the session system, performed satisfactorily and the bottle count as well, being optimally executed but the results of both functions, were again affected by the stronger latency, which severely affected the bottle counts and made the success rate for logging complicated cocktail mixes non-existent only being able to count the simpler ones.

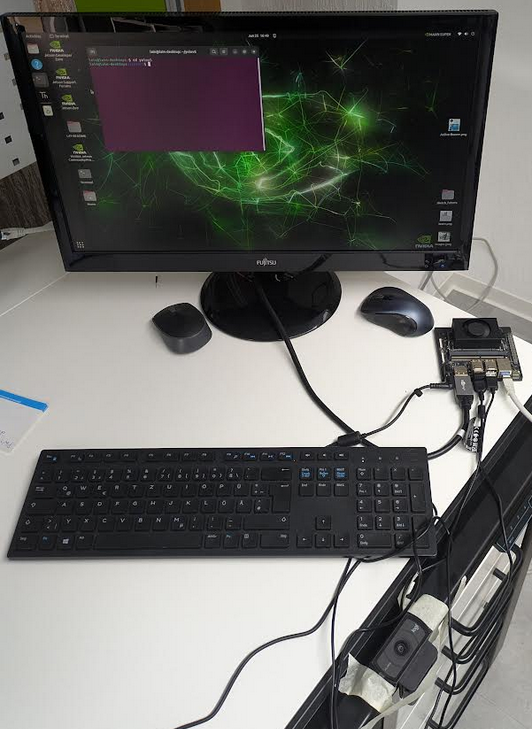


Figure 37 Experimental set-up for advanced test

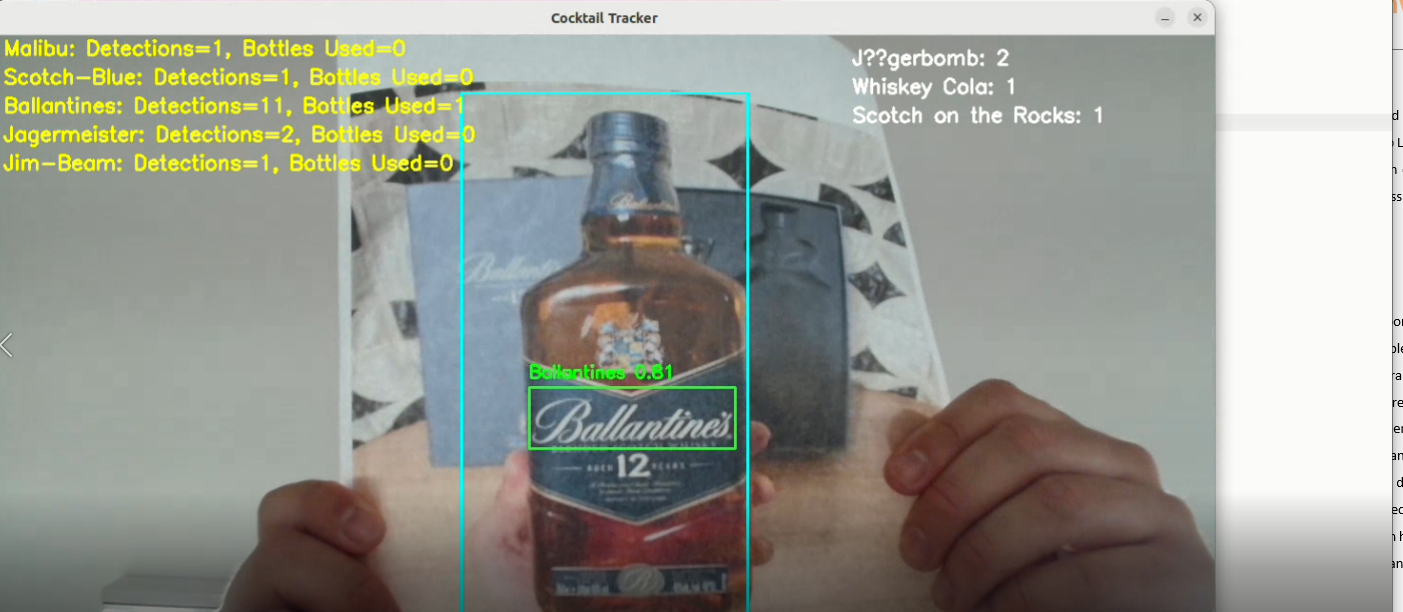


Figure 38 Example of advanced test with cocktail recipes. Adapted from [25].

