People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
M'Hamed Bougara University of Boumerdes
Faculty of Sciences
Department of Computer Science

Final Year Project Thesis
Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science

Field of Study: Computer Science **Specialization:** Computer Systems

Real-Time Detection and Recognition of ISO Container Identification from Video Streams using Deep Learning

Prepared by:

SEFSAF Younes
BOUDJELLALI Abderrahmane
LAIB Abderaouf
BENKHELIFA Abdallah

Jury Members:

DJEDDAI Selma	Examiner
KEDJAR Saâdia	Academic Supervisor
Hamza Rassoul	Industrial Supervisor

Academic Year: 2025 – 2026

Words of Thanks

Through these few lines, we would like to express our sincere thanks and gratitude, first and foremost, to our supervisor, University Professor Mrs. Kedjar Saâdia, and our External Supervisor Mr. Hamza Rassoul , for their guidance and valuable advice throughout the realization of this project.

We also extend our thanks to all those who, through their support, availability, and encouragement, gave us the motivation and courage to successfully complete this work.

We would like to express our deepest gratitude to our parents, who have been a constant source of unconditional support throughout our academic journey. Their love, patience, encouragement, and belief in us were essential in bringing this project to fruition. We are profoundly grateful for their sacrifices, understanding, and continuous support.

Our heartfelt thanks also go to the honorable jury members for the interest they have shown in our end-of-study project (PFE) by accepting to examine and evaluate this work, as well as for their valuable remarks and constructive feedback.

Finally, we cannot conclude without expressing our sincere gratitude to all the professors of the Faculty of Sciences of Boumerdes and the Department of Computer Science for their dedication, guidance, and assistance throughout our years of study.

Contents

Words of Thanks **i**

GENERAL INTRODUCTION **1**

1 Concepts and Technologies **2**

1.1 Introduction . . . 2
1.2 Project Context . . . 2
1.2.1 Smart Logistics . . . 2
1.2.2 Smart Ports . . . 2
1.2.3 Need for Automation . . . 2
1.3 Overview of AI and Computer Vision . . . 3
1.3.1 Artificial Intelligence (AI) . . . 3
1.3.2 Machine Learning . . . 3
1.3.3 Deep Learning . . . 3
1.3.4 Convolutional Neural Networks (CNN) . . . 3
1.4 Object Detection . . . 4
1.4.1 Introduction to Object Detection . . . 4
1.4.2 Traditional Detection Methods . . . 4
1.4.3 Deep Learning-Based Detection . . . 5
1.4.4 YOLO Definition . . . 5
1.4.5 YOLO Principle and Architecture . . . 5
1.4.6 The Reason for Using YOLO . . . 6
1.5 Optical Character Recognition (OCR) . . . 6
1.5.1 OCR Definition . . . 6
1.6 Conclusion . . . 6

2 System Analysis and Design **7**

2.1 Introduction . . . 7
2.2 Requirements Analysis . . . 7
2.2.1 UML Presentation . . . 7
2.2.2 System Requirements Analysis . . . 7
2.3 Use Case Diagram . . . 8

2.3.1 Use Case Description . . . 8
2.4 Conceptual study . . . 9
2.4.1 Data Dictionary . . . 9
2.4.2 Class Table . . . 9
2.4.3 Relationship Table . . . 11
2.5 Classes Diagram . . . 12
2.6 Sequence Diagrams . . . 12
2.6.1 Video Sampling Sequence Diagram . . . 12
2.6.2 Object Detection Sequence Diagram . . . 13
2.6.3 OCR Processing Sequence Diagram . . . 14
2.6.4 FastAPI Result Delivery Sequence Diagram . . . 15
2.7 Conclusion . . . 16

3 System Methodology and Implementation 17
3.1 Introduction . . . 17
3.2 Problem Statement . . . 17
3.2.1 Manual Identification Challenges . . . 17
3.2.2 Environmental Challenges . . . 17
3.3 Project Objectives . . . 17
3.4 Methodology . . . 18
3.4.1 Dataset Collection . . . 18
3.4.2 Model Training . . . 18
3.4.3 Image Preprocessing . . . 19
3.4.4 Text Recognition using Paddle-OCR . . . 20
3.4.5 Asynchronous Processing Pipeline . . . 20
3.4.6 System Workflow . . . 21
3.5 Implementation . . . 21
3.5.1 Technology Stack . . . 21
3.5.2 The Core Microservices . . . 21
3.5.3 FastAPI Backend . . . 24
3.5.4 Data Models and Communication . . . 24
3.5.5 Docker Deployment . . . 25
3.6 Conclusion . . . 25

4 RESULTS AND DISCUSSION 26
4.1 Introduction . . . 26
4.2 Results . . . 26
4.2.1 Object Detection Results and Evaluation . . . 26
4.2.2 OCR Recognition Results . . . 27
4.3 Error Analysis and Discussion . . . 28

4.4 Conclusion 29

GENERAL CONCLUSION **30**

GENERAL INTRODUCTION

Manual identification of shipping containers in ports is a slow and costly process. It often leads to human errors, which reduces the efficiency of modern logistics. Environmental factors like poor lighting, rust, and motion blur make it even harder to read container numbers. By applying Artificial Intelligence (AI) and computer vision, we can turn this real-world logistics challenge into a computer science problem. In computer science, this is solved as an object detection and text recognition problem.

Our solution

Our solution relies on Deep Learning and Optical Character Recognition (OCR). First, it uses a fast object detection model, YOLO26, to find and crop the container's ISO code from a video stream. Then, the system cleans the image and uses PaddleOCR to read the text. It is designed to accurately read text whether it is written horizontally or vertically. The entire system runs smoothly in real-time using a microservices architecture so it does not slow down the video feed.

Chapter summaries

- **Chapter 1** reviews the basic concepts of smart logistics, deep learning, YOLO, and the OCR technologies used in this project.
- **Chapter 2** details the system analysis and design, presenting the functional requirements, UML models, and data structures.
- **Chapter 3** presents the methodology and practical implementation, covering the data pipeline, selected technologies, microservices architecture, and system deployment.
- **Chapter 4** analyzes the final results of the system, discusses its accuracy under different conditions, and examines the main causes of any errors.

Chapter 1

Concepts and Technologies

1.1 Introduction

This chapter presents the theoretical background and the main technologies used in the development of the automatic container ID detection and recognition system. Understanding the concepts related to smart logistics, computer vision, and artificial intelligence is essential to justify the methodological choices adopted in this project.

The chapter first introduces the project context through smart ports, and the growing need for automation in container tracking. It then presents the fundamental concepts of deep learning and object detection, with a focus on the YOLO model. The principles of Optical Character Recognition (OCR) and the PaddleOCR tool are also discussed. This chapter provides the theoretical foundation for the implementation described in the following chapters.

1.2 Project Context

1.2.1 Smart Logistics

Smart logistics refers to the integration of advanced digital technologies — including IoT, AI, and real-time tracking systems — to improve the efficiency and visibility of supply chain operations. With the continuous growth of global trade, container transportation has become a fundamental element in international logistics, making accurate tracking and fast identification essential for ensuring smooth goods movement and reducing operational delays. [1]

1.2.2 Smart Ports

Smart ports are modern maritime terminals that leverage advanced digital technologies such as artificial intelligence, IoT, and automation systems to optimise port operations. These technologies enable real-time monitoring of container movements, faster loading and unloading processes, and improved security. By reducing manual intervention and increasing operational efficiency, smart ports play a key role in modern logistics and international trade. [2]

1.2.3 Need for Automation

The increasing volume of container traffic in modern ports makes manual identification processes slow, costly, and prone to errors. Automation is therefore essential to accelerate container tracking, reduce human intervention, improve accuracy, and enhance safety in port operations. Automated systems based on AI and computer vision provide faster and more reliable solutions for real-time container identification. [3]

1.3 Overview of AI and Computer Vision

1.3.1 Artificial Intelligence (AI)

Artificial Intelligence (AI) is a branch of computer science that enables computers or computer-controlled robots to perform tasks commonly associated with intelligent beings. It encompasses systems with the ability to reason, discover meaning, generalize, and learn from past experience — capabilities that are central to the automation of complex real-world tasks. [4]

1.3.2 Machine Learning

Machine learning is a branch of Artificial Intelligence that focuses on developing models and algorithms that let computers learn from data without being explicitly programmed for every task. In simple words, ML teaches systems to think and understand like humans by learning from the data. [5]

1.3.3 Deep Learning

Deep learning is a subset of machine learning driven by multilayered neural networks whose design is inspired by the structure of the human brain. It is widely used in image recognition, speech processing, and natural language understanding. Powered by large datasets and GPU-accelerated computing, deep learning models automatically learn hierarchical feature representations from raw data, eliminating the need for manual feature engineering. [6]

