Dissertation/Report submitted to the

UNIVERSITY OF MOHAMED BOUDIAF – M'SILA, ALGERIA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

in partial fulfillment of the requirements for the degree of

Master's in Computer Science

Specialization: Computer Science

Option: Artificial Intelligence and Computer Vision

Presented by

LAOUFI Abderrahmane

Entitled

End-to-End YOLO-Based Object Detection: From Model Training to Deployment

Under the supervision of

MOKHTARI Rabah

Jury Members

Dr. Bounif Mohamed	University of M'sila	President
Dr. Kamel Mohamed	University of M'sila	Examiner

June, 2026

Abstract

This dissertation studies an end-to-end YOLOv8-based object detection workflow, from computer vision foundations and dataset preparation to model comparison and an illustrative application prototype. Two experiments were conducted. The first experiment trained YOLOv8 variants for ship detection in maritime images, while the second used a custom 64-image egg dataset annotated manually with LabelImg. The models were compared using Precision, Recall, mAP@0.5, and mAP@0.5:0.95. In both experiments, YOLOv8 large was selected according to the stricter mAP@0.5:0.95 criterion, although the ship experiment also showed that YOLOv8 medium remained competitive on mAP@0.5 and Recall. The dissertation also presents GrabAndGo as an illustrative prototype using a pretrained YOLO-style detector to show how object detection can support cashierless shopping workflows. This application is included as an example of possible deployment, not as the main objective of the research or as a production-ready retail system. The work highlights the importance of dataset quality, annotation consistency, metric-based model selection, and careful system-level framing when applying object detection models.

Keywords: YOLOv8, object detection, ship detection, egg detection, cashierless shopping.

Résumé

Ce mémoire étudie un flux de travail de détection d'objets basé sur YOLOv8 de bout en bout, depuis les fondements de la vision par ordinateur et la préparation des jeux de données jusqu'à la comparaison des modèles et la présentation d'un prototype d'application illustratif. Deux expériences ont été menées. La première expérience a entraîné des variantes de YOLOv8 pour la détection de navires dans des images maritimes, tandis que la seconde a utilisé un jeu de données personnalisé de 64 images d'œufs annotées manuellement avec LabelImg. Les modèles ont été comparés à l'aide de la précision, du rappel, de mAP@0.5 et de mAP@0.5:0.95. Dans les deux expériences, YOLOv8 large a été sélectionné selon le critère plus strict mAP@0.5:0.95, bien que l'expérience de détection des navires ait également montré que YOLOv8 medium restait compétitif sur mAP@0.5 et le rappel. Le mémoire présente également GrabAndGo comme un prototype illustratif utilisant un détecteur préentraîné de type YOLO afin de montrer comment la détection d'objets peut soutenir des flux de travail de magasinage sans caisse. Cette application est incluse comme exemple de déploiement possible, et non comme objectif principal de la recherche ni comme système de vente au détail prêt pour la production. Le travail met en évidence l'importance de la qualité des données, de la cohérence des annotations, de la sélection des modèles fondée sur les métriques et d'un cadrage rigoureux au niveau du système lors de l'application de modèles de détection d'objets.

Mots-clés : YOLOv8, détection d'objets, détection de navires, détection d'œufs, magasinage sans caisse.

الملخص

تدرس هذه الأطروحة سير عمل متكامل لاكتشاف الكائنات مبني على نموذج YOLOv8، بدءاً من أسس الرؤية الحاسوبية وإعداد مجموعات البيانات، وصولاً إلى مقارنة النماذج ونموذج تطبيقي توضيحي. أُجريت تجربتان في هذا السياق؛ تناولت التجربة الأولى تدريب متغيرات YOLOv8 للكشف عن السفن في الصور البحرية، في حين اعتمدت التجربة الثانية على مجموعة بيانات مخصصة مؤلفة من 64 صورة للبيض، جرى تعليقها يدوياً باستخدام أداة LabelImg. وقد قُورنت النماذج وفق مقاييس الدقة والاستدعاء ومتوسط الدقة المتوسط mAP@0.5 وmAP@0.5:0.95. وفي كلتا التجربتين، جرى اختيار نموذج YOLOv8 الكبير استناداً إلى معيار mAP@0.5:0.95 الأكثر صرامة، وإن كانت تجربة السفن قد كشفت أيضاً أن نموذج YOLOv8 المتوسط ظل منافساً من حيث مقياسَي mAP@0.5 والاستدعاء. كما تقدم الأطروحة نظاماً باسم GrabAndGo بوصفه نموذجاً أولياً توضيحياً يستعين بكاشف بأسلوب YOLO مدرب مسبقاً، لإظهار كيف يمكن لاكتشاف الكائنات أن يدعم سير العمل في بيئات التسوق بدون كاشيرين. ويُدرج هذا التطبيق كمثال على إمكانيات النشر الفعلي، لا بوصفه هدفاً رئيسياً للبحث أو نظاماً جاهزاً للإنتاج في بيئة التجزئة. ويُسلط البحث الضوء على أهمية جودة البيانات، واتساق التعليقات التوضيحية، واختيار النماذج وفق المقاييس، والصياغة الدقيقة على مستوى النظام عند تطبيق نماذج اكتشاف الكائنات.

الكلمات المفتاحية: YOLOv8، اكتشاف الأجسام، اكتشاف السفن، اكتشاف البيض، التسوق دون أمين صندوق.

Dedication

"I am deeply grateful to my parents, family, and friends, whose unwavering support, encouragement, and love have played an instrumental role in my journey of completing this thesis. Their constant belief in me, even during the most challenging times, has been a never-ending source of inspiration and motivation. I owe them everything for their guidance, sacrifice, and faith in my abilities. This achievement would not have been possible without their love and steadfast support. With heartfelt appreciation and immense love, I dedicate this work to them, acknowledging their immeasurable impact on my life"

Acknowledgements

I would like to extend my sincere gratitude to my supervisor Professor Rabah Mokhtari, without whom I would've been lost, and whose support, inspiration, and guidance has led me through my studies and project with ease. I thank him for his efforts, and the enormous help he provided along the way.

I would also like to extend my gratitude to the University of ``Mohamed Boudiaf -- M'sila" for providing me with the countless opportunities, necessary education, and support to build my future.

Lastly, I am grateful to the jury members, Dr. Kamel Mohamed and Dr. Bounif Mohamed, for taking their valuable time to evaluate my project, and for their feedback.

Contents

List of Tables

List of Figures

Introduction

Computer vision is a central branch of artificial intelligence because it enables machines to extract structured information from images and video streams. Within this field, object detection is especially important because the system must not only classify a visual object but also localize it with a bounding box. This capability is useful in focused tasks such as maritime monitoring, food-quality inspection, smart surveillance, and application prototypes that need visual feedback, but it does not by itself solve broad automation problems such as complete retail checkout [1], [2].

This dissertation focuses on an applied YOLO-based object-detection workflow. Its actual contribution consists of two YOLO experiments and one demonstration prototype. The first experiment investigates ship detection in sea images using a dataset prepared and trained in a Kaggle notebook environment. The second experiment studies egg detection using a custom dataset created and annotated manually. The third part presents the *GrabAndGo* desktop application as an illustrative prototype inspired by cashierless shopping workflows. This prototype connects a pretrained YOLO-style detector to a desktop interface, but it is included as a demonstration of integration rather than as a validated retail-automation solution.

The main objectives of this research are summarized as follows:

- explain the core concepts of computer vision and object detection that support YOLO-based systems;
- describe the practical workflow used to prepare datasets, annotate data, train multiple YOLOv8 variants, and compare their performance;
- analyze the ship detection and egg detection experiments and justify the final model selections;
- present the architecture, deployment choices, and limitations of the GrabAndGo demonstration prototype.

The adopted methodology follows an applied experimental approach. First, a review of computer vision, convolutional neural networks, object detection paradigms, YOLO architectures, and evaluation metrics is presented in Chapter 1. Next, the training workflow is documented in Chapter 2, including data collection, annotation, preprocessing, model configuration, training, validation, and optimization decisions summarized from the experiment notebooks. Then, Chapter 3 describes the design of the GrabAndGo prototype as a demonstration application that integrates a pretrained YOLO-based detector in a cashierless-shopping-inspired interface. The dissertation ends with a general conclusion that summarizes the main findings, boundaries, and future directions.

Overall, this work demonstrates a limited but coherent path from YOLO-based experimentation to prototype integration. The evidence comes from the ship and egg detection experiments, while GrabAndGo shows how a detector can be connected to a user-facing workflow. Therefore, the dissertation should be read as a study of YOLO experimentation plus prototype demonstration, not as proof of a general-purpose retail automation system.

Chapter 1

Artificial Intelligence and Computer Vision

Introduction

This chapter presents the theoretical background needed to understand the practical work of the dissertation. It starts with artificial intelligence as the broad field, then introduces machine learning and deep learning as the main computational approaches behind modern computer vision. After that, the chapter explains image representation, object detection, YOLO architecture, evaluation metrics, and the main application areas related to the proposed work.

1.1 Artificial Intelligence

Artificial intelligence (AI) is the broad computational field concerned with building systems that can interpret data, learn from experience, and support decision-making. In this dissertation, AI is not treated as a general philosophical topic, but as the foundation for visual perception systems that can recognize and localize objects in images.

The proposed work belongs to practical narrow AI: the system is designed for specific detection tasks rather than general reasoning. This focus is important because the later YOLO-based experiments depend on measurable visual outputs, including object classes, bounding-box locations, confidence scores, and inference behavior in application-oriented settings [1], [3].

1.2 Machine Learning

Machine learning provides the data-driven mechanism through which an AI system improves its predictions from examples. Instead of manually defining all visual rules, a model

learns statistical relationships from annotated data and applies them to unseen images. For object detection, this means learning from images paired with class labels and bounding boxes.

The experimental part of this dissertation is mainly based on supervised learning. The ship and egg datasets provide labeled examples, and the YOLO models use these annotations to optimize localization and classification performance. Other learning paradigms, such as unsupervised, semi-supervised, and reinforcement learning, are relevant in the wider field, but they are secondary to the supervised detection workflow used here [3].

1.2.1 Applications of Machine Learning

Machine learning is used in many domains, including fraud detection, recommendation systems, medical diagnosis, forecasting, speech recognition, and image analysis. In this dissertation, the most relevant application is computer vision: machine learning converts annotated visual data into models that can detect ships, eggs, or retail products under realistic imaging conditions.

1.3 Deep Learning

Deep learning is the machine learning form most closely connected to modern computer vision. It uses multi-layer neural networks to learn hierarchical representations from raw data. In image-based tasks, early layers capture simple patterns like edges and textures, while deeper layers combine them into object parts and higher-level semantic features [3], [4].

For this dissertation, deep learning is important because YOLO detectors rely on convolutional feature extraction, multi-scale feature fusion, and trainable detection heads. These components allow the model to process an image once, producing both object categories and bounding boxes. Therefore, the following sections move from general deep learning concepts toward computer vision, object detection, and the YOLOv8 architecture used in the experiments.

1.4 Computer Vision

Computer vision is the discipline that enables computers to extract meaning from visual data such as images, video streams, and camera feeds. Its goal is to transform raw pixels into structured information that can support recognition, localization, measurement, tracking, and decision-making. Classical computer vision relied heavily on handcrafted features and geometric rules, whereas modern computer vision is driven largely by deep learning models that learn features automatically from large datasets. In object detection tasks, the system must determine *what* the object is and *where* it is located inside the image, making the problem

more complex than image classification alone [3], [5].

1.5 Image Representation and Processing

Digital images are numerical representations of visual scenes. Before an image can be used by a learning system, it must be represented in a form that preserves useful spatial and color information while remaining computationally manageable. Each image is usually stored as a matrix of pixel values, where the position of each value describes its location and the value itself describes intensity or color. In grayscale images, one channel is sufficient to describe brightness, while color images commonly use three channels such as red, green, and blue. This structured representation allows algorithms to analyze edges, textures, shapes, and object boundaries. Processing operations such as resizing, normalization, filtering, and contrast adjustment help prepare the image for later analysis. For deep learning models, careful representation and preprocessing are essential because the quality and consistency of the input directly influence feature extraction, training stability, and detection accuracy.

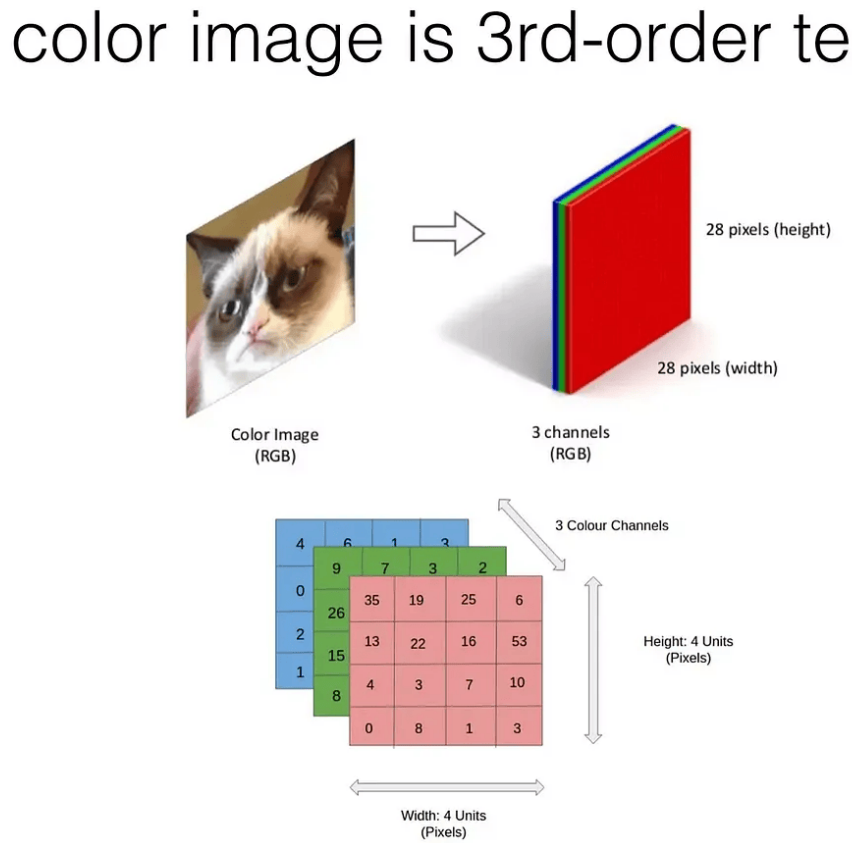


Figure 1.1: Image representation and processing interface illustrating how visual information is represented and prepared for computer vision workflows.

1.5.1 Pixels, Color Spaces, and Channels

A digital image is composed of pixels arranged on a rectangular grid. Each pixel stores an intensity value in grayscale images or multiple values in color images. The number of pixels defines the spatial resolution of the image, while the numeric range used to store each pixel determines how much visual detail can be represented. The most common color model is RGB, where each pixel is represented by red, green, and blue channels. Other color spaces such as HSV, LAB, or YCbCr are often used when illumination robustness or color separation is important. In practical vision pipelines, the selected color space must remain consistent between training and inference so that the model receives data in the same representation it learned from. Channel-based representation is particularly relevant in deep learning because convolution filters process spatial correlations across channels to extract informative patterns such as edges, textures, and semantic structures [5].

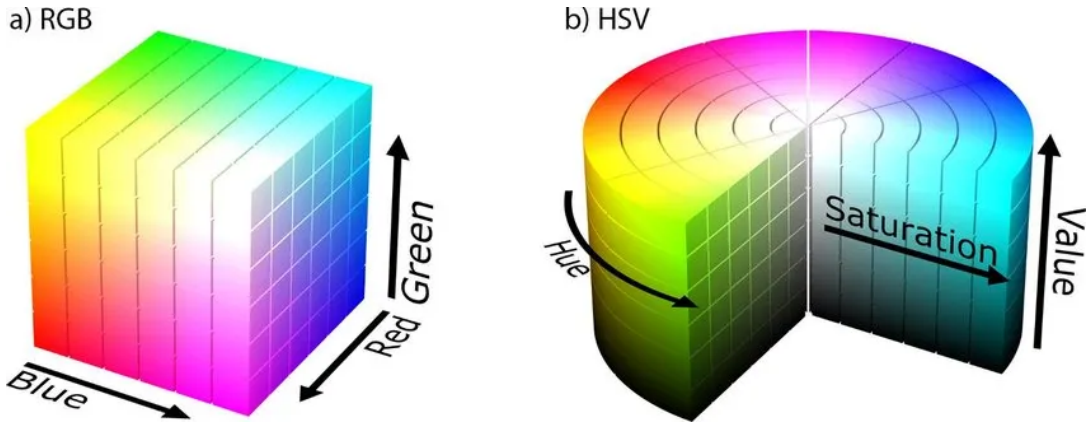


Figure 1.2: Pixels, color spaces, and channels illustrating how digital images are organized into spatial grids and color-channel representations.

1.5.2 Image Preprocessing Techniques

Image preprocessing improves data quality before training or inference. Common operations include resizing, normalization, denoising, contrast enhancement, sharpening, histogram equalization, and geometric adjustments such as cropping or padding. In learning-based pipelines, preprocessing helps standardize the input resolution, reduce irrelevant variation, and improve training stability. For object detection, preprocessing is also important because bounding-box coordinates must remain consistent after transformations.