The complete logs for all tests conducted during the experiment can be found on the project´s GitHub.

## Model Comparison and Analysis

Model I performed consistently during training and testing, showing substantial precision, recall, and detection rates. However, it struggled in realistic scenarios due to severe class confusion, which limited its practical use and reduced the chances of improvement through further training.

Model II, despite lower training and validation metrics, performed better in real-world conditions. This is likely due to a more diverse training dataset, which helped it develop better class distinction under harsher conditions. Model I, in contrast, may have overfitted to ideal scenarios and suffered from mislabeled or empty class entries. Overall, Model II is better suited for deployment due to its real-world reliability and reduced need for retraining.

## Benefits and Advantages

The complete framework is upgradable, user-friendly, and adaptable to work with various devices and operating systems, making it affordable. It can operate with any kind of data the user wants to create a Deep Learning Model with. In addition, the existing YOLO platforms allow for adaptability to work with other newer versions, and the source code requires minimal adaption to this end. The software accessibility enables the use of other applications, even beyond the scope of this study.

## Limitations and Risks

While versatile, the system is confined to operating exclusively with YOLO and leaves little room for similar Deep Learning algorithms or frameworks. The weights don´t function as the more complete AI applications that can be continuously trained without the supervision needed during the training process, making the system's performance very intertwined with the user´s capacity to acquire and prepare the necessary data, as well as their technical knowledge of the training process and execution. This hinders the potential of this project to develop a product that can compete with other tools that rely on AI and are equally accessible and user-friendly. Furthermore, user errors are present throughout the entire pipeline, creating a potential risk that produces inaccuracies, which ultimately affect inventory costs for both large and small venues. In short, due to the system's requirements, being in the hands of an inexperienced user accustomed to products with better documentation, the system can be a liability rather than a helpful tool.

## Lessons Learned

The process for acquiring pre-labelled datasets must be meticulous, involving the manual verification of all images, particularly those used for training and validation. Combining datasets is a viable option, but it requires the same level of detail when organizing the file structure. Given that the purpose of a Real-Time Object Detection System is to operate under realistic conditions, it is insufficient to rely solely on training metrics and simulated tests. The models must be thoroughly examined and deployed under various conditions while modifying the parameters of the source code and continuously retraining the models. These actions are highly challenging due to the given time constraints. Therefore, it is essential to significantly improve every step of the pipeline so that the final product requires as few iterations as possible.

# Conclusion

While the primary objectives of this project, namely designing and deploying a compact, low-cost, and user-friendly Real-Time Object Detection System for cocktail and bottle logging in a bar, have been achieved, the broader vision for a fully robust and commercial product remains an open path for further development. This report establishes the essential framework for training, deploying, and evaluating YOLOv8-based detection models, providing a replicable structure for both technical and non-technical users to iterate and build upon.

Despite some inconclusive or uneven results, especially when comparing simulated and real-world environments, the system successfully demonstrates how foundational Deep Learning and Data Analysis concepts can be applied to practical, everyday business use cases, such as venues that require precise data collection to optimize their inventories. The performance of both Model I and Model II highlights the complexity of correlating training metrics with realistic situations, underscoring the need for more adaptive training sets, rigorous testing, and iterative fine-tuning. Additionally, it demonstrates the importance of refining the data gathering process involved in collecting high-quality images for training, which utilizes sophisticated and advanced algorithms such as YOLO.

Furthermore, this study highlights the ongoing evolution and democratization of Computer Vision and AI-driven technologies. From grassroots DIY setups to large-scale industrial solutions, the field continues to expand and diversify. This project makes a modest contribution to that evolution by demonstrating a viable and scalable framework for innovative logging systems, and by highlighting the crucial importance of accessible tools, reproducible workflows, and informed human oversight.

Aside from this remarkable trend, the project´s development also confirms the ever-growing process of Data Analysis applications in task automation and how these methods offer great potential to help either individual users or large-scale operations improve their services and logistics, such as this project´s main case study.