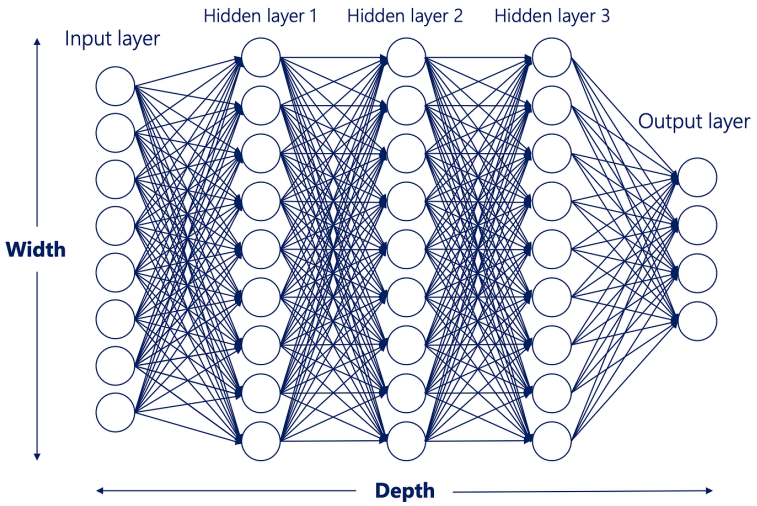


Figure 1.1: How Deep Learning works [7]

1.3.4 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others. They form the backbone of all modern object detectors and OCR recognition engines used in this project. [8]

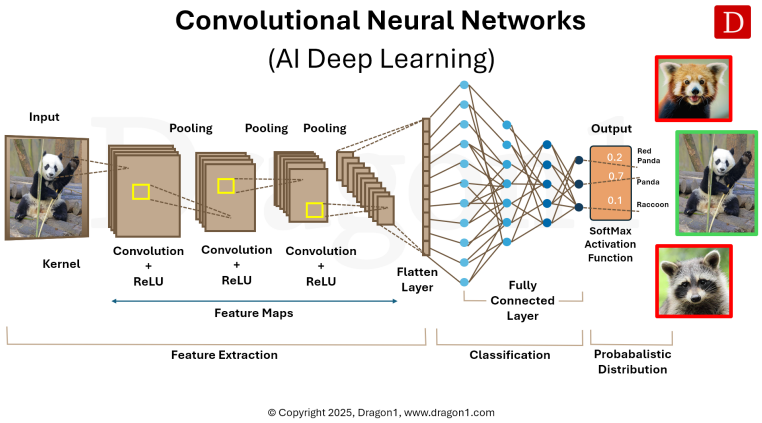


Figure 1.2: How CNN works [9]

1.4 Object Detection

1.4.1 Introduction to Object Detection

Object detection is a computer vision task that aims to locate objects in digital images. As such, it is an instance of artificial intelligence that consists of training computers to see as humans do, specifically by recognizing and classifying objects according to semantic categories. Object localization is a technique for determining the location of specific objects in an image by demarcating the object through a bounding box. [10]

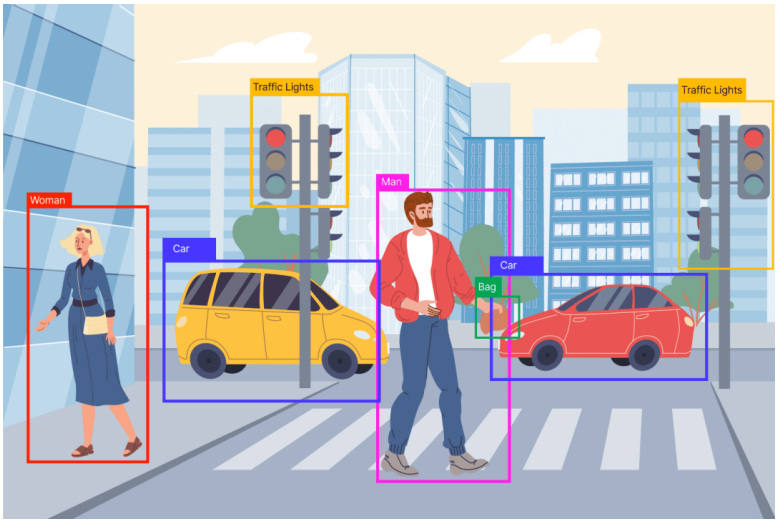


Figure 1.3: How Object Detection works [11]

1.4.2 Traditional Detection Methods

Before deep learning, object detection relied on manually designed features and classical machine learning algorithms. These methods involved three stages:

- **Sliding Window:** Scans the image at various positions and scales to identify candidate regions.
- **Feature Extraction (HOG)** [12]: Captures object shapes and contours using image gradient histograms.
- **Classification (SVM)** [13]: Analyses extracted features to identify the object class.

Although widely used in earlier systems, these methods suffer from high computational cost and limited robustness in complex environments — particularly under low illumination, noise, rust, or partial occlusion, which are common in container images.

1.4.3 Deep Learning-Based Detection

With the rise of deep learning, object detection became dramatically more accurate and efficient. CNN-based detectors automatically learn discriminative visual features — edges, shapes, textures — directly from data. Modern detectors fall into two families:

- **Two-Stage Detectors (e.g., Faster R-CNN)** : First propose candidate regions, then classify each one. High accuracy but slower inference.
- **Single-Stage Detectors (e.g., YOLO, SSD):** Predict bounding boxes and class probabilities in a single forward pass. Faster and well-suited for real-time applications.

1.4.4 YOLO Definition

YOLO (You Only Look Once) [14], was proposed by Joseph Redmon et al. in 2015 to solve the speed limitations of existing detectors. At the time, Fast R-CNN took 2–3 seconds per image, making real-time use impossible. YOLO reformulates detection as a single regression problem: the entire image is processed in one forward pass, producing bounding boxes and class probabilities simultaneously.

1.4.5 YOLO Principle and Architecture

The core of YOLO's architecture consists of a single CNN that simultaneously predicts bounding boxes and class probabilities. Its key components are:

- **Grid Division:** The input image is divided into an $S \times S$ grid. Each cell is responsible for detecting objects whose centre falls within it.
- **Bounding Box Prediction:** Each grid cell predicts B bounding boxes, each with an associated confidence score.
- **Class Prediction:** Each cell also predicts class probabilities C, conditioned on an object being present.
- **Final Predictions:** Non-maximum suppression (NMS) filters overlapping boxes to produce clean final detection. [15]

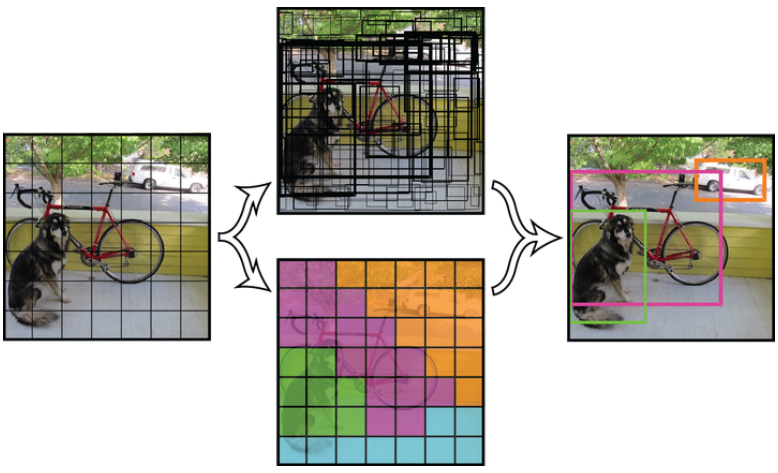


Figure 1.4: YOLO architecture and prediction mechanism [16]

1.4.6 The Reason for Using YOLO

YOLO was selected for this project because it offers an excellent balance between speed and accuracy, which is essential for real-time container detection in smart ports.

1.5 Optical Character Recognition (OCR)

1.5.1 OCR Definition

Optical Character Recognition (OCR) is a technology that converts images of text — whether printed, handwritten, or appearing in natural scenes — into machine-readable strings. By analyzing visual patterns, OCR engines can recognize individual characters and reconstruct the underlying text with high accuracy in a matter of milliseconds, enabling automated data entry and document processing at scale. [17]

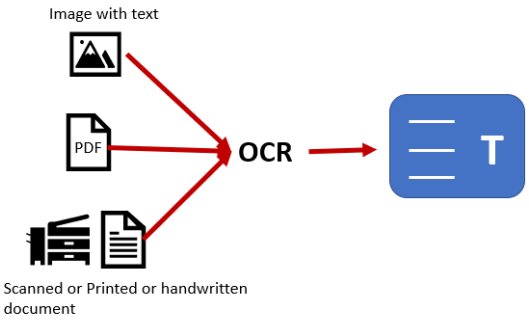


Figure 1.5: How OCR works [18]

1.6 Conclusion

This chapter explained the main ideas behind the new tracking system. First, it looked at how modern shipping and logistics need automation to work faster. Then, it explained the basic rules of deep learning, showing how computers can automatically find important details in images. Next, it covered the YOLO system, which finds shipping containers in real-time. Finally, it explained how PaddleOCR reads the text on those containers and turns them into standard ISO tracking numbers. All of these ideas will be used to design and build the system in the next chapters.

Chapter 2

System Analysis and Design

2.1 Introduction

This chapter focuses on the analysis and design of the proposed system. It details the requirements analysis, modeling of the system using UML diagrams, and the data structures necessary for its operation. A clear conceptual design is crucial for building a scalable and reliable architecture.

2.2 Requirements Analysis

2.2.1 UML Presentation

UML (Unified Modeling Language) is a graphical modeling language used to represent software systems from structural, behavioral, and interaction perspectives. In this project, sequence diagrams are used to illustrate the interactions between the different microservices of the proposed architecture.

