FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Efficient Object Detection on Edge Devices

Luís Miguel da Costa Ferreira

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. João Manuel R. S. Tavares

July 25, 2025

Resumo

A necessidade de informações precisas e em tempo real sobre operações industriais, como o estado de peças em produção, ferramentas, máquinas, materiais, operadores e veículos de transporte, tornou-se cada vez mais essencial nos ambientes de manufatura moderna. Essa demanda é ainda mais intensificada pela integração desses elementos em sistemas de gêmeos digitais, que exigem uma troca de dados fluida e precisa para garantir um monitoramento eficaz e uma tomada de decisão informada. Em resposta a esses desafios, técnicas de aprendizado de máquina, particularmente o aprendizado profundo, emergem como ferramentas transformadoras, permitindo análises avançadas baseadas em imagens e fornecendo insights acionáveis em ambientes industriais complexos e dinâmicos.

Esta Dissertação busca atender a essas demandas por meio do desenvolvimento de modelos de detecção de objetos baseados em aprendizagem de máquina e visão computacional. O objetivo é fornecer informações confiáveis e precisas sobre ambientes industriais, garantindo que as soluções desenvolvidas sejam não apenas eficazes, mas também eficientes o suficiente para operar em dispositivos de borda. Ao longo da pesquisa, diversos modelos de aprendizado profundo serão projetados e treinados com conjuntos de dados específicos para cenários industriais. Esses modelos passarão por extensos testes e avaliações com múltiplas métricas, assegurando que atendam aos requisitos de velocidade, precisão e eficiência necessários para implementação em dispositivos embarcados.

A qualidade dos resultados será avaliada utilizando métricas amplamente adotadas na detecção de objetos, como Precision, F1-score, Recall, mean Average Precision (mAP), Box Loss (para medir a precisão da localização das caixas preditas) e o tempo de inferência, que é essencial para aplicações em tempo real.

Palavras-chave: Aprendizagem de Máquina, Detecção de Objetos, Aprendizagem Profundo, Visão Computacional

Abstract

The need for accurate and real-time information about industrial operations, such as the status of parts in production, tools, machines, materials, operators, and transport vehicles, has become increasingly essential in modern manufacturing environments. This demand is further intensified by the integration of these elements into digital twin systems, which require seamless and precise data exchange to ensure effective monitoring and informed decision-making. In response to these challenges, machine learning techniques, particularly deep learning, have emerged as transformative tools, enabling advanced image-based analysis and delivering actionable insights in complex and dynamic industrial settings.

This Dissertation aims to address these demands through the development of object detection models based on machine learning and computer vision. The goal is to provide reliable and accurate information about industrial environments, ensuring that the proposed solutions are not only effective but also efficient enough to operate on edge devices. Throughout the research, various deep learning models will be designed and trained using datasets tailored to industrial scenarios. These models will undergo extensive testing and evaluation using multiple metrics to ensure they meet the required standards of speed, accuracy, and efficiency necessary for deployment on embedded devices.

The quality of the results will be assessed using widely adopted object detection metrics such as Precision, F1-score, Recall, mean Average Precision (mAP), Box Loss (to evaluate the accuracy of predicted bounding box locations), and inference time, which is critical for real-time applications.

Keywords: Machine Learning, Object Detection, Deep Learning, Computer Vision

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

The specific Sustainable Development Goals mentioned have the following names:

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

SGD	**Target**	**Contribution**	**Performance Indicators and Metrics**
8	8.2	Enhancing the efficiency of object detection in industrial environments increases the reliability of information, which in turn contributes to achieving higher levels of economic productivity.	Cost Reduction Production Output
9	9.5	Enhance scientific research in the area of object detection on edges devices	Scientific Impact Real-World Deployment Success

Acknowledgements

In this chapter, I would like to express my gratitude to everyone who contributed directly and indirectly to the completion of this project.

I am deeply grateful to Professor João Manuel R. S. Tavares for his unwavering attention, support, and guidance throughout the entire supervision period. His insights and expertise were invaluable to the successful development of this work.

I would also like to extend my heartfelt thanks to my family for their constant encouragement, understanding, and support during this journey. Their belief in me has been a source of strength and motivation throughout this challenging yet rewarding process.

To everyone who has contributed in any way to the realization of this project, my sincere thanks.

Luís Miguel da Costa Ferreira

"What we anticipate seldom occurs: but what we least expect generally happens."

Benjamin Disraeli

Contents

List of Figures

List of Tables

List of Acronyms

mAP	mean Average Precision
SDG	Sustainable Development Goal
CNN	Convolutional Neural Network
AI	Artificial Intelligence
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
IoU	Intersection over Union
YOLO	You Only Look Once
R-YOLO	Robust You Only Look Once
FCNet	Feature Calibration Network
QTNet	Quasi-Translation Network
ROI	Region of Interest
COCO	Common Objects in Context
DETR	Detection Transformer
ViT	Vision Transformer
MDFA	Multi-Dimension Feature Attention
RFCA	Receptive-Field Coordinated Attention
SDK	Software Development Kit
MS-PAFPN	Multi-Scale Path Aggregation Feature Pyramid Network
WS-Fusion	Weighted Shuffling Fusion
SPPFormer	Spatial Pyramid Pooling Former
TFLite	TensorFlow Lite
UI	User Interface
FLOPs	Floating-Point Operations per Second
MB	Megabytes
DFL	Distributed Focal Loss
NMS	Non-Maximum Suppression

Chapter 1

Introduction

This chapter begins by presenting the context of the work, the motivation behind the Dissertation, and the problem along with its significance. It then outlines the main contributions of the research, both direct and indirect, and concludes with an overview of the Dissertation's structure.

1.1 Context

This work focuses on image-based analysis for object identification in industrial environments. This project is set within the field of computer vision and artificial intelligence, especially deep learning methods. These technologies allow for automated analysis of visual data, making it possible to recognize and classify various objects in complex and dynamic industrial settings, such as parts in production, tools, machinery, and raw materials. This capability is increasingly essential as industries adopt digital twin systems, which demand rapid and reliable data exchange.

1.2 Motivation

The motivation behind this project lies in the growing demand for accurate information regarding the operational status of industrial facilities. As industries advance towards automation and data-driven decision-making, the ability to track and manage resources efficiently has become crucial. By integrating advanced object recognition within industrial environments, industries can enhance productivity, minimize downtime, and ensure better safety standards. This work not only supports operational improvements but also aligns with the trends of Industry 4.0, where interconnected systems and smart technologies are key.

1.3 Problem and its Importance

The core problem faced in this dissertation is the need for a robust and efficient system to identify various objects within industrial settings through image analysis. The dynamic and complex nature of industrial environments, where multiple objects overlap or frequently change positions, poses

significant challenges. Addressing this problem is crucial as it has direct implications for the efficiency and safety of industrial processes. Enhanced object recognition contributes to more reliable operations, optimized resource use, and reduced operational risks, ultimately supporting a safer and more efficient industrial landscape.

1.4 Main Contributions

In the scope of this Dissertation, the main achieved contributions are as follows:

- Development of Efficient Object Detection Models: The creation and optimization of deep learning-based models designed to accurately identify and classify objects in dynamic industrial environments, with an emphasis on edge-based processing for real-time performance.
- Foundation for Future Research in Industrial Tool Detection: Serving as a basis for future studies, this work provides valuable insights into industrial tool detection using imaging data, guiding researchers in selecting the most appropriate models for specific operational environments.

1.5 Dissertation Structure

In addition to the introduction, this dissertation contains 5 more chapters.

- Chapter 2 presents and explains the concepts required to understand this work;
- Chapter 3 the state of the art is described and related work is presented;
- Chapter 4 details the process carried out throughout this Dissertation to accomplish the intended objective;
- Chapter 5 displays the obtained results along with a critical analysis of their significance;
- Chapter 6 conclude and recommend possible directions for future research.

Chapter 2

Theoretical Fundaments

This chapter aims to present some theoretical aspects and fundamental knowledge that are relevant for understanding the scientific or engineering domain in which this work is contextualized.It begins by explaining what Artificial Intelligence is, along with its subfields: Machine Learning, Deep Learning, and Convolutional Neural Networks (CNN). Next, the concept of Transfer Learning is introduced. Following this, the concept of Computer Vision is introduced, with a particular focus on one of its subfields, object detection. Lastly, the chapter addresses evaluation metrics, providing an overview of several commonly used metrics for assessing model performance. The chapter concludes with a summary of the main concepts presented.

2.1 Artificial intelligence

Artificial Intelligence (AI) refers to the simulation of human cognitive processes by machines, particularly computer systems. Among the prominent applications of AI are expert systems, natural language processing, speech recognition, and machine vision. The development and training of AI systems require specialized hardware and software infrastructure. Although no single programming language is synonymous with AI, languages like Python, R, and Java are commonly used due to their versatility and the wide range of libraries they provide. AI systems typically function by processing substantial volumes of labeled training data, identifying correlations and patterns within the data, and using these patterns to generate predictive insights. For instance, a chatbot trained on textual dialogues can learn to engage in realistic exchanges with users, and an image recognition system can learn to identify and classify objects by analyzing extensive image datasets. In general, AI programming is structured around three primary cognitive functions: learning, reasoning and adaptation [3]. AI is incorporated into a variety of different types of technology, as depicted in Figure 2.1.

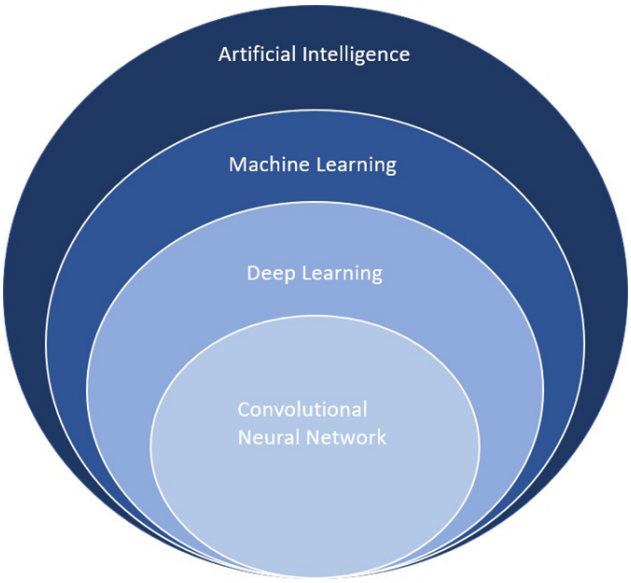

Figure 2.1: Representation of an AI (adapted [50]).

2.1.1 Machine learning

Machine learning is a subfield of artificial intelligence, which is the science of enabling computers to act without explicit programming. There are three main classes of machine learning techniques:

- Supervised Learning: In supervised learning, the training set consists of pairs of input and desired output, and the goal is that of learning a mapping between input and output spaces [47]. This approach is commonly used for tasks where the expected output is well-defined, such as classification and regression.
- Unsupervised Learning: In unsupervised learning, the training set consists of unlabeled inputs, that is, of inputs without any assigned desired output [47]. This type of learning is typically used for clustering, association, and anomaly detection, where the goal is to uncover hidden structures within the data.
- Reinforcement Learning: Reinforcement learning lies, in a sense, between supervised and unsupervised learning [47]. Datasets are also unlabeled, but after performing one or more actions, the AI system receives feedback in the form of rewards or penalties. This feedback loop allows the system to learn optimal actions over time, making reinforcement learning well-suited for dynamic environments and sequential decision-making tasks.

Although each algorithm has advantages and limitations, no single algorithm works for all problems.

2.1.2 Deep Learning

Deep learning is a subset of machine learning that enables computers to perform tasks typically associated with human cognition, such as speech recognition, image identification, and predictive

analysis.Deep learning is characterized by learning large neural network-style models with multiple layers of representation [26]. This layered approach allows for progressively complex feature extraction and abstraction, enabling deep learning models to achieve high levels of accuracy in a wide range of applications.

2.1.3 Convolutional neural network

Convolutional neural network (CNN), often called ConvNet, has deep feed-forward architecture and has astonishing ability to generalize in a better way as compared to networks with fully connected layers. A CNN is inspired by the human brain and mimics the way biological neurons communicate with one another. In a CNN, each node (or perceptron) functions as a simplified model of a neuron. This node encodes a linear regression model composed of input data, weights, a bias (or threshold), and an output. The weights and biases adjust through training, enabling the network to learn and recognize patterns in complex data [54]. One of the defining characteristics of CNNs is their variable number of layers, which can be adjusted based on the complexity of the task at hand. Figure 2.2 shows the representation of a CNN.

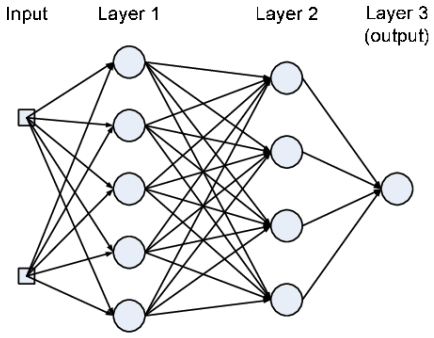

Figure 2.2: Representation of a common CNN.

2.2 Transfer Learning

Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second related task. This approach is particularly beneficial when the target task has limited data, allowing the model to leverage knowledge acquired from solving a different problem with a richer dataset [36]. Figure 2.3 shows a simplified diagram of this process, illustrating how a pretrained model on a source task can be fine-tuned and adapted to improve performance on a target task with less available data.

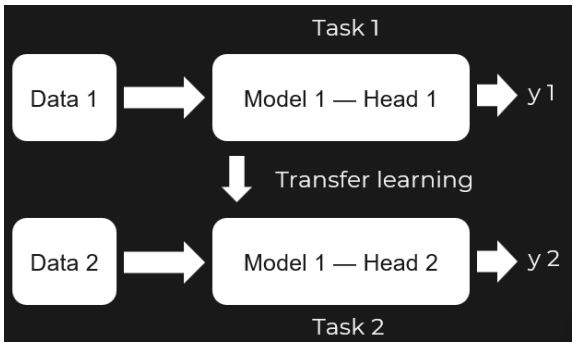

Figure 2.3: Diagram illustrating Transfer Learning (adapted [60]).

The core idea of transfer learning lies in the ability of deep neural networks to learn and encode general features in their early layers. These features, such as edges, shapes, and textures, are often transferable across different vision tasks. By retaining these learned features and adapting only the later, task-specific layers, the model can generalize to a new domain with fewer data and less training time.

Transfer learning offers several practical advantages:

- **Reduced Training Time:** Models converge faster by starting with pre-trained weights instead of initializing from scratch;
- **Improved Performance:** Leveraging previously learned features often leads to higher accuracy, especially with small datasets;
- **Lower Data Requirements:** It allows the training of robust models even when the target dataset is limited.

Transfer learning is particularly effective when:

- The new task is related to the original task used for pre-training;
- There is a shortage of labeled data for the target task;
- Computational resources are limited, and training from scratch would be infeasible.

There are several practical approaches to implementing transfer learning in deep learning:

1. **Training a Model to Reuse It:** When data for task 1 is scarce, a model can be trained on task 2, which has an abundance of data. The trained model is then reused for task 1 by either using it directly or by modifying and retraining some of its layers. This approach is common when both tasks share the same input format [17].
2. **Using a Pre-Trained Model:** A widely used method is to adopt a model pre-trained on a large dataset, such as ImageNet or COCO. Frameworks like Keras and PyTorch provide many pre-trained models ready for use in feature extraction or fine-tuning. Depending on the application, some layers can be frozen while others are retrained to adapt to the new task. This method is frequently applied in real-world deep learning projects [17].