Ultimately, the groundwork established here can serve as a stepping stone for future iterations, enabling more accurate detection, more intelligent automation, and real-time insights that support both operational efficiency and strategic decision making in fast-paced environments.

# 

# Appendix

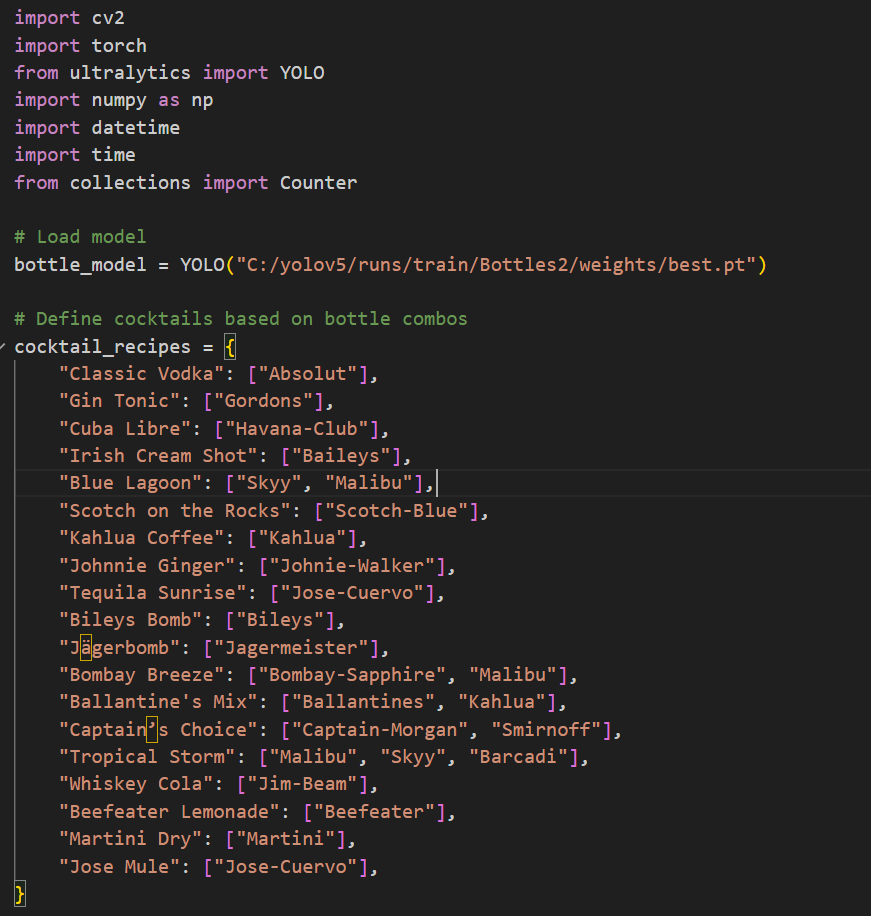


Figure 39 Source code I. The recipes were gathered from Difford´s Guide [47]