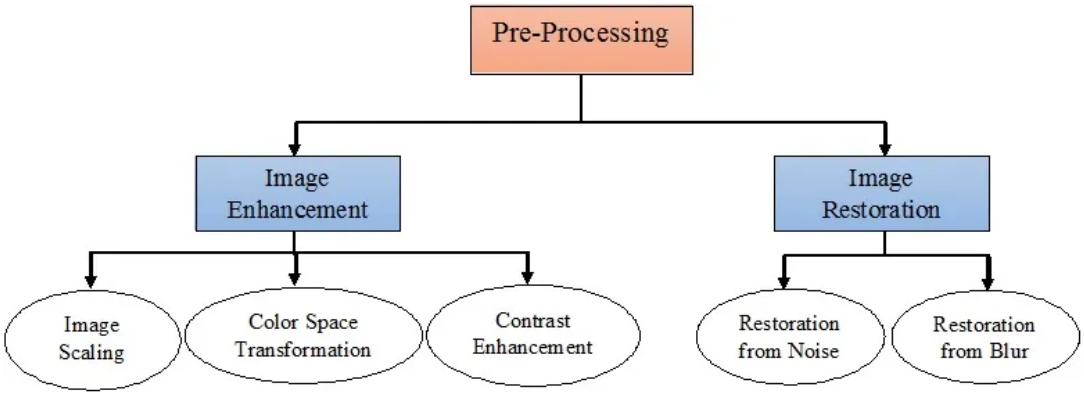


Figure 1.3: Image preprocessing techniques illustrating common operations used to prepare visual data before training or inference.

1.6 Deep Learning in Computer Vision

The general principles of deep learning are especially important in computer vision because visual tasks require models to learn spatial patterns from large numbers of pixels. In this context, convolutional architectures convert raw images into hierarchical feature maps that support recognition, localization, and object detection.

1.6.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks are specialized neural architectures designed for image data. They use convolution layers to learn local spatial patterns, pooling layers to reduce dimensionality, and nonlinear activations to model complex visual relationships. CNNs are particularly effective because they exploit local connectivity and weight sharing, which reduce the number of parameters while preserving spatial structure. Landmark results such as LeNet and later large-scale CNNs demonstrated that convolution-based learning became the foundation for modern detection, segmentation, and recognition systems [4], [6].

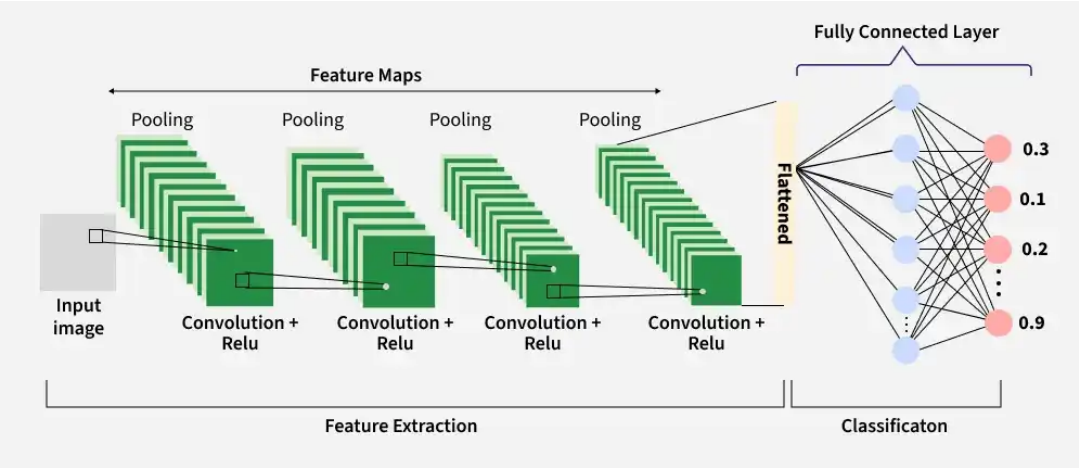


Figure 1.4: Convolutional neural network architecture illustrating convolution, pooling, and feature-extraction stages used in visual recognition tasks.

1.7 Object Detection Techniques

Object detection methods have evolved from handcrafted pipelines to deep end-to-end detectors. This evolution was driven by the need for better accuracy, better robustness, and eventually real-time performance. Unlike image classification, object detection must solve two tasks at the same time: identifying the object category and locating the object with a bounding box. Early approaches depended on manually designed features and sliding-window search strategies, which made them sensitive to scale changes, lighting variation, clutter, and occlusion. Modern deep learning detectors learn feature representations directly from annotated images and can detect multiple objects of different sizes in one scene. They also combine classification confidence with localization quality, making them more suitable for real-world applications such as surveillance, autonomous navigation, medical imaging, and smart retail systems. The development from traditional methods to CNN-based and YOLO-style detectors shows how object detection moved from slow, handcrafted recognition pipelines toward fast, trainable, and deployment-ready visual perception systems.

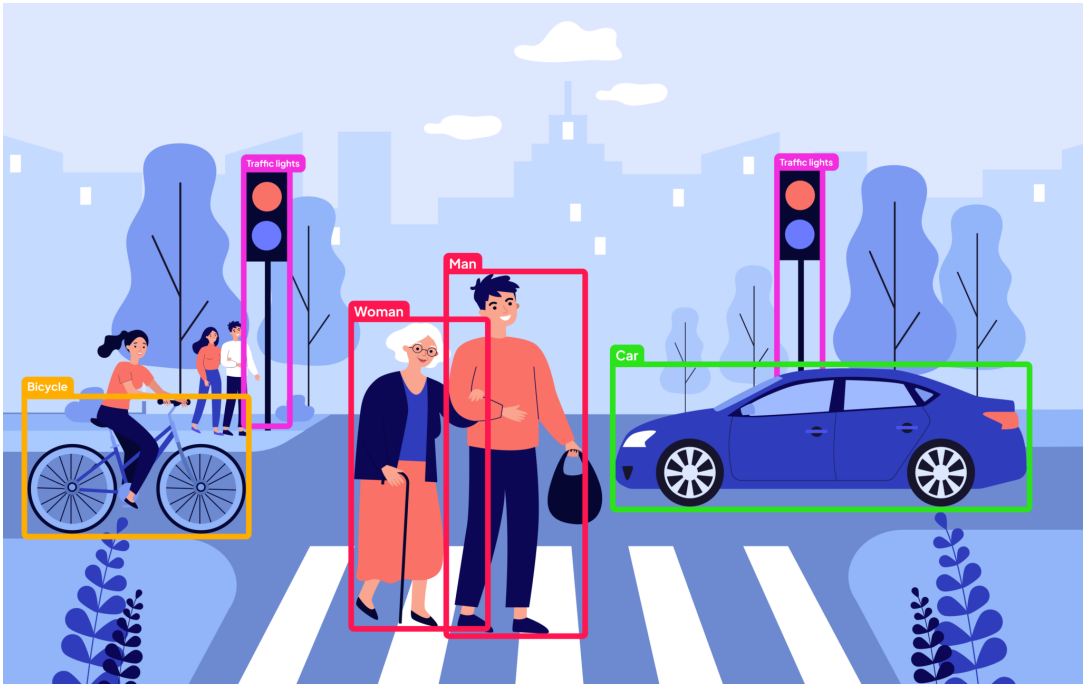


Figure 1.5: Object detection techniques illustrating the localization and classification of objects using bounding boxes in visual scenes.

1.7.1 Traditional Methods (HOG, Haar Cascades)

Traditional object detection methods relied on engineered features and classical classifiers. Haar cascade detectors used intensity differences over rectangular regions combined with boosted classifiers for fast face and object detection [7]. Histogram of Oriented Gradients (HOG) descriptors captured local edge orientation statistics and were often paired with support vector machines for pedestrian and object detection [8]. These approaches were influential, but their accuracy and adaptability were limited when scenes became complex, cluttered, or highly variable.

Figure 1.6: Traditional object detection methods illustrating Haar cascade-style feature extraction and classical detection behavior.

1.7.2 Deep Learning-Based Detection

Deep learning-based detectors replaced manual feature engineering with learned visual representations. By jointly learning features and decision boundaries from annotated examples, these systems achieved much stronger generalization across scale changes, occlusion, appearance variation, and background clutter. This shift established a new state of the art and opened the door to industrial object detection systems.

1.7.3 Two-Stage Detectors (R-CNN Family)

Two-stage detectors first generate candidate object regions and then classify and refine them. The R-CNN family began with region proposals followed by CNN-based classification, then evolved through Fast R-CNN and Faster R-CNN to achieve better efficiency and end-to-end training [9], [10], [11]. Mask R-CNN later extended this family by adding a mask-prediction branch for instance segmentation while preserving the detection framework [12]. These models are known for strong accuracy, especially when precise localization is needed, but they are usually slower than one-stage methods.

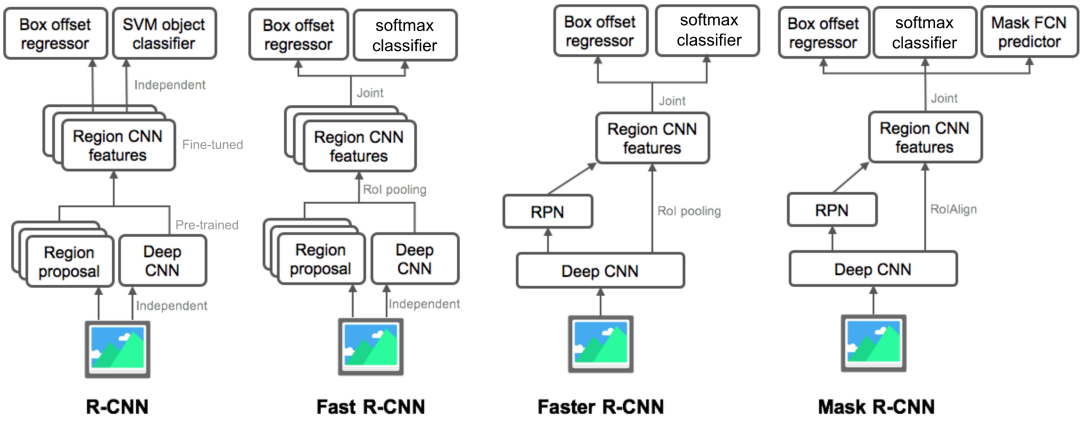


Figure 1.7: R-CNN family overview illustrating the progression from R-CNN to Fast R-CNN, Faster R-CNN, and Mask R-CNN.

1.7.4 One-Stage Detectors

One-stage detectors predict class probabilities and bounding boxes directly from the image in a single pass. This design removes the explicit proposal stage and significantly improves inference speed. Instead of examining a small set of candidate regions, these models make dense predictions over grid cells or feature-map locations and then filter the results using confidence thresholds and non-maximum suppression. Popular one-stage families include SSD and YOLO [13]. Their main advantage is real-time operation, which is crucial for applications such as retail monitoring, autonomous systems, and surveillance. However,

because localization and classification are solved at the same time, one-stage detectors depend strongly on multi-scale feature extraction, balanced training losses, and high-quality annotations to maintain accuracy in crowded or small-object scenes.

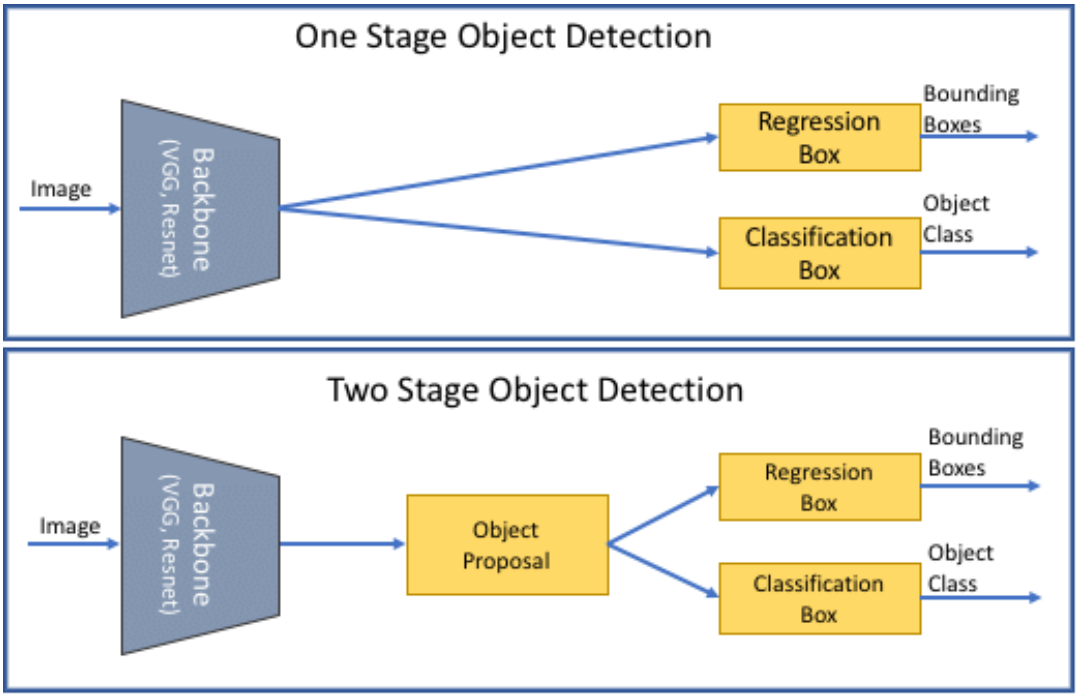


Figure 1.8: Comparison between one-stage and two-stage detector workflows, highlighting the direct-prediction design of one-stage methods.

1.7.5 SSD (Single Shot Detector)

SSD (Single Shot Detector) is an important one-stage detector because it predicts object classes and bounding boxes in a single forward pass [13]. It uses multiple feature maps at different scales so that objects of different sizes can be detected more effectively. SSD also relies on default boxes, or anchor boxes, as reference templates during localization and classification. Compared with two-stage methods, it offers better speed while preserving competitive accuracy on many benchmarks. For this reason, SSD is often considered a major step between early one-stage detectors and more advanced YOLO-based systems.

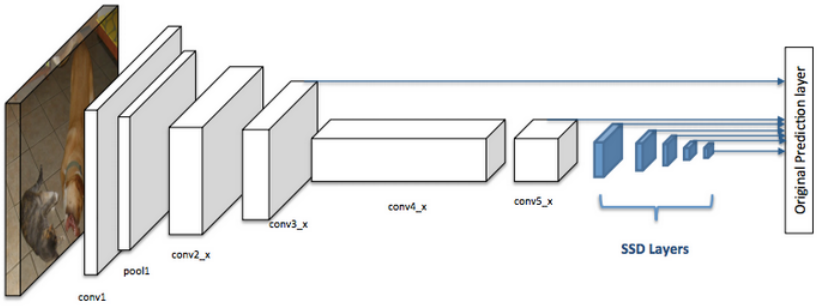


Figure 1.9: SSD architecture illustrating the single-shot object detection workflow with multi-scale feature maps and direct bounding-box prediction.

1.8 YOLO (You Only Look Once) Architecture

YOLO reformulated object detection as a one-stage problem in which object localization and classification are predicted directly from image features in a single forward pass [2]. This design is important for the dissertation because the experimental workflow compares YOLOv8 variants on custom ship and egg datasets, where both detection accuracy and practical inference behavior matter. A modern YOLO detector is usually organized into three connected parts: a backbone that extracts visual features, a neck that fuses features across scales, and a head that predicts bounding boxes and class scores.

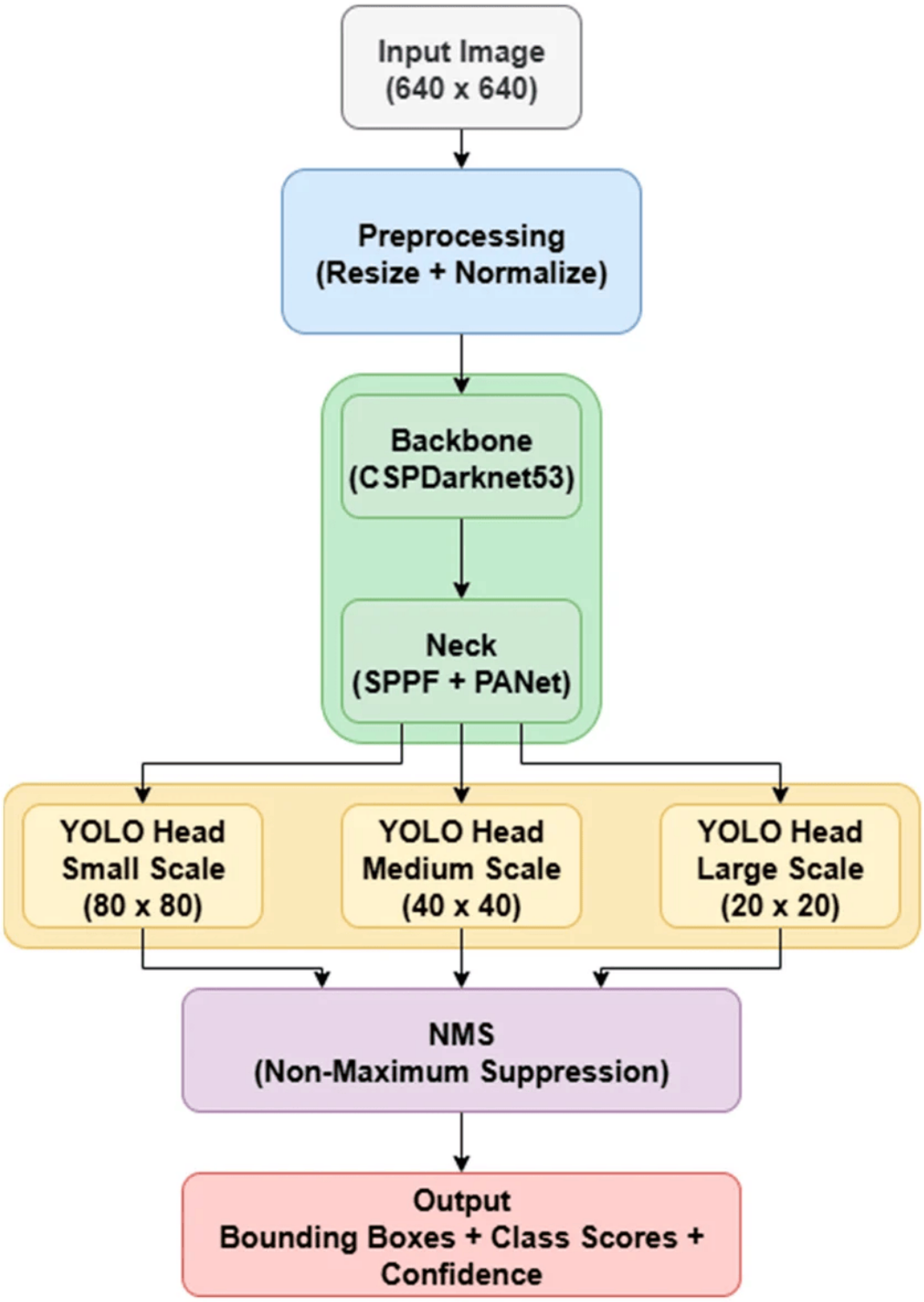


Figure 1.10: Basic YOLO architecture illustrating the main stages of feature extraction and detection prediction in a one-stage object detector.