3. **Feature Extraction:** In this method, a pre-trained model is used to extract useful features from the input data, which are then passed to a new classifier. Also known as representation learning, this technique often yields superior results compared to using hand-crafted features, as the model learns the most relevant patterns directly from data [17].

In conclusion, transfer learning not only improves performance and efficiency but also plays a key role in enabling object detection systems to be deployed in real-world industrial settings with limited data and constrained hardware environments.

2.3 Computer Vision

Computer vision seeks to develop software that enables computers to interpret and extract specific information from images, videos or other types of media [48]. By using AI techniques, computer vision systems can significantly enhance their accuracy and efficiency, enabling more complex analyses and robust performance. Figure 2.4 demonstrates the relationship between computer vision and artificial intelligence. Techniques such as object detection, object identification, object tracking, event detection, image classification, and object classification, among others, make computer vision applicable to a wide range of fields, from industrial automation and medical imaging to autonomous vehicles and security systems, where accurate visual information processing is essential.

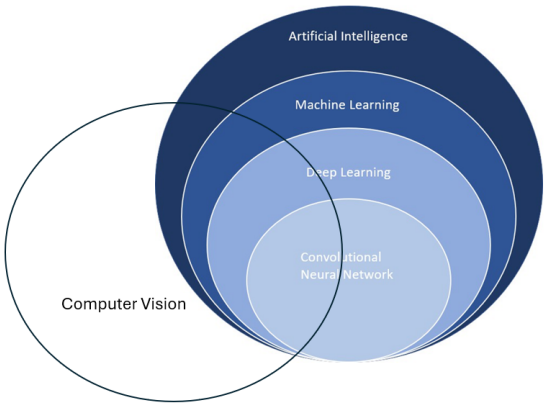

Figure 2.4: Diagram illustrating the relationship between computer vision and artificial intelligence.

2.3.1 Object Detection

Unlike classification, which simply predicts the class present in an image, object detection not only determines the class but also identifies the object's precise location within the image. For instance, when building a classifier for objects A and B, the classifier will return both the predicted class (A or B) and a confidence score (indicating the certainty of the prediction).

However, if the image contains both A and B, a potential solution is to train multi-class classifiers that can identify multiple classes simultaneously. To mark the object's position, a rectangular

bounding box is created, encapsulating the object within the image. In this scenario, however, the classifier alone does not provide the exact location of objects A or B in the image. The task of determining an object's location within an image is known as localization.

Therefore, object detection involves both identifying the class of an object and determining its location within the image, often across multiple classes. Figure 2.5 shows an example of object detection with two objects.

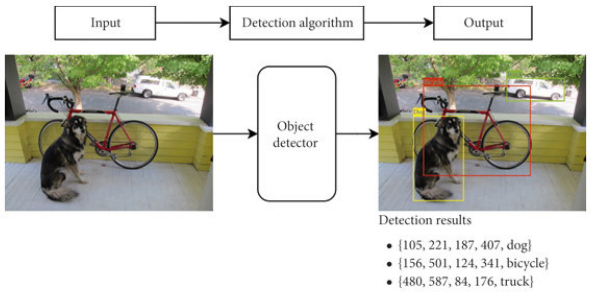

Figure 2.5: Object detection in an example image (from [21]).

2.4 Evaluation Metrics

To thoroughly assess the performance of the trained models, a comprehensive set of evaluation metrics will be employed. These metrics are crucial for measuring the models accuracy, efficiency, and overall effectiveness in addressing the problem at hand. By analyzing these metrics, we can gain valuable insights into the strengths and limitations of the models, ensuring their suitability for real-world applications.

A classification model aims to make predictions based on past occurrences. To evaluate such a model, a separate dataset (distinct from the training set) must be used to test its accuracy. However, it is insufficient to simply count the number of correct predictions to determine whether the model is effective. Depending on the problem being studied, different metrics must be applied to assess its performance. The specific metrics to be used are defined as follows:

2.4.1 Basic Metrics

- **True Positive (TP):** When the method predicts that the class is positive, and upon verification, the actual class is indeed positive;
- **True Negative (TN):** When the method predicts that the class is negative, and upon verification, the actual class is indeed negative;
- **False Positive (FP):** When the method predicts that the class is positive, but upon verification, the actual class is found to be negative;
- **False Negative (FN):** When the method predicts that the class is negative, but upon verification, the actual class is found to be positive.

The confusion matrix, illustrated in Figure 2.6, complements the definitions provided above by visually organizing the relationship between the predicted and actual values of a classification model. It highlights the four key outcomes (TP, TN, F, and FN) which were previously described. This matrix provides an intuitive and structured way to evaluate the performance of the model, showcasing how well the predictions align with the actual classifications.

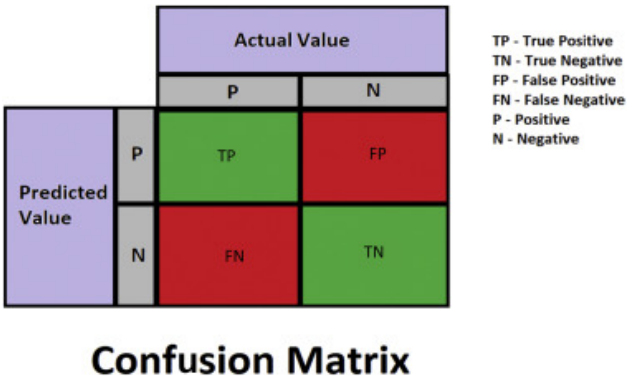

Figure 2.6: Example of a confusion matrix (from [32]).

2.4.2 Accuracy

Accuracy is a metric that evaluates how frequently a machine learning model makes correct predictions. It is computed by dividing the number of correct predictions by the total number of predictions made:

$$Accuracy: \quad Acc = \frac{TP+TN}{TP+TN+FP+FN}. \tag{2.1}$$

2.4.3 Recall

Recall [38], also referred to as sensitivity, is a metric used to evaluate a model's effectiveness in correctly identifying positive instances. It is computed as the ratio of **TP** to the total actual positive instances, which comprises the sum of **TP** and **FN**. This metric underscores the proportion of correctly identified positives relative to all actual positives, highlighting the model's ability to minimize missed cases:

$$Recall: \quad R = \frac{TP}{TP+FN}. \tag{2.2}$$

2.4.4 Precision

Precision [38] is a metric that evaluates how accurately a machine learning model predicts the positive class. It is calculated by dividing the number of **TP** by the total number of instances the model predicted as positive, which includes both **TP** and **FP**:

$$Precision: \quad P = \frac{TP}{TP+FP}. \tag{2.3}$$

2.4.5 F1-score

Precision and recall represent a trade-off: improving one metric often results in a decrease in the other. Increasing precision requires a more stringent classifier, which may classify even actual positive samples as negative, thereby reducing recall. On the other hand, increasing recall involves a more lenient classifier, allowing more samples to be classified as positive, including some borderline negative cases, which lowers precision. Ideally, the objective is to maximize both precision and recall to achieve the best classifier.

The F1-score [45] addresses this balance by combining precision and recall through their harmonic mean. Maximizing the F1-score inherently maximizes both metrics. As a result, the F1-score has become a preferred evaluation metric for researchers, often used alongside accuracy to provide a comprehensive assessment of model performance.The F1-score is calculated:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{2.4}$$

2.4.6 Intersection over Union

The Intersection over Union (IoU), also referred to as the Jaccard Index, is a widely used performance metric for evaluating the accuracy of annotation, segmentation, and object detection algorithms. It quantifies the overlap between the predicted bounding box or segmented region and the ground truth bounding box or annotated region in a dataset. By focusing on the geometric properties of objects (such as widths, heights, and positions) IoU represents them as regions and calculates a normalized measure based solely on their areas (or volumes, in the case of 3D shapes) [43]. This normalization ensures IoU is invariant to scale, making it a robust and versatile metric for various applications.

IoU is calculated as the ratio of the area of intersection between the predicted and ground truth regions to the area of their union. A higher IoU value indicates a better alignment between the predicted and actual regions, reflecting a more accurate model. This process is illustrated in Figure 2.7.

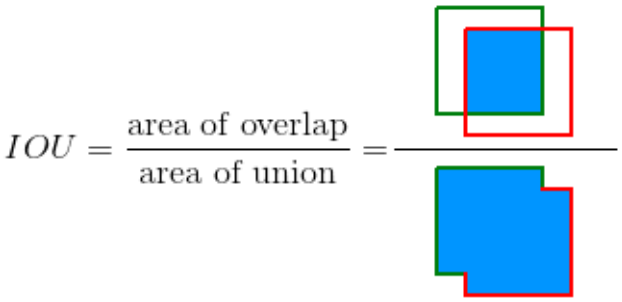

Figure 2.7: IoU (from [35]).

2.5 Summary

This Chapter explored Artificial Intelligence (AI), its foundational concepts, and key areas such as machine learning, deep learning, and computer vision. Machine learning techniques are categorized into supervised, unsupervised, and reinforcement learning, each addressing distinct challenges. Deep learning, particularly through Convolutional Neural Networks (CNNs), excels in processing complex visual data using layered architectures inspired by biological neurons.

Transfer Learning was also addressed as a technique that allows models pre-trained on large datasets to be fine-tuned for specific tasks with relatively less data and computational resources. This approach significantly accelerates training time and often improves performance, especially in domains where annotated data is limited.

Computer vision integrates these AI techniques for tasks like object detection and tracking, essential for applications in industrial and real-world scenarios. Object detection uniquely combines classification and localization, enabling precise identification of objects within images.

To evaluate the performance of AI models, metrics like accuracy, precision, recall, F1-score, and Intersection over Union (IoU) are applied, ensuring reliability and adaptability in practical deployments.

Chapter 3

Literature Review

Technology has been evolving over the years, leading to the creation of new innovations or the improvement of pre-existing ones.

This chapter presents a review of the state of the art in solutions and projects that explore object detection technologies and their integration into edge computing devices. The analysis focuses on identifying the main advances in the field of computer vision, with an emphasis on deep learning models applied to object detection, as well as the challenges and solutions associated with implementation on resource-constrained devices, such as edge devices.

In addition, this chapter introduce the searching method used to conduct the literature review and conclude with a summary of the key findings.

3.1 Searching Method

The research was conducted entirely within the Scopus database using the following keywords: object detection, deep learning, computer vision and machine learning. This initial query yielded 454 results. A noticeable trend indicates that publications related to artificial intelligence and object detection have grown significantly over the years, as shown in Figure 3.1.

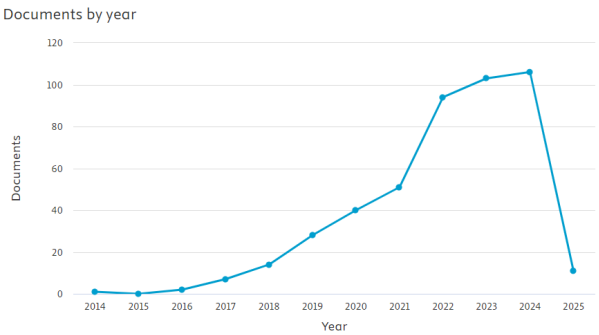

Figure 3.1: Documents retrieved using the specified keywords.

To focus on the most recent advancements in the field, a filtering process was applied. First, all documents published prior to 2024 were excluded, removing 301 results. Subsequently, a manual screening was conducted based on the titles and abstracts of the remaining documents, eliminating an additional 73 irrelevant papers. Moreover, review and survey papers were excluded, resulting in the removal of 17 more documents.

After completing this comprehensive filtering process, 24 relevant papers were identified and selected as the basis for the state-of-the-art analysis in this study. The methodology employed is summarized in Figure 3.2.

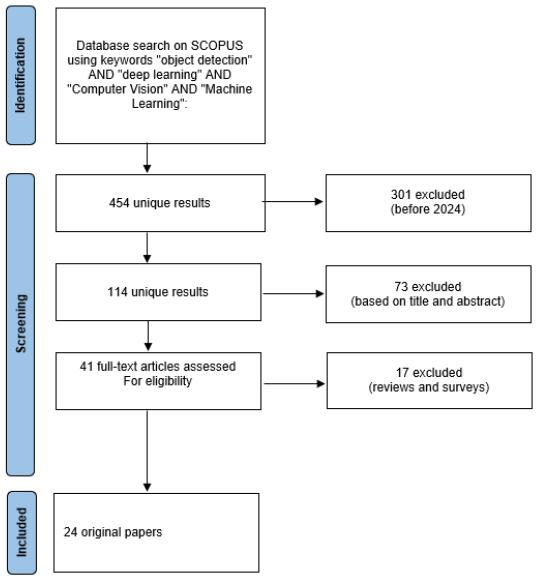

Figure 3.2: PRISMA diagram illustrating the outcomes of the conducted literature review.

3.2 Related work

Among the 24 documents reviewed, the models utilized in the studies can be categorized into the following types: You Only Look Once (YOLO), Detectron2, Vision-DETR and Swin Transformer.

3.2.1 Models

In this section, a brief explanation is provided about the models found in the literature review.

3.2.1.1 You Only Look Once

You Only Look Once (YOLO) is one of the most widely used object detection models, recognized for its remarkable speed and accuracy. Unlike traditional detection methods that apply sliding windows or region proposals, YOLO frames the detection task as a single regression problem [42].

3.2.1.2 Detectron2

Developed by Facebook AI, Detectron2 is a state-of-the-art library for object detection and instance segmentation tasks. It supports a variety of architectures, such as Faster R-CNN, Mask R-CNN, RetinaNet, and Cascade R-CNN, allowing users to train and deploy custom models efficiently. Detectron2 incorporates advanced features, including multi-scale training, dense region proposals, and anchor-based detection, which contribute to its robustness and adaptability across diverse datasets [61].

3.2.1.3 Vision-DETR

Vision-DETR is a transformer-based architecture designed for object detection. Unlike traditional methods, Vision-DETR eliminates the need for anchor boxes by leveraging self-attention mechanisms to directly predict object locations and class labels [7]. Its design simplifies the detection pipeline by encoding global spatial relationships and efficiently handling objects at different scales. The attention-based approach enhances its adaptability to complex object arrangements and challenging scenes [64].

3.2.1.4 Swin Transformer

Swin Transformer is a hierarchical vision transformer architecture designed for image classification, object detection, and semantic segmentation. Unlike traditional vision transformers, Swin Transformer introduces a shifted window mechanism to partition images into non-overlapping local windows and compute self-attention within these regions. This approach reduces computational complexity by focusing attention locally, while the shifted windows ensure cross-window connections and global context. Its hierarchical structure enables multi-scale feature representation, making it highly effective for handling objects of varying sizes. The Swin Transformer is particularly well-suited for dense prediction tasks due to its ability to capture both local and global dependencies efficiently [29].

3.2.2 Selected works

[2] selected YOLOv8 due to its speed and capability for multi-scale object detection. The architecture of YOLOv8 builds upon its predecessors in the YOLO family, utilizing a convolutional neural network divided into two primary components: the backbone and the head. For their study, the ExDARK dataset was employed, with data pre-processing techniques applied to enhance image quality and improve detection performance.