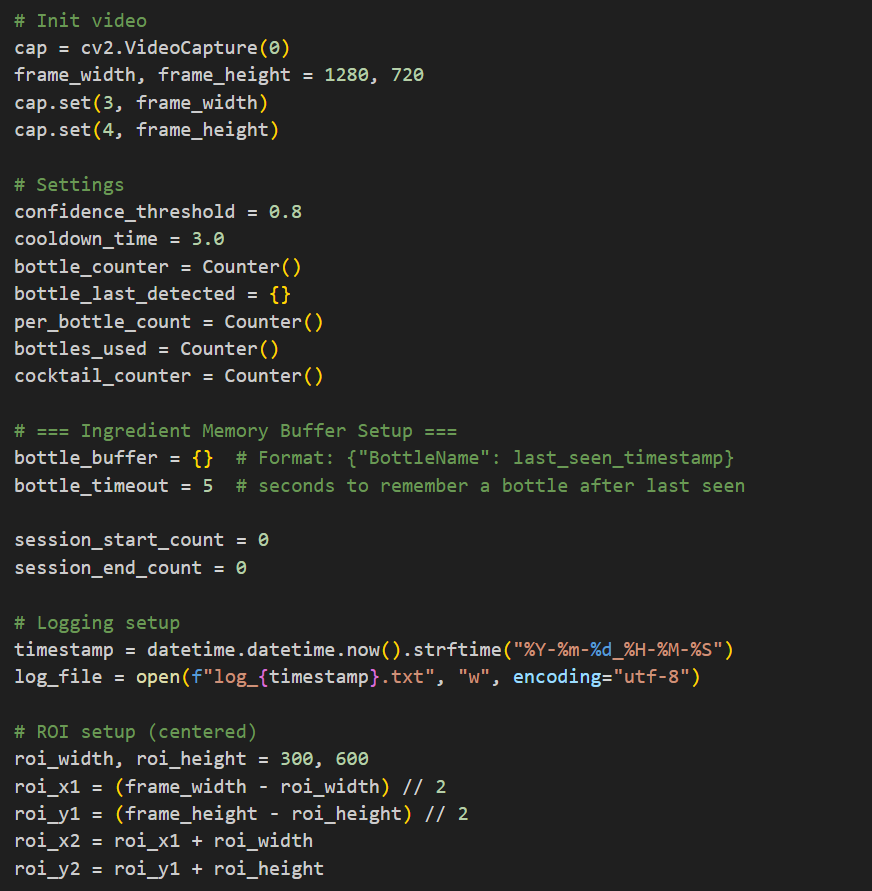


Figure Source code II

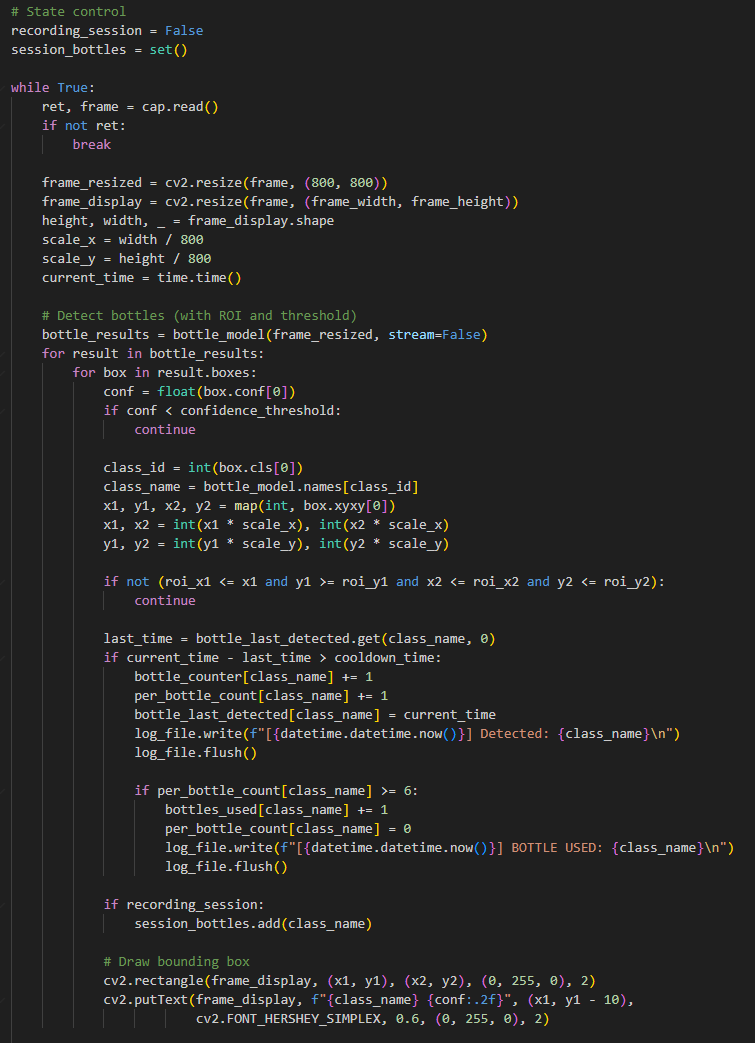


Figure Source code III

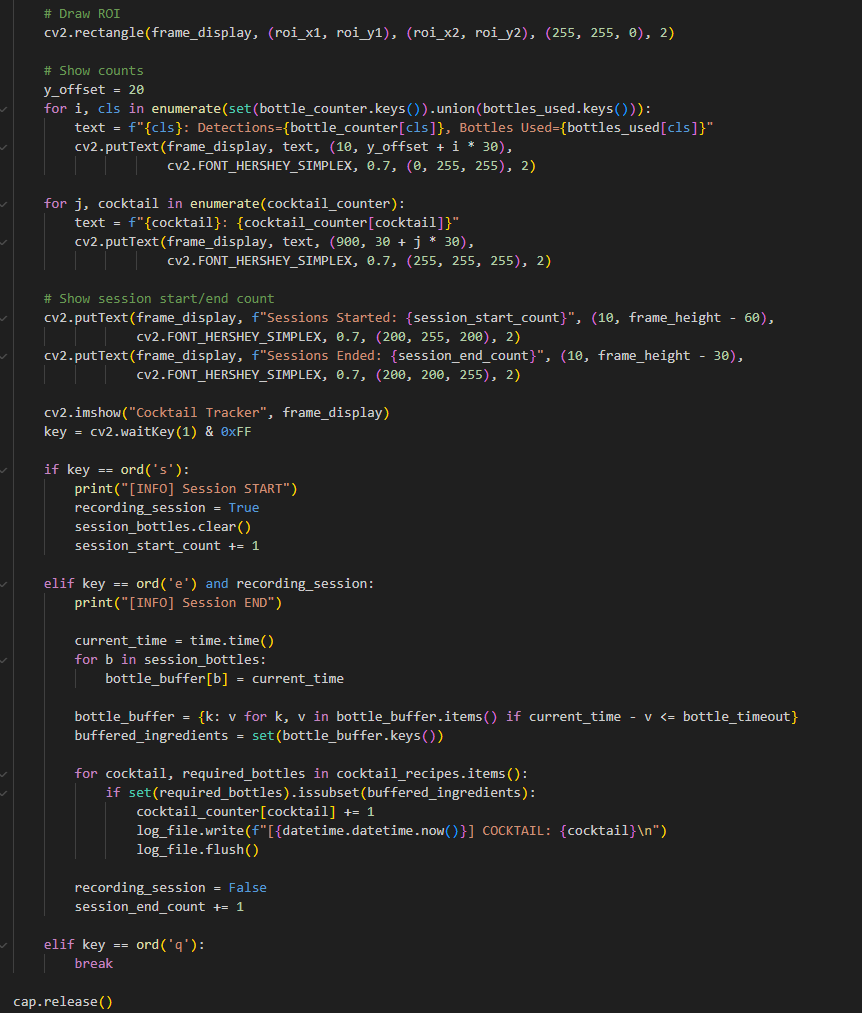


Figure 42 Source code IV

# Bibliography

[1] “Diwan et al. - 2023 - Object detection using YOLO challenges, architect.pdf.”

[2] “Preprint PDF.” Accessed: Nov. 27, 2024. [Online]. Available: http://arxiv.org/pdf/2304.00501v6

[3] “Cascella et al. - 2024 - Employing the Artificial Intelligence Object Detec.pdf.”

[4] “Full Text PDF.” Accessed: Nov. 29, 2024. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-981-13-6794-6.pdf

[5] “DEEP LEARNING, THE FINAL STAGE OF AUTOMATION AND THE END OF WORK (AGAIN)?,” *Psychosociological Issues Hum. Resour. Manag.*, vol. 5, no. 2, p. 154, 2017, doi: 10.22381/PIHRM5220176.

[6] M. L. Ali and Z. Zhang, “The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection,” *Computers*, vol. 13, no. 12, Art. no. 12, Dec. 2024, doi: 10.3390/computers13120336.

[7] “Zaidi et al. - 2021 - A Survey of Modern Deep Learning based Object Dete.pdf.”

[8] X. Jiang, A. Hadid, Y. Pang, E. Granger, and X. Feng, Eds., *Deep Learning in Object Detection and Recognition*. Singapore: Springer, 2019. doi: 10.1007/978-981-10-5152-4.

[9] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A Survey of Modern Deep Learning based Object Detection Models,” 2021, *arXiv*. doi: 10.48550/ARXIV.2104.11892.