2.2.2 System Requirements Analysis

Functional Requirements

The implemented system must support the following functionalities:

- Real-time acquisition of video frames from RTSP camera streams.
- Automatic detection of container identification regions using YOLO.
- OCR-based extraction of ISO container codes from detected regions.
- Support for both vertical and horizontal text orientations.
- Real-time delivery of OCR results through REST APIs and streaming interfaces.
- Validation of extracted codes according to the ISO 6346 format.

Non-Functional Requirements

The system must also satisfy several non-functional requirements:

- High processing performance with low latency.
- Reliable operation during continuous real-time video analysis.
- Modular design for easier maintenance and future extensions.
- Accurate recognition under different environmental conditions.

- Containerized deployment using Docker for portability and reproducibility.

2.3 Use Case Diagram

The use case diagram illustrates how the system interacts with external entities, called actors. It specifies the events that take place within the system, but does not provide details on the implementation of these events.

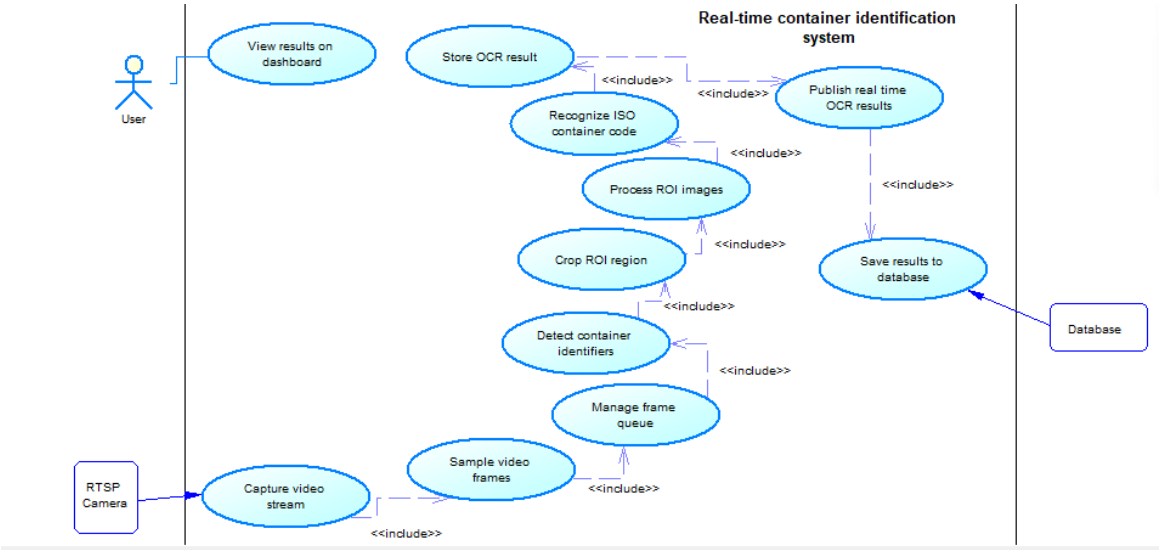


Figure 2.1: Use Case diagramme

2.3.1 Use Case Description

- **Capture Video Stream:** The RTSP camera sends live video frames to the system.
- **Sample Video Frames:** The system extracts frames from the video stream at regular intervals.
- **Manage Frame Queue:** The extracted frames are stored in a queue for processing.
- **Detect Container Identifiers:** The YOLO model detects the container code region in each frame.
- **Crop ROI Region:** The detected container identifier region is cropped from the image.
- **Process ROI Images:** The cropped image is preprocessed to improve OCR accuracy.
- **Recognize ISO Container Code:** The OCR module reads the ISO container code from the processed image.
- **Store OCR Result:** The recognized OCR result is temporarily stored by the system.
- **Publish Real-Time OCR Results:** The system sends OCR results in real time for live monitoring.
- **Save Results to Database:** The OCR results are saved in the database.
- **View Results on Dashboard:** The user can view OCR results through the dashboard interface.

2.4 Conceptual study

2.4.1 Data Dictionary

The data dictionary provides a detailed list of the information contained within the system. For each data element, the following are specified: Name, Type, Size, Description.

Table 2.1: Updated Data Dictionary Based on the Class Diagram

Data	**Type**	**Size**	**Attribute / Code**
Video identifier	AN	50	video_id
Frame identifier	AN	50	frame_id
Frame timestamp	N	10	timestamp
Encoded frame image (Base64)	AN	255	image_b64
ROI bounding box coordinates	AN	100	roi_bbox
Detection confidence score	N	10	confidence
ROI cropped image (Base64)	AN	255	roi_image_b64
Detected ISO text	A	30	iso_text
OCR confidence score	N	10	ocr_confidence
Camera source URL	AN	255	source
Frames per second	N	10	fps
YOLO confidence threshold	N	10	confidence
Redis frames queue	AN	100	frames_queue
Redis ROIs queue	AN	100	rois_queue
Redis OCR results queue	AN	100	ocr_results_queue
OCR Pub/Sub channel	AN	100	ocr_results_channel
OCR result status	A	20	status
Maximum staleness age	N	10	max_age

2.4.2 Class Table

The class table describes the various classes (entities) identified during the system analysis. For each class, the following are specified:

- Attributes (properties)
- Methods (operations/functions)

Table 2.2: Class Table

Classe	**Attributs**	**Méthodes**
Camera	- source : str - cap : Optional[VideoCapture]	+ open() : bool + read_frame() : Optional[ndarray] + release() : void + fps : float
FrameProducer	- video_id : str - redis : Redis	+ push_frame(frame : ndarray) : void
FrameJob *«dataclass»*	+ video_id : str + frame_id : str + timestamp : float + image_b64 : str	+ to_dict() : dict + from_dict(data) : FrameJob + is_stale(max_age) : bool
ISOZoneDetector	- model : YOLO - confidence : float	+ detect(frame : ndarray) : list[dict] + crop_roi(frame : ndarray, bbox : list[int]) : Optional[ndarray]
Detection *«dataclass»*	+ video_id : str + frame_id : str + timestamp : float + roi_bbox : list[int] + confidence : float + roi_image_b64 : str	+ to_dict() : dict + from_dict(data) : Detection + is_stale(max_age) : bool
ISOCodeReader	- ocr : PaddleOCR	+ read(image : ndarray) : Optional[Tuple[str, float]]
OCRResult *«dataclass»*	+ video_id : str + frame_id : str + timestamp : float + roi_bbox : list[int] + confidence : float + iso_text : str + ocr_confidence : float + roi_image_b64 : str	+ to_dict() : dict + from_dict(data) : OCRResult
APIRouter *«module»*		+ get_results(limit) : dict + get_result_by_frame(frame_id) : dict + stream() : EventSourceResponse + health() : dict
Redis *«infrastructure»*	+ frames_queue : List + rois_queue : List + ocr_results_queue : List + ocr_results_channel : Channel(Pub/Sub)	

2.4.3 Relationship Table

The relationship table models the relationships and associations between the different classes in the system. For each link, the following are identified:

- Source class
- Target class
- Cardinality (or multiplicity, e.g., 1 or 0..n)

This table completes the static view by specifying the relationships and dependencies between the classes.

Table 2.3: Relationship Table

Lien (Action)	**Classes**	**Cardinalités**
feeds frames	Camera — FrameProducer	$1 \rightarrow 1$
creates	FrameProducer — FrameJob	$1 \rightarrow 0..*$
RPUSH frames_queue	FrameProducer — Redis	$1 \rightarrow 1$
BRPOP frames_queue	Redis — ISOZoneDetector	$1 \rightarrow 1$
creates per frame	ISOZoneDetector — Detection	$1 \rightarrow 0..*$
RPUSH rois_queue	ISOZoneDetector — Redis	$1 \rightarrow 1$
BRPOP rois_queue	Redis — ISOCodeReader	$1 \rightarrow 1$
creates per ROI	ISOCodeReader — OCRResult	$1 \rightarrow 0..1$
RPUSH + PUBLISH ocr_results	ISOCodeReader — Redis	$1 \rightarrow 1$
LPOP + SUBSCRIBE ocr_results	Redis — APIRouter	$1 \rightarrow 1$
exposes	APIRouter — OCRResult	$1 \rightarrow 0..*$