The author [4] used R-YOLO (Robust You Only Look Once), designed to enhance the resilience of the YOLO object detector. This system integrates two key components: a feature calibration network (FCNet) and an image quasi-translation network (QTNet). Together, these networks facilitate a gradual adaptation from normal weather conditions to adverse weather environments, improving detection performance in challenging scenarios.

In [31], two models were employed: YOLO v5 for object detection and DeepSORT for object tracking. YOLO v5 builds upon its predecessors by introducing advanced features such as mosaic data augmentation, adaptive anchor box computation, and enhanced feature pyramids, which significantly enhance its detection accuracy and robustness. Renowned for its balance between speed and precision, YOLO v5 is highly effective in applications that demand fast and reliable object detection, including video analysis and autonomous systems. The model processes input in the form of video frames, where each frame is treated as an image to detect objects. Additionally, they affirm that setting up parameters is essential and should be based on the distance between the camera and the Region of Interest (ROI). This ensures optimal performance and accuracy in detecting objects at varying distances.

The study [28] employed YOLOv5 on the Common Objects in Context (COCO) dataset to identify objects within images. Additionally, the system integrates Google Text-to-Speech to generate spoken descriptions of detected objects in Konkani, enhancing usability for the target user group. To improve the model's performance, the dataset was augmented using transformations such as rotation, scaling, resizing, and flipping, which are crucial for YOLOv5. These transformations helped increase the dataset's variability, ensuring the model could effectively handle various orientations and perspectives of objects.

Building upon this, another study by [34] proposed a novel model that combines DETR (Detection TRansformer) and ViT (Vision Transformer), replacing the traditional ResNet backbone with a fully Transformer-based architecture. This approach was designed to recognize objects in images and videos using only Transformer components. DETR demonstrates strong performance in object detection by leveraging the Transformer architecture and eliminating the need for complex intermediate steps. Similarly, ViT, as a Transformer-based architecture, has achieved significant breakthroughs in image classification. The model was trained and evaluated on the PASCAL VOC 2012 dataset, which includes 20 object classes.

In this study [8], a YOLOv8 model was utilized to detect and classify powerline faults. The preprocessing phase involved extracting images from data collected via UAVs and converting them into high-quality, resized images. A pretrained YOLOv8 model, trained on the COCO dataset, was employed for object detection, ensuring robust and accurate identification of powerline faults.

In this work [9], different variations of YOLOv5 (YOLOv5s, YOLOv5m, and YOLOv5m6) were employed to address error recognition in manual cable assembly. All models were pretrained on the COCO dataset and subsequently fine-tuned for the specific task. The training process utilized synthetic data, which was designed to enhance the model's ability to identify errors effectively in cable assembly scenarios.

Shifting focus to an earlier version of the YOLO framework, [51] explored the use of YOLOv3 integrated with the DarkNet framework to detect and classify objects. The system then converts the detected objects into text to assist blind and visually impaired individuals. The model was applied to images extracted from video captured by a camera operating at 30 frames per second, leveraging OpenCV for video processing and frame extraction.

In study [5] introduced a TinyML model designed for sidewalk obstacle detection to assist

blind and visually impaired individuals. The model was optimized for deployment on resource-constrained IoT devices, achieving a compact size of 1.93 MB. To develop the TinyML model, the study utilized and tested YOLOv5 and YOLOv8 architectures, tailoring them for use in low-resource environments. Preprocessing steps included resizing images to match the model's input dimensions, normalizing pixel values for consistency, and annotating data to classify obstacles such as potholes and cones. With an inference speed of 96.2 milliseconds on a standard CPU, the system facilitates real-time processing, enhancing urban navigation for the visually impaired.

In their work [30], the authors proposed an intelligent system for tank detection using the YOLOv5 model. The dataset consisted of images containing tanks, sourced from the Roboflow website. To enhance the diversity and robustness of the model, data augmentation techniques such as rotation, flipping, and scaling were applied. Additionally, the labeling of the dataset was performed manually to ensure accuracy and consistency in the training process.

In contrast to YOLOv5, a study by [25] applied YOLOv8 for identifying look-alike drugs. The dataset consisted of three different generic drugs, and various versions of YOLOv8 and YOLOv9 were trained and compared. Among these, YOLOv8-nano emerged as the most effective model for drug identification.

Recent work by [39] explores the use of YOLOv7 to detect and recognize number plates in images of moving or stationary vehicles. The model was fine-tuned with a large dataset of images, and performance was compared with other versions of YOLO (v5 and v6). Image preprocessing, including resizing and noise removal, was applied to enhance image quality. For data augmentation, transformations such as rotation, brightness and contrast adjustment, and noise addition were used. The best results were achieved with the pretrained YOLOv7-x weights.

In the study by [58], YOLOv4 was utilized for the automated detection of bone fractures. The researchers employed the MURA dataset, which is specifically designed for medical imaging tasks. This approach leveraged YOLOv4's capabilities to detect fractures in radiographic images, showcasing the effectiveness of deep learning models in the medical field for automating the diagnosis process. The study highlights the potential of computer vision to assist in healthcare applications.

In the study by [49], YOLOv8, specifically the YOLOv8m.pt variant, was used for object detection due to its balance between speed and accuracy. Initially, the researchers considered using YOLOv5 but chose YOLOv8 for its improved performance. The study highlighted that lighting conditions and the orientation of detected objects both significantly influenced the model's accuracy, demonstrating the importance of environmental factors in object detection tasks.

In a similar study, [44] employed YOLOv8 for immature green apple detection. They utilized a pretrained YOLOv8m-seg model, which was fine-tuned to meet the specific needs of their study. While the model demonstrated strong performance in detecting and segmenting immature green apples in orchard environments, challenges arose in situations where the apples' green color blended with background objects, such as leaves, leading to occasional detection failures. This highlights the model's limitations in handling complex backgrounds.

In the study by [62], YOLOv8 was used to accurately identify gestures made by football referees. They introduced the FRGR-YOLOv8s model, specifically designed for gesture recognition in football referees. The dataset included both referees and various background elements. To address challenges with long-distance image capture, downsampling through convolution was applied during feature extraction. Additionally, to manage complex backgrounds, they integrated the Global Attention Mechanism on top of YOLOv8s, improving the model's performance.

In the work by [46], YOLOv7 was employed for monitoring and identifying employees who disregard safety procedures. The dataset used was Pictor-v3, with four classes selected to reduce annotation time. After YOLOv7 detects the employees, the image is cropped, and a CNN-based classifier, specifically the VGG-16 model, is used to classify the worker based on the cropped image. This two-step process enhances the accuracy of monitoring safety compliance.

The study in [41] focuses on using YOLOv4 for real-time quality monitoring in welding processes on moving steel joists. The system processes video feeds with frames in grayscale, smoothed using Gaussian blur. The dataset includes six labeled classes, with part of it manually annotated. The model achieves an inference time of 0.0083 seconds per frame, with the overall framework processing time of 0.0167 seconds, ensuring fast detection of missing welds.

The article [40] presents a low-cost system using the YOLOv3 model to monitor water consumption and identify ear tags in calves. The system processes video frames at 15 frames per second, recognizing animals near drinkers. YOLOv3 detects regions of interest containing drinking actions and identifies ear tags based on the calf's visual appearance when captured by the camera. This approach ensures efficient monitoring with high performance in livestock management.

The study [27] employs the CSYOLOv3 model for real-time detection of whether individuals are wearing safety helmets. A pretrained model was fine-tuned on helmet datasets sourced from Roboflow and Kaggle, covering diverse conditions such as varied lighting, angles, and perspectives. The dataset includes images without helmets to minimize false positives. While the model performed well, challenges arose in accurately localizing and distinguishing overlapping objects, particularly when multiple objects shared the same grid cell.

In their work [6], the authors present an automated system for detecting marine oil spills using the YOLOv8 model. Pretrained on the COCO dataset, the model was fine-tuned for the specific task. Due to the limited size of the dataset, data augmentation techniques were applied, including rescaling, to generate additional images. To ensure data quality, filtering was performed using the SEID metric, and all samples were manually annotated. This approach enhances detection accuracy in challenging marine environments.

In their article [63], the authors developed YOLOv8-RFMD, an enhanced model for detecting mulberry leaf diseases in natural environments. The dataset, sourced from Kaggle, included diverse conditions and was augmented using techniques such as flipping, noise addition, and brightness adjustment. They introduced the Multi-Dimension Feature Attention (MDFA) and Receptive-Field Coordinated Attention (RFCA) modules, improving feature extraction and replacing YOLOv8's bottleneck. Additionally, they replaced the CIOU loss function with NWD

and demonstrated superior performance compared to YOLOv8n. The lightweight model is optimized for mobile and embedded devices.

In their study [37], the authors developed a system for automated car damage detection using YOLOv5-based models.The use of YOLOv5 was motivated by its efficient inference times on CPUs. They designed 10 pretrained models optimized through a process that includes selecting object detectors, adjusting confidence thresholds, and combining predictions. To maintain efficiency, they applied the PSO algorithm to limit the number of models while ensuring accuracy. A prediction box merging process was introduced, where intersecting predictions were unified based on a voting system.To optimize for real-world scenarios where high inference times could pose a challenge, the authors leveraged parallelization, allowing them to manage the maximum times effectively.

In this study [59], the authors developed a novel approach for multi-category sorting of plastic waste using a Swin Transformer model. This model classifies waste into five categories and demonstrates its effectiveness through detection result visualizations on municipal solid waste streams and principal component analysis of classification scores. Pre-trained on the COCO dataset, the Swin Transformer excels in feature extraction but faces challenges with small, rare, or overlapping objects. In the study conducted by [22], YOLOv5 was employed for detecting slugs on lettuce and performing 3D localization. The authors outlined three key stages in their methodology: (1) preparation of the SOL dataset, (2) training of the YOLOv5 framework, and (3) conducting real-time testing. The dataset consisted of images captured under varying lighting conditions, weather, and different lettuce varieties, all of which were manually annotated. The pixel coordinates of the center of the bounding box were calculated and subsequently converted into three-dimensional coordinates using the Software Development Kit (SDK) of the Intel RealSense™ D405 camera. The authors tested different versions of YOLOv5, namely YOLOv5n, YOLOv5s, YOLOv5m, and YOLOv5l, with the best-performing model being YOLOv5l.

In this work by [23] utilized the RS-YOLO model for object detection in road scenes. Their model, based on YOLOv8, introduced several enhancements to improve performance, including the MS-PAFPN (Multi-Scale Path Aggregation Feature Pyramid Network), WS-Fusion (Weighted Shuffling Fusion), and SPPFormer (Spatial Pyramid Pooling Former). The researchers utilized two datasets, SODA10M and KITTI, for training and evaluation. Comparative analyses with other models demonstrated that RS-YOLO exhibited exceptional generalization performance and robustness, particularly when applied to unknown and complex road scenes.

After reviewing all these related works, the results are summarized in Table 3.1, which presents an overview of each study, including the author, model used, accuracy, precision, mAP, and other relevant metrics. For studies involving multiple models, the table highlights the best-performing model identified in the respective work. This comprehensive summary aims to provide a clear comparison of the methodologies and outcomes achieved across the different studies.

Table 3.1: Summary of related works and their performance metrics.

Article	Author	Year	Model	Accuracy	Precision	mAP_50:95	Other Metrics
[2]	Aref Bahia Salah Zriguib Mounir Zriguic Henri Nicolas	2024	YOLOv8	–	77.90	71.20	–
[4]	Bharat Siva Varma P. Adimoolam, Prathap Marna, Yamini Lahari Vengala, Anantharamaiah Sundar, V. S. Divya Ram Pavan Kumar M.V.T.	2024	R-YOLO	98.00	94.00	89.60	f1_score=88.90
[31]	Mallikharjuna Rao, K Agrawal, Deepesh Reyya, Shikhar Varma, Priykrit	2024	YOLO v5	–	–	–	–
[28]	Likhitha P. Dessai, Sukanya Naik, Aftab Rauf Shetgaonkar, Pratiksha Chari, Kishan Nagesh	2024	YOLOv5s	81.00	100.00	56.60	f1_score=57.00
[34]	Le, Van-Tuong-Lan	2024	V-DETR	34.50	24.0	44.40	–
[8]	Chandaliya, Mahavir Goli, Teja Sree Kotha, Swapna Gao, Jerry	2024	YOLOv8	83.40	86.70	86.10	–
[9]	Conrad, Jonas Stauffer, Tobias Meng, Xuanting Ferchow, Julian Meboldt, Mirko	2024	YOLOv5m6	98.30	98	79.40	–
[5]	Boussihmed, Ahmed El Makkaoui, Khalid Ouahbi, Ibrahim Maleh, Yassine Chetouani, Abdelaziz	2024	YOLOv5n	50.20	58.50	31.50	Speed CPU = 96.20 ms
[25]	Kittisorayut, Sorayus Nonsiri, Sarayut Kamin, Pichitchai Makdee, Supawee	2024	YOLOv8-n	99.97	99.10	–	mAP_50=99.50 f1_score=99.40
[39]	Prakash, Akshay Babu, Ashin Hari Narayanan A.G.	2024	YOLOv7	92.00	–	–	–
[58]	Verma, Abhimanyu Kumar, Vansh Sharmila Yadav, R.K.	2024	YOLOv4	–	63.50	–	–
[49]	Tan, Jia He Goh, Ching Pang	2024	YOLOv8	82.00	–	–	–
[44]	Sapkota, Ranjan Ahmed, Dawood Churuvija, Martin Karkee, Manoj	2024	YOLOv8M	88.00	90.00	–	mAP_75=91.00 f1_score=89.00
[62]	Shen, Yanfei Shen, Yuanyuan Yang, Zhiyuan	2024	FRGR-YOLOv8s	88.90	89.30	77.30	–
[46]	Shahin, Mohammad Chen, F. Frank Hosseinzadeh, Ali Koodiani, Hamid Khodadadi Bouzary, Hamed Rashidifar, Rasoul	2024	YOLOv7	59.90	62.10	–	11.79 fps f1_score=80.00
[41]	Raoofi, Hamed Sabahnia, Asa Barbeau, Daniel Motamedi, Ali	2024	YOLOv4	99.00	92.00	76.70	121 fps f1_score=95.00

Article	Author	Year	Model	Accuracy	Precision	mAP_50:95	Other Metrics
[40]	Pretto, Andrea Savio, Gianpaolo Gottardo, Flaviana Uccheddu, Francesca Concheri, Gianmaria	2024	YOLOv3	87.00	90.00	–	mAP_50=89.00 f1_score=88.00 Average IoU=74
[27]	Li, Jun Li, Yuanyuan Villaverde, Jocelyn F. Chen, Xiaoji Zhang, Xiaozhi	2024	CSYOLOv3	90.00	93.00	60.00	–
[6]	Cai, Yuepeng Chen, Lusheng Zhuang, Xuebin Zhang, Bolin	2024	YOLO-v8	77.40	80.20	–	Test time per image=8.20ms
[63]	Zhang, Ming Yuan, Chang Liu, Qinghua Liu, Hongrui Qiu, Xiulin Zhao, Mengdi	2024	YOLOv8-RFMD	89.50	92.60	67.80	model size 5.45 MB
[37]	Pérez-Zarate, Sergio A. Corzo-García, Daniel Pro-Martín, Jose L. Álvarez-García, Juan A. Martínez-del-Amor, Miguel A. Fernández-Cabrera, David	2024	YOLOv8	–	–	–	Speed CPU = 1110.00 ms
[59]	Wang, Zhengyu Ye, Linhai Chen, Feng Zhou, Tao Zhao, Youcai	2024	Swin Transformer	86.80	98.60	–	mAP_50=81.00
[22]	Hassanzadehtalouki, Mohammadreza Nasirahmadi, Abozar Wilczek, Ulrike Jungwirth, Oliver Hensel, Oliver	2024	YOLOv5l	98.60	98.90	78.60	–
[23]	Jiao, Bowen Wang, Yulin Wang, Peng Wang, Hongchang Yue, Haiyang	2025	RS-YOLO	91.30	93.21	78.51	–

3.3 Discussion and Trends

The analysis of the 24 selected papers highlights several trends and insights regarding object detection on edge devices. These trends include the predominance of specific model families, the emergence of novel architectures, deployment challenges, and future research directions.