[10] “Preprint PDF.” Accessed: Nov. 27, 2024. [Online]. Available: http://arxiv.org/pdf/1506.02640v5

[11] “YOLOv8 vs Mask R-CNN: In-depth Analysis and Comparison,” Labelvisor. Accessed: Dec. 03, 2024. [Online]. Available: https://www.labelvisor.com/yolov8-vs-mask-r-cnn-in-depth-analysis-and-comparison/

[12] “YOLOv8 vs Faster R-CNN: A Comparative Analysis.” Accessed: Nov. 27, 2024. [Online]. Available: https://keylabs.ai/blog/yolov8-vs-faster-r-cnn-a-comparative-analysis/

[13] “Full Text PDF.” Accessed: Mar. 24, 2025. [Online]. Available: https://www.mdpi.com/2073-431X/13/12/336/pdf?version=1734336408

[14] Ultralytics, “YOLOv8.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/models/yolov8

[15] “YOLOv8: A Complete Guide - viso.ai.” Accessed: Jun. 26, 2025. [Online]. Available: https://viso.ai/deep-learning/yolov8-guide/

[16] Ultralytics, “YOLOv8 vs YOLOv6-3.0: A Technical Comparison.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/compare/yolov8-vs-yolov6

[17] “Jetson Orin Nano Developer Kit Getting Started Guide,” NVIDIA Developer. Accessed: Mar. 21, 2025. [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit

[18] “jetson-orin-datasheet-nano-developer-kit-3575392-r2.pdf.” Accessed: Mar. 21, 2025. [Online]. Available: https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2

[19] “Ultralytics · GitHub.” Accessed: Mar. 22, 2025. [Online]. Available: https://github.com/ultralytics

[20] Ultralytics, “YOLOv8 vs DAMO-YOLO: A Technical Comparison.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/compare/yolov8-vs-damo-yolo

[21] Ultralytics, “YOLOv5 vs. YOLOv8: A Detailed Comparison.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/compare/yolov5-vs-yolov8

[22] A. Ottaiano, “13 Types of Cocktail Glasses And Their Uses,” DrinksWorld. Accessed: Jun. 26, 2025. [Online]. Available: https://drinksworld.com/types-of-cocktail-glasses/

[23] A. Gibson, “5 Creative Cocktail Garnish Ideas for Your Restaurant,” Food Supplier & Distributor | Restaurant Supply. Accessed: Jun. 26, 2025. [Online]. Available: https://www.usfoods.com/great-food/food-trends/5-creative-cocktail-garnish-ideas-for-your-restaurant.html

[24] “www Dataset > Overview,” Roboflow. Accessed: Jun. 26, 2025. [Online]. Available: https://universe.roboflow.com/hailid3111-gmail-com/www-nthwg

[25] “geonui Dataset > Overview,” Roboflow. Accessed: Jun. 26, 2025. [Online]. Available: https://universe.roboflow.com/new-workspace-gfhju/geonui

[26] “How to Train YOLOv8 Object Detection on a Custom Dataset,” Roboflow Blog. Accessed: Jun. 26, 2025. [Online]. Available: https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/

[27] “C920S PRO HD WEBCAM,” Logitech. Accessed: Jun. 26, 2025. [Online]. Available: https://www.logitech.com/en-us/shop/p/c920s-pro-hd-webcam

[28] N. Torabi, “Managing ML/AI projects for Product Leaders and Managers — Part 2: Managing ML projects,” Medium. Accessed: Jun. 26, 2025. [Online]. Available: https://neemz.medium.com/managing-ml-ai-projects-for-product-leaders-and-managers-part-2-managing-ml-projects-3f8d78462d22

[29] D. Karani, “Roles in ML Team and How They Collaborate With Each Other,” neptune.ai. Accessed: Jun. 26, 2025. [Online]. Available: https://neptune.ai/blog/roles-in-ml-team-and-how-they-collaborate

[30] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, “Operationalizing Machine Learning: An Interview Study,” Sep. 16, 2022, *arXiv*: arXiv:2209.09125. doi: 10.48550/arXiv.2209.09125.