2.5 Classes Diagram

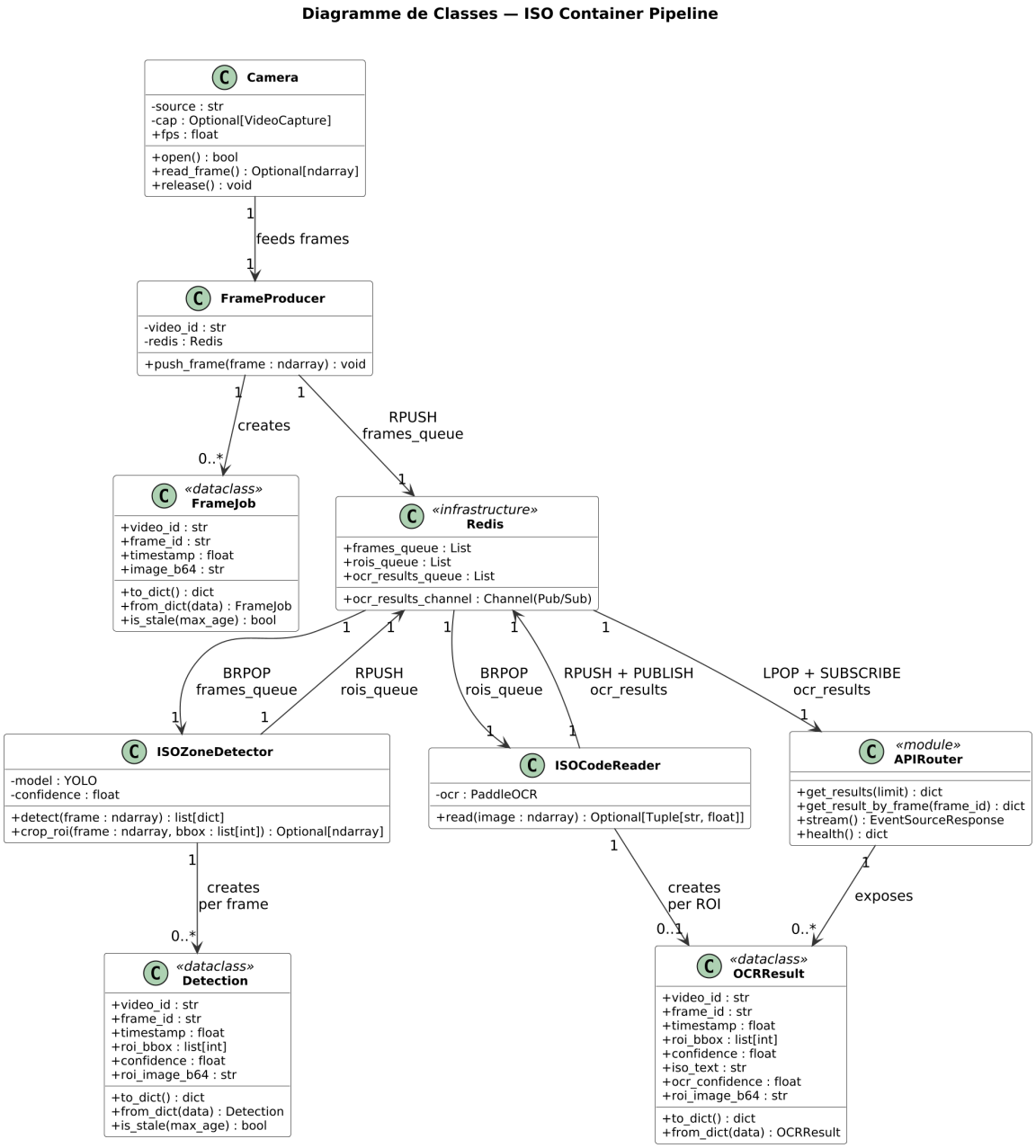


Figure 2.2: classes Diagramme

2.6 Sequence Diagrams

A sequence diagram provides a chronological representation of the interactions between system elements, actors, and objects.

2.6.1 Video Sampling Sequence Diagram

The video sampling service continuously captures frames from the RTSP camera stream at configurable intervals. Each valid frame is compressed into JPEG format, encoded using Base64, and transmitted to the Redis frames_queue.

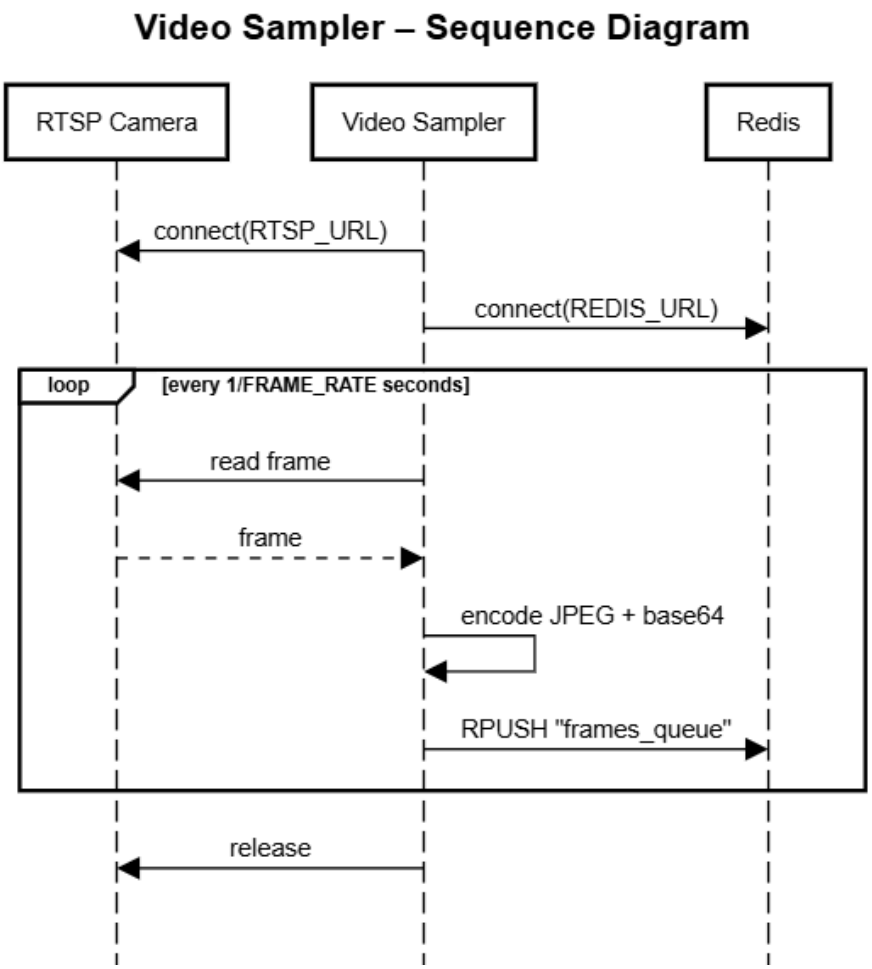


Figure 2.3: Vidio sampler sequence diagramme

2.6.2 Object Detection Sequence Diagram

The Object Detection Service consumes frames from Redis and applies the YOLO model to detect container identification regions. Valid detections are cropped and forwarded to the rois-queue,stale frames (>5s) are discarded.

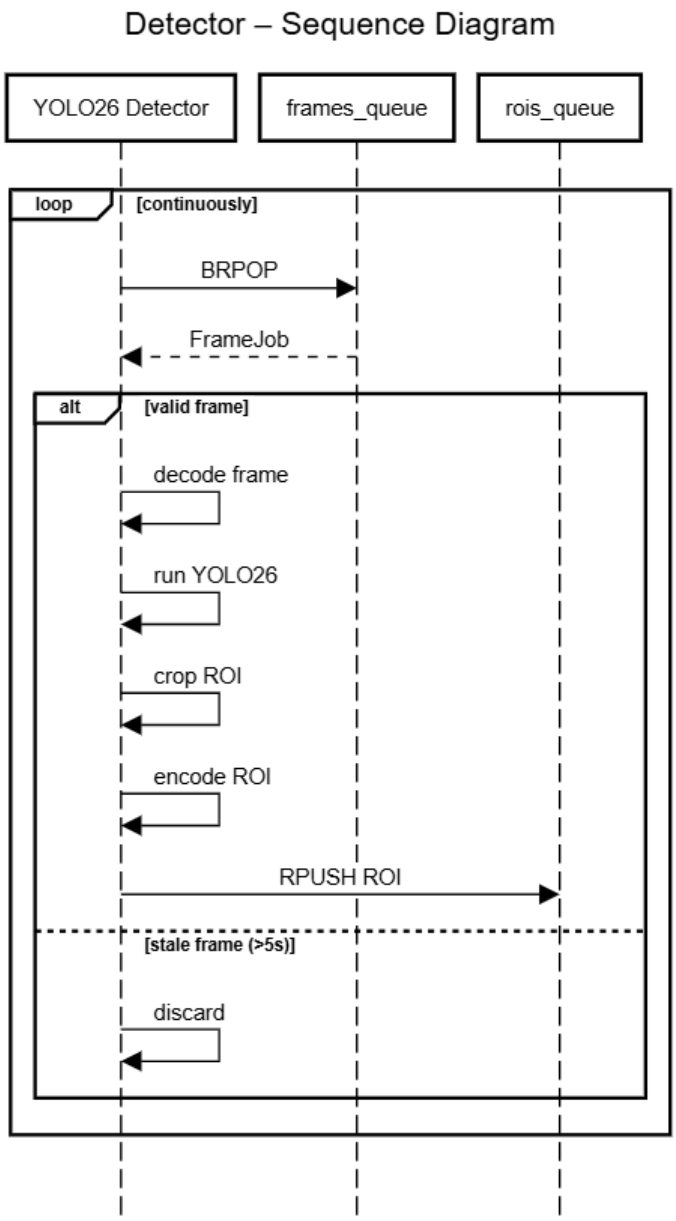


Figure 2.4: YOLO sequence diagramme

2.6.3 OCR Processing Sequence Diagram

The OCR Worker Service processes cropped container regions using PaddleOCR to recognize ISO container identification codes. The extracted text is validated before being published to the result queue, invalid results are discarded.