1.8.1 Evolution of YOLO Models

The YOLO family evolved from the original unified detector toward architectures with stronger feature extraction, better multi-scale prediction, and easier deployment. YOLOv3 improved feature learning and prediction at multiple scales, while later versions such as YOLOv4 and the Ultralytics YOLO ecosystem improved training strategies, feature aggregation, and usability [14], [15], [16]. In this dissertation, YOLOv8 is the main experimental detector because it provides pretrained variants, an accessible training interface, validation tools, and export options within the same framework.

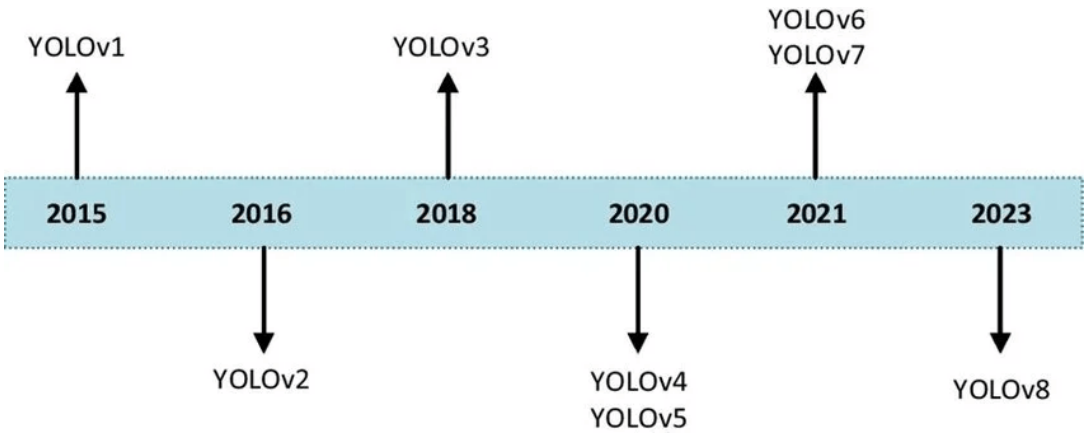


Figure 1.11: Evolution of YOLO models showing the progression of YOLO-based detectors across major versions and architectural improvements.

1.8.2 YOLOv8 Architecture in This Dissertation

YOLOv8 keeps the real-time, one-stage detection philosophy but introduces architectural changes that are directly relevant to training on custom datasets. The backbone and neck extract and fuse features at different spatial resolutions, allowing the detector to respond to objects of different sizes. The detection head then produces localization and classification outputs from these fused feature maps. For the ship dataset, this multi-scale behavior is useful because vessels may appear at different distances and image scales. For the egg dataset, it is useful because the detector must separate compact objects from backgrounds that may contain shadows, reflections, or similar textures.

The YOLOv8 design also supports the comparison of different model sizes, such as small, medium, and large variants. Larger variants generally provide more representational capacity, while smaller variants are more efficient. Therefore, the architecture must be understood not only as a network diagram, but also as a set of design choices that affect the accuracy--speed trade-off measured in the experiments.

1.8.3 Anchor-free Detection Design

Earlier YOLO versions and other one-stage detectors commonly used anchor boxes: predefined bounding-box templates with specific scales and aspect ratios that guide the model during prediction [13], [14], [15]. This approach can be effective, but it introduces an additional design dependency because anchors may need to match the distribution of object sizes in the dataset.

YOLOv8 uses an anchor-free split detection head [16]. Instead of predicting offsets relative to predefined anchor-box templates, it predicts boxes from feature-map locations, or anchor points. This reduces the need for dataset-specific anchor tuning and simplifies training when object sizes vary across datasets. In the context of this dissertation, anchor-free prediction is relevant because the ship and egg datasets differ strongly in object scale, background complexity, and visual appearance. A design that does not depend on manually selected anchor boxes is therefore more convenient for comparing YOLOv8 variants under the same experimental protocol.

1.8.4 C2f Module for Efficient Feature Extraction

YOLOv8 uses the C2f module, a faster Cross Stage Partial-style bottleneck with two convolutions, as a core feature-extraction block [16]. Its purpose is to improve feature reuse and gradient flow while keeping computation efficient. In practice, this matters because object detection requires both low-level detail, such as edges and textures, and higher-level semantic patterns, such as object shapes.

Compared with heavier feature-extraction blocks, C2f helps the network maintain a useful balance between accuracy and computational cost. This is important for the dissertation experiments because the selected YOLOv8 model should not only obtain high mAP values, but also remain practical for a deployment-oriented application. Efficient feature extraction is especially relevant when comparing model sizes, because the large model may improve accuracy while the small or medium model may provide better speed and lower resource demand.

1.8.5 Task-aligned Assignment During Training

During training, the detector must decide which predictions should be treated as positive matches for each ground-truth box. Older strategies often relied more heavily on fixed spatial rules, anchor matching, or IoU thresholds. These strategies can create weak alignment between the classification objective and the localization objective, especially when several candidate predictions overlap the same object.

YOLOv8 uses task-aligned assignment in its detection loss implementation [16]. The task-aligned idea comes from Task Alignment Learning, which combines classification confidence and localization quality so that positive samples are selected according to how well they serve both tasks [17]. In practical terms, this encourages the model to learn from candidates that are not only close to the object but also reliable for the correct class. For the experiments in this dissertation, this is important because high mAP requires both correct class prediction and accurate box placement.

1.8.6 Decoupled Detection Head

YOLOv8 uses a split, decoupled detection head in which bounding-box regression and class prediction are processed through separate branches [16]. This differs from earlier coupled heads where localization and classification information were more tightly mixed. Separating these branches is useful because the two tasks require different information: localization benefits from precise spatial features, while classification benefits from semantic category features.

For custom object detection, the decoupled head helps the model optimize box placement and class confidence more independently. This is directly connected to the dissertation's evaluation metrics. Precision and recall depend on correct class decisions and confidence thresholds, while IoU and mAP depend strongly on localization quality. A decoupled head therefore supports the balanced evaluation used later when comparing YOLOv8 model variants.

1.8.7 YOLO Detection Pipeline

The YOLOv8 detection pipeline begins with image preprocessing, including resizing and normalization. The image is then passed through the backbone and neck to obtain multi-scale feature maps. The decoupled, anchor-free head predicts bounding-box distributions and class scores from these feature maps. During training, task-aligned assignment selects useful positive samples and the loss function updates the model. During inference, predictions are decoded, confidence filtering is applied, and non-maximum suppression removes redundant overlapping boxes.

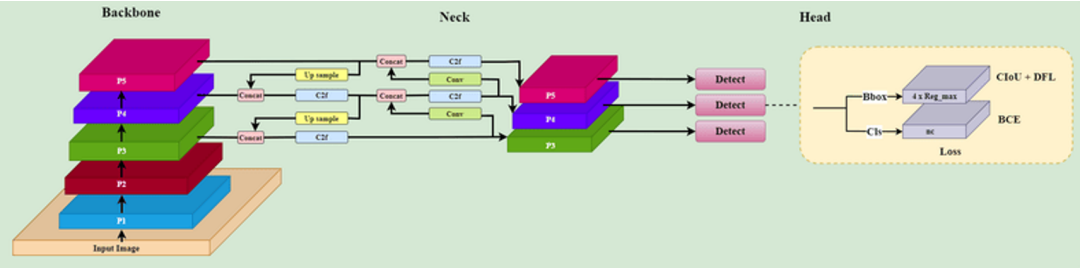


Figure 1.12: YOLOv8 object-detection pipeline showing the main architecture stages used to extract features and generate detection predictions.

1.8.8 Advantages and Limitations

YOLOv8 offers several advantages for the work in this dissertation: fast one-stage inference, pretrained model variants, anchor-free prediction, efficient C2f-based feature extraction, task-aligned training assignment, and a decoupled head for classification and localization. These properties make it suitable for comparing object detection performance on the selected datasets and for later integration into the GrabAndGo-style application.

However, YOLOv8 still depends strongly on annotation quality, class balance, image diversity, and suitable training configuration. Small objects, crowded scenes, motion blur, shadows, and visually similar backgrounds can still reduce detection quality. For this reason, the experimental chapters evaluate the models using several metrics rather than relying on accuracy alone.

1.9 Evaluation Metrics for Object Detection

Reliable evaluation is essential when comparing detection models because a detector must balance localization quality and classification quality. In image classification, evaluation mainly checks whether the predicted class is correct. In object detection, however, the model must correctly identify the object class and also place the bounding box close enough to the real object location. Therefore, a prediction is considered useful only when it satisfies both conditions: the class label is correct and the overlap between the predicted box and the ground-truth box is acceptable.

Object detection evaluation is usually based on matching predicted boxes with annotated ground-truth boxes. Each prediction has a class label, a confidence score, and bounding-box coordinates. According to the selected confidence threshold and IoU threshold, predictions are counted as true positives, false positives, or false negatives. These counts are then used to compute metrics such as Precision, Recall, F1-score, IoU, and mAP. Using several metrics together gives a more complete view of detector performance than relying on a single score.

For the experiments in this dissertation, evaluation metrics are important because YOLO models may behave differently depending on the dataset, object size, background complexity, and confidence threshold. A model with high precision may produce few false detections but miss some objects, while a model with high recall may detect more objects but also generate more false alarms. For this reason, the evaluation process must analyze both detection correctness and localization quality.

1.9.1 Confidence Score and Detection Thresholds

The confidence score expresses how certain the model is about a predicted object. During inference, YOLO produces many candidate bounding boxes, and each candidate is associated with a confidence value. A confidence threshold is then used to remove weak predictions. If the threshold is too high, the model may ignore correct detections with moderate confidence. If the threshold is too low, the model may keep many incorrect detections. Selecting a suitable threshold is therefore important for balancing false positives and missed objects.

Detection evaluation also depends on the IoU threshold used to decide whether a predicted box matches a ground-truth box. For example, a prediction may have the correct class but still be rejected if its bounding box is too far from the annotated object. This shows why confidence and localization must be interpreted together when evaluating object detectors.

1.9.2 Intersection over Union (IoU)

Intersection over Union measures the overlap between a predicted bounding box and the ground-truth bounding box. It is computed as the ratio between the area of intersection and the area of union. A higher IoU indicates better localization because it means that the predicted box covers the real object more accurately. If the predicted box and the ground-truth box overlap perfectly, the IoU value is equal to 1. If they do not overlap, the IoU value is equal to 0.

IoU is one of the most important metrics in object detection because it connects prediction quality with spatial accuracy. It is often used to determine whether a predicted detection should be counted as a true positive. For example, if the IoU threshold is set to 0.5, the predicted box must overlap the ground-truth box by at least this amount to be accepted as a correct localization. The metric is expressed as:

$$\mathrm{IoU} = \frac{\mathrm{Area}(B_p \cap B_{gt})}{\mathrm{Area}(B_p \cup B_{gt})} \tag{1.1}$$

where B_p is the predicted box and B_{gt} is the ground-truth box.

1.9.3 Precision, Recall, F1-score

Precision measures how many predicted detections are correct, while recall measures how many true objects are successfully detected. Precision is important when false alarms are costly, such as in security monitoring or automatic checkout systems. Recall is important when missing objects is more serious, such as in safety-critical detection or inspection tasks. In object detection, these metrics reveal different aspects of performance: a model may be precise but miss many objects, or it may detect many objects but generate more false positives.

F1-score combines precision and recall into a single harmonic mean. It is useful when a balanced comparison is needed between models, especially when neither false positives nor false negatives should dominate the evaluation. However, F1-score should still be interpreted together with IoU and mAP because it does not fully describe localization quality by itself. The metrics are computed as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1.2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{1.3}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1.4}$$

where TP, FP, and FN denote true positives, false positives, and false negatives.

1.9.4 Why Accuracy Was Not Used

Although accuracy is a fundamental metric in many machine learning tasks, it is not the most informative measure for object detection. In detection problems, the number of background regions is usually much larger than the number of true objects, which means a model can obtain a high accuracy score even when it fails to localize objects correctly. A detector may also predict the correct class but produce a poor bounding box, and simple accuracy would not adequately represent this localization error.

Accuracy also does not reflect bounding-box quality, overlap quality, confidence ranking, or the balance between missed detections and false alarms. For example, a model that misses several small objects may still appear acceptable if only global image-level correctness is considered. For this reason, object detection studies rely more on Precision, Recall, F1-score, IoU, and mAP, because these metrics evaluate both classification correctness and localization performance in a more meaningful way.

1.9.5 Mean Average Precision (mAP)

Mean Average Precision is one of the standard metrics for object detection. It summarizes the precision-recall behavior across classes and confidence levels. Average Precision (AP) is first computed for each class by measuring the area under the precision-recall curve. The mean value across all classes is then reported as mAP. This makes mAP useful because it evaluates how well the detector ranks its predictions and how consistently it performs across object categories.

Common reporting forms include mAP@0.5 and mAP@0.5:0.95. The mAP@0.5 metric uses an IoU threshold of 0.5 and is less strict, while mAP@0.5:0.95 averages performance across several IoU thresholds from 0.5 to 0.95. The second form is more demanding because it rewards detectors that not only find objects, but also localize them with higher precision. In practical model comparison, mAP@0.5:0.95 often provides the best overall indicator of balanced detector quality. For one class, the average precision is obtained from the area under the precision-recall curve, where $p(r)$ denotes precision as a function of recall r:

$$\mathrm{AP} = \int_0^1 p(r)\, dr \tag{1.5}$$

and the mean average precision over N classes is:

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i \tag{1.6}$$

1.10 Applications of Object Detection

Object detection has become a key enabling technology in many sectors because it provides direct spatial awareness of relevant entities in a scene. Unlike image classification, which assigns a single label to an entire image, object detection identifies both the category and the position of each object. This makes it suitable for applications where decisions depend not only on recognizing what appears in an image, but also on knowing where each object is located, how many objects are present, and how they relate spatially to one another.

Modern detection systems are used in real-time camera feeds, mobile applications, industrial inspection platforms, autonomous systems, and decision-support tools. Their usefulness comes from the ability to convert unstructured visual data into measurable information such as object counts, bounding-box coordinates, confidence scores, and event alerts. In practical deployments, object detection can reduce manual inspection effort, improve response time, increase operational safety, and support automation in environments where visual monitoring is required.

The most important application areas for this dissertation are those that require fast and reliable visual recognition under realistic conditions. These include retail monitoring, surveillance, transportation, agriculture, food-quality inspection, and maritime or industrial monitoring. In each case, the detector must handle variations in lighting, object scale, camera angle, occlusion, background complexity, and image quality.

1.10.1 Retail and Smart Shopping Systems

In retail, object detection supports shelf monitoring, product counting, stock auditing, customer behavior analysis, and cashierless shopping systems. Real-world retail shelf images highlight the complexity of dense scenes where many products appear simultaneously, overlap visually, and must be localized accurately. Detection models can identify products on shelves, verify whether items are misplaced, estimate stock availability, and support automatic checkout by recognizing selected objects from camera input. This application area is directly relevant to the GrabAndGo prototype discussed in this dissertation, where visual detection contributes to a smoother shopping process and reduces the need for manual product entry.

Figure 1.13: Retail and smart shopping systems illustrating product detection in a dense retail environment.

1.10.2 Surveillance and Security

In surveillance and security, detection models identify people, vehicles, suspicious objects, or restricted events in real time. They can be used to monitor entrances, restricted areas, public spaces, parking zones, and industrial sites. By detecting objects continuously, these

systems can generate alerts, assist human operators, and support later video analysis. The ability to operate continuously and at low latency makes YOLO-style detectors particularly attractive for camera-based monitoring systems, especially when rapid response is required.

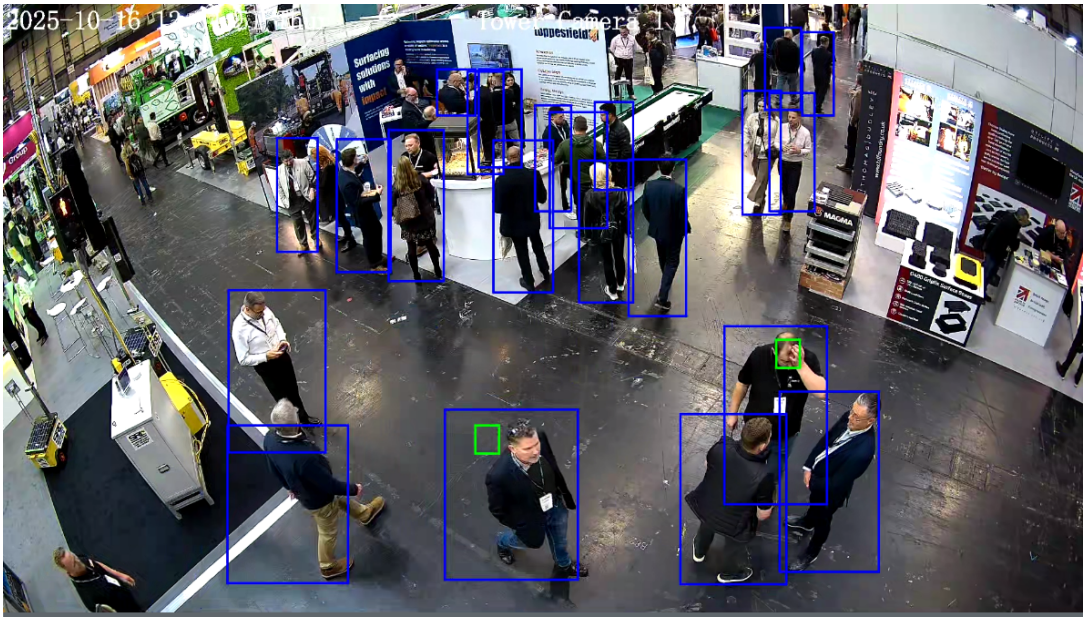


Figure 1.14: Retail CCTV surveillance example illustrating how camera-based monitoring can support object detection in security and retail environments.