3.3.1 Prevalence of YOLO-Based Architectures

Among the 24 reviewed papers, 20 utilized models from the YOLO family, making it the most widely adopted architecture. Versions range from YOLOv3 [40, 51], YOLOv4 [41, 58], and YOLOv5 [9, 22, 28, 31], to YOLOv7 [39, 46], YOLOv8 [2, 6, 8, 25, 44, 49, 62, 63] and even a custom YOLO-based model like RS-YOLO [23].

This dominance is likely due to YOLO's real-time performance, simplicity of implementation, and adaptability. For instance, YOLOv8 achieved high mAP scores such as 86.1% [8] and F1-scores reaching 99.4% [25], while lightweight versions like YOLOv5n were effectively deployed on constrained devices, as seen in [5] with an inference time of 96.2 ms on CPU.

3.3.2 Transformer-Based Models Gaining Traction

Although YOLO variants dominate, some works have adopted Transformer-based approaches. Vision-DETR was employed in [34], achieving an mAP_50:95 of 44.4%, while Swin Transformer was used in [59], reporting an accuracy of 86.8% and mAP_50 of 81.0%. These models benefit from global attention mechanisms, allowing superior performance in complex scenes, but often demand higher computational resources, which limits their current usage on edge platforms.

3.3.3 Performance Metrics and Deployment Readiness

Performance evaluation across the studies varied, but the most common metrics included accuracy, precision, and mAP. The highest reported mAP_50:95 was 89.6% by R-YOLO in [4], while accuracy values ranged from 50.2% [5] to 99.97% [25].

Real-time capability was emphasized in works such as [41], achieving an inference time of 0.0083 s/frame with YOLOv4 for welding quality inspection. However, high inference times like 1110 ms in [37] show that not all approaches are yet suitable for real-time applications on edge devices.

3.3.4 Emerging Trends and Future Directions

From the reviewed works, several forward-looking trends emerge:

- **Model compression and optimization** are becoming essential. Works like [63] and [5] show success with models under 6 MB, indicating growing interest in lightweight deployments;
- **Hybrid architectures**, such as combining CNNs and Transformers [34], aim to balance performance and efficiency;
- **Multi-model fusion and ensemble methods**, as seen in [37], improve robustness but may increase inference time;
- **Increased application diversity**, from agriculture [44] to industrial safety [46], demonstrates the adaptability of detection pipelines with proper fine-tuning;
- **Edge-focused evaluation**, including inference time and model size, is gaining relevance, beyond traditional accuracy-based metrics.

In summary, object detection for edge computing is a rapidly maturing field, with YOLO models at the forefront due to their speed and flexibility. However, the emergence of attention-based architectures, lightweight deployments, and tailored applications suggests an ongoing evolution towards more intelligent, context-aware, and efficient systems.

3.4 Summary

Through this study, three essential components for addressing the identified problem were identified.

The first component is the selection of a dataset containing a sufficient number of high-quality images. When the dataset is insufficient in size or quality, data augmentation becomes necessary. This technique involves artificially expanding the training set by generating modified versions of the existing data through methods such as scaling, rotation, and flipping. Additionally, low-quality datasets require preprocessing to improve their suitability for model training.The selection of a high-quality dataset is crucial because it directly impacts the accuracy and reliability of the model's performance. A robust dataset ensures the model can learn meaningful patterns, making it capable of generalizing well to unseen data. Datasets with insufficient size or poor quality can lead to overfitting, where the model memorizes specific examples rather than learning general features. Moreover, a diverse dataset enhances the model's ability to handle variations such as lighting, angles, or object appearances, which are critical for real-world applications.

The second component is the selection of the model and its associated training parameters. This step is crucial as the chosen model and configuration directly impact key performance metrics, such as recall, precision, and mAP, as well as the model's size and processing speed. Pre-trained models, combined with fine-tuning and transfer learning, provide an efficient approach, saving time while optimizing performance. An intriguing method observed in one of the studies proposed combining predictions from ten models, where the ensemble approach yielded improved accuracy and robustness by leveraging the strengths of multiple architectures. Models like YOLO remain popular due to their speed, accuracy, and adaptability, particularly in real-time or high-precision tasks. A well-chosen and optimized model, or ensemble, ensures both efficiency and quality in diverse applications.

Finally, the third component focuses on the evaluation metrics employed to measure the model's performance. Using well-defined and comprehensive metrics is essential for reliably assessing how effectively the model addresses the task at hand. Metrics such as mAP (mean average precision) are particularly valuable, as they provide a detailed understanding of the trade-off between precision and recall. By leveraging such metrics, researchers can gain actionable insights, identify areas for improvement, and fine-tune the model to achieve optimal performance tailored to the problem's requirements.

Chapter 4

Methodology

This chapter outlines the methodology employed in this Dissertation to develop an efficient and fast object detection system for industrial applications, for edges devices. The approach involves training and evaluating two distinct models, YOLOv11 (yolov11n and yolov11s) and YOLOv12 (yolov12n and yolov12s), using a custom dataset. A rigorous and systematic comparison will be conducted to assess their performance in terms of accuracy, efficiency, and suitability for real-world deployment. Based on these evaluations, a single model will be selected as the final solution.

4.1 Overview

The Figure 4.1 illustrates the methodological workflow adopted in this study, encompassing 3 key phases: (1) dataset creation/selection, (2) model training (yolov11n, yolov11s, yolov12n and yolov12s), and (3) model evaluation.

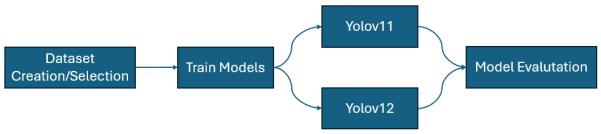

Figure 4.1: Methodology Workflow.

4.2 Dataset

Before selecting the dataset, the objects to be detected were first defined as follows: Forklift, Pallet, Hammer, Pliers, Screwdriver and Wrench. Once these objects were identified, the Roboflow platform was used to obtain pre-annotated datasets. A total of 8 distinct datasets [10–16, 18] were combined, resulting in a final dataset comprising 4,151 images, which was split as follows: 70% for training, 15% for validation and 15% for testing. For each part, there were two corresponding files,one containing the images and another with the labels. The label files contained annotations

specifying the class and location of each object. The first value represents the class ID, ranging from 0 to 5 in this project, followed by four normalized values that correspond to the bounding box coordinates. Figure 4.2 illustrates the annotation format for an image containing a single object, in this case, one object of class 'Forklift'. In contrast, Figure 4.3 demonstrates the annotation for an image with multiple objects present, specifically, seven objects of class 'Pallet'.

```
0 0.49140625 0.53671875 0.459375 0.421875
```

Figure 4.2: Annotation example for a single object of class 'Forklift' present in the image.

```
1 0.3859375 0.12265625 0.1859375 0.0515625
1 0.7609375 0.125 0.184375 0.0515625
1 0.928125 0.12734375 0.1421875 0.0484375
1 0.05078125 0.1515625 0.1015625 0.0484375
1 0.196875 0.1609375 0.190625 0.0515625
1 0.57421875 0.1671875 0.1859375 0.0484375
1 0.38671875 0.16953125 0.1875 0.0515625
```

Figure 4.3: Annotation example for multiple objects of class 'Pallet' present in the image (7 objects).

Roboflow offers an option to perform this split, however, in this dataset, it was not possible to use it as the split did not account for the images of each class. It tried to split the dataset into 70% training, 15% validation, and 15% testing in total, without ensuring that each class was present in each of these divisions. As a result, there were cases where a class was not included in one of the splits. To address this issue, a custom script was written to properly separate the dataset into the desired training, validation, and testing subsets, ensuring that all classes were represented in each subset.Table 4.1 outlines the distribution of images across the training, validation, and test sets for each class. Additionally, Table 4.2 presents the total number of objects per class in the dataset, further breaking down the instances observed in each split. Figure 4.4 provides a visual representation of some images included in the dataset, where each image has a resolution of 640×640 pixels.

Table 4.1: Number of images per class across training, validation, and testing sets.

Classe	**Total**	**Train**	**Validation**	**Test**
Forklift	861	602	135	124
Pallet	639	447	95	97
Hammer	725	502	111	112
Plier	611	420	88	103
Screwdriver	743	508	119	116
Wrench	649	439	103	107
Total	**4151**	**2891**	**625**	**635**

Table 4.2: Number of instances per class across training, validation, and testing sets.

Classe	**Total**	**Train**	**Validation**	**Test**
Forklift	866	603	137	126
Pallet	35433	25070	4943	5420
Hammer	861	584	139	138
Plier	657	441	102	114
Screwdriver	817	556	129	132
Wrench	1508	1033	238	237
Total	**40142**	**28287**	**5688**	**6167**

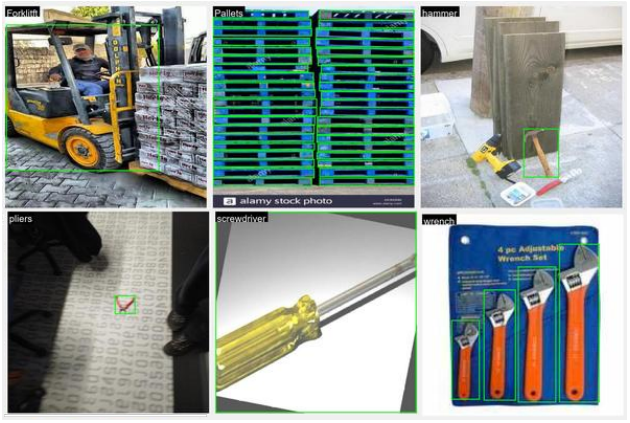

Figure 4.4: Examples of the images used in this study.

The dataset contains images with backgrounds related to industrial environments, but it also includes images with non-industrial backgrounds, such as plain white settings that isolate the object from any contextual scenery.

Regarding preprocessing, only image resizing to 640×640 pixels was performed to ensure consistency in model input. No other processing techniques were applied, as the original datasets had already undergone preprocessing by their respective authors.

The images within the dataset encompass both individual images of each object and images containing multiple objects, though, as noted, images containing more than one class are rare. The dataset is formatted for YOLOv11, using an export option available on the Roboflow platform.

While the distribution of images across classes is relatively balanced, there is a significant imbalance in instance frequency per class. The pallets class appears 35,433 times, whereas the average number of instances for the other classes is approximately 942. To address this imbalance, the initial approach considered was applying class weights during training. However, this feature could not be implemented in YOLOv11.

One potential strategy involved removing images from the 'Pallet' class to reduce class imbalance, however, this was not implemented, as it could have compromised the diversity of the dataset. Another considered approach was to apply data augmentation techniques, but this was

also set aside, since it would have significantly increased the dataset size, leading to longer training times and potentially affecting the overall feasibility of the project.

Ultimately, no corrective measures were applied to address this class imbalance. Nonetheless, during model training, it was observed that this imbalance did not significantly affect the final network performance.

4.3 Models selection

While conducting the state-of-the-art review for this Dissertation, focused on object detection in industrial environments with an emphasis on speed and efficiency for edge devices, as well as models capable of handling multi-class detection, it was observed that the majority of studies employed the YOLO architecture. This preference is due to YOLO's ability to deliver high speed, precision, and accuracy, making it particularly well-suited for deployment on edge devices.

For this reason, the models selected for this work are also based on the YOLO architecture, specifically, YOLOv11 and YOLOv12, the latest versions available.

4.3.1 YOLOv11

YOLO11 is the second latest iteration in the Ultralytics YOLO series of real-time object detectors, redefining what's possible with cutting-edge accuracy, speed, and efficiency. Building upon the impressive advancements of previous YOLO versions, YOLO11 introduces significant improvements in architecture and training methods, making it a versatile choice for a wide range of computer vision tasks. [24].Figure 4.5 compares YOLOv11 with other YOLO versions.

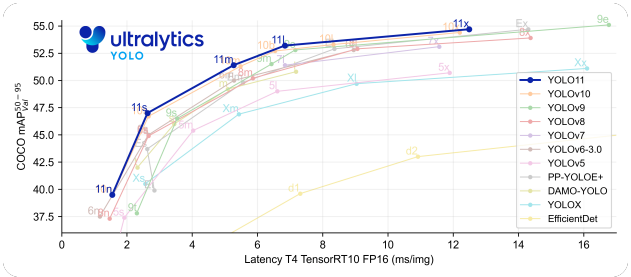

Figure 4.5: Comparative analysis of YOLOv11 performance against earlier YOLO architectures (from [24]).

YOLO is capable of doing a wider range of computer vision tasks including object detection, image segmentation, pose estimationand and so on. Ultralytics released five YOLO11 models according to the size and 25 models across all tasks:

- **YOLOv11n:** Nano variant designed for small and lightweight tasks;
- **YOLOv11s:** A slight upgrade over the Nano variant, offering improved accuracy;
- **YOLOv11m:** Medium variant, optimized for general-purpose use.

- **YOLOv11l:** Large variant, providing higher accuracy at the cost of increased computation;
- **YOLOv11x:** Extra-large variant, delivering maximum accuracy and performance.

Table 4.3 illustrates certain aspects of the variations in YOLOv11.

Table 4.3: YOLO11 Detection Performance on COCO val2017 [24]

Model	**size (pixels)**	**mAPval 50-95**	**T4 TensorRT (ms)**	**params (M)**	**FLOPs (B)**
YOLO11n	640	39.5	1.5	2.6	6.5
YOLO11s	640	47.0	2.5	9.4	21.5
YOLO11m	640	51.5	4.7	20.1	68.0
YOLO11l	640	53.4	6.2	25.3	86.9
YOLO11x	640	54.7	11.3	56.9	194.9

YOLO11 Architecture is an upgrade over YOLOv8 architecture with some new integrations and parameter tuning.The Figure 4.6 ilustrate YOLO11 Architecture.

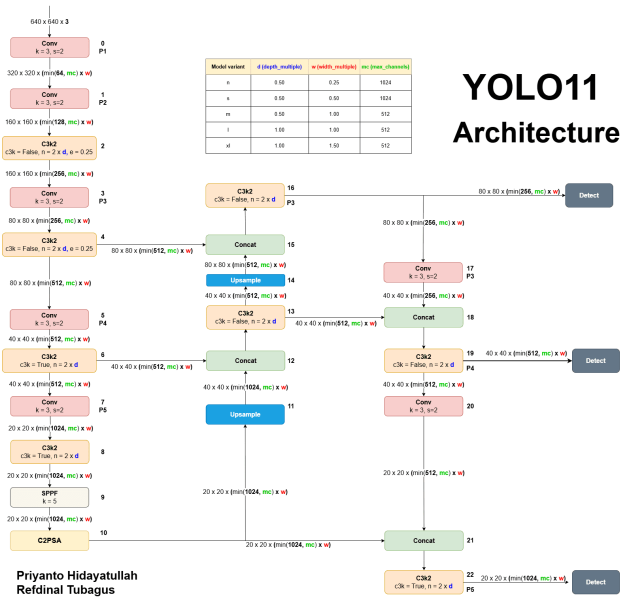

Figure 4.6: YOLO11 Architecture [24].