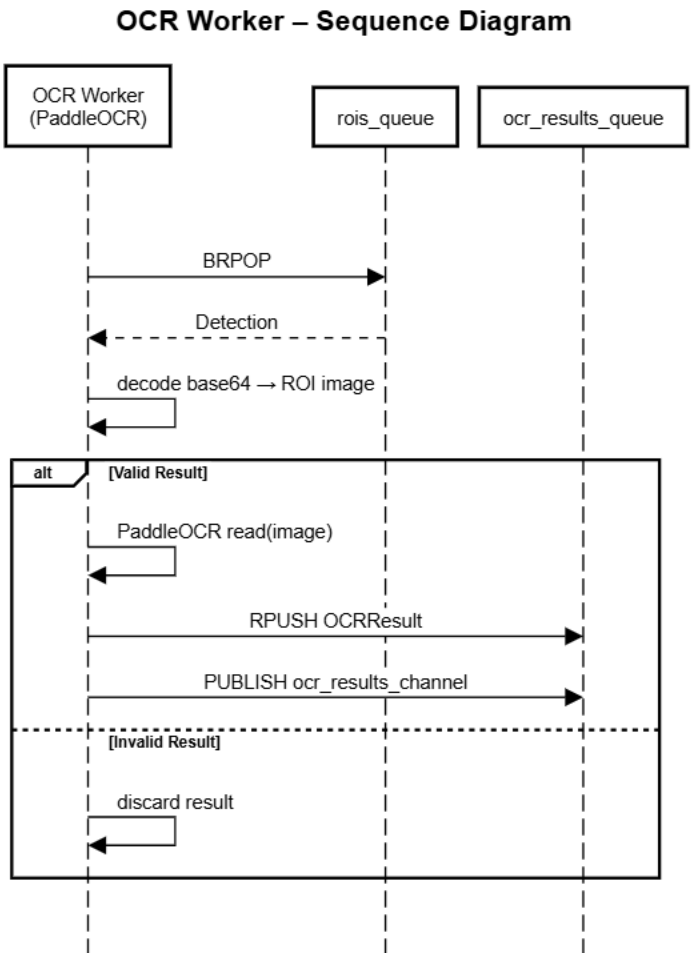


Figure 2.5: OCR sequence diagramme

2.6.4 FastAPI Result Delivery Sequence Diagram

The FastAPI backend provides both historical OCR retrieval and real-time streaming services. OCR results are temporarily stored in memory and streamed to dashboard clients using Server-Sent Events (SSE).

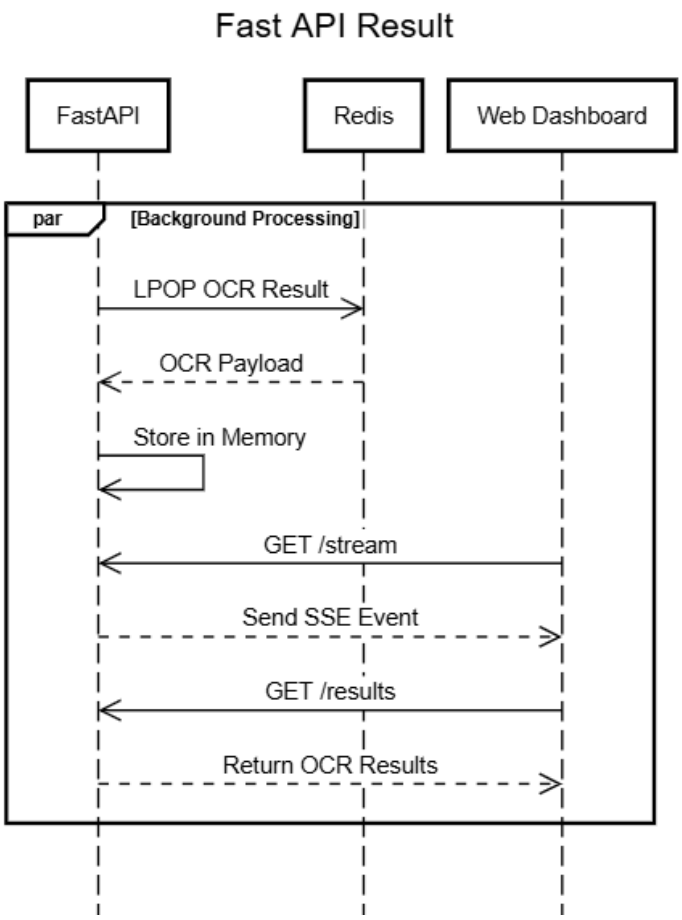


Figure 2.6: FastApi sequence diagramme

2.7 Conclusion

This chapter established the functional and non-functional requirements of the system, illustrating the architecture and data flows through comprehensive UML diagrams. These models directly inform the technological and architectural choices made during the implementation phase.

Chapter 3

System Methodology and Implementation

3.1 Introduction

This chapter presents the design and methodology of the proposed system for real-time detection and recognition of ISO 6346 container codes from video streams. It describes the complete processing pipeline, starting from data preparation and object detection to image preprocessing and OCR-based text extraction.

It also covers the system implementation, including the technology stack, microservices architecture, asynchronous communication using Redis, and deployment using Docker. The goal is to show how all components work together to achieve accurate and real-time container identification.

3.2 Problem Statement

3.2.1 Manual Identification Challenges

Manual container identification is slow and prone to human errors, especially when processing large numbers of containers. Misreading similar characters and manual data entry can cause delays and inaccurate records.

3.2.2 Environmental Challenges

Container identification in port environments is strongly affected by several environmental factors that reduce the quality of captured images and directly impact the performance of OCR systems.

- **Lighting Impact:** Low-quality light can drop accuracy by up to 30%.
- **Surface Damage:** Rust, dirt, and scratches make IDs hard to read.
- **Motion Blur and Glare:** Caused by moving trucks, cranes, or strong lighting sources, further complicating image acquisition.

3.3 Project Objectives

The main objective of this project is to develop an intelligent system capable of automatically detecting and recognizing ISO 6346 container identification codes from real-time video streams.

The proposed system aims to:

- Detect the container identification region using YOLO.
- Pre-process the cropped image to improve text readability.
- Recognize container codes using OCR techniques.
- Handle both horizontal and vertical text orientations.
- Provide real-time processing through asynchronous micro-services communication.

3.4 Methodology

The proposed methodology is based on a real-time processing pipeline designed to automatically detect and recognize ISO 6346 container identification codes from video streams. The system combines object detection, image preprocessing, optical character recognition, and asynchronous communication techniques to ensure accurate and efficient processing under real-world port conditions.

3.4.1 Dataset Collection

The first step in the proposed methodology consists of collecting a representative dataset of container images that reflects real-world operating conditions. Since the objective of this project is to detect and recognize container identifiers, the dataset must include images where the identification code is clearly visible under different scenarios.

The system uses the ACCLR dataset obtained from Kaggle, it includes both real-world container images and synthetic scene-text samples, offering diversity in lighting, tilt, noise, orientation, color variation, background clutter, and font styles.

3.4.2 Model Training

The object detection task is modeled as a supervised learning regression and classification problem, trained to locate the single class ISO-Zone. To achieve real-time processing on edge devices, the YOLO single-stage detection framework was selected. YOLO processes the image in a single step by directly predicting bounding boxes b=(x,y,w,h,c),which represent the object location, size, and confidence score. This approach allows fast and efficient detection, making YOLO suitable for high-speed real-time video streams.

The system uses YOLO26 [19] as the object detection model due to its high inference speed and suitability for real-time applications. The model is trained to detect the container identification region directly from video frames. During training, the dataset is divided into training and validation subsets to evaluate the model's generalization capability.

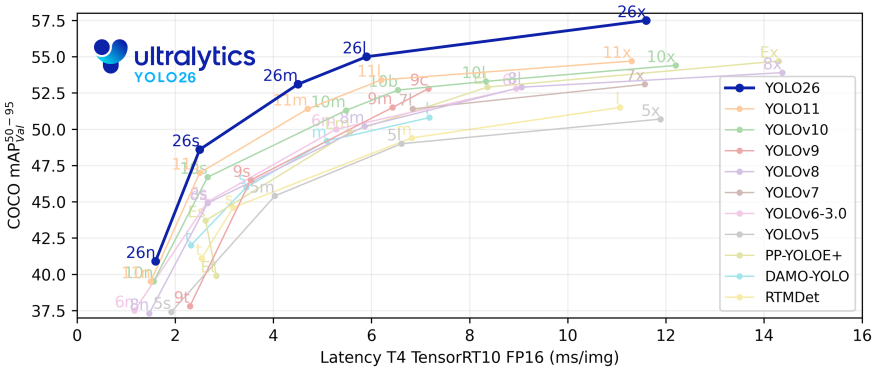


Figure 3.1: Benchmark evaluating YOLO26 against previous YOLO versions [19]

The model optimizes a composite multi-task loss function Ltotal to penalize localization and classification errors: Ltotal= box LCIoU+ cls Lcls+ dfl Ldfl.This loss formulation enables the model to improve bounding box accuracy while minimizing classification errors during training.The model performance is evaluated using several metrics, including:

- Precision (P=TP/TP+FP): Evaluates false positive suppression
- Recall (R=TP/TP+FN): Measures sensitivity (avoiding missed containers).
- Intersection over Union (IoU): Measures bounding box alignment.
- mAP@0.5: The mean Average Precision calculated at an IoU threshold of 0.5, representing the overall localization quality.
- mAP@0.5:0.95: The mean Average Precision calculated across IoU thresholds ranging from 0.5 to 0.95, representing the model's overall detection and localization performance.

Once trained, the detector predicts the bounding box coordinates surrounding the ISO container code.