1.10.3 Autonomous Vehicles and Transportation

In autonomous vehicles and intelligent transportation systems, object detection is used to identify pedestrians, vehicles, traffic signs, lanes, obstacles, and other road elements. These detections help the system understand the surrounding environment and support decisions such as braking, lane keeping, collision avoidance, and path planning. Because transportation scenes change quickly, detection models must combine high accuracy with fast inference, which makes real-time architectures such as YOLO especially useful in this application area [18].

1.10.4 Agriculture and Food Quality Inspection

In agriculture and food-quality inspection, object detection is used to recognize crops, fruits, pests, diseased leaves, animals, eggs, and other biological objects. It can support yield estimation, automated counting, quality grading, defect detection, and monitoring of production processes. For example, detection models can locate eggs in images, separate them from the background, and provide visual information that can later be used for counting or classification. These applications are important because agricultural and food images often

contain irregular shapes, variable lighting, natural backgrounds, and objects that differ in size or appearance.

1.10.5 Maritime and Industrial Monitoring

In maritime and industrial environments, object detection can identify ships, containers, safety equipment, workers, machines, and obstacles. Maritime scenes are challenging because objects may appear at long distances, under fog, reflections, waves, or changing illumination. Industrial scenes can also be complex because objects may be partially occluded, visually similar, or arranged densely on production lines. In both cases, object detection supports monitoring, counting, safety control, and automated decision-making by locating important objects in visual data.

1.11 Conclusion

This chapter presented the theoretical basis required for the rest of the dissertation. It introduced artificial intelligence, machine learning, and deep learning before focusing on computer vision, image representation, preprocessing, CNNs, and the main families of object detection approaches. Special attention was given to the YOLO architecture and its evaluation metrics because these concepts provide the foundation for understanding the practical training workflow described in the next chapter and the deployment choices made for the GrabAndGo application.

Chapter 2

Model Training and Dataset Preparation

2.1 Introduction

This chapter describes the experimental workflow followed in this research to train YOLO-based detectors. The chapter covers dataset collection, annotation, preprocessing, model training, validation, and optimization. It documents the concrete steps performed in two experiments: ship detection in sea images and egg detection in photographs. Both experiments were implemented in notebook-based workflows, and the final decisions were based on metric comparison, especially mAP@0.5 and mAP@0.5:0.95 [2], [16].

2.2 Dataset Collection for the Experiments

This dissertation includes two practical object detection experiments. Therefore, dataset collection is discussed according to the datasets actually used during training and evaluation: a ship dataset and a custom egg dataset.

2.2.1 Data Sources

Two datasets were used in the experimental part: a ship dataset prepared inside a Kaggle environment and a custom egg dataset created specifically for this work. The ship dataset was selected to evaluate detection performance in maritime scenes, while the egg dataset was created to test model behavior on a small, manually annotated dataset.

2.2.2 Data Acquisition Strategy

The first practical step of the research was the creation of a Kaggle notebook dedicated to ship detection. The motivation was to build a detector capable of identifying ships in sea images. After the notebook environment was prepared, the ship dataset was uploaded and reorganized for YOLO training. In the second experiment, a suitable egg dataset compatible with the intended workflow was not found, so a custom dataset containing 64 images was built manually. This strategy made it possible to test YOLOv8 under both relatively structured and data-limited conditions.

2.3 Data Annotation and Labeling

Object detection performance depends heavily on annotation quality because the model learns directly from the bounding boxes and class labels provided during training.

2.3.1 Annotation Tools

Annotation tools are essential in object detection because they transform raw images into supervised training samples by associating each object with a bounding box and a class label. A clear annotation workflow improves label consistency, reduces localization errors, and makes the dataset easier to export in a format compatible with YOLO training.

2.3.2 LabelImg

LabelImg is a desktop annotation tool used in this research to annotate the custom egg dataset image by image. It allowed manual drawing of bounding boxes around egg instances and direct checking of object boundaries, class names, and label consistency. This manual control was important because the egg dataset was small and required careful verification before training [19].

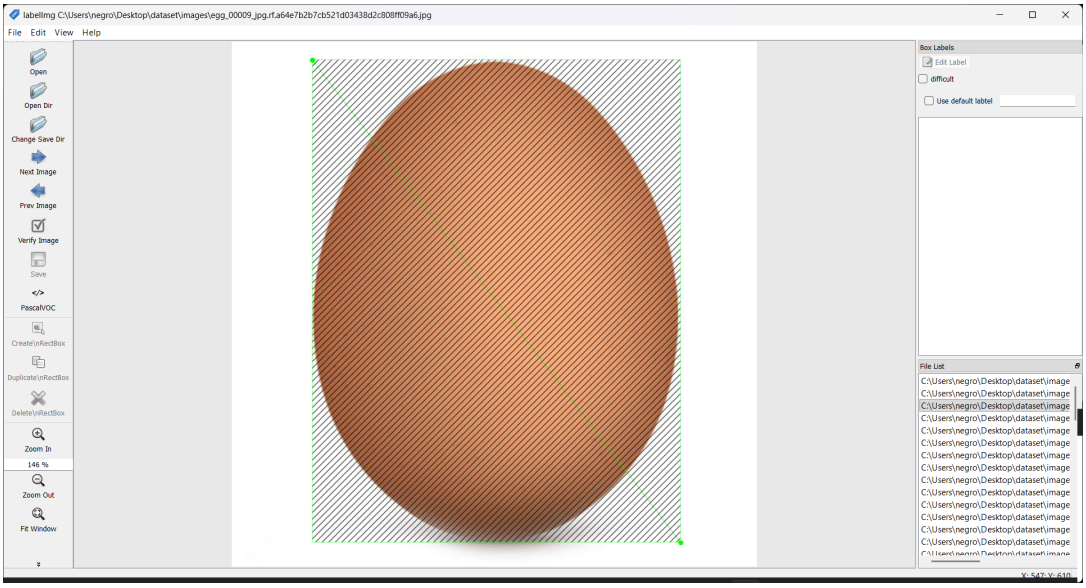


Figure 2.1: LabelImg annotation interface used for manually drawing bounding boxes around egg instances in the custom dataset.

2.3.3 Roboflow

Roboflow (`app.roboflow.com`) is a web-based computer vision platform that supports the full dataset lifecycle, from image upload and annotation to preprocessing, augmentation, dataset versioning, and model deployment [20], [21]. In this research, Roboflow was used in two roles. First, as a dataset management tool, it provided a structured project environment where images could be organized, preprocessed with operations such as resizing and auto-orientation, and exported in YOLO-compatible format. A key feature of the platform is dataset versioning, which creates a frozen snapshot of the dataset at each preparation stage, allowing reproducible exports for training. Second, Roboflow served as the inference deployment backend for the GrabAndGo prototype through its hosted serverless API [21].

2.3.4 Annotation Formats (YOLO, COCO)

Two common annotation formats are YOLO and COCO [2], [22]. YOLO stores normalized bounding-box coordinates in compact text files, which makes it convenient for Ultralytics training pipelines [16]. COCO is richer and more expressive but also more verbose. In the ship experiment, XML annotations were converted to YOLO format before training. In the egg experiment, annotations produced with LabelImg were organized in a YOLO-compatible structure for direct use by the training framework.

2.4 Data Preprocessing

Data preprocessing ensures that all samples are suitable for model optimization and inference.

2.4.1 Image Resizing and Normalization

Input images must be resized to the dimensions expected by the selected model configuration. In the ship experiment, training used an image size of 768 pixels, while the egg experiment used 640 pixels. Resizing creates a uniform training input, while normalization helps stabilize the learning process by constraining the data range and improving convergence [3], [5].

2.4.2 Data Augmentation Techniques

Data augmentation introduces controlled variation into the training set through operations such as scaling, translation, flipping, color adjustment, and mosaic composition. These techniques reduce overfitting and improve generalization, especially when the available data is limited. Augmentation is particularly important in small custom datasets such as the egg dataset, where diversity must be increased artificially [3], [23].

2.5 Dataset Splitting

A proper train-validation-test split is required to measure learning progress and assess generalization fairly [3].

2.5.1 Training, Validation, and Testing Sets

In the ship detection experiment, 621 labeled images were prepared and split into 434 training images, 93 validation images, and 94 testing images. This corresponds to an approximate 70%/15%/15% partition. In the egg detection experiment, the custom dataset of 64 images was split into 44 training images, 10 validation images, and 10 testing images. Such partitions allow the models to learn on the majority of the data while preserving independent subsets for tuning and evaluation.

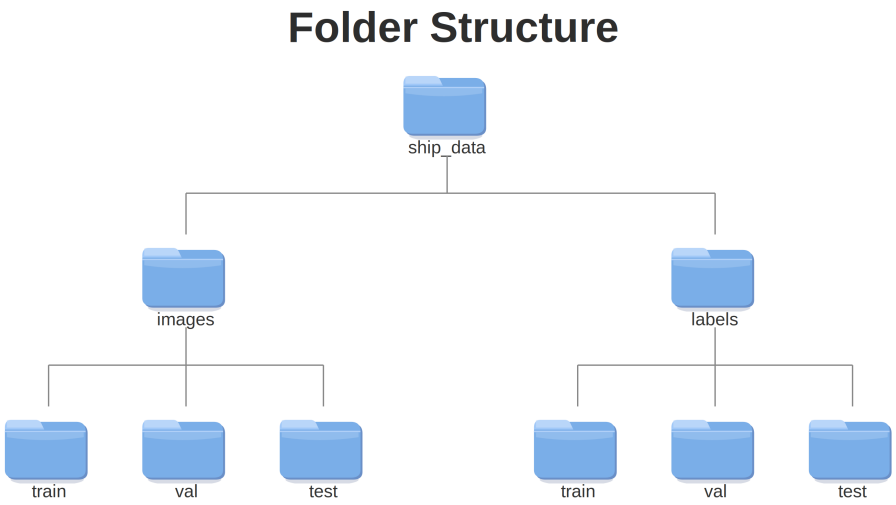


Figure 2.2: Ship dataset folder structure illustrating the training, validation, and testing split used for YOLO training. The egg dataset followed the same train/validation/test organization with fewer images.

2.5.2 Handling Class Imbalance

Class imbalance can degrade detector performance when some classes appear much more frequently than others. In the ship and egg experiments, the tasks were effectively single-class detection problems, which simplified the class-distribution challenge. However, imbalance can still appear through scene diversity, object count per image, and scale variation. In practice, careful sampling, augmentation, and validation monitoring help reduce this issue.

2.6 YOLO Model Training

The main training stage of the research was carried out using the Ultralytics YOLO framework imported directly inside the notebooks [16]. This stage connected the prepared datasets, configuration files, pretrained model weights, and evaluation outputs into a single experimental workflow. Each notebook loaded the corresponding dataset structure, initialized the selected YOLOv8 variant, applied the defined training parameters, and saved the resulting weights and metric summaries. This organization made it possible to compare model scales under controlled conditions while keeping the training process practical, repeatable, and closely linked to the validation results reported later in the chapter.

2.6.1 Model Selection (YOLOv8)

This dissertation focuses on YOLOv8 because it provides a modern and convenient implementation for training, evaluation, and deployment [2], [15], [16]. In the ship detection experiment, three YOLOv8 variants were trained: YOLOv8 large, YOLOv8 medium, and YOLOv8 small. In the egg detection experiment, only YOLOv8 large and YOLOv8 medium were trained because the dataset was small and the goal was to compare whether the medium configuration might generalize better with limited data. In both experiments, the large model produced the best final results.

2.6.2 Training Configuration

The ship detection experiment was executed in a Kaggle notebook. After importing the YOLO model from Ultralytics, training was launched for the three YOLOv8 variants [16]. The ship experiment used 150 epochs, image size 768, batch size 16, optimizer SGD, learning rate 0.005, deterministic training, seed 42, and freezing of the first 10 layers. The egg detection experiment used 50 epochs, image size 640, batch size 16, optimizer SGD, learning rate 0.005, deterministic training, seed 42, and no layer freezing. The large and medium variants were compared under the same general setup in the egg experiment, while the ship experiment compared all three scales under a consistent configuration.

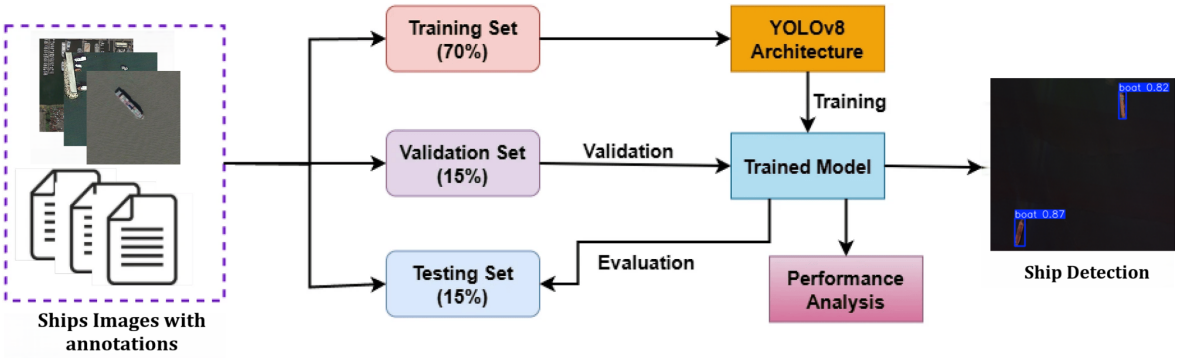


Figure 2.3: YOLOv8 architecture used as a reference for the ship-detection training workflow.

2.6.3 Hyperparameter Tuning

Hyperparameter tuning involves selecting appropriate training settings such as batch size, learning rate, number of epochs, optimizer choice, input resolution, and frozen layers [3], [16]. In the conducted experiments, tuning focused on selecting stable configurations suitable for the available hardware and dataset size. Rather than exploring a very large search space, the research compared multiple model scales under controlled settings and selected the configuration that achieved the strongest evaluation metrics. Using SGD, the parameter update rule can be written as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}_{\text{total}} \tag{2.1}$$

where θ represents the model parameters, η is the learning rate, and $\nabla_\theta \mathcal{L}_{\text{total}}$ is the loss gradient [3].

2.7 Model Evaluation and Validation

Evaluation was performed by reading the final training metrics generated by the notebooks and comparing the models quantitatively [16].

In addition to aggregate metrics such as precision, recall, and mAP, standard confusion matrices were used to inspect the prediction behavior of the YOLOv8 large and YOLOv8 medium models. The diagonal cells represent correct detections, while the off-diagonal and background-related cells highlight missed objects or incorrect predictions. These visualizations therefore complement the numerical metrics by showing where each model succeeds and where detection errors remain.

2.7.1 Performance Metrics Analysis

For model selection, mAP@0.5:0.95 was treated as the primary criterion because it evaluates localization quality across several IoU thresholds. Precision, recall, and mAP@0.5 were still considered as supporting indicators, especially when two models produced very close strict-localization scores.

Ships Detection Experiment

The ship detection experiment compared three YOLOv8 variants. The final metrics are summarized in Table 2.2.

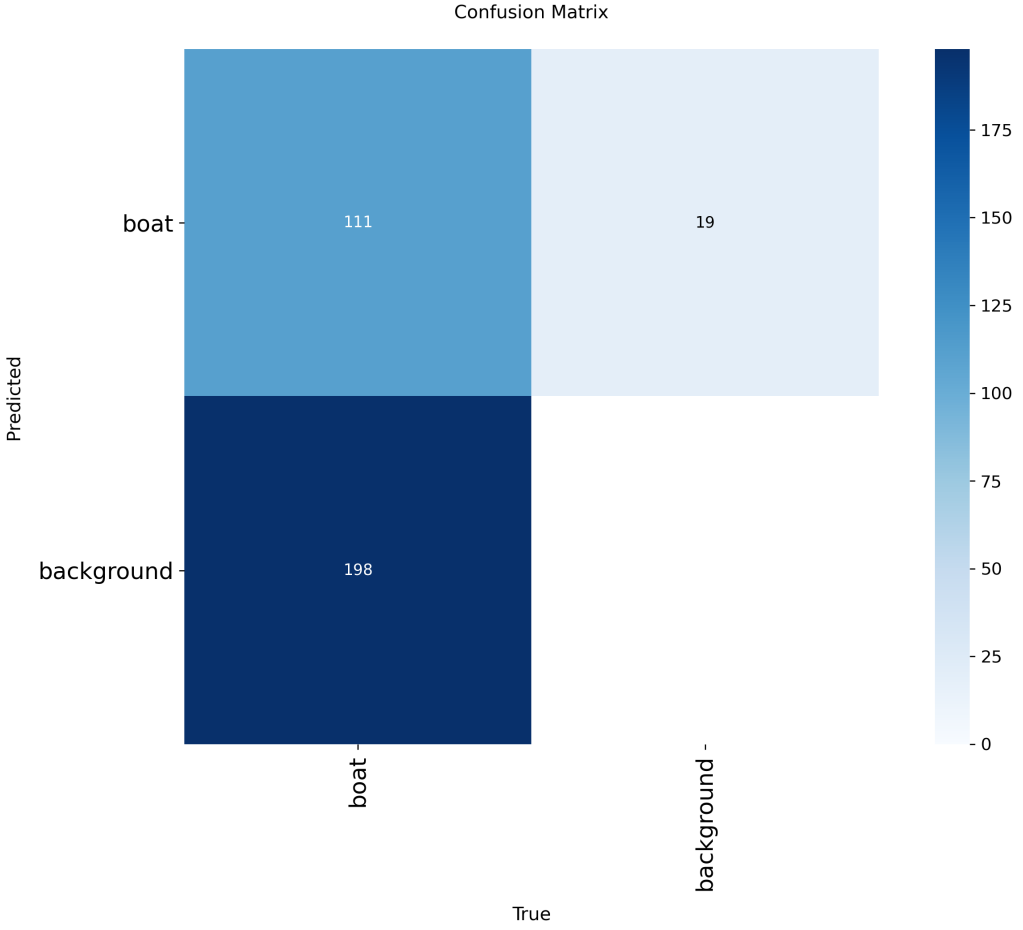


Figure 2.4: Standard confusion matrix for the YOLOv8 small model for ship detection.