The improvement of yolov11 addresses 5 features [24]:

- **Enhanced Feature Extraction:** YOLOv11 features an enhanced backbone and neck architecture, improving feature extraction capabilities for more accurate object detection. [24]
- **Optimized Efficiency and Speed:** Refined architectural enhancements and optimized training pipelines contribute to faster processing speeds while preserving a balanced trade-off between accuracy and performance.

- **Greater Accuracy with Fewer Parameters:** YOLOv11m achieves higher mAP on the COCO dataset while utilizing 22% fewer parameters than YOLOv8m, making it more computationally efficient without sacrificing accuracy. [24]
- **Adaptability Across Environments:** YOLOv11 can be deployed across a wide range of environments, including edge devices, cloud platforms, and systems equipped with NVIDIA GPUs, ensuring flexibility and scalability for diverse applications. [24]
- **Broad Range of Supported Tasks:** YOLOv11 supports a wide range of computer vision tasks, including object detection, instance segmentation, image classification, pose estimation and oriented object detection, making it a versatile solution for various applications.

4.3.2 YOLOv12

YOLOv12 introduces an attention-centric architecture that diverges from the traditional CNN-based approaches employed in previous YOLO models, while still preserving the real-time inference speed critical for many applications. This model achieves state-of-the-art object detection accuracy through innovative advancements in attention mechanisms and the overall network architecture, all while maintaining real-time performance [52] [53]. Figure 4.7 compares YOLOv12 with others YOLO versions.

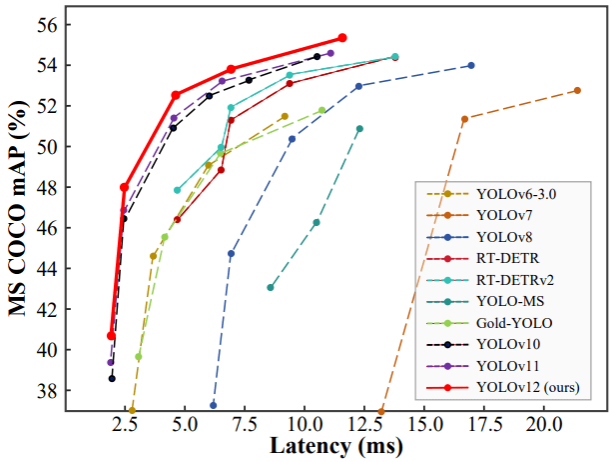

Figure 4.7: Comparative analysis of YOLOv12 performance against earlier YOLO architectures (from [52] [53]).

Table 4.4 shows the performance comparison between YOLO12 variants.

Table 4.4: YOLO12 Detection Performance on COCO val2017 [52] [53]

Model	**size (pixels)**	**mAPval 50-95**	**T4 TensorRT (ms)**	**params (M)**	**FLOPs (B)**
YOLO12n	640	40.6	1.64	2.6	6.5
YOLO12s	640	48.0	2.61	9.3	21.4
YOLO12m	640	52.5	4.86	20.2	67.5
YOLO12l	640	53.7	6.77	26.4	88.9
YOLO12x	640	55.2	11.79	59.1	199.0

YOLOv12 introduces two significant advancements:

- **Area Attention Mechanism:** A novel self-attention mechanism that efficiently handles large receptive fields by dividing the feature map into smaller, manageable regions. This allows the model to process each segment independently, reducing computational complexity when compared to conventional attention mechanisms. Despite this segmentation, the model retains a wide receptive field, enabling it to capture global contextual information without relying on complex operations [52] [53].
- **Residual Efficient Layer Aggregation Networks (R-ELAN):** An improved feature aggregation module derived from ELAN, specifically designed to overcome optimization challenges commonly faced in large-scale attention-driven architectures [52] [53].

However, YOLOv12 also introduces a notable limitation:

- **Hardware Compatibility Constraint:** The optimal performance of YOLOv12 relies on FlashAttention, a mechanism that is only supported by newer GPU architectures, including NVIDIA's Turing, Ampere, Ada Lovelace, and Hopper series (e.g., Tesla T4, RTX 20/30/40-series, A100, H100). Consequently, devices equipped with older GPUs cannot fully benefit from this optimization. In such cases, users must fall back to standard attention kernels, which may result in a loss of inference speed and efficiency.

4.4 Model Exportation for Edge Deployment

In order to deploy trained models on edge devices, it is necessary to convert them from their original format (PyTorch `.pt`) into a format compatible with such devices. These devices are often resource-constrained in terms of memory, computational power, and battery life. Therefore, models must be optimized to ensure efficient execution. Several standardized formats exist for edge deployment, including:

- **TensorFlow Lite** (`.tflite`);
- **Core ML** (`.mlmodel`);

- **PyTorch Mobile** (`.ptl`);
- **ONNX Runtime** (`.onnx`).

For this study, **TensorFlow Lite (TFLite)** was selected due to the following advantages:

- Broad compatibility with mobile and embedded systems;
- Support for various optimization techniques, including quantization;
- Native integration with the YOLO framework and its export utilities.

4.4.1 Conversion

The YOLO framework simplifies this workflow by providing a built-in `export` function that automates the entire conversion. In this Dissertation, the export process was executed using Google Colab. Although the following steps are described to illustrate what occurs internally, it is important to note that these are automatically handled by the YOLO `export` function, requiring no manual implementation from the user:

1. **Initial Export:** The trained PyTorch model is first exported to an intermediate format, in this case the ONNX format.
2. **TFLite Conversion:** The second step consists of converting the model to the TensorFlow Lite format (`.tflite`), simplifying deployment and enabling support for:
    - Low-latency inference;
    - Deployment in constrained environments.

This methodology ensures that the model maintains a balance between detection accuracy and computational efficiency. The streamlined workflow offered by the YOLO framework simplifies this multi-step conversion and guarantees a seamless transition from training to real-world deployment on edge devices.

4.5 Computational Infrastructure and Implemented Resources

4.5.1 Computational Infrastructure

In this project, a custom desktop computer was utilized, equipped with a 12th Gen Intel® Core™ i5-12400F processor (2.50 GHz), an NVIDIA GeForce RTX 3060 Ti graphics card, and 32 GB of RAM. The system operated on Microsoft Windows 11 Home 64-bit. The model training process was conducted using the GPU to enhance computational efficiency and accelerate training performance. Additionally, a Redmi 12 smartphone was used in this Dissertation to test the behavior of the model on edge devices. The device features a MediaTek Helio G88 octa-core CPU (max

2.00 GHz) and 12 GB of RAM, running on the Android operating system. This setup allowed for evaluating the real-time inference performance and efficiency of the trained object detection model in a resource-constrained environment. Additionally, Google Colab was utilized during the model export phase. Colab is a cloud-based Jupyter Notebook platform that offers free access to computing resources, without requiring any setup [20]. In this context, Colab was used due to compatibility issues encountered locally, mismatches between library versions required by the YOLO export function. These issues made it difficult to perform the export process on the desktop machine, whereas Colab provided a consistent and pre-configured environment in which the export operation could be executed successfully.

4.5.2 Hyperparameter Evaluation

To train the object detection models, a set of well-defined hyperparameters was selected to achieve an effective balance between detection performance and computational efficiency. The training process was conducted over 100 epochs, with a batch size of 16. An input image resolution of 640×640 pixels was adopted, consistent with the default input size of YOLO models. This choice supports compatibility with pretrained weights while preserving high-resolution detection capabilities. Additionally, a patience parameter of 20 was configured to enable early stopping if no performance improvements were observed over consecutive epochs, thereby optimizing training time and preventing overfitting.

The training was conducted using GPU acceleration (device='cuda'), leveraging the processing power of an NVIDIA RTX 3060 Ti to significantly speed up the training process. Additionally, 8 worker threads were allocated for data loading.

In addition to the custom settings, the following default values provided by the YOLO framework were used:

- **Optimizer:** auto - which allows automatic selection based on the model configuration. This affects convergence speed and training stability;
- **Initial Learning Rate (lr0):** 0.01 – determines how rapidly the model weights are updated;
- **Final learning rate (lrf):** 0.01 – the final learning rate is calculated as `lr0 * lrf`, controlling decay over time with schedulers;
- **Weight Decay:** 0.0005 – used to penalize large weights and prevent overfitting;
- **Box Loss Weight:** 7.5 – emphasizes the importance of accurately predicting bounding box coordinates;
- **Classification Loss Weight:** 0.5 – influences the importance of correct class prediction in the total loss function.

Figure 4.8 shows an example of a random batch used by the model during training.

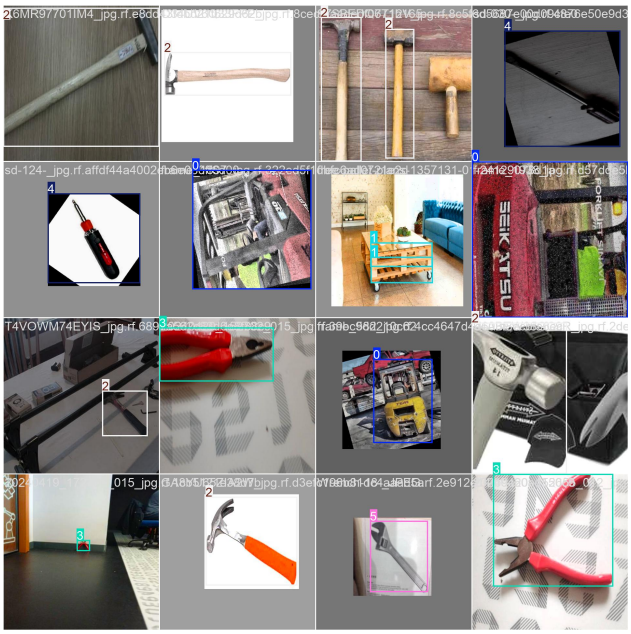

Figure 4.8: Random batch used by the model during training.

4.5.3 Development language

4.5.3.1 Python

Python is a high-level programming language widely used in the development of AI applications due to its simplicity, readability, and extensive ecosystem of specialized libraries. Frameworks such as TensorFlow, PyTorch, and Scikit-learn facilitate the creation and training of machine learning and deep learning models [19]. Additionally, Python integrates seamlessly with data processing tools such as NumPy and Pandas, making it particularly suitable for handling large datasets [33]. In the context of this Dissertation, Python was employed for the development and training of the YOLO-based object detection model, ensuring efficient implementation and optimization of the detection pipeline.

4.5.3.2 Kotlin

Kotlin is an expressive and concise programming language that reduces common code errors and easily integrates into existing apps officially recommended by Google for Android development. Fully interoperable with Java, Kotlin provides significant improvements in syntax, null safety, and support for functional programming paradigms [1]. Within this Dissertation, Kotlin was utilized to develop the Android application integrating the YOLO object detection model. Through Android Studio, Kotlin enabled the implementation of the user interface and the integration of the AI model for real-time image processing and object detection, ensuring both efficiency and responsiveness in mobile deployment.

4.6 Model Testing and Deployment

The application was developed with the following components:

- **Development Environment**: Android Studio;
- **Programming Language**: Kotlin;
- **Machine Learning Framework**: TensorFlow Lite;

4.6.1 Functionality

The application offers the following features:

- Single-image detection mode;
- Option to select an image from the device or capture a photo using the camera for object detection;
- Visual annotation of detected objects;
- Display of performance metrics:
  - Confidence scores for each detection;
  - Inference time (milliseconds);
  - Total number of detected objects;
  - Total processing time from image input to detection output.
- Configurable maximum detection threshold.

4.6.2 User Interface

The user interface (UI) is structured as follows:

1. Selection of an existing image or capture of a new photo via the device camera;
2. Execution of model inference on the selected or captured image;
3. Visualization of detection results with bounding boxes and labels;
4. Display of relevant performance metrics.

This testing framework enables a comprehensive assessment of the model's performance, while also showcasing its applicability in real-world edge computing scenarios. Figure 4.9 presents the initial layout of the application's interface and its main functionalities.

Figure 4.9: Initial interface of the mobile application.

4.7 Summary

This Chapter presented the methodology used to develop a fast and efficient object detection system tailored for industrial applications and edge device deployment. The process involved three main phases: dataset preparation, model training and model evaluation.

A custom dataset was created using images from Roboflow, comprising 4,151 images and six object classes: Forklift, Pallet, Hammer, Pliers, Screwdriver and Wrench. The dataset was carefully split into training, validation, and test sets in a 70%, 15%, and 15% ratio, respectively, ensuring all classes were represented in each subset. Only minor preprocessing was applied. Although a class imbalance was identified, no corrective measures were implemented, as it did not significantly affect the model's performance.

Four models (yolov11n, yolov11s, yolov12n and yolov12s) were selected based on a literature review highlighting YOLO's popularity for edge applications due to its speed and accuracy. YOLOv11 introduced improved architectural components and supports a wide range of computer

vision tasks, while YOLOv12 focused on attention-based mechanisms to achieve state-of-the-art accuracy with real-time performance.

Experiments were conducted using a desktop computer with an NVIDIA RTX 3060 Ti GPU and a Redmi 12 smartphone for edge testing. Training used 100 epochs with a batch size of 16 and an image size of 640×640. The implementation was carried out using Python for model training and Kotlin for Android application development, allowing seamless integration of real-time object detection on mobile devices.

Chapter 5

Results and Discussion

This chapter presents the experimental findings obtained from the training and evaluation of different YOLO-based object detection models. The objective is to provide a comprehensive view of their performance using quantitative metrics and visual outputs. Initially, the results are presented in detail. Afterwards, a preliminary test was conducted using images from the validation folder to assess how each model performs on unseen data. This is followed by a critical discussion to interpret their significance and implications for practical deployment. Subsequently, the results obtained from the deployment on an edge device will be demonstrated. Finally, a summary of this chapter will be provided.

5.1 Results

This chapter presents the results obtained from the training of the YOLO models (yolov11n, yolov11s, yolov12n and yolov12s). The evaluation metrics and training outputs are primarily derived from the artifacts automatically generated by the YOLO framework at the end of the training process.

Performance indicators were extracted from the `results.csv` file located in the training directory, ensuring precise numerical values for metrics.

Additionally, graphical outputs such as confusion matrices and plots tracking the evolution of key metrics throughout the training process were generated and used to support the visual analysis.

The model architecture summary displayed at the end of each training session was used to collect information regarding the number of Layers, total Parameters, computational cost in Floating-Point Operations Per Second (FLOPs), the total Training Time and Model Size providing valuable insights into the resource demands and scalability of each variant.

5.1.1 Evaluation metrics

The results obtained are presented in Table 5.1, which includes several evaluation metrics provided by the YOLO framework. These metrics include: Box Loss (localization performance), Precision, Recall, F1-score, mAP50 (mAP at 50% IoU threshold), mAP50-95 (mAP across IoU

thresholds ranging from 0.5 to 0.95), and Inference time. Together, these metrics provide a comprehensive quantitative assessment of model performance across three critical dimensions: localization accuracy (Box Loss), classification reliability (Precision/Recall/F1-score) and overall detection quality (mAP values), while the Inference time metric evaluates computational efficiency.