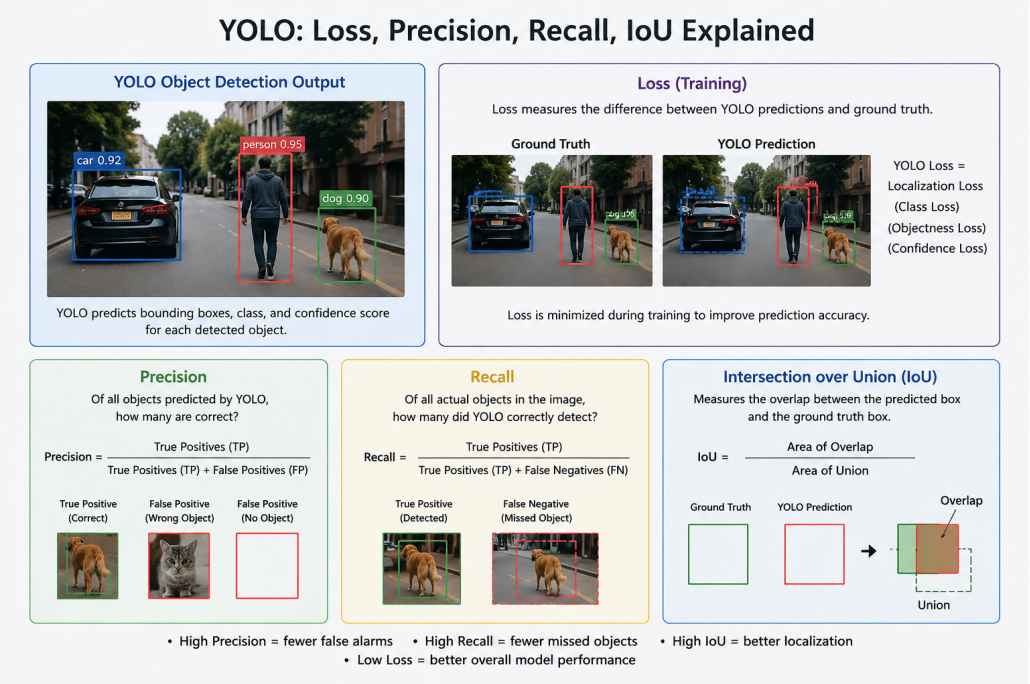


Figure 3.2: Visual Explanation of YOLO Object Detection Performance Metrics

3.4.3 Image Preprocessing

After detection, the cropped ROI undergoes several preprocessing operations to improve text readability before OCR processing. The preprocessing pipeline includes:

- **Resizing:** Scaling the image down to an optimal size to reduce the computational load and accelerate the system's processing speed.
- **Grayscale Conversion:** Converting the image from full color to shades of gray. This simplifies the image for the computer by removing unnecessary color data.
- **Noise Removal:** Eliminating minor visual artifacts, camera blur, and background dirt to clarify the target area.
- **Binarization:** Converting the image into a strict black-and-white format, where every pixel is either completely black or completely white.

3.4.4 Text Recognition using Paddle-OCR

Paddle-OCR is a state-of-the-art open-source Optical Character Recognition (OCR) framework developed by Baidu and built on the PaddlePaddle deep learning platform. It supports over 80 languages and is based on a CRNN (CNN + RNN) architecture, which combines convolutional feature extraction with sequential character decoding. The system operates through three main modular stages: text detection, where a DB (Differentiable Binarization) detector identifies and localizes text regions within an image; orientation classification (optional), which detects and corrects rotated or upside-down text regions to ensure proper alignment; and text recognition, where a CRNN-based model interprets each detected region and outputs the corresponding character sequence.

In the proposed system, the preprocessed Region of Interest (ROI) is fed into the OCR pipeline to extract ISO container identifiers. Since container codes may appear in both horizontal and vertical orientations, the system first analyzes the geometric properties of the ROI to determine text orientation. For vertically oriented text, characters are segmented and interpreted sequentially from top to bottom, while for horizontally oriented text, the ROI is corrected for skew and processed as a continuous text line. The extracted text is then passed through a post-processing stage where it is validated against the ISO 6346 standard format to ensure structural correctness and reliability of the detected container identifier.

3.4.5 Asynchronous Processing Pipeline

To ensure real-time performance and scalability, the system adopts a producer-consumer microservices architecture based on asynchronous communication. The processing pipeline is divided into independent services, where each service is responsible for a dedicated task within the workflow:

- Video sampling
- Object detection
- OCR processing
- Result distribution

This modular design improves fault isolation, simplifies maintenance, and allows each component to scale independently according to workload requirements.

Communication between services is handled through asynchronous message passing using Redis message queues. FIFO (First-In, First-Out) queues are used to distribute tasks sequentially between processing stages, while a Publish-Subscribe mechanism enables real-time streaming of OCR results to connected clients and backend services.

In high-throughput streaming environments, variations in processing speed may lead to queue accumulation and increased system latency. To mitigate this issue, each consumer service validates the timestamp of incoming data before processing. Frames whose age exceeds a predefined temporal threshold are automatically discarded.

This staleness-control strategy prevents outdated data from congesting the pipeline and ensures that the system prioritizes recent frames, thereby maintaining high temporal responsiveness and stable real-time performance.

3.4.6 System Workflow

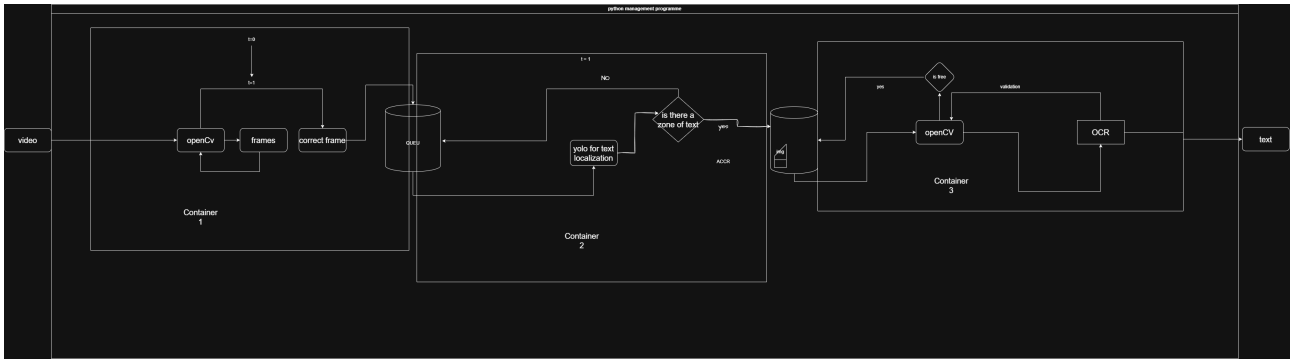


Figure 3.3: System workflow

3.5 Implementation

3.5.1 Technology Stack

The technology stack was carefully selected to balance high-performance AI inference with rapid backend development and robust container orchestration.

Machine Learning Technologies

- **YOLO26:** Used for real-time detection of container identification regions.
- **PaddleOCR:** Used for optical character recognition (OCR) of ISO container codes.
- **OpenCV:** Used for image processing and preprocessing operations.

Backend Technologies

- **Python:** Main programming language used for AI processing and backend development.
- **FastAPI:** Asynchronous framework used to build high-performance REST APIs and streaming services.

Messaging and Deployment Technologies

- **Redis:** Used as the message broker for asynchronous communication between microservices.
- **Docker & Docker Compose:** Used to containerize and orchestrate the different system services.

3.5.2 The Core Microservices

Video Sampler Service

The Video Sampler Service is responsible for capturing video frames from RTSP camera streams or local video files and sending them to the processing pipeline.

The service utilizes the OpenCV library (`cv2.VideoCapture`) to establish a connection with the video source and continuously read incoming frames. Instead of processing every frame

produced by the camera (typically 30 FPS), the service applies temporal downsampling by selecting frames at a configurable rate (e.g., 2 FPS). This strategy significantly reduces computational load, memory consumption, and network overhead while preserving sufficient visual information for downstream detection tasks.

Each selected frame is compressed into JPEG format using OpenCV (`cv2.imencode`) with a predefined quality factor, and subsequently encoded into a Base64 string to enable safe transmission across distributed services.

Each frame is associated with structured metadata, including:

- a unique video session identifier,
- a unique frame identifier,
- a timestamp used for synchronization and staleness control,
- and the encoded image payload.

This metadata is encapsulated into a `FrameJob` object following a standardized JSON contract before being pushed to the Redis `frames_queue` using an LPUSH operation.

To improve system reliability, the service continuously monitors the camera connection. In the event of stream interruption, the video interface is safely released and automatic reconnection attempts are performed to maintain continuous pipeline operation.

Object Detection Service

The Object Detection Service is responsible for consuming video frames from the Redis queue and detecting the ISO code region on shipping containers. The service operates continuously using a Redis consumer loop that listens to the `frames_queue` channel for incoming frames.

Real-Time Frame Consumption with Staleness Control
To maintain real-time performance, the system avoids processing outdated frames. If a frame is too old (e.g., > 5 seconds), it is discarded immediately to prevent backlog and latency accumulation.

Region of Interest (ROI) Cropping
After detection, the system extracts only the relevant part of the image using bounding box coordinates. Before sending the ROI to the OCR service, it is enhanced to improve text readability. The preprocessing pipeline includes:

- Resizing to a standard width.
- Grayscale conversion.
- Contrast enhancement using CLAHE (Contrast Limited Adaptive Histogram Equalization).