Figure 2.4 illustrates the raw classification counts for the lightweight YOLOv8 small variant. The model successfully identified 111 true ship (``boat'') instances but demonstrated a high rate of localization or background confusion, misclassifying 198 background elements as ships. Conversely, it missed 19 actual ships, categorizing them as background. This indicates that while YOLOv8 small maintains a reasonable baseline for true positive detections, its lower capacity makes it highly susceptible to false positives in complex backgrounds.

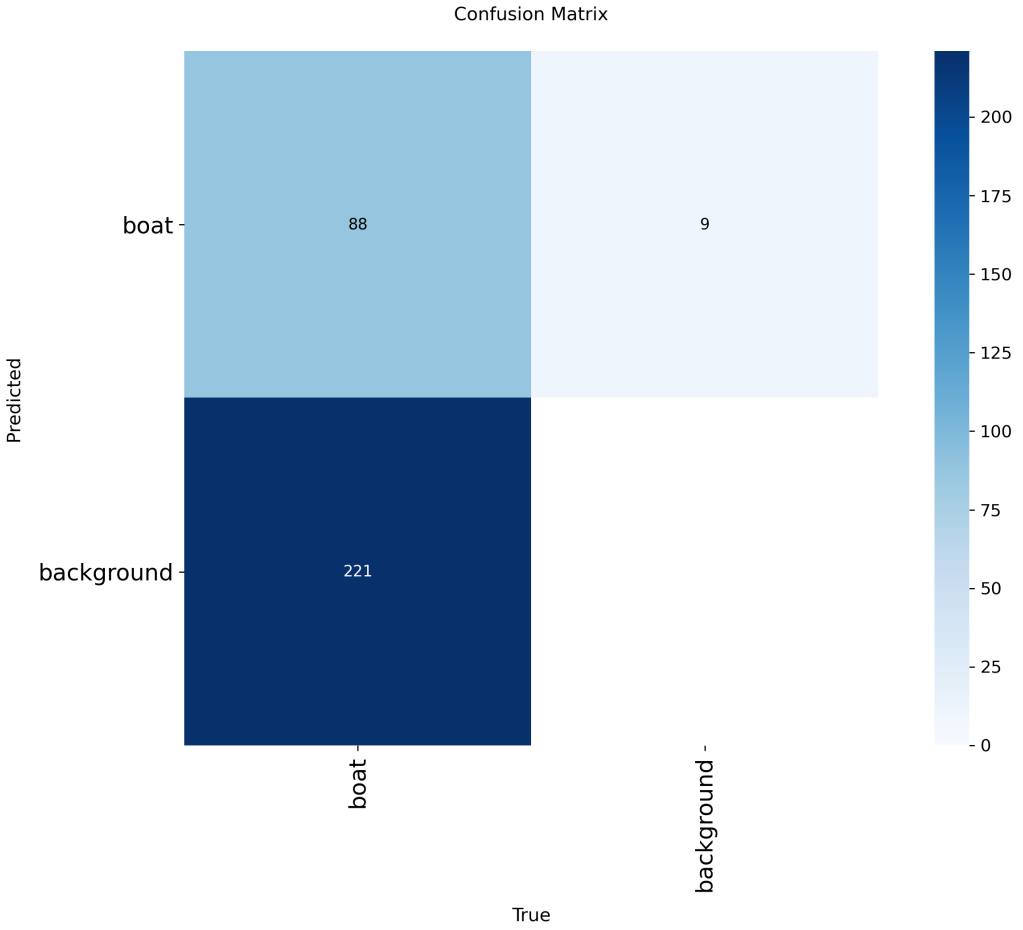


Figure 2.5: Standard confusion matrix for the YOLOv8 medium model for ship detection.

Figure 2.5 displays the raw performance metrics for the medium-capacity YOLOv8 medium model. Out of the true ship targets, the model correctly captured 88 instances while missing 9 as background misses. However, background interference remains a significant challenge for this architecture, yielding 221 false positives where background features were incorrectly flagged as ships. This profile suggests that the medium architecture slightly tightens its true negative gap but still struggles with environmental noise and false alarms in the scene.

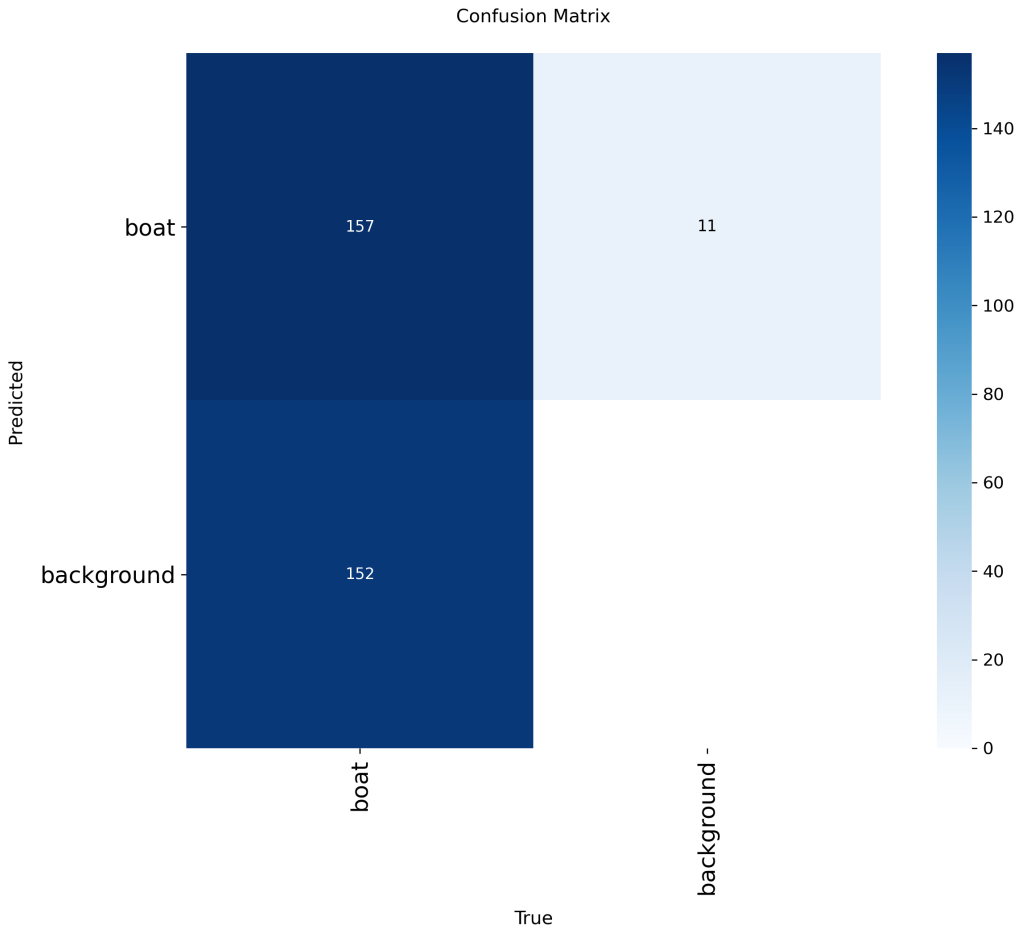


Figure 2.6: Standard confusion matrix for the YOLOv8 large model for ship detection.

Figure 2.6 represents the raw detection performance of the large-scale YOLOv8 large model, which features a higher parameter count. Scaling up the network architecture yields a noticeable improvement in capturing positive features, achieving the highest true positive count among the three models with 157 correct ship classifications. It minimized background misses down to 11 instances. While false positives from the background are still present (152 instances), they are noticeably lower than the medium variant, demonstrating that the deeper feature extraction capabilities of the large model better differentiate true ship structures from surrounding background textures.

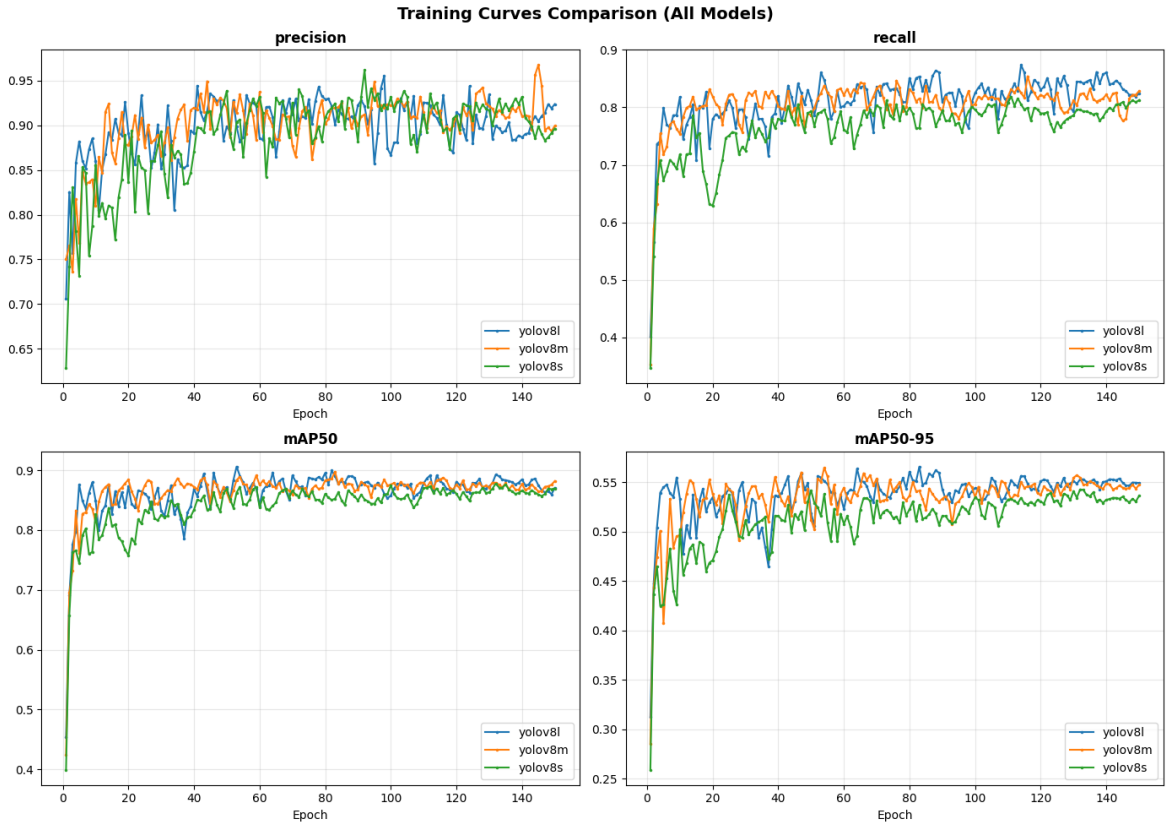


Figure 2.7: Training-curve comparison for the ship-detection experiment across YOLOv8 large, medium, and small models. The figure summarizes precision, recall, mAP@0.5, and mAP@0.5:0.95 over the training epochs.

Table 2.1: Raw confusion-matrix counts for the ship-detection models.

Model Architecture	True Class	Predicted: Ship	Predicted: Background
YOLOv8s	**Ship**	111	19
	Background	198	*N/A*
YOLOv8m	**Ship**	88	9
	Background	221	*N/A*
YOLOv8l	**Ship**	157	11
	Background	152	*N/A*

The following table summarizes the raw evaluation counts across the `YOLOv8s`, `YOLOv8m`, and `YOLOv8l` architectures explicitly evaluated for ship detection.

Table 2.2: Final validation metrics for the ship detection experiment.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
YOLOv8 large	0.92383	0.82432	0.87051	0.54915
YOLOv8 medium	0.89950	0.82848	0.88199	0.54718
YOLOv8 small	0.89648	0.81273	0.86985	0.53623

In the ship experiment, YOLOv8 medium achieved the highest mAP@0.5 (0.88199) and recall (0.82848), while YOLOv8 large achieved the highest precision (0.92383) and the highest mAP@0.5:0.95 (0.54915). The difference on this stricter metric was small, since YOLOv8 medium reached 0.54718. Therefore, YOLOv8 large was selected for ship detection because it performed best on the primary metric, while YOLOv8 medium remained a very competitive alternative when recall or mAP@0.5 is prioritized. The corresponding training curves are shown in Figure 2.7, while representative qualitative detections on held-out test images are shown in Figure 2.11.

Egg Detection Experiment

The egg detection experiment compared two YOLOv8 variants. The final metrics are summarized in Table 2.4.

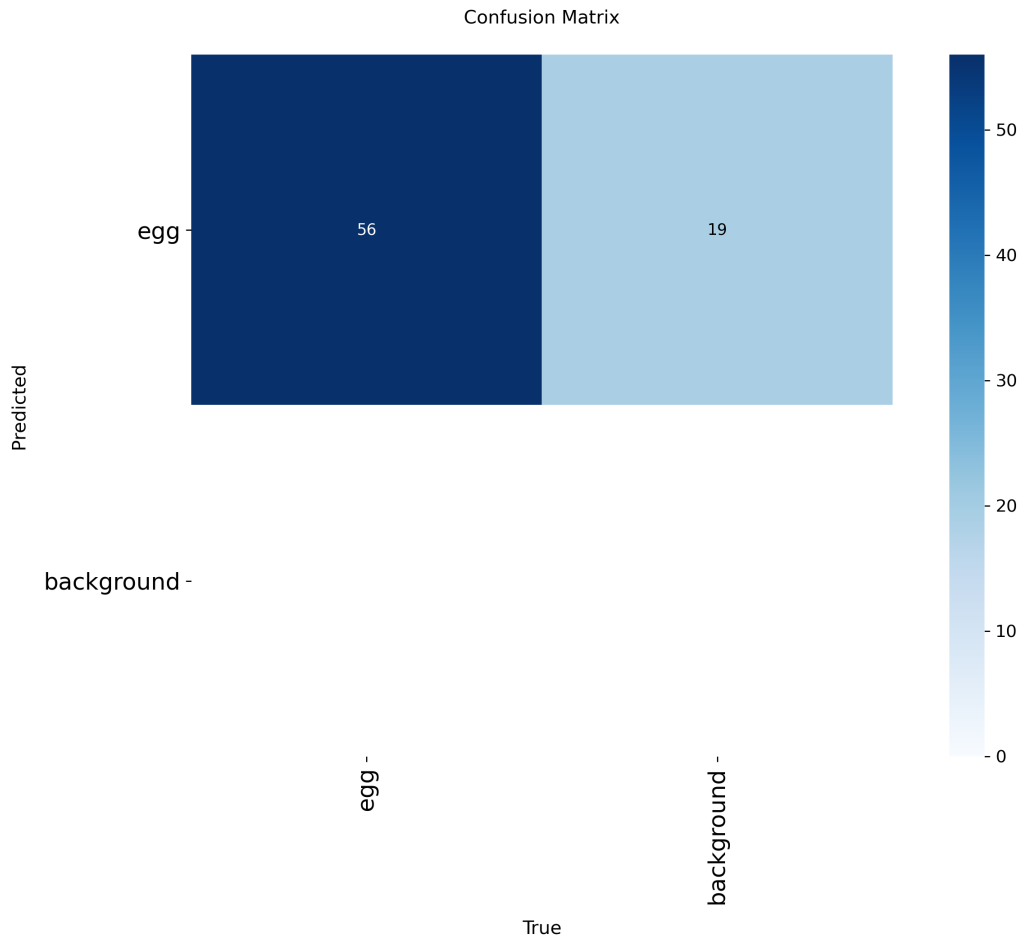


Figure 2.8: Standard confusion matrix for the YOLOv8 medium model for egg_detection model.

Figure 2.8 outlines the raw classification counts for the medium-capacity YOLOv8 medium architecture during egg detection. The model effectively identified 56 true positive egg instances. However, it struggled slightly with localization or background confusion, misclassifying 19 background regions as eggs (false positives). Notably, for this specific test run, the model recorded 0 instances of actual eggs being completely missed and categorized as background, indicating high sensitivity to the target class despite the background interference.