Table 5.1: Evaluation Metrics Results of Trained Models.

Model	**Box Loss**	**Precision (%)**	**Recall (%)**	**F1-score (%)**	**mAP50 (%)**	**mAP50-95 (%)**	**Inference (ms)**
YOLOv11n	0.73	93.02	79.65	85.82	86.53	68.53	1.7
YOLOv12n	0.72	92.48	79.69	85.61	86.12	69.00	2.5
YOLOv11s	0.68	91.87	81.93	86.62	87.46	70.06	4.6
YOLOv12s	0.68	92.21	80.28	85.83	87.60	69.66	4.1

5.1.2 Normalized confusion matrix

Table 5.2 breaks down the per-class precision scores for each model, highlighting their performance on individual object categories: Forklift, Pallets, Hammer, Pliers, Screwdriver and Wrench.

Table 5.2: Precision per classe of Trained Models.

Modelo	**Forklitft (%)**	**Pallets (%)**	**Hammer (%)**	**Pliers (%)**	**Screwdriver (%)**	**Wrench (%)**
Yolov11n	98	95	76	84	82	69
Yolov12n	98	95	73	82	80	72
Yolov11s	97	95	74	87	81	72
Yolov12s	97	95	77	84	81	70

Figure 5.1 complement the results presented in Table 5.2 by illustrating the normalized confusion matrices of the trained models.

A normalized confusion matrix is a visual representation of a classification model's performance, where the values are expressed as proportions rather than absolute counts. Each row corresponds to the actual class, while each column represents the predicted class. Values along the main diagonal indicate the proportion of correctly classified instances for each class, whereas off-diagonal values show the proportion of misclassifications.

This representation provides a more intuitive understanding of how well each class is being recognized and highlights specific patterns of confusion between classes, which may not be evident from numerical metrics alone.

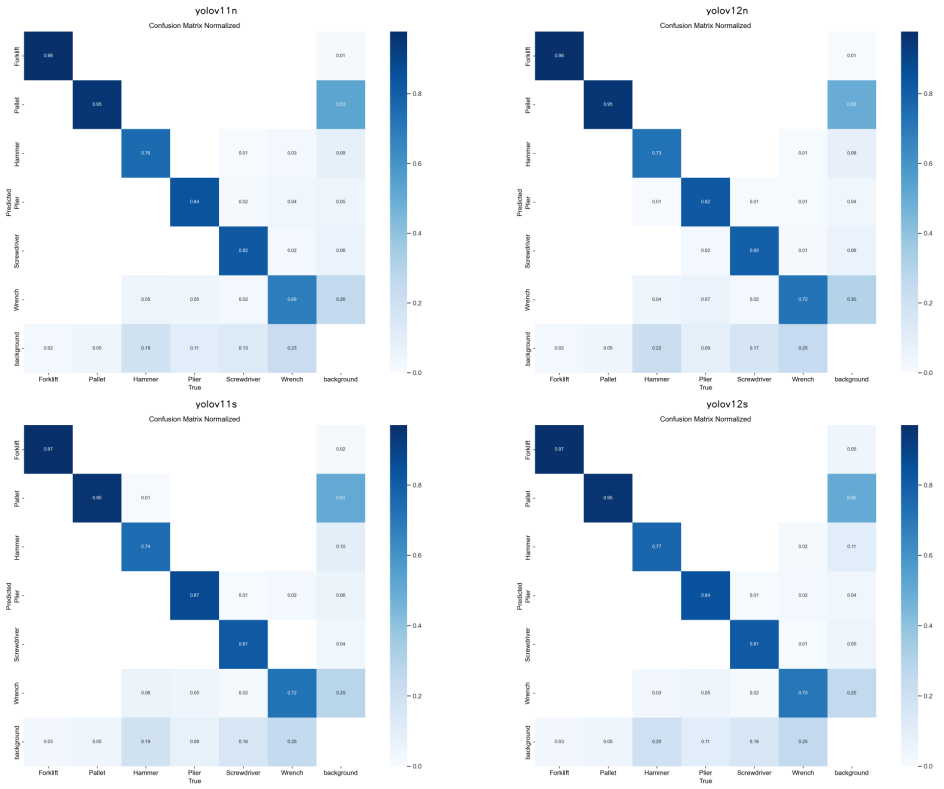

Figure 5.1: Confusion matrix of the models trained in this study.

5.1.3 Loss

The Figures 5.2, 5.3, and 5.4 illustrate the evolution of three key loss components throughout the training process of the YOLO models. These losses reflect distinct aspects of the model's learning dynamics.

Figure 5.2 illustrate the Localization loss (box_loss) where Represents the error associated with predicting the coordinates of the bounding boxes. It evaluates how accurately the model can localize objects within the image by comparing the predicted box positions to the ground truth.

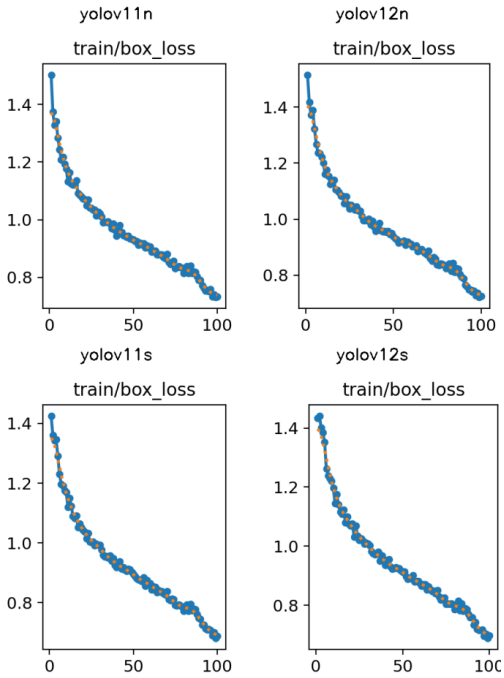

Figure 5.2: Localization loss of the trained models.

Figure 5.3 illustrates the classification loss (cls_loss), which measures the model's ability to correctly classify detected objects into their respective categories. This loss penalizes incorrect class predictions and is crucial for ensuring semantic accuracy in object detection.

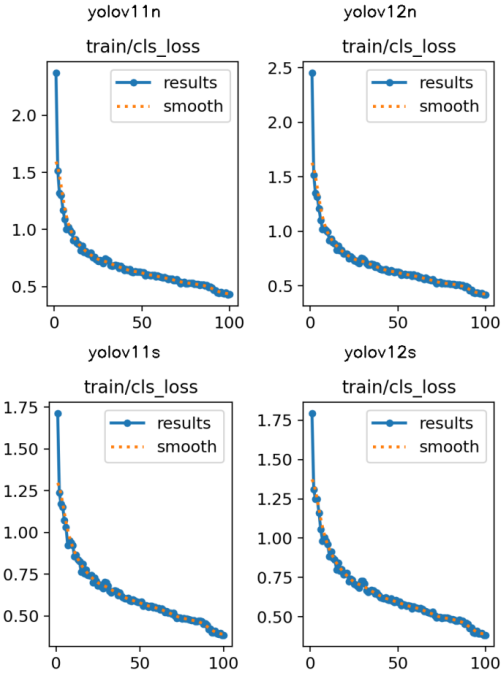

Figure 5.3: Classification loss of the trained models.

Figure 5.4 illustrates Distributed Focal Loss (dfl_loss), which builds on the focal loss and is designed to address class imbalance by reducing the influence of easily classified examples. DFL improves detection performance by distributing the focal loss across multiple scales and classes, enabling the model to better focus on difficult-to-detect objects. This distribution helps stabilize the learning process and increases effectiveness in complex detection scenarios [56].

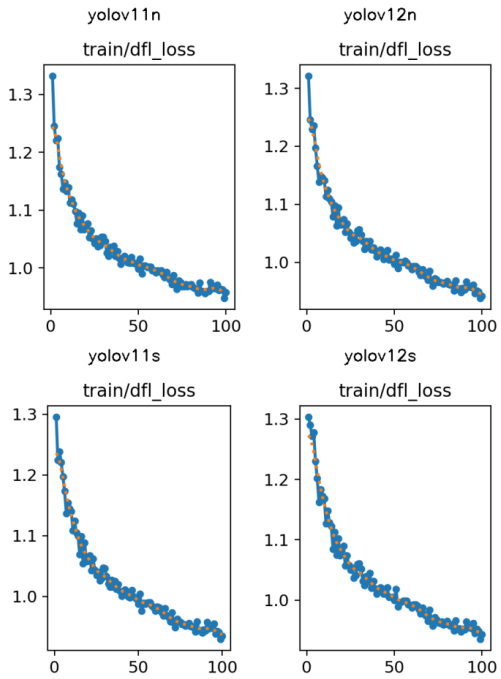

Figure 5.4: Distributed Focal Loss of the trained models.

5.1.4 Model complexity

The computational efficiency of object detection models is critical for real-world deployment. Table 5.3 presents a comparative analysis of the models trained across five key metrics:

- **Layers**: Represents the architectural depth of the model, i.e., the total number of computational layers (e.g., convolutional, pooling, etc.) involved in feature extraction and prediction;
- **Parameters**: Denotes the total number of trainable weights within the model. This value directly impacts the memory footprint and model capacity;
- **FLOPs**: Indicates the amount of computation required to process a single image, measured in billions of operations;
- **Training Time**: Refers to the total time, in hours, required to train each model configuration on the given dataset and hardware setup;
- **Model Size**: Specifies the storage size of the model on disk, in megabytes MB, which affects the model's deployability on memory-constrained or embedded devices.

Table 5.3: Comparison of Model Complexity.

Model	Layers	Parameters	FLOPs	Training Time (h)	Model Size (MB)
YOLOv11n	100	2,583,322	6.3	1.392	5.20
YOLOv12n	159	2,557,898	6.3	1.392	5.25
YOLOv11s	100	9,415,122	21.3	1.895	18.28
YOLOv12s	159	9,233,202	21.2	4.736	18.04

5.2 Discussion

5.2.1 Evaluation Metrics

Table 5.1 presents the performance metrics of the four trained YOLO models. Each variant reveals distinct strengths depending on the evaluation criterion, making them suitable for different application scenarios.

YOLOv11n delivers the fastest inference time at only 1.7 ms, making it an excellent candidate for real-time applications. It achieves the highest precision (93.02%) and a strong F1-score (85.82%), indicating that it is very effective at minimizing false positives. However, it has the highest box loss (0.73) among the models, suggesting less accurate object localization.

YOLOv12n offers slightly better box loss (0.72) and mAP50-95 (69.00%) compared to YOLOv11n, which suggests better generalization across IoU thresholds. Nevertheless, it comes with a higher inference time (2.5 ms) and a marginally lower F1-score (85.61%), implying a small trade-off between accuracy and speed.

YOLOv11s emerges as the most balanced and overall best-performing model. It achieves the lowest box loss (0.68), highest F1-score (86.62%), and the highest mAP50-95 (70.06%), demonstrating superior localization and consistent detection performance. Although it has a higher inference time (4.6 ms), the improved accuracy justifies its use in scenarios where precision and robustness are more critical than latency.

YOLOv12s also performs well, with competitive precision (92.21%), a strong F1-score (85.83%) and the highest mAP50 (87.60%). Its box loss (0.68) is on par with YOLOv11s, indicating precise bounding box predictions. With a slightly faster inference time (4.1 ms), it offers a good compromise between accuracy and efficiency, though it doesn't outperform YOLOv11s overall.

5.2.2 Normalized confusion matrix

Through Figure 5.1, which presents the confusion matrices for each model, it is evident that the majority of classes are detected correctly, with most models achieving accuracy rates above 70% for common object categories.

The 'Pallet' class achieved the second highest precision across all models, reaching 95%. This strong performance can largely be attributed to the high number of labeled instances for this class, with more than 35,000 examples across training, validation, and test sets, which provides the models with ample exposure and learning opportunities. However, this dominance may present certain

drawbacks. As observed in Figure 5.1, in scenarios where no objects are present (background), the models occasionally misclassify background regions as 'Pallet', with rates ranging between 50% and 53%. This pattern suggests a potential case of overfitting in the training dataset, where the models have learned to over-rely on common patterns or textures associated with 'Pallet', even when these cues appear in irrelevant contexts.

Beyond these general observations, the confusion matrices provide a more granular view of each model's behavior. All models exhibit particularly strong performance in identifying the 'Forklift' and 'Pallet' classes, consistently achieving true positive rates exceeding 95%. However, classification accuracy declines for smaller tools such as 'Wrench' and 'Hammer', which are more prone to misclassification. For the 'Wrench' class, the lowest true positive rates range between 69% and 72%, while for 'Hammer', they vary from 73% to 77%. Across all evaluated models, a consistent pattern of misclassification is observed regarding the 'Wrench' class. The 'Wrench' is the second most frequently misclassified object as background, with normalized confusion values typically ranging between 25% and 30%. This pattern is evident across the confusion matrices of all four YOLO variants, indicating a systematic weakness in accurately distinguishing this object from the background class.

Among the evaluated models, the 's' variations demonstrate the most consistent and balanced performance across all object classes. These models achieve higher precision values along the main diagonal of the confusion matrix, particularly for the 'Hammer' and 'Wrench' categories, indicating improved class discrimination. In contrast, the 'n' variations exhibit overall lower performance across most classes, with the exception of 'Forklift' and 'Pallet', where they match the performance of the 's' variants. This suggests that while the 'n' models are competitive for more frequent or visually distinct classes, the 's' models provide superior generalization and precision for less represented or more challenging classes.

5.2.3 Loss

Through Figures 5.2, 5.3 and 5.4, it is possible to observe that all models exhibit a consistent decrease in loss values as training progresses.

5.2.3.1 Localization Loss

Figure 5.2 displays the evolution of the localization loss (box_loss) over time. All four YOLO variants achieve low final loss values, indicating improved bounding box precision. However, differences in initial loss levels can be noted between models. Models from the 's' variant consistently start with lower localization losses, while the 'n' variants begin with higher loss values.

This distinction is directly linked to model complexity, as summarized in Table 5.3. For example, YOLOv11n and YOLOv12n contain approximately 2.5 million parameters, whereas YOLOv11s and YOLOv12s possess around 9 million parameters. The increased number of parameters and layers in the 's' models equips them with greater capacity to extract and process complex spatial features at earlier stages of training.

As a result, the more complex models are capable of capturing detailed patterns sooner, which translates into better initial bounding box predictions and consequently lower starting loss values.

5.2.3.2 Classification Loss

Figure 5.3 presents the evolution of the classification loss (cls_loss) over the training epochs for the four YOLO variants. Across all models, a sharp decrease in loss values is observed during the initial epochs, reflecting rapid early learning as the networks begin to distinguish between object classes. Following this initial phase, the rate of decrease becomes more gradual, indicating a transition to more refined adjustments in the models' predictions.

Despite this general pattern, the difference in the initial loss values is clearly noticeable. Similar to the behavior observed in the localization loss, the 's' variants (YOLOv11s and YOLOv12s) begin with lower classification loss values when compared to the 'n' variants. This distinction once again reflects the influence of model complexity.

5.2.3.3 Distributed Focal Loss

Figure 5.4 shows the evolution of the Distributed Focal Loss (DFL) for the lightweight variants of the YOLOv11 and YOLOv12 architectures, specifically the *n* and *s* models. All versions begin training with DFL values around 1.3, which may be attributed to dataset imbalance and the presence of challenging or ambiguous samples, scenarios where DFL is particularly sensitive. These initial values reflect the models initial difficulty in handling class distribution disparities and learning from harder instances in the early stages of training.

Overall, all models exhibit a continuous decrease in loss over the 100 epochs, reflecting an effective learning process. However, it is important to note that the convergence of the loss tends to slow down after approximately 60-70 epochs, indicating that additional training beyond this point may not yield substantial improvements unless adjustments are made to the training strategy.

Finally, although all models achieve similar DFL loss values (around 0.9), no clear advantage of the YOLOv12 versions over the YOLOv11 counterparts is observed based solely on the DFL evolution.

5.2.4 Model complexity

Table 5.3 presents a comparative overview of the four YOLO models evaluated in this study, detailing their architectural depth (number of layers), number of parameters, computational complexity (measured in FLOPs), total training time in hours and storage size in MB.

The models are divided into two categories based on their size and complexity: the nano ('n') variants and the small ('s') variants. The YOLOv11n and YOLOv12n models are the most lightweight architectures, featuring approximately 2.56 to 2.58 million parameters, a computational cost of 6.3 FLOPs, and compact model sizes of about 5.20 MB. Both models completed training in roughly 1.39 hours.

In contrast, the YOLOv11s and YOLOv12s models are larger and more complex, with over 9 million parameters and computational costs around 21.2 FLOPs. These models required longer training times, with YOLOv11s training in about 1.90 hours and YOLOv12s in approximately 4.74 hours. Their storage sizes are significantly larger as well, at 18.28 MB and 18.04 MB, respectively.

Inference times also vary significantly between these models, as detailed in Table 5.1. The YOLOv11n achieved the fastest inference at 1.7 milliseconds per image, followed by YOLOv12n at 2.5 milliseconds. The small variants have higher inference times, with 4.6 milliseconds for YOLOv11s and 4.1 milliseconds for YOLOv12s.

5.3 Model validation

This section presents the results obtained when the trained models were evaluated using the validation dataset. The objective is to analyze, in detail, how each model performed in terms of *Precision*, *Recall*, *F1-score*, *mAP@50* and *mAP@50-95*, both per class and overall. This helps reinforce the selection of the most suitable model for deployment on edge devices.

The evaluation was conducted using the following command:

```
from ultralytics import YOLO
model = YOLO("runs/detect/modelX/weights/best.pt")
model.val()
```

where `modelX` corresponds to the following model identifiers:

- YOLOv11n
- YOLOv12n
- YOLOv11s
- YOLOv12s

In addition to the quantitative evaluation, a qualitative analysis was performed using two images from batch 16 of the validation dataset for each model. This visual inspection helps verify how well each model detects and localizes fractures under different conditions, complementing the numerical metrics.

5.3.1 YOLOv11n

Table 5.4 presents the performance of the YOLOv11n model on the validation dataset. This model is one of the two most lightweight among the tested versions, with only 2.58 million parameters and 6.3 FLOPs.

Despite its compact architecture, YOLOv11n achieved strong performance, with an overall precision of **93.4%**, recall of **79.5%** and an F1-score of approximately **85.8%**. The model also

obtained a mAP@50 of **86.6%** and a mAP@50-95 of **68.5%**, reflecting a solid balance between detection accuracy and localization precision across varying IoU thresholds.

Table 5.4 reveals that YOLOv11n performs exceptionally well on the 'Forklift' and 'Pallet' classes, achieving precision values of **97.8%** and **98.3%**, respectively, along with mAP@50 scores exceeding **96%**. The 'Pallet' class, being the most represented in the dataset, likely benefits from the high number of training instances, which may lead to signs of overfitting, where the model becomes overly specialized in detecting this dominant class. In contrast, the 'Forklift' class has a number of instances comparable to other object categories, suggesting that its strong performance could be due to either mild overfitting or, alternatively, the model's effective learning of the distinctive visual features associated with forklifts.

In contrast, performance was more modest on underrepresented classes such as 'Hammer' and 'Wrench', which recorded lower mAP@50-95 values of **59.3%** and **49.6%**, respectively. This indicates difficulties in generalizing to less frequent objects or those with ambiguous visual features.

In terms of speed, YOLOv11n demonstrated excellent efficiency, requiring only **2.3 milliseconds per image** for inference, with a total processing time (including pre-processing and post-processing) averaging **3.8 milliseconds per image**.

Table 5.4: YOLOv11n results on the validation dataset

Class	**Images**	**Instances**	**Precision**	**Recall**	**F1-score**	**mAP@50**	**mAP@50-95**
Forklift	135	137	0.978	0.958	0.968	0.981	0.788
Pallet	95	4943	0.983	0.945	0.964	0.967	0.778
Hammer	111	139	0.891	0.655	0.755	0.809	0.593
Plier	88	102	0.937	0.814	0.871	0.890	0.766
Screwdriver	119	129	0.955	0.791	0.865	0.840	0.688
Wrench	103	238	0.858	0.608	0.711	0.708	0.496
All	**625**	**5688**	**0.934**	**0.795**	**0.859**	**0.866**	**0.685**

Figures 5.5 and 5.6 illustrate batches of 16 images from the validation set, with object detection performed by the trained model. These batches correspond to one of the default outputs generated by the `model.val()` function from the Ultralytics YOLO framework. As this output is consistent across different models when using the same validation dataset, the same set of images is shown for all models, enabling a fair visual comparison of their detection performance.

In Figure 5.5, the model correctly identified objects in 14 out of the 16 images. The two incorrect predictions involved duplicate detections: in both cases, the model predicted two overlapping instances of an object when only one was actually present. These false positives occurred in the image at row 2, column 1, and in the image at row 2, column 2.

Figure 5.5: YOLOv11n detection outputs on a 16-image batch from the validation set, example 1.

In Figure 5.6, the model successfully detected all objects across the 16 images. The confidence scores for these predictions ranged from as low as 0.4 (for the 'Pallet' class) to a perfect 1.0 (for the 'Screwdriver' class), demonstrating the model's ability to detect a variety of objects with varying confidence levels.

Figure 5.6: YOLOv11n detection outputs on a 16-image batch from the validation set, example 2.

5.3.2 YOLOv12n

Table 5.5 shows the results of the YOLOv12n model evaluated on the validation dataset. As the second most lightweight model in the series, YOLOv12n is designed for efficient edge deployment while offering a slightly deeper architecture compared to YOLOv11n.

The model achieved an overall precision of **92.5%**, recall of **79.7%** and an F1-score of approximately **85.6%**. It demonstrated particularly strong performance on the 'Pallet' and 'Forklift' classes, which represent the highest and moderately represented categories in terms of training instances, respectively. The strong results on the 'Pallet' class may indicate some degree of overfitting due to its dominance in the dataset, whereas the performance on the 'Forklift' class may reflect either mild overfitting or the model's ability to effectively learn distinctive features of this category.

In contrast, lower performance was observed on less frequent classes such as 'Hammer' and 'Wrench', with mAP@50-95 values of **61.0%** and **49.1%**, respectively, suggesting challenges in generalizing to underrepresented or visually ambiguous objects.

Regarding inference speed, YOLOv12n required an average of **3.6 milliseconds per image**, representing a slight increase compared to YOLOv11n but still preserving a real-time processing capability.

Table 5.5: YOLOv12n results on the validation dataset

Class	Images	Instances	Precision	Recall	F1-score	mAP@50	mAP@50-95
Forklift	135	137	0.978	0.963	0.970	0.979	0.806
Pallet	95	4943	0.979	0.948	0.963	0.967	0.780
Hammer	111	139	0.923	0.662	0.771	0.810	0.610
Plier	88	102	0.948	0.804	0.870	0.899	0.791
Screwdriver	119	129	0.936	0.787	0.855	0.821	0.664
Wrench	103	238	0.786	0.618	0.692	0.692	0.491
All	**625**	**5688**	**0.925**	**0.797**	**0.856**	**0.861**	**0.690**

Figures 5.7 and 5.8 display batches of 16 images from the validation set, showcasing the predictions made by the YOLOv12n model. These visualizations correspond to one of the default outputs generated by the `model.val()` function from the Ultralytics YOLO framework. Since the validation dataset is fixed, the same image batches are used across different models, enabling a direct and fair visual comparison of detection accuracy and consistency.

In Figure 5.7, the model correctly detected objects in 15 out of the 16 images. The only incorrect prediction involved duplicate detections, where the model predicted two overlapping instances of an object when only one was actually present. This error occurred in the image located at row 4, column 3.

Figure 5.7: YOLOv12n detection outputs on a 16-image batch from the validation set, example 1.

In Figure 5.8, the model successfully detected all objects across the 16 images. The confidence scores for these predictions ranged from as low as 0.4 (for the 'Pallet' class) to a perfect 1.0 (for the 'Screwdriver' class), demonstrating the model's ability to detect a variety of objects with varying confidence levels.

Figure 5.8: YOLOv12n detection outputs on a 16-image batch from the validation set, example 2.

It is worth noting that in the image at row 2, column 4, the model detected two additional 'Pallet' objects, positioned in the upper right corner. These objects were not annotated in the corresponding ground truth labels shown in Figure 5.9. However, since these detected objects clearly correspond to pallets that were missed in the annotations, their detection by the model is considered correct.

Figure 5.9: Ground truth annotations on a 16-image batch from the validation set, example 2.

5.3.3 YOLOv11s

Table 5.6 presents the performance metrics for the YOLOv11s model evaluated on the validation dataset. YOLOv11s incorporates a more complex architecture and a greater number of parameters than its lighter counterparts, targeting a more balanced trade-off between detection accuracy and computational efficiency.

The model achieved an overall precision of **91.9%**, recall of **81.9%** and an F1-score of approximately **86.6%**. These figures represent a significant improvement compared to YOLOv11n and YOLOv12n, reflecting the benefits of its increased representational capacity.

Notably, the model performed best on the 'Pallet' and 'Forklift' classes, which are among the most frequent in the training dataset. The 'Pallet' class reached a mAP@50 of **97.1%** and the 'Forklift' class achieved **98.6%**, suggesting that the model successfully captured the distinct visual patterns of these object types. While such high performance is promising, it may also indicate potential overfitting to these dominant classes.

Conversely, detection performance was more modest for underrepresented objects such as 'Hammer' and 'Wrench', which obtained mAP@50-95 values of **60.9%** and **53.3%**, respectively. These results are consistent with earlier observations across the YOLOv11 series, highlighting the model's ongoing difficulty in generalizing to visually ambiguous or sparsely represented categories.

Despite its more demanding architecture, YOLOv11s maintained a competitive inference speed, processing each image in approximately **4.8 milliseconds**. This real-time capability, combined with its accuracy, makes YOLOv11s a strong candidate for scenarios that require both responsiveness and reliable object detection.

Table 5.6: YOLOv11s results on the validation dataset

Class	**Images**	**Instances**	**Precision**	**Recall**	**F1-score**	**mAP@50**	**mAP@50-95**
Forklift	135	137	1.000	0.959	0.979	0.986	0.796
Pallet	95	4943	0.983	0.952	0.967	0.971	0.800
Hammer	111	139	0.913	0.669	0.772	0.804	0.609
Plier	88	102	0.868	0.853	0.860	0.917	0.785
Screwdriver	119	129	0.962	0.793	0.869	0.836	0.681
Wrench	103	238	0.785	0.689	0.734	0.738	0.533
All	**625**	**5688**	**0.919**	**0.819**	**0.866**	**0.875**	**0.701**

Figures 5.10 and 5.11 display batches of 16 images from the validation set, showcasing the predictions made by the YOLOv11s model. These visualizations are part of the standard output generated by the `model.val()` function from the Ultralytics YOLO framework. Since this output is consistent across different models when using the same validation dataset, the same image batches are presented, enabling a fair visual comparison of detection performance.

In Figure 5.10, the YOLOv11s model accurately identified all objects within the 16-image batch. The confidence scores for these detections varied between 0.5 and 0.9, both recorded for instances of the 'Forklift' class. This indicates the model's consistent capacity to detect objects from this class despite differences in visual conditions or occlusions.

Figure 5.10: YOLOv11s detection outputs on a 16-image batch from the validation set, example 1.

Similarly, in Figure 5.11, the model also achieved complete object detection across all images. The confidence values ranged from 0.4, observed in a 'Pallet' detection, to a peak of 0.9 across multiple classes. These results underscore the model's robustness and adaptability in identifying a diverse range of objects with varying levels of certainty, even under less favorable detection conditions.

Figure 5.11: YOLOv11s detection outputs on a 16-image batch from the validation set, example 2.

5.3.4 YOLOv12s

Table 5.7 presents the evaluation results of the YOLOv12s model on the validation dataset. As one of the more accurate variants in the lightweight series, YOLOv12s balances architectural complexity and real-time performance.

The model achieved an overall precision of 92.2%, recall of 80.3% and an F1-score of approximately 85.8%. This positions it as one of the strongest performers in the series in terms of balanced detection performance, showing robust generalization across object categories.

In terms of class-specific performance, the model yielded exceptional results for both the 'Pallet' and 'Forklift' classes, which are among the most frequently represented in the dataset. The 'Pallet' class reached a near-perfect mAP@50 of 96.9% and an F1-score of 96.6%, while 'Forklift' also maintained high precision and recall, with a corresponding mAP@50-95 of 80.3%. These results suggest that YOLOv12s effectively captures distinguishing features of dominant classes, though a slight risk of overfitting may still be present due to data imbalance.

Conversely, as observed in previous models, lower detection quality was evident in underrepresented categories such as 'Hammer' and 'Wrench'. These classes recorded F1-scores of 76.9% and 70.8%, respectively, with corresponding mAP@50-95 values of 60.5% and 52.3%. This reinforces the model's relative difficulty in generalizing to infrequent or visually ambiguous objects, despite the increased model complexity.

From a computational perspective, YOLOv12s maintained a real-time processing capability, with an average inference time of **7.0 milliseconds per image**. While this is higher than earlier variants such as YOLOv11n and YOLOv12n, it is a justified trade-off given the improved accuracy across most classes.

Table 5.7: YOLOv12s results on the validation dataset

Class	**Images**	**Instances**	**Precision**	**Recall**	**F1-score**	**mAP@50**	**mAP@50-95**
Forklift	135	137	0.944	0.956	0.950	0.974	0.803
Pallet	95	4943	0.982	0.950	0.9657	0.969	0.796
Hammer	111	139	0.895	0.674	0.7689	0.828	0.605
Plier	88	102	0.948	0.814	0.8759	0.902	0.787
Screwdriver	119	129	0.959	0.791	0.8669	0.843	0.667
Wrench	103	238	0.807	0.631	0.7082	0.740	0.523
All	**625**	**5688**	**0.922**	**0.803**	**0.8584**	**0.876**	**0.697**

Figures 5.12 and 5.13 illustrate batches of 16 images from the validation set, with object detection performed by the trained model. These batches correspond to one of the default outputs generated by the `model.val()` function from the Ultralytics YOLO framework. As this output is consistent across different models when using the same validation dataset, the same set of images is shown for all models, enabling a fair visual comparison of their detection performance.

In Figure 5.12, the model correctly identified objects in 14 out of the 16 images. The two incorrect predictions involved duplicate detections: in both cases, the model predicted two overlapping instances of an object when only one was actually present, both belonging to the class 'Forklift'. These false positives appeared in the image located at row 2, column 4, and in the image at row 4, column 3.

Figure 5.12: YOLOv12s detection outputs on a 16-image batch from the validation set, example 1.

Similarly, in Figure 5.13, the model also correctly identified objects in 14 out of the 16 images. The two errors again consisted of duplicate detections, this time for the class 'Pallet'. These false positives were found in the images at row 1, column 4, and row 2, column 4.

Figure 5.13: YOLOv12s detection outputs on a 16-image batch from the validation set, example 2.

5.4 Model deployment

For deployment within the developed application, the model selected was YOLOv11n, primarily due to its low inference time, making it well-suited for real-time object detection tasks. In addition to its speed, this model also exhibited satisfactory performance across key evaluation metrics, justifying its choice for integration.

The code developed in this Dissertation was inspired by the work presented in [57].

During implementation, the PyTorch model (.pt) was exported to the TensorFlow Lite (.tflite) format, resulting in two versions: 'best_float32.tflite' and 'best_float16.tflite'. The 'float32' variant was ultimately chosen for deployment, as it provides superior precision and numerical stability, despite its higher memory consumption and slightly slower inference compared to the 'float16' version.

Upon testing within the application, the model demonstrated effective object detection capabilities when evaluated on the pre-defined test dataset. However, its performance was notably inconsistent when exposed to live images captured in real-world conditions. Several issues were observed, including the repeated detection of the same object multiple times, the false detection of objects where none were present, incorrect class predictions (misclassifications), and in some cases, a complete failure to detect the object. These challenges suggest limitations in the model's

generalization capabilities, underscoring the need for greater dataset variability and potential domain adaptation strategies to enhance performance in uncontrolled environments.

Figures 5.14, 5.15, 5.16, 5.17, 5.18, and 5.19 illustrate the model's performance when applied to images from the predefined test dataset.

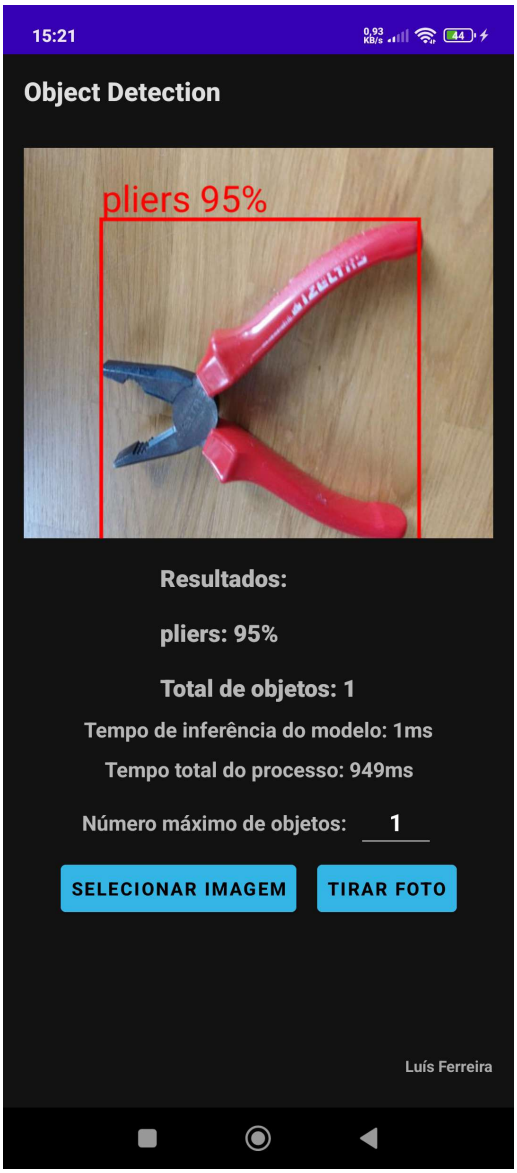

Figure 5.14: Test Image of object 'pliers'.

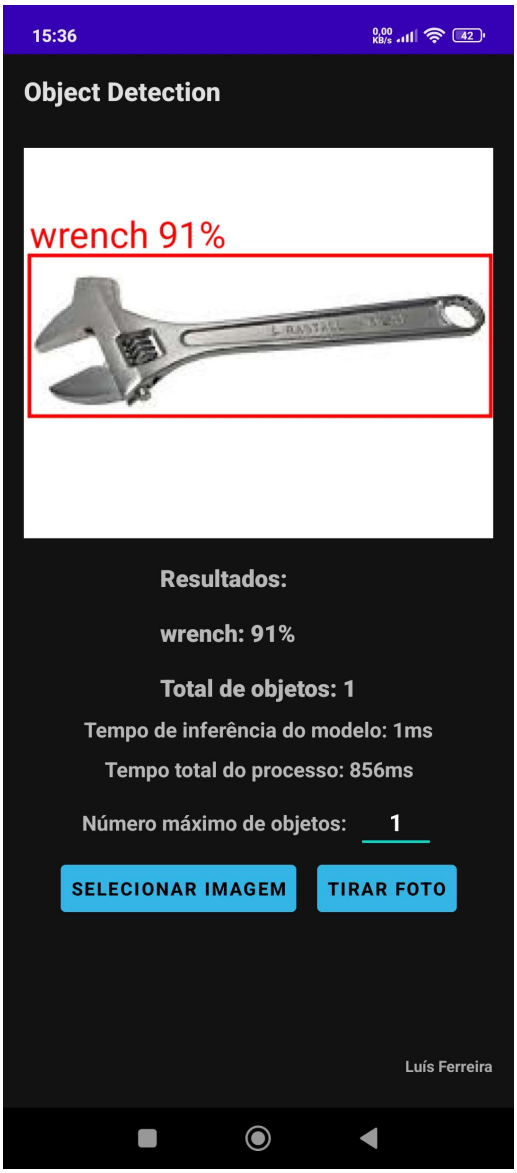

Figure 5.15: Test Image of object 'wrench'.

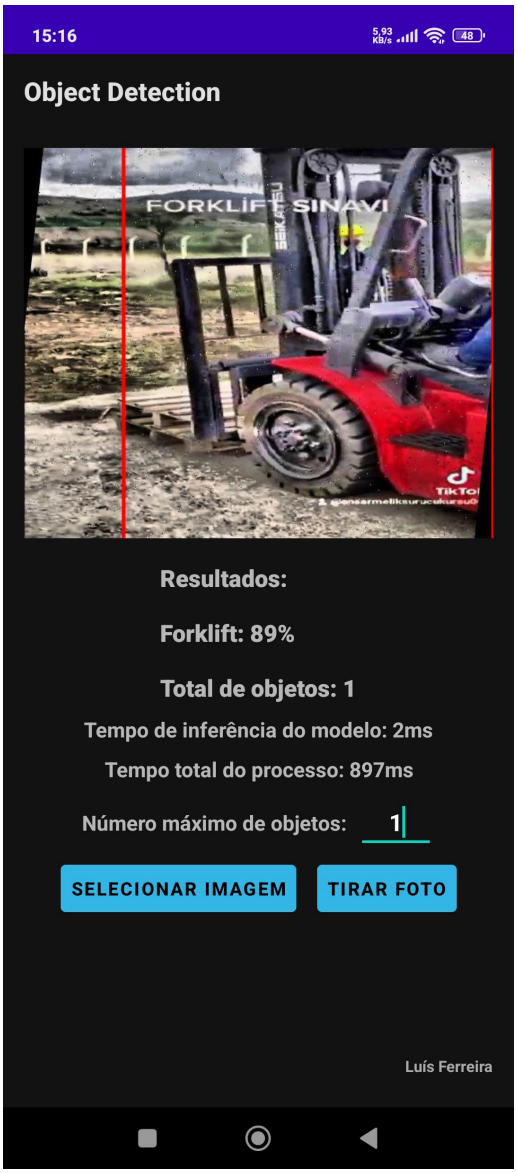

Figure 5.16: Test Image of object 'Forklift'.

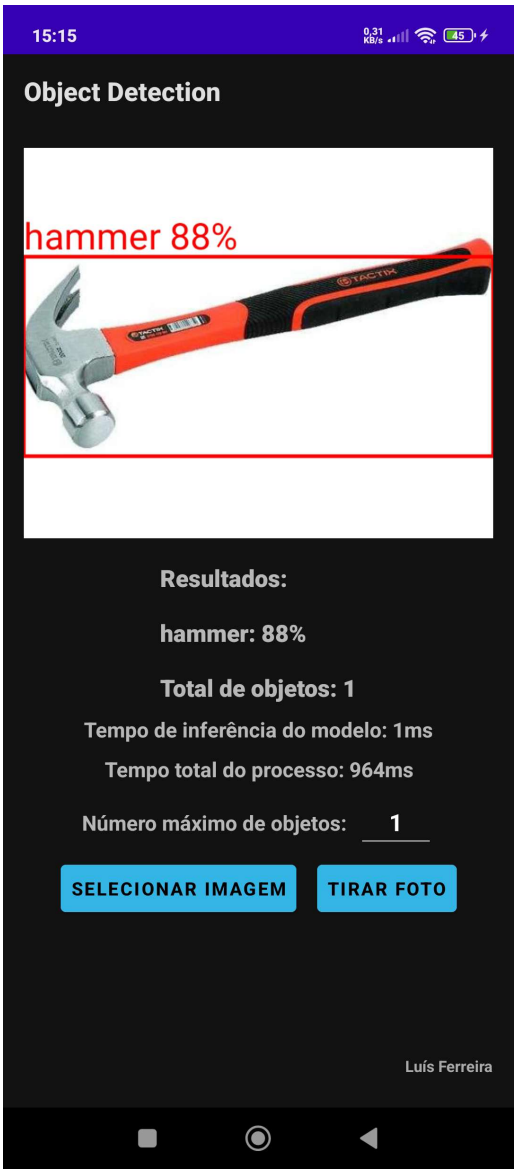

Figure 5.17: Test Image of object 'Hammer'.

Figure 5.18: Test Image of object 'Pallets'.

Figure 5.19: Test Image of object 'screwdriver'.

Figure 5.20 illustrates how the model responds to the detection of the same object, specifically the 'screwdriver', under varying background conditions. It is evident that the background significantly influences the model's detection performance, which can be attributed to the types of backgrounds present in the training dataset. Additionally, it can be observed that the localization of the object is not accurate, with bounding boxes not being tightly fitted around the object. This imprecision may stem from limited variability in object positioning during training or from insufficiently annotated samples that fail to capture diverse real-world contexts.

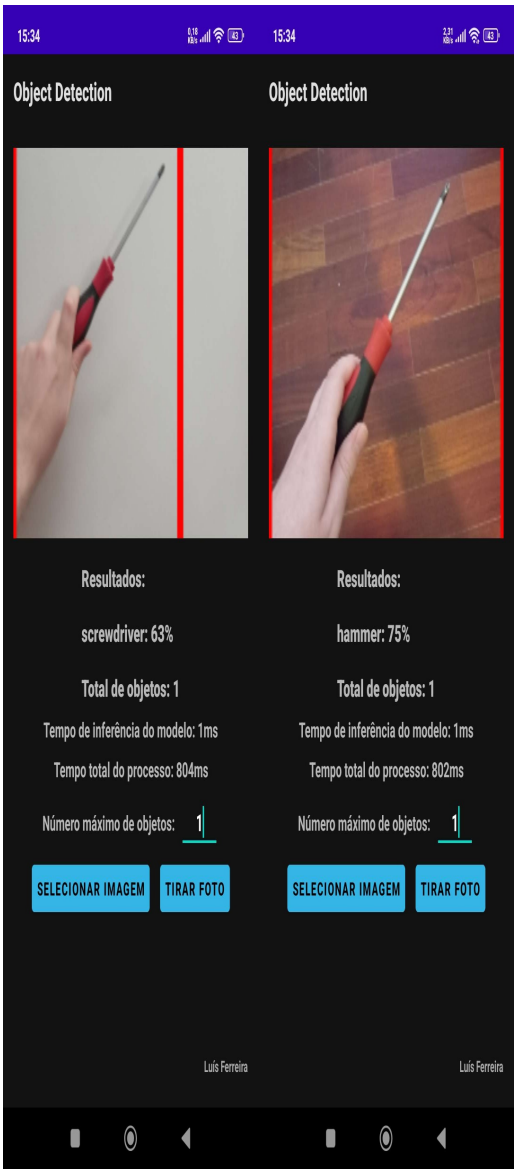

Figure 5.20: 'Screwdriver' with background white and background brown.

Figure 5.21 demonstrates how the model responds when presented with multiple instances of the same object, in this case the 'wrench'. Initially, the model struggled to detect all six instances correctly, often predicting duplicate detections of the same object. However, by increasing the maximum object detection limit by one, the model was able to successfully identify all six objects.

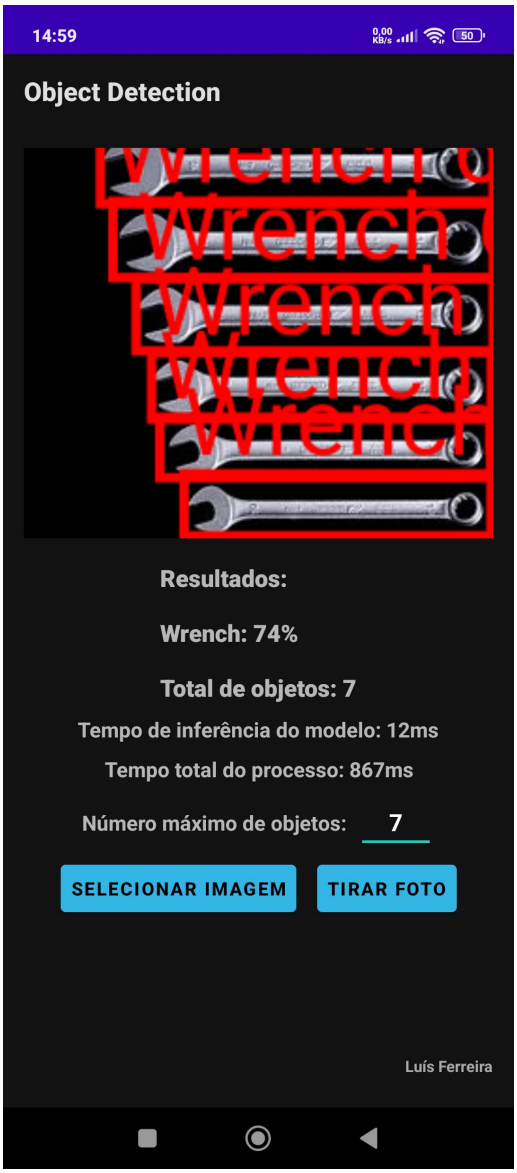

Figure 5.21: Detection example with multiple wrench instances.

Figure 5.22 illustrates a failure case where the model attempted to detect two 'Screwdriver' objects, but incorrectly produced duplicate detections for the same object.

Figure 5.23 shows that, when increasing the maximum number of detectable objects, the model was able to identify the two actual 'Screwdriver' objects. However, it generated a total of ten detections, most of which were overlapping predictions of the same objects. This indicates a clear case of over-detection due to the absence of filtering redundant bounding boxes.