Object Detection Model Training Configuration
To achieve accurate and reliable detection of ISO container codes, the YOLOv26 model (experiment name: `container_detector`) was trained using a configuration specifically adapted for robust text region localization.

- **Image Size and Batch Processing:** All input images were resized to `640 × 640` pixels (`imgsz=640`). The model processed data in batches of 8 images per step (`batch=8`), balancing memory usage and training stability.
- **Training Process:** The training was configured to run for a maximum of 300 epochs (`epochs=300`). An early stopping mechanism with a patience value of 50 epochs (`patience=50`) was employed to prevent overfitting. Training automatically stops if there is no improve-

ment in the validation metrics for 50 consecutive epochs.

- **Data Augmentation Settings:** Horizontal flipping was enabled with a probability of 0.5 (`fliplr=0.5`), allowing the model to become more robust to left–right variations. Vertical flipping was disabled (`flipud=0.0`) since the detected regions contain readable text; flipping vertically would make the text unreadable, which could negatively impact the model's learning.
- **Single-Class Mode:** The model was trained to detect a single target class (`single_cls=true`), simplifying the task and focusing the model's learning on accurately localizing the container code region.
- **Checkpoint and Configuration:** Pretrained weights (`model=yolo26l.pt`, `pretrained=true`) were used to accelerate convergence. The full training configuration was stored in `yolo26_exp17.yaml` including optimizer settings, augmentation strategies, and learning rate scheduling.
  - Learning Rate: `lr0 = 0.01`, `lrf = 0.01`
  - Regularization: `weight_decay = 0.0005`, `momentum = 0.937`
  - Mosaic Augmentation: enabled during early training (`mosaic=1.0`) and disabled for the last 10 epochs (`close_mosaic=10`)
  - Additional Augmentations: RandAugment, HSV transformations (`hsv_h=0.015`, `hsv_s=0.7`, `hsv_v=0.4`), and random erasing (`erasing=0.4`)
  - NMS-Free Design: Non-Maximum Suppression was disabled (`nms=false`) to enable end-to-end direct prediction, reducing latency and ensuring predictable inference times

```python
def main():
    model26 = YOLO("yolo26l.pt")
    results26 = model26.train(
    data=str(data_yaml),
    epochs=300,
    imgsz=640,
    batch=8,
    device=0,
    workers=8,
    project="runs/train27",
    name="yolo26_exp",
    pretrained=True,
    optimizer="auto",
    lr0=0.01,
    lrf=0.01,
    momentum=0.937,
    weight_decay=0.0005,
    patience=50,
    amp=True,
```

Figure 3.4: YOLO Model Training Setup

OCR Worker Service

The OCR Worker Service is a microservice responsible for extracting ISO container codes from preprocessed ROI images in real time. It operates as a consumer in a Redis-based queue system,

continuously retrieving ROI jobs from the `rois_queue`.

OCR is performed using PaddleOCR, which processes each ROI after determining its text orientation. Orientation is inferred by comparing the height and width of the image:

- **Vertical Text Strategy:** When vertical text is detected, the system applies a character-level recognition approach to ensure accurate extraction in constrained or stacked layouts. The process begins by applying contour detection using cv2.findContours on the binarized image. Bounding boxes are then extracted using cv2.boundingRect, and irrelevant detections are filtered out based on geometric constraints such as minimum width, height, and aspect ratio.
  Once valid character regions are identified, they are sorted according to their vertical position using the Y-coordinate to preserve the correct reading order from top to bottom. Each character is then cropped from the original image using NumPy slicing and slightly padded to reduce boundary noise.
  To standardize input for OCR inference, each character is centered on a square canvas generated using `np.zeros`, ensuring consistent dimensions across all samples. The character image is then inverted using `cv2.bitwise_not` to match the expected input format of PaddleOCR. The OCR engine is executed in recognition-only mode using `ocr.ocr({ocr_ready_img}, det=false, cls=false)`, producing a predicted character along with a confidence score. All recognized characters are filtered to retain only alphanumeric values and concatenated sequentially to reconstruct the final container identifier.
- **Horizontal Text Strategy:** To process horizontal text, the system first calculates the skew angle and rotates the image to ensure perfect horizontal alignment. Once aligned, the entire Region of Interest (ROI) is evaluated by PaddleOCR using `ocr.ocr({rotated_img}, det=true, cls=false)` for both text detection and recognition. The detected text blocks are subsequently sorted from left to right according to their x-coordinate positions. Finally, the extracted strings are filtered to retain only alphanumeric characters and concatenated to produce the final output.

The extracted text is validated using the ISO 6346 regular expression pattern `[A-Za-z]{4}\d{7}`. Only valid results are accepted. Finally, recognized container codes are published to the `ocr_results_queue` and broadcast through the `ocr_results_channel` for real-time downstream consumption.

3.5.3 FastAPI Backend

The FastAPI backend provides REST APIs and real-time streaming services for OCR results. This design allows the API to handle high volumes of concurrent requests without blocking. The system exposes three core REST endpoints:

- `GET /health`: This is a way to check if the API is working correctly and if it is connected to the Redis message broker.
- `GET /results`: This gives users a list of the OCR results that the system has processed. For example it might give them the 50 items in the list. It does this by getting the data from the queue so users always get the latest information.
- `GET /results/{frame_id}`: This lets users ask for the results for a specific video frame.

3.5.4 Data Models and Communication

The microservices communicate asynchronously through Redis queues and Pub/Sub channels. LPUSH and BRPOP operations are used for queue communication, while Redis Pub/Sub is used for real-time event streaming.

The system has a 3-Stage Pipeline with three queues:

- **frames_queue:** Holds raw video frames sent from the video sampler, waiting to be processed by the Object Detection (YOLO) worker.
- **rois_queue:** Holds the cropped images of containers (Regions of Interest) identified by the detector, waiting for the OCR worker.
- **ocr_results_queue:** Holds the final extracted ISO container codes, waiting for the FastAPI backend to send them to the user.

3.5.5 Docker Deployment

The complete architecture is containerized using Docker to ensure portability and reproducibility across different environments. Each microservice runs inside an independent container:

- Video Sampler Container
- Detection Container
- OCR Worker Container
- FastAPI Container
- Redis Container

Docker Compose is used to orchestrate the multi-container environment, manage networking between services, and simplify deployment procedures. This deployment strategy allows the system to be easily scaled and maintained in production environments.

3.6 Conclusion

This chapter presented the design and implementation of the proposed real-time system for ISO 6346 container code detection and recognition. It detailed the full pipeline, including object detection, image preprocessing, OCR processing, and asynchronous microservices communication using Redis and Docker-based deployment. Each component was designed to ensure accuracy, efficiency, and real-time performance in challenging port environments.

In the next chapter, we will present the experimental results and evaluate the performance of the complete pipeline under different conditions.

Chapter 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the test results and evaluates the performance of the proposed real-time container identification system. It uses standard metrics to analyze how well the YOLO detection model and the PaddleOCR recognition stage work.

4.2 Results

4.2.1 Object Detection Results and Evaluation

Figure 4.1 illustrates the evolution of training and validation metrics during the learning process.

First, the training losses (`box_loss`, `cls_loss`, and `dfl_loss`) steadily decrease, indicating that the model learns effectively over time. The validation losses follow a similar downward trend, which confirms good generalization performance and suggests that the model does not suffer from significant overfitting.

Furthermore, Precision and Recall increase rapidly during the early training stages and stabilize at approximately 99%, demonstrating highly reliable object detection performance.

The mAP@0.5 reaches approximately 99.5%, while the more strict metric mAP@0.5:0.95 reaches approximately 90.4%, confirming that the predicted bounding boxes remain accurate under progressively stricter IoU thresholds.

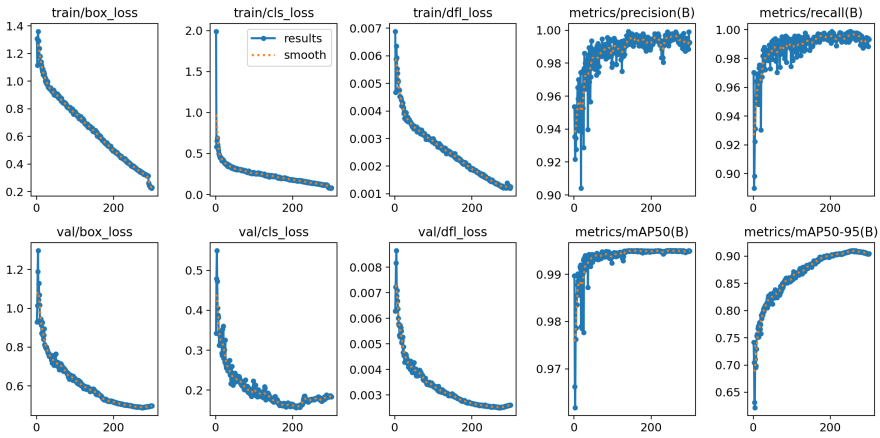


Figure 4.1: Training and validation metrics during model training

At the final training epoch, the obtained results are summarized as follows:

- **Precision:** 0.99259
- **Recall:** 0.99342
- **mAP@0.5:** 0.99493
- **mAP@0.5:0.95:** 0.90442

Confusion Matrix Analysis

The confusion matrix shown in Figure 4.2 illustrates the detection performance of the trained model on the validation dataset.