[31] “Build the Perfect AI Team: A Guide to Ideal Team Structure.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.index.dev/blog/ideal-ai-team-structure

[32] R. Olivares, R. Noel, S. M. Guzmán, D. Miranda, and R. Munoz, “Intelligent Learning-Based Methods for Determining the Ideal Team Size in Agile Practices,” *Biomimetics*, vol. 9, no. 5, Art. no. 5, May 2024, doi: 10.3390/biomimetics9050292.

[33] “YOLOV8 Label Format: Step-by-Step Guide | YOLOV8.” Accessed: Jun. 26, 2025. [Online]. Available: https://yolov8.org/yolov8-label-format/

[34] “Weights & Biases,” W&B. Accessed: Jun. 26, 2025. [Online]. Available: httpss://wandb.ai/tensorgirl/yolov8-collect-and-label/reports/Collect-and-Label-Images-to-Train-a-YOLOv8-Object-Detection-Model--Vmlldzo2MzQ1Nzcw

[35] K. Naminas, “Object Detection Evaluation Metric Explained.” Accessed: Jun. 26, 2025. [Online]. Available: https://labelyourdata.com/articles/object-detection-metrics

[36] “Mean Average Precision (mAP) Explained: Everything You Need to Know.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.v7labs.com/blog/mean-average-precision

[37] Ultralytics, “YOLO Performance Metrics.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/guides/yolo-performance-metrics

[38] soumyadip, “YOLO Loss Function Part 2: GFL and VFL Loss.” Accessed: Jun. 26, 2025. [Online]. Available: https://learnopencv.com/yolo-loss-function-gfl-vfl-loss/

[39] “(20) Losses and Their Weights in Yolov8 | LinkedIn.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.linkedin.com/pulse/losses-weights-yolov8-dsaisolutions-x1ggf/

[40] ultralytics, “Is the explanation of DFL Loss correct? · Issue #21048 · ultralytics/ultralytics,” GitHub. Accessed: Jun. 26, 2025. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/21048

[41] Ultralytics, “CLI.” Accessed: Jun. 26, 2025. [Online]. Available: https://docs.ultralytics.com/usage/cli

[42] ultralytics, “Yolov8 Confusion Matrix FN, FP, TP, TN values · Issue #1325 · ultralytics/ultralytics,” GitHub. Accessed: Jun. 26, 2025. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/1325

[43] “Object Detection - Supervision.” Accessed: Jun. 26, 2025. [Online]. Available: https://supervision.roboflow.com/metrics/detection/

[44] “Understanding the Confusion Matrix in Machine Learning - GeeksforGeeks.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

[45] “Discussion on precision and recall values of YOLOv8 - Discussion,” Ultralytics. Accessed: Jun. 26, 2025. [Online]. Available: https://community.ultralytics.com/t/discussion-on-precision-and-recall-values-of-yolov8/66

[46] “How to Evaluate Deep Learning Models: Key Metrics Explained | DigitalOcean.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.digitalocean.com/community/tutorials/deep-learning-metrics-precision-recall-accuracy

[47] “Difford’s Guide - the home of discerning drinkers.” Accessed: Jun. 26, 2025. [Online]. Available: https://www.diffordsguide.com/

# 

# Affidavit

The author(s) hereby declare in their word of honour

* that they prepared this project report independently and without any outside help.
* that they used no sources or resources other than those indicated.
* that they marked any verbatim quotes and paraphrased text by other authors within the work where they appear.
* that they did not submit it elsewhere for examination purposes.

I am/we are fully aware that a false declaration will have legal consequences.

The English text in this document only serves the purpose of providing information   
on the contents of the corresponding German text.

Only the German version of this affidavit is legally binding.

Die Autoren erklären hiermit ehrenwörtlich,

* diesen Projektbericht selbstständig und ohne fremde Hilfe angefertigt,
* keine anderen als die angegebenen Quellen und Hilfsmittel benutzt,
* die Übernahme wörtlicher und sinngemäßer Zitate aus der Literatur an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet,
* die Arbeit mit gleichem Inhalt bzw. in wesentlichen Teilen noch nicht anderweitig für Prüfungszwecke vorgelegt zu haben.

Ich bin mir/wir sind uns bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Schweinfurt,27.06.2025 

Ort, Datum Unterschrift