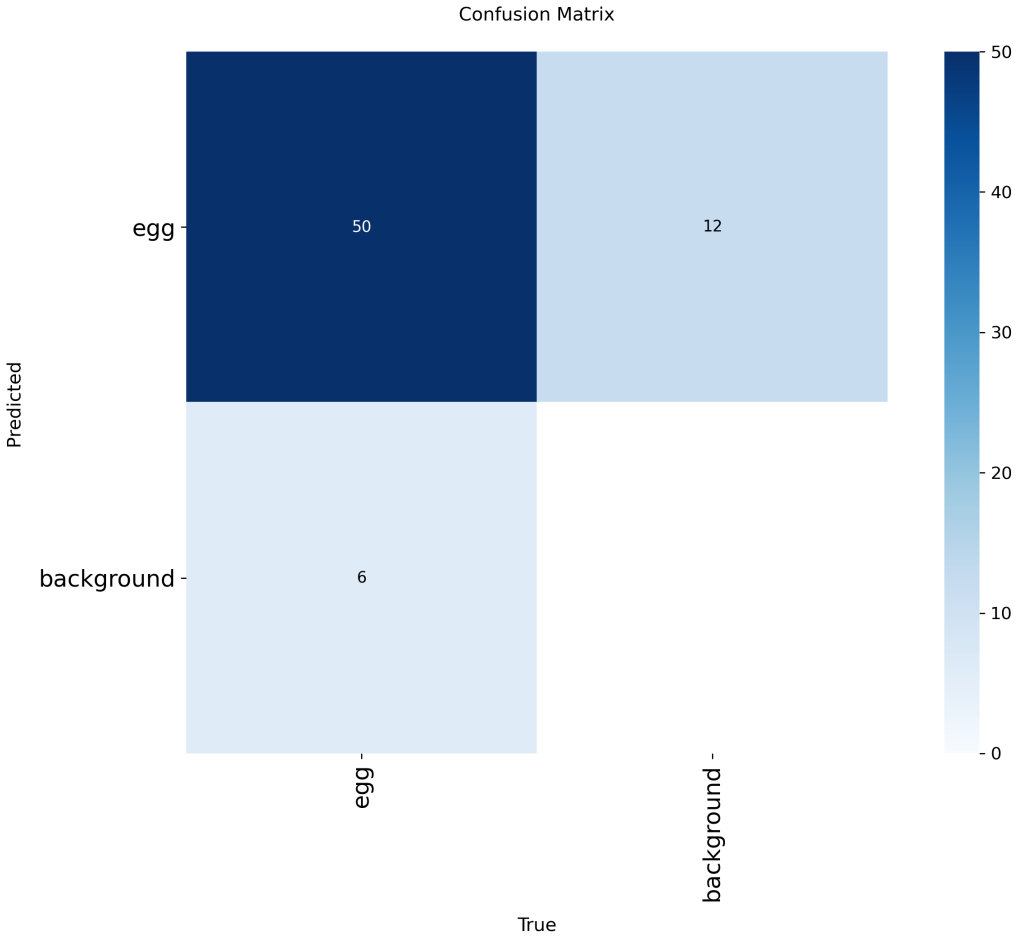


Figure 2.9: Standard confusion matrix for the selected YOLOv8 large model for egg_detection.

Figure 2.9 presents the performance of the high-capacity YOLOv8 large model on the egg dataset. The architecture successfully detected 50 true positive egg instances. In comparison to the medium variant, YOLOv8 large demonstrated better background discrimination, reducing the background-to-egg false positives down to 12 instances. However, this configuration introduced a minor trade-off in false negatives, completely missing 6 true egg instances by classifying them as background.

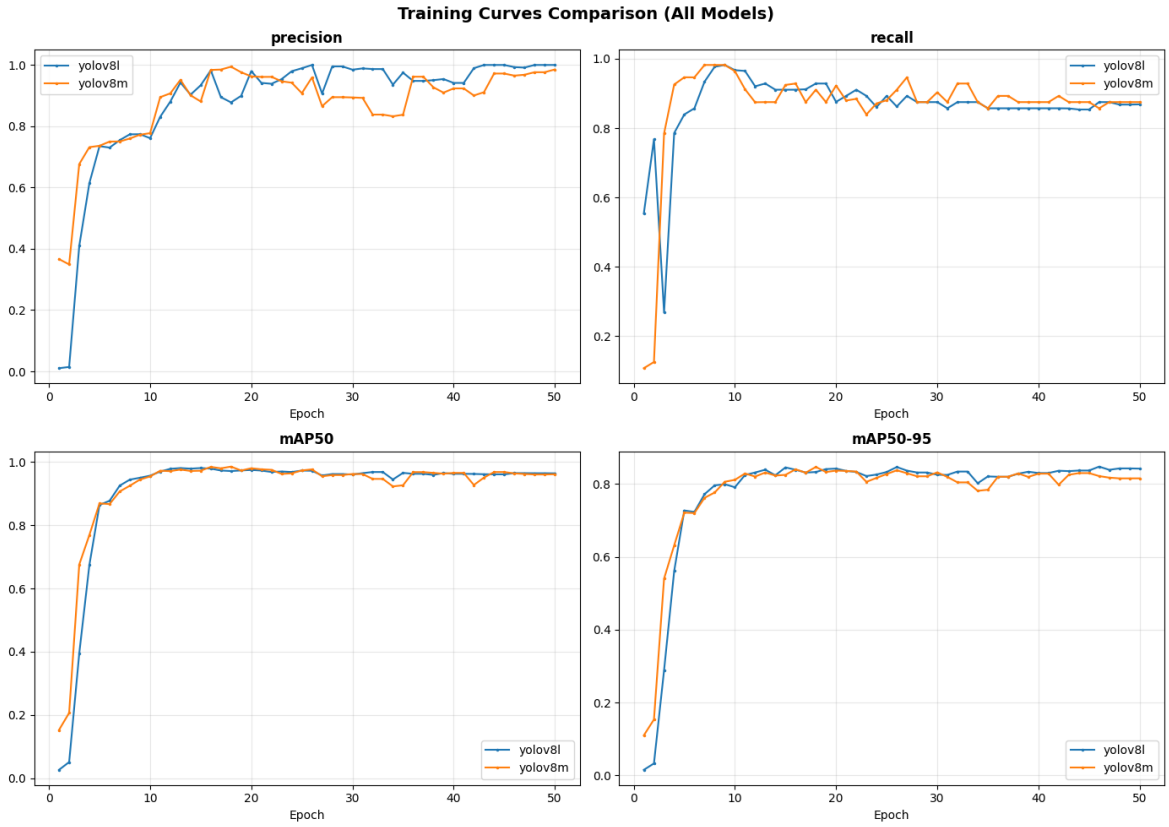


Figure 2.10: Training-curve comparison for the egg-detection experiment across YOLOv8 large and medium models. The figure summarizes precision, recall, mAP@0.5, and mAP@0.5:0.95 over the training epochs.

Table 2.3: Raw confusion-matrix counts for the egg-detection models.

Model Architecture	True Class	Predicted: Egg	Predicted: Background
YOLOv8m	**Egg**	56	19
	Background	*N/A*	*N/A*
YOLOv8l	**Egg**	50	12
	Background	6	*N/A*

The following table summarizes the raw evaluation counts across the `YOLOv8m`, and `YOLOv8l` architectures explicitly evaluated for egg detection.

Table 2.4: Final validation metrics for the egg detection experiment.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
YOLOv8 large	1.00000	0.86846	0.96454	0.84252
YOLOv8 medium	0.98536	0.87500	0.96169	0.81578

In the egg experiment, YOLOv8 large achieved the best precision (1.00000), mAP@0.5 (0.96454), and mAP@0.5:0.95 (0.84252). YOLOv8 medium produced a slightly higher recall value (0.87500 compared with 0.86846), but its mAP@0.5:0.95 was lower at 0.81578. Based on the primary mAP@0.5:0.95 criterion, YOLOv8 large was therefore selected as the best model for egg detection. The corresponding training curves are shown in Figure 2.10, while representative qualitative detections on held-out test images are shown in Figure 2.11.

2.7.2 Held-Out Test-Set Inference Evidence

The values in Tables 2.2 and 2.4 are validation-set metrics generated during training. The notebooks did not run a separate quantitative test-split validation call. Instead, the selected `best.pt` models were applied to held-out test images using `model.predict`. Therefore, the available test-set evidence consists of qualitative bounding-box outputs, detection counts, and inference-speed logs rather than separate test mAP values.

Table 2.5: Held-out test-set inference evidence for the selected YOLOv8 large models.

Evidence item	**Ship detection**	**Egg detection**
Test images	10 sampled images from 94	all 10 test images
Confidence threshold	0.5	0.2
Input size	768×768 px	640×640 px
Detection range	1--7 ships per image	1--26 eggs per image
Inference speed	≈ 59.4 ms/image (~ 16.8 FPS)	≈ 46.4 ms/image (~ 21.6 FPS)

2.7.3 Error Analysis

Error analysis is necessary to understand why one model performs better than another. In the ship detection task, possible errors include missed ships in cluttered sea backgrounds, localization errors on distant objects, and reduced recall in difficult environmental conditions. In the egg detection task, the main challenges are limited dataset size, annotation sensitivity, overlap between objects, and shape similarity. These observations explain why stronger feature extraction in the large model improved the primary strict-localization metric, even when a medium model remained competitive on recall or mAP@0.5.

2.8 Model Optimization

After evaluation, model optimization focuses on adapting the detector for practical use.

2.8.1 Pruning and Quantization

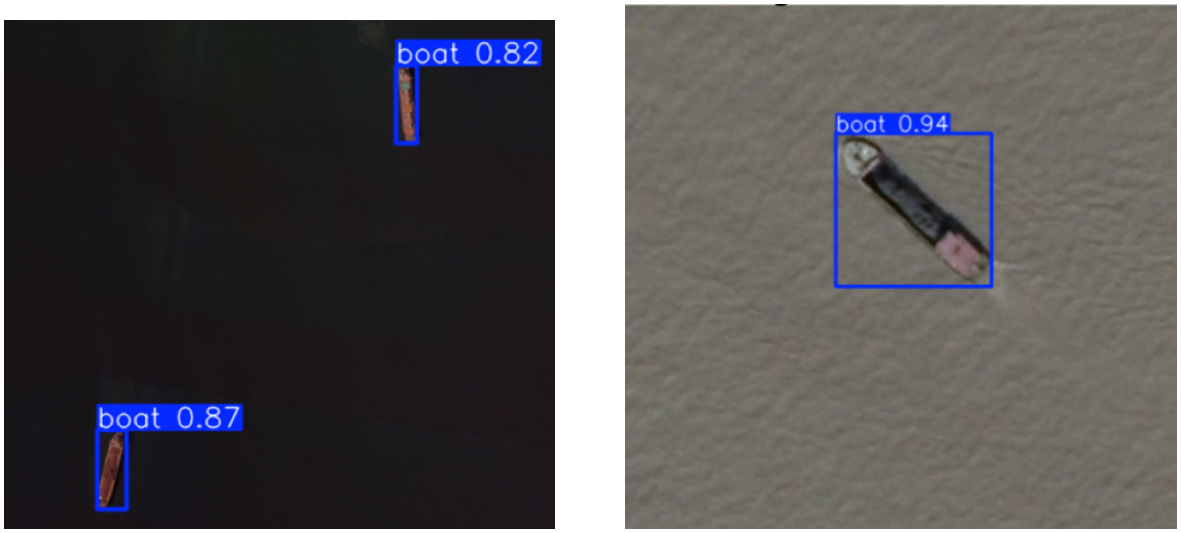
Pruning and quantization are common techniques for reducing computational cost and model size [24]. Although the GrabAndGo prototype relied on a hosted inference pipeline built around a pretrained model rather than deep edge compression, these techniques remain important for future versions when lower memory usage or faster inference on lightweight hardware is required.

2.8.2 Speed vs Accuracy Trade-offs

The experiments show the classical trade-off between model size and predictive quality. Smaller models usually offer faster inference and lower resource consumption, while larger models can deliver stronger strict-localization performance. In the ship experiment, YOLOv8 medium achieved the best mAP@0.5 and recall, but YOLOv8 large was selected because it had the highest precision and the highest mAP@0.5:0.95. In the egg experiment, YOLOv8

large was also superior on the main selection metric. Since the main goal of the research was to prioritize reliable detection quality under the stricter mAP@0.5:0.95 criterion, YOLOv8 large was selected in both experiments.

Ship-detection test results



Egg-detection test results

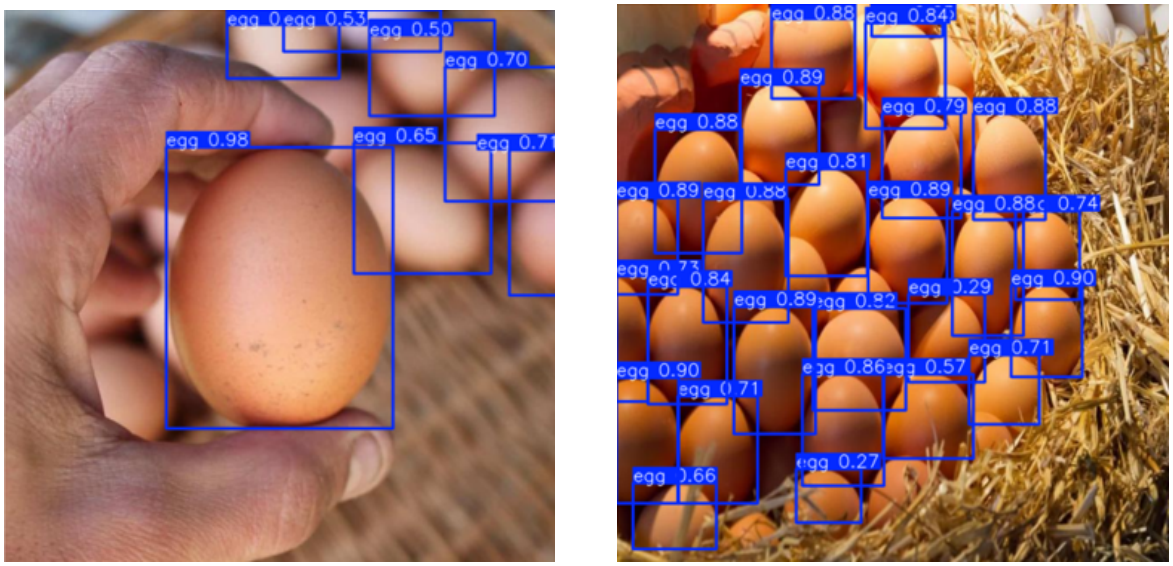


Figure 2.11: Qualitative test results for the selected YOLOv8 large detectors. The top row shows representative ship-detection outputs, while the bottom row shows representative egg-detection outputs on held-out test images.

2.9 Conclusion

This chapter documented the complete training workflow followed in the research. It explained how the datasets were prepared, annotated, preprocessed, and split, then described how YOLOv8 models were trained and compared in notebook-based experiments. The ship detection task started from a Kaggle notebook and compared three YOLOv8 variants, while the egg detection task relied on a custom 64-image dataset annotated manually with LabelImg and compared two variants. YOLOv8 large was selected in both experiments according to the stricter mAP@0.5:0.95 criterion, while the ship results also showed that YOLOv8 medium remained competitive on mAP@0.5 and recall.

Chapter 3

GrabAndGo — A YOLO-Based Cashierless Shopping Application

3.1 Introduction

Following the discussion of computer vision fundamentals in Chapter 1 and the model-training workflow in Chapter 2, this chapter presents *GrabAndGo* as an illustrative application prototype. The prototype uses a pretrained YOLO-style detector to show what object detection models can enable in a cashierless shopping scenario. Its role is to demonstrate a possible use case in which detected products can be associated with a customer session inside a desktop application workflow.

The chapter describes the system architecture, software components, object-detection integration, customer workflow, core functionalities, implementation details, prototype demonstration, and limitations of this example application.

3.2 System Architecture

GrabAndGo was designed as a modular system linking the graphical user interface, local application logic, persistent data storage, and the computer vision inference component. This separation makes the system easier to maintain because the interface, database operations, and detection pipeline can evolve independently.

3.2.1 Hardware Components

The prototype relies on a standard desktop computing environment, a camera for scene acquisition, and network connectivity for communication with the deployed detection service. In a production environment, additional components such as barcode backup scanners, customer identification modules, and store-side multi-camera installations could be in-

tegrated to improve reliability and coverage.

3.2.2 Software Stack

The software stack combines a React.js and TypeScript frontend, a Tauri desktop application layer, Rust backend commands, local SQLite-based persistence, and a pretrained YOLO-based inference pipeline. The application is presented as a demonstration layer for YOLO-based detection concepts rather than as a separately validated retail-detector experiment. This architecture allows the application to remain lightweight while delegating detection to a specialized inference component serving the pretrained model [16], [21].

3.2.3 Frontend User Interface

The graphical interface is built using React.js with TypeScript and bundled with Vite. This stack provides a responsive desktop experience while still allowing the interface to be developed with modern web technologies. The frontend is responsible for displaying the camera feed, rendering detection boxes, managing user interactions, presenting shopping-cart information, and showing dashboard statistics. Visual components such as Lucide React icons and Recharts visualizations support a clearer administrative interface for monitoring application activity.

3.2.4 Backend and Database

The core application logic is managed by the Tauri backend. Rust commands connect the frontend to system-level operations and local data storage. The application uses SQLite through the `rusqlite` library to maintain structured tables for:

- customers and account balances;
- products, inventory levels, and pricing;
- transactions and historical purchase records;
- active virtual shopping carts;
- application settings such as camera configuration and detection thresholds.

This local database layer links visual detections to concrete retail entities and supports persistent transaction history.

3.2.5 Computer Vision Inference Engine

The computer vision layer is designed as a multi-stage, multi-model detection and tracking framework built around the YOLO family of single-shot detectors. Instead of treating all visual entities with one model, the pipeline separates the problem into two parallel streams: a customer detection and tracking stream, and a product detection stream. A fusion and visualization layer then combines their outputs into annotated frames and structured data that the application can consume.

Stream A focuses on customer detection. It uses a fine-tuned YOLOv8 detector, identified in the Roboflow workspace as `people-detection-o4rdr/7`, to detect people in retail-like scenes. Its bounding boxes are passed to ByteTrack, a SORT-family multi-object tracker, which maintains persistent customer identities across frames using Kalman filtering and Hungarian assignment. This layer is important because cashierless shopping requires identity persistence rather than isolated per-frame detections.

Stream B focuses on product detection. It uses YOLOv8 Large, `YOLOv8l-640`, pretrained on the COCO dataset, as a general-purpose object detector for items that may appear in retail scenes, such as bottles, cups, books, phones, and packaged goods. In the prototype, these COCO classes act as product-like proxies. A production version would replace this stream with a SKU-specific detector trained on the actual store inventory.

3.2.6 Roboflow Inference SDK

The Roboflow Inference SDK provides a Python interface to the hosted detection models deployed on Roboflow's infrastructure. When a frame is captured by the application, it can be forwarded to the hosted endpoint or to a local inference component configured with the same model identifiers. The inference response contains bounding boxes, class labels, confidence scores, per-frame counts, and tracking information where applicable. This hosted approach removes the need to bundle model weights inside the desktop application and allows the detection layer to be updated independently from the interface and database logic.

3.3 Object Detection Integration

Object detection is the technical core of GrabAndGo because it provides the visual intelligence needed to identify customers and products automatically. The prototype uses a YOLO-centric architecture because YOLO performs detection in a single forward pass, making it suitable for real-time video streams. YOLOv8 also uses an anchor-free design, which helps the detector generalize across different object scales, from full-body customers to smaller product-like objects.

3.3.1 Multi-Model Fusion Strategy

A key design decision is the use of two separate detection streams instead of a single generic model. The custom person stream improves customer detection under retail-specific poses such as reaching, bending, or carrying items, while the COCO-based product stream provides general product-like object detection for the prototype. Before counting products, person detections from the product stream are filtered out so that a customer is not counted as both a person and an item.

Table 3.1: Multi-model detection streams used in the Roboflow-based prototype.

Stream	**Model**	**Classes**	**Purpose**
A	Custom YOLOv8 person detector	Person	Optimized for customer detection and tracking in retail scenes.
B	YOLOv8-L COCO	80 generic classes	General object detection used as a proxy for product-like items.

3.3.2 Real-Time Detection Pipeline

The real-time pipeline begins when the application captures frames from the camera interface. Each selected frame is sent to the detection layer, where Stream A detects customers and Stream B detects product-like objects. ByteTrack then receives the customer bounding boxes and assigns persistent tracking identifiers across frames. The fusion layer combines people tracks, product detections, confidence scores, and class labels, then renders annotated frames and prepares structured JSON output for the application.

3.3.3 Product Identification

Once detections are returned, each recognized product-like object can be compared with the product catalog stored in the database. In the current prototype, the YOLOv8-L COCO stream detects generic object categories rather than exact store SKUs, so its role is to demonstrate the product-detection pathway rather than complete commercial identification. Accurate SKU-level product identification would require a custom dataset containing the actual store inventory, packaging variations, shelf views, and camera angles.

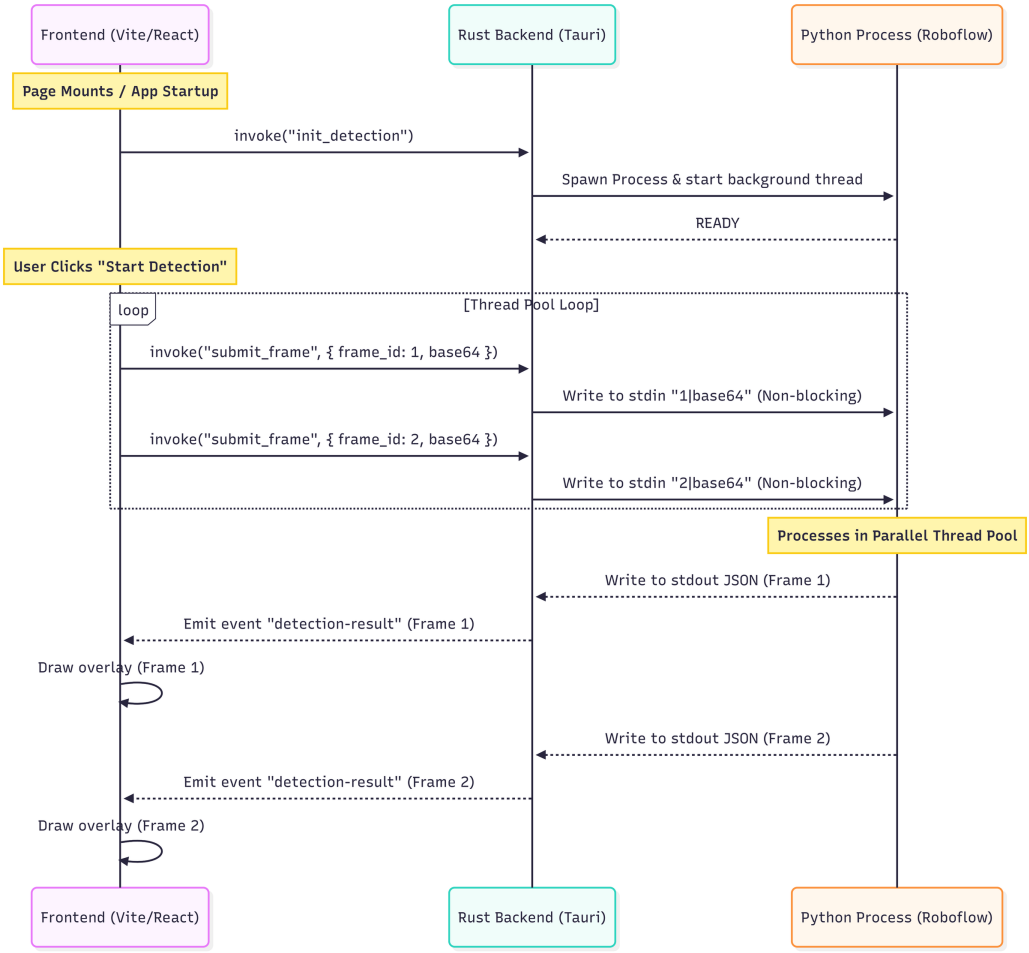


Figure 3.1: Detection sequence diagram for GrabAndGo.

3.3.4 People and Product Tracking

The live detection module supports the cashierless-inspired workflow by assigning persistent identities to detected customers. ByteTrack first matches high-confidence person detections to existing tracks using IoU-based Hungarian matching, then uses lower-confidence detections to recover tracks that may be partially occluded by shelves, products, or other customers. A Kalman filter predicts motion between frames so that short occlusions do not immediately break identity chains. Each customer receives a persistent `tracker_id`, allowing the system to visualize people with color-coded bounding boxes and enabling future logic such as associating a tracked customer with a nearby product event [25].

3.4 Customer Interaction Workflow

The application workflow follows the logic of a cashierless shopping experience where products are detected automatically and associated with the active user session.

3.4.1 Entry and Authentication

At the beginning of the shopping process, the customer enters the system and an authenticated session is established. This session acts as the container for all selected items and future billing events. Authentication may be implemented through login credentials, QR code scanning, or account-based identification.

3.4.2 Item Selection and Virtual Cart

As the customer selects products, the camera feed captures the scene and submits relevant frames to the detection service. The application tracks recognized products and updates the virtual cart dynamically. The cart aggregates detected products, updates quantities, and calculates the running checkout price in real time.

3.4.3 Checkout Calculation

After the session is complete, the prototype can compute a checkout total from products that have been detected and confirmed in the virtual cart. In this dissertation, this step is treated as an interface and data-flow demonstration. A production cashierless system would require stronger evidence, including SKU-level recognition, customer--item association across time, payment integration, and error-handling procedures.

3.5 Core Functionalities

GrabAndGo is structured around several modular interfaces that support both operational use and administrative management.

3.5.1 Live Detection and Virtual Cart

The live detection interface captures camera input, displays detected people and products, and renders bounding boxes over the video feed. At the same time, the virtual cart records the detected products associated with the active session. This module represents the main bridge between computer vision output and retail action.

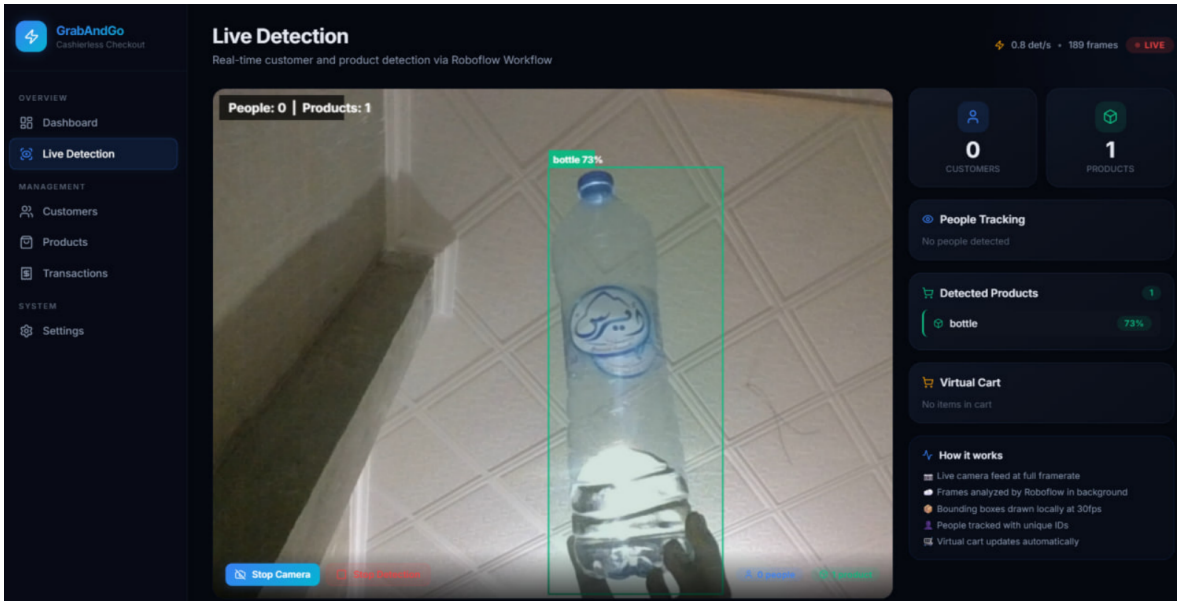


Figure 3.2: Live Detection interface visualizing the camera feed, detected objects, bounding boxes, and virtual-cart updates.

3.5.2 Dashboard and Analytics

The dashboard provides a high-level overview of system activity. It can display total registered customers, active shopping sessions, daily revenue, total processed transactions, and other operational indicators. A dashboard view is useful because it turns application events into business intelligence that administrators can use to monitor store performance.

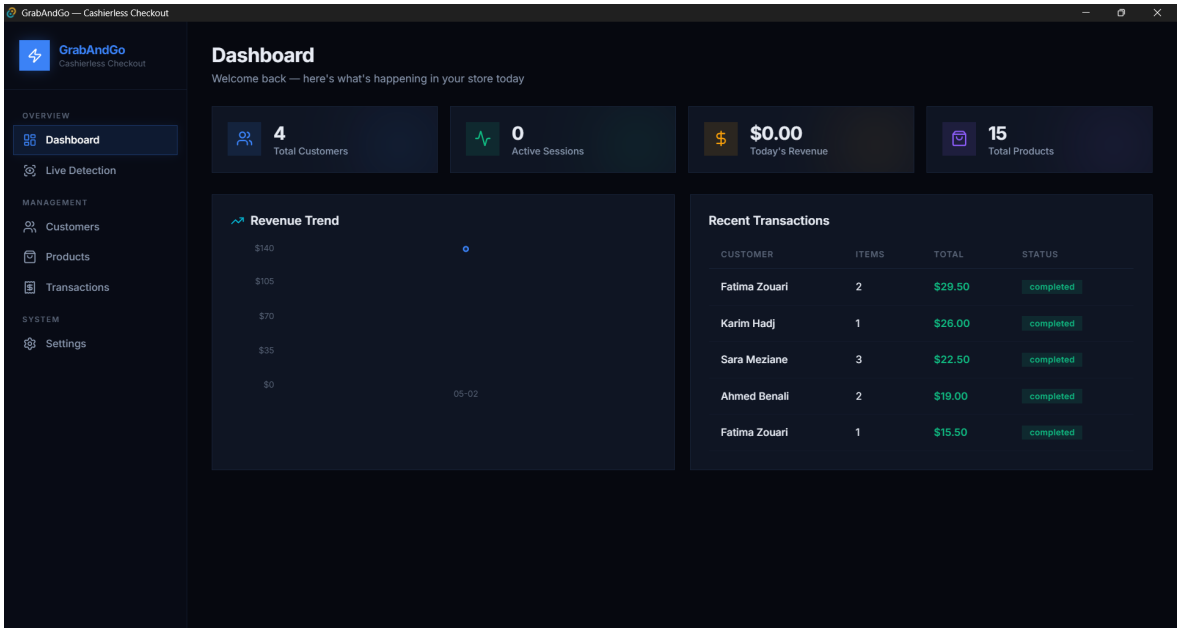


Figure 3.3: Dashboard and Analytics interface showing operational indicators and visual summaries for monitoring application activity.

3.5.3 Inventory and Product Management

The product-management interface allows administrators to manage the store catalog. Users can add new items, categorize products, define prices, and monitor stock levels. Search and filtering features make inventory management more efficient, especially as the number of supported products increases.

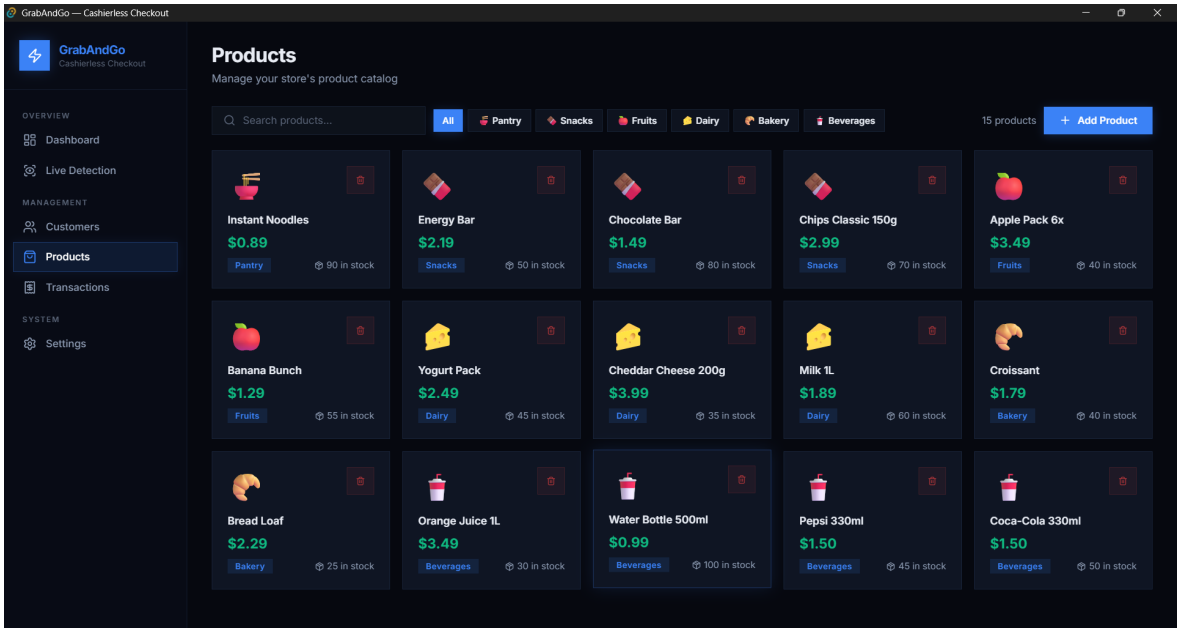


Figure 3.4: Inventory and Product Management interface showing the product catalog, item details, pricing, and stock-management controls.

3.5.4 Customers

The customer interface stores customer details, contact information, account balances, and account status. It allows administrators to review registered customers, verify their account state, and maintain the information required to associate shopping activity with the correct customer profile.

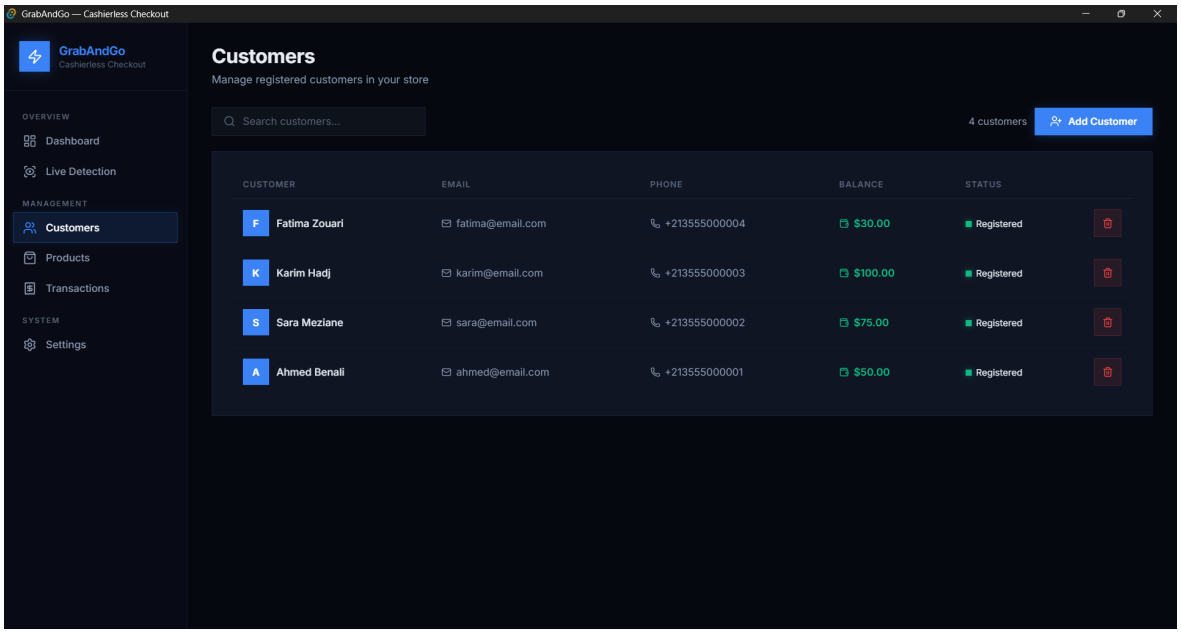


Figure 3.5: Customers interface showing registered customer records, contact details, account balances, and account status controls.

3.5.5 Transaction Management

The transaction-management interface records completed checkouts, including purchased items, total cost, and the customer associated with each transaction. These records provide traceability and support financial review by keeping a structured history of checkout operations.

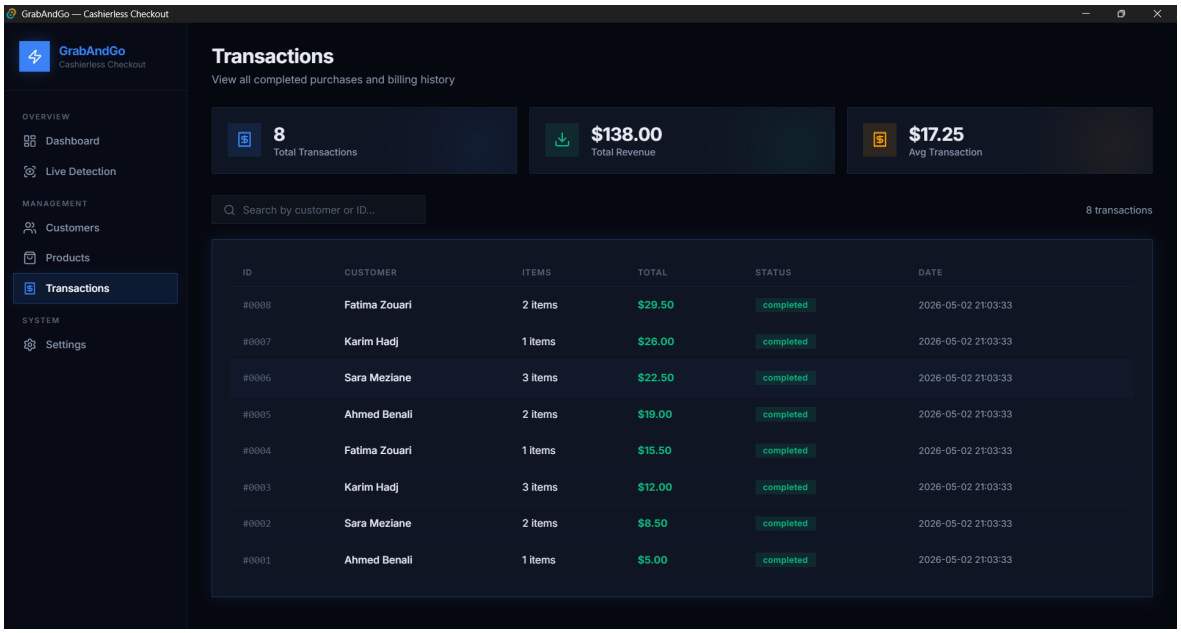


Figure 3.6: Transaction Management interface showing completed checkout records, purchased items, customer associations, and transaction totals.

3.6 Backend and Database Design

The backend layer is responsible for organizing product information, customer information, and transaction history.

3.6.1 Product Database

The product database stores product identifiers, names, categories, prices, inventory counts, and any mapping required between API predictions and application items. This database is the bridge between visual detections and commercial actions.

3.6.2 Transaction Management

Transaction management records the products associated with each customer session, keeps track of billing events, and supports order review. A reliable transaction layer is necessary to guarantee consistency between detections, basket updates, and final payment records.

3.7 Deployment of the Model

Deployment connects the YOLO-based detection and tracking stack to a practical service that can be consumed by the application. In this prototype, Roboflow is used to test and expose the detection workflow because it provides a convenient hosted interface for sending images, receiving predictions, and inspecting the returned JSON data before full application integration.

3.7.1 Edge vs Cloud Deployment

Two common deployment strategies are edge deployment and cloud deployment. Edge deployment offers lower latency and stronger local control, while cloud or hosted deployment simplifies maintenance and scalability. In this work, the selected strategy was hosted deployment through the Roboflow serverless API because it allowed the YOLO-based detector to be integrated rapidly into the desktop application without building a custom local inference server [21]. This choice is especially useful during prototyping because the model can be tested and updated independently from the React, Tauri, and Rust application layers.

3.7.2 Roboflow Serverless Deployment

To deploy the detection workflow through Roboflow, the model identifiers are configured inside a Roboflow project workspace. The person-detection stream uses the fine-tuned `people-detection-o4rdr/7` detector, while the product-detection stream uses `YOLOv8l-640`

as a general COCO-based detector. Once deployed, the hosted endpoint accepts image inputs and returns standardized JSON output containing detected classes, confidence values, bounding-box coordinates, live counts, and tracker identifiers where tracking is enabled. This output contract can feed a higher-level association module that reasons about which tracked customer was closest to a product detection when an item disappeared from a shelf.

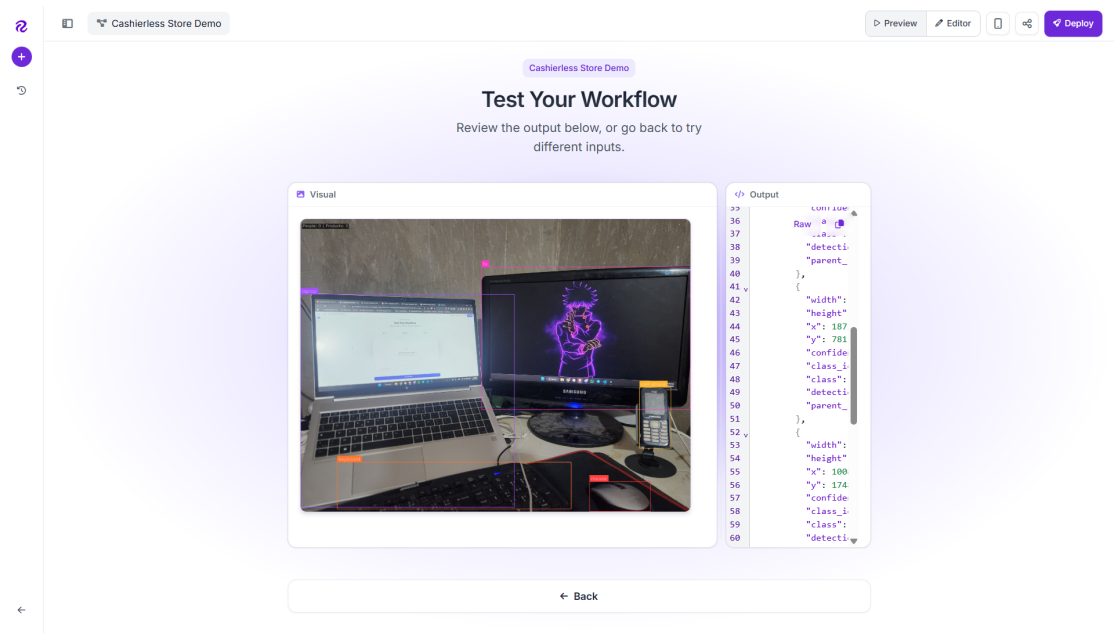


Figure 3.7: Roboflow object-detection test interface showing a sample detection image with bounding boxes and the corresponding raw JSON output.

This testing interface illustrates why hosted deployment was suitable for the prototype: it provides an accessible way to send images to a deployed detector, inspect bounding-box predictions, and verify the structured JSON response before integrating the endpoint into the application workflow. The left side of the interface visualizes detections on the input image, while the right side exposes the raw JSON response that the application can parse for counts, bounding boxes, class names, confidences, and tracking metadata.

3.7.3 Performance Optimization for Real-Time Use

For real-time application behavior, performance must be balanced across network latency, API response time, image size, confidence thresholds, and application-side processing. Practical optimization includes selecting suitable frame rates, reducing unnecessary requests, filtering low-confidence detections, and limiting expensive rendering operations so that the interface remains responsive.

3.8 Implementation Details

Developing a real-time computer vision application requires overcoming technical challenges related to process communication, inference latency, and interface rendering.

3.8.1 Inter-Process Communication

The application uses a command-based flow to connect the React frontend, Rust backend, and inference component. When detection is enabled, the frontend captures camera frames and sends them to Rust through Tauri commands. The backend forwards frame data to the inference component, waits for the structured JSON response, and returns parsed results to the frontend. This response may include per-frame product counts, customer `tracker_id` values, bounding-box coordinates, class names, and confidence scores. This explicit communication reduces coupling between the interface and the model execution layer.

3.8.2 Optimized Interface Rendering

To reduce visual stuttering caused by inference latency, the frontend can separate video rendering from bounding-box rendering through a dual-canvas approach. A hidden canvas captures raw video frames for backend processing, while a visible overlay canvas draws bounding boxes and labels using the HTML5 Canvas API. This approach keeps the user interface interactive even when inference responses are slower than the display frame rate.

3.9 Prototype Demonstration

The GrabAndGo prototype was examined qualitatively as an example of how a pretrained YOLO-based detector can be connected to an application workflow. The observations in this section are descriptive and are not intended to replace a full production-level system evaluation.

3.9.1 Detection and Latency Considerations

From a technical standpoint, the usefulness of such a prototype depends on the quality of the pretrained detector and the responsiveness of the inference service. Detection quality would need to be measured with metrics such as precision, recall, mAP, or basket-level correctness in a full evaluation. Latency would need to be measured by timing how fast the desktop application receives and processes detection results. In this dissertation, these aspects are discussed as design considerations rather than reported as final application benchmarks.

3.9.2 User Experience Testing

A future user experience evaluation should examine whether the workflow feels natural and efficient. Key aspects include clarity of the interface, speed of basket updates, transparency of automated billing, and confidence in the product recognition process. For a cashierless application, usability is as important as pure model accuracy because the system must inspire trust during shopping.

3.10 Challenges and Limitations

Several challenges remain in such a system. The product stream is not SKU-specific because the COCO-based YOLOv8-L model detects generic object categories rather than exact products from a store catalog. The prototype also does not include hand detection, grasp-event classification, or pose/keypoint reasoning, so it cannot yet determine whether a customer actually picked up an item. In addition, shelf-zone logic is not implemented; empty-shelf detection would require polygon zones, time-in-zone analytics, and product-state monitoring over time. Finally, the backend association module is still incomplete because the current workflow processes frame-level detections rather than correlating tracker trajectories with product state changes across a full shopping session.

3.11 Future Improvements

Future improvements may include training a SKU-specific YOLO model on the actual store inventory, adding hand or pose detection to recognize interaction events, defining shelf zones for spatial reasoning, and implementing a backend association module that links customer tracks with product disappearances over time. A full cashierless system would need spatial--temporal reasoning to answer questions such as which `tracker_id` was closest to a product detection when that product disappeared from the shelf. Other extensions include tighter multi-camera integration, local fallback inference, inventory synchronization, richer analytics dashboards, and stronger real-time optimization.

3.12 Conclusion

This chapter described GrabAndGo as an illustrative example application rather than the main purpose of the research. The prototype combines a YOLO-centric detection stack, a custom person-detection stream, a generic product-like detection stream, ByteTrack-based identity persistence, a React and Tauri desktop interface, Rust backend logic, local database management, and a cashierless-shopping-inspired workflow. The chapter showed how ob-

ject detection concepts can move from notebook-based experimentation toward a user-facing demonstration pipeline for visual detection, customer tracking, virtual cart display, and checkout calculation. The current prototype is not a production Amazon Go-style system and should not be interpreted as a complete automated retail solution; it is a modular proof-of-concept that identifies the additional work needed for stronger retail-specific evaluation.

General Conclusion

This dissertation examined an applied YOLO-based object detection workflow, moving from theoretical foundations and dataset preparation to model comparison and prototype integration. The work began with a review of computer vision concepts, deep learning in vision, object detection paradigms, and the YOLO architecture. This theoretical basis made it possible to justify the methodological choices adopted later in the experimental chapters and to frame the demonstration prototype appropriately.

The practical contribution of the research was demonstrated through two experiments. The first experiment used a Kaggle notebook to prepare and train a ship detector for maritime scenes. After uploading the dataset and converting annotations to a YOLO-compatible structure, three YOLOv8 variants were trained and compared. The second experiment focused on egg detection and required building a custom dataset of 64 images because a directly compatible dataset was not available. These images were annotated manually with LabelImg and used to train two YOLOv8 variants. In both experiments, the large variant was selected according to the stricter mAP@0.5:0.95 criterion, while the ship experiment also showed that the medium variant remained competitive in mAP@0.5 and recall.

GrabAndGo was presented as an illustrative desktop prototype rather than the main objective of the dissertation. Inspired by cashierless shopping workflows, the application shows one possible way that a pretrained YOLO-style detector can be connected to a user-facing interface. This example demonstrates prototype-level integration only: it was not evaluated as a complete checkout system, does not identify exact retail SKUs, and does not prove that general retail automation has been solved. The core contribution of the dissertation remains the YOLO-based training, comparison, and evaluation workflow.

Overall, the results show that YOLO-based detection is suitable for the two experimental tasks studied and can motivate deployment-focused prototypes that use pretrained detectors for demonstration purposes. The study also highlights the importance of dataset quality, annotation accuracy, metric-based model comparison, and careful system-level framing. Future work can extend this foundation through larger and more diverse datasets, stronger evaluation on task-specific test sets, SKU-level product data, multi-camera tracking, and more advanced real-time optimization.

References

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Springer, 2022, ISBN: 978-3-030-34372-9. DOI: 10.1007/978-3-030-34372-9

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ``You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, ``Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ``Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[7] P. Viola and M. Jones, ``Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, pp. I--511--I–518.

[8] N. Dalal and B. Triggs, ``Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ``Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[10] R. Girshick, ``Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[11] S. Ren, K. He, R. Girshick, and J. Sun, ``Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick, ``Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2961–2969.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed, ``SSD: Single shot MultiBox detector,'' in *European Conference on Computer Vision*, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2

[14] J. Redmon and A. Farhadi, ``YOLOv3: An incremental improvement,'' *arXiv preprint arXiv:1804.02767*, 2018.

[15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, ``YOLOv4: Optimal speed and accuracy of object detection,'' *arXiv preprint arXiv:2004.10934*, 2020.

[16] Ultralytics, *Ultralytics YOLO documentation*, Official documentation for the Ultralytics training framework used in this research, 2026. Accessed: May 11, 2026. [Online]. Available: https://docs.ultralytics.com/

[17] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang, ``TOOD: Task-aligned one-stage object detection,'' in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 3510–3519.

[18] J. Janai, F. Güney, A. Behl, and A. Geiger, ``Computer vision for autonomous vehicles: Problems, datasets and state of the art,'' *Foundations and Trends in Computer Graphics and Vision*, vol. 12, no. 1--3, pp. 1–308, 2020. DOI: 10.1561/0600000079

[19] Tzutalin, *LabelImg*, Git code, 2015. Accessed: May 11, 2026. [Online]. Available: https://github.com/HumanSignal/labelImg

[20] Roboflow, *Introduction to Roboflow annotate*, Official documentation, 2026. Accessed: May 11, 2026. [Online]. Available: https://docs.roboflow.com/annotate

[21] Roboflow, *Deploy a model or workflow*, Official documentation, 2026. Accessed: May 11, 2026. [Online]. Available: https://docs.roboflow.com/deploy

[22] T.-Y. Lin et al., ``Microsoft COCO: Common objects in context,'' in *European Conference on Computer Vision*, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48

[23] C. Shorten and T. M. Khoshgoftaar, ``A survey on image data augmentation for deep learning,'' *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019. DOI: 10.1186/s40537-019-0197-0

[24] S. Han, H. Mao, and W. J. Dally, ``Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,'' in *International Conference on Learning Representations*, 2016.

[25] Y. Zhang, P. Sun, Y. Jiang, D. Yu, and F. Weng, ``ByteTrack: Multi-object tracking by associating every detection box,'' in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022, pp. 1–21. DOI: 10.1007/978-3-031-20047-2_1

Appendix A

Appendix A: List of Acronyms

- **AI**: Artificial Intelligence
- **AP**: Average Precision
- **API**: Application Programming Interface
- **C2f**: Cross Stage Partial Bottleneck with two convolutions (YOLOv8 module)
- **CCTV**: Closed-Circuit Television
- **CNN**: Convolutional Neural Network
- **COCO**: Common Objects in Context
- **CV**: Computer Vision
- **DL**: Deep Learning
- **FN**: False Negative
- **FP**: False Positive
- **FPS**: Frames Per Second
- **HOG**: Histogram of Oriented Gradients
- **HSV**: Hue, Saturation, Value
- **IoU**: Intersection over Union
- **JSON**: JavaScript Object Notation
- **LAB**: CIELAB color space (Lightness, a, b)
- **ML**: Machine Learning

- **NMS**: Non-Maximum Suppression
- **PANet**: Path Aggregation Network
- **QR**: Quick Response (code)
- **R-CNN**: Region-based Convolutional Neural Network
- **RGB**: Red, Green, Blue
- **SDK**: Software Development Kit
- **SGD**: Stochastic Gradient Descent
- **SKU**: Stock Keeping Unit
- **SORT**: Simple Online and Realtime Tracking
- **SPPF**: Spatial Pyramid Pooling - Fast
- **SSD**: Single Shot MultiBox Detector
- **TP**: True Positive
- **UI**: User Interface
- **XML**: Extensible Markup Language
- **YCbCr**: Luminance, Chrominance Blue, Chrominance Red color space
- **YOLO**: You Only Look Once
- **YOLOv8**: You Only Look Once version 8
- **mAP**: Mean Average Precision