The model correctly identified 944 target regions while producing only 18 incorrect detections. The absence of significant false negatives indicates that the model successfully detects nearly all container-code regions present in the images.

Overall, these results demonstrate the robustness of the detector and confirm its suitability as the first stage of the OCR-based container identification pipeline.

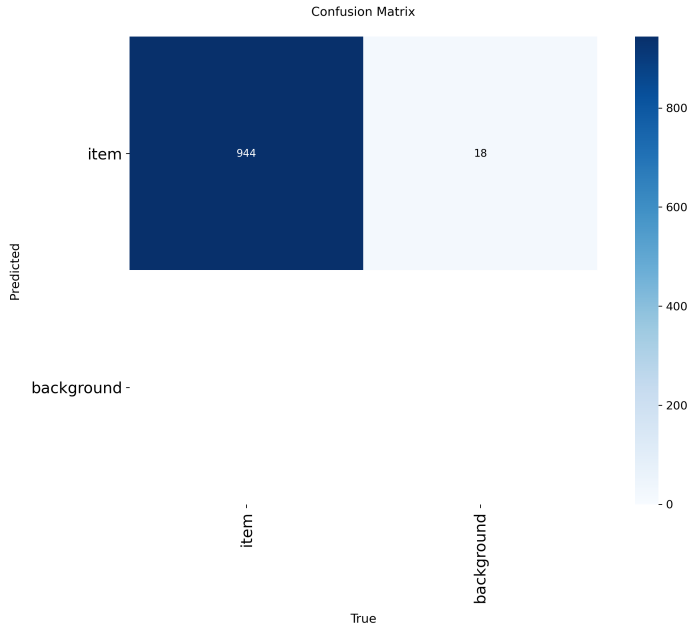


Figure 4.2: Confusion matrix of the trained detection model

Discussion

The obtained results demonstrate that the proposed YOLO-based detector achieves excellent localization performance for container-code region detection. Overall, the detector provides reliable and tightly localized bounding boxes, making it highly suitable as the first stage of the OCR pipeline.

4.2.2 OCR Recognition Results

The Optical Character Recognition (OCR) module was evaluated on the test video dataset under different environmental and capture conditions. The obtained results indicate that the proposed system achieved high recognition accuracy for most of the detected container IDs.

However, most recognition errors were caused by poor detection. If the bounding box was incomplete, blurry, or misaligned, the OCR module generated incorrect readings.

Figure 4.3 shows an ideal run of the OCR pipeline. As seen in the terminal logs, the system correctly processes the extracted Region of Interest (ROI) and separates clear text parts: the alphabetic owner code ("TEMU") and the numeric serial number including its check digit ("6372145"). Both parts were detected with a confidence score of 1.00.

In addition, the post-processing step successfully combines the extracted text and validates it using system rules, confirming that a valid container ID was obtained.



Figure 4.3: ROI image output example

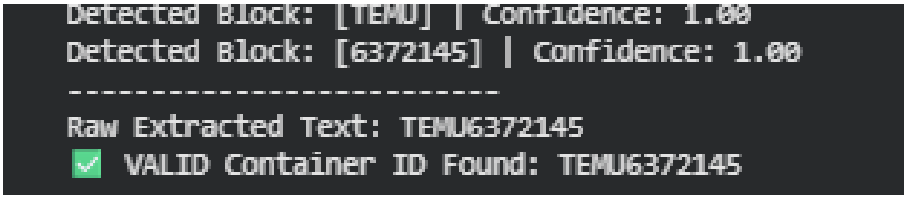


Figure 4.4: OCR result example

4.3 Error Analysis and Discussion

Although the proposed system achieved strong detection and recognition performance, several errors were observed during real-world deployment. These errors mainly originate from environmental conditions, object detection limitations, and OCR sensitivity to input quality.

To maintain real-time responsiveness, the system applies a staleness-control mechanism that discards frames older than 5 seconds. While this prevents queue congestion and excessive latency, it may also lead to missed detections under heavy processing load. When the processing speed becomes lower than the frame acquisition rate, some frames are discarded before reaching the detection stage, even if they contain clearly visible ISO codes.

The YOLO detector mainly produced three categories of errors:

- **Missed detections:** Occur when the ISO code region is too small, partially occluded, blurred, or poorly illuminated.
- **False positives:** Occur when visually similar regions, such as logos or painted markings, are incorrectly detected as ISO code regions.
- **Partial bounding boxes:** Occur when the predicted ROI does not fully cover the container code, causing some characters to be truncated before OCR processing.

The OCR stage is also highly dependent on the quality of the cropped ROI. Several factors reduce OCR accuracy, including JPEG compression artifacts, low-resolution crops caused by distant containers, and image upscaling that cannot recover lost visual details. In some cases, CLAHE enhancement amplifies image noise, while grayscale conversion removes useful color information that could help distinguish text from the background. These effects reduce character sharpness and lower OCR confidence.

In addition, OCR performance is tightly coupled with YOLO detection quality. Errors intro-

duced during detection directly propagate to the recognition stage:

- Missed detection → no OCR result generated
- Partial bounding box → incomplete or truncated text
- Misaligned ROI → noisy or incorrect OCR output
- Blurry frame → low-confidence recognition

However, experimental observations showed that the system remains reliable under moderately difficult environmental conditions. For example, the pipeline continues to operate correctly during normal rain conditions, moderate lighting variations, or when the ISO code region appears slightly small in the frame. Most failures occur only in severe conditions, such as strong motion blur, heavy rain, extreme low lighting, strong occlusion, very distant containers, or highly degraded container surfaces.

Overall, the experimental results demonstrate that the proposed pipeline remains robust under normal operating conditions and moderately challenging environments. Despite certain limitations in extreme scenarios, the system maintains high detection and recognition performance suitable for near real-time smart port applications.

4.4 Conclusion

This chapter evaluated the performance of the proposed real-time ISO container identification system through both quantitative metrics and qualitative analysis. The experimental results showed that the YOLO-based detector achieved high localization accuracy, while the OCR module successfully recognized most container codes under normal operating conditions.

The analysis also highlighted the main limitations of the system, particularly under challenging environments involving motion blur, low illumination, occlusion, and degraded container surfaces.

The obtained results confirm the effectiveness of combining deep learning object detection with OCR technologies for automated container identification from video streams.

GENERAL CONCLUSION

This project focused on the design and implementation of an intelligent system for the automatic detection and recognition of ISO 6346 container codes from video streams. The proposed solution combines deep learning, optical character recognition, and a microservices architecture to reduce manual identification effort in smart port and logistics environments.

The developed system is composed of four main services: a Video Sampler for frame acquisition, a YOLO-based Detection Service for identifying ISO code regions, an OCR Worker using PaddleOCR for text extraction, and a FastAPI backend for result delivery and real-time monitoring. Communication between services is ensured through Redis queues and Pub/Sub channels, while Docker and Docker Compose provide a portable and scalable deployment environment.

Experimental results demonstrated that the YOLO detector achieved high localization performance, with excellent Precision, Recall, and mAP scores. The OCR module also produced reliable recognition results under normal operating conditions. However, the evaluation highlighted several limitations related to motion blur, low lighting, occlusion, low-resolution inputs, and the dependency between detection and OCR quality.

Despite these challenges, the proposed pipeline proved capable of delivering robust near real-time performance suitable for industrial applications. The modular architecture allows future improvements to be integrated easily, including enhanced preprocessing, geometric correction, more advanced OCR models, and feedback mechanisms between detection and recognition stages.

Overall, this work demonstrates the effectiveness of combining computer vision, deep learning, and distributed system design to solve real-world container identification problems in modern logistics systems.

Bibliography

[1] Bogwasi, T. (2026). *Smart Logistics*. Brimco.

[2] AllRead. (2022). *What is a Smart Port?*. AllRead.

[3] Bolevich, M. (2026). *How AI, IoT, and automation are transforming smart ports*. Interesting Engineering.

[4] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[5] Khan, M. S., & Imran, A. (2024). The art of seeing: A computer vision journey into object detection. *Advances in Machine Learning & Artificial Intelligence*, 5(2).

[6] Campesato, O. (2020). *Artificial intelligence, machine learning, and deep learning*. Mercury Learning and Information.

[7] Gadre, V. (2023, July 20). *Deep learning explained: A beginner's guide*. Medium.

[8] Keita, Z. (2026, March 26). What are convolutional neural networks? A complete CNN guide. *DataCamp*.

[9] Dragon1. (n.d.). *Convolutional neural networks definition*. Dragon1

[10] Murel, J., & Kavlakoglu, E. (n.d.). What is object detection? *IBM*.

[11] Samer. (n.d.). *Object detection: From traditional techniques to modern*. LinkedIn.

[12] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1, 886–893.

[13] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

[14] Singh, R. (2024, October 14). Understanding and implementing Faster R-CNN. *Medium*.

[15] mitthudon. (2025, July 23). How does YOLO work for object detection? *GeeksforGeeks*.

[16] Ankushsharma. (2020, June 25). *YOLO V1 architecture*. Medium.

[17] Kundu, R. (2023, January 17). YOLO algorithm for object detection explained [+Examples]. *V7 Labs*.

[18] Goodwin, A. (n.d.). *Application of OCR in archive digitization*. Wondershare.

[19] Ultralytics. (2026). *Ultralytics YOLO26 Documentation*. Retrieved from `https://docs.ultralytics.com/models/yolo26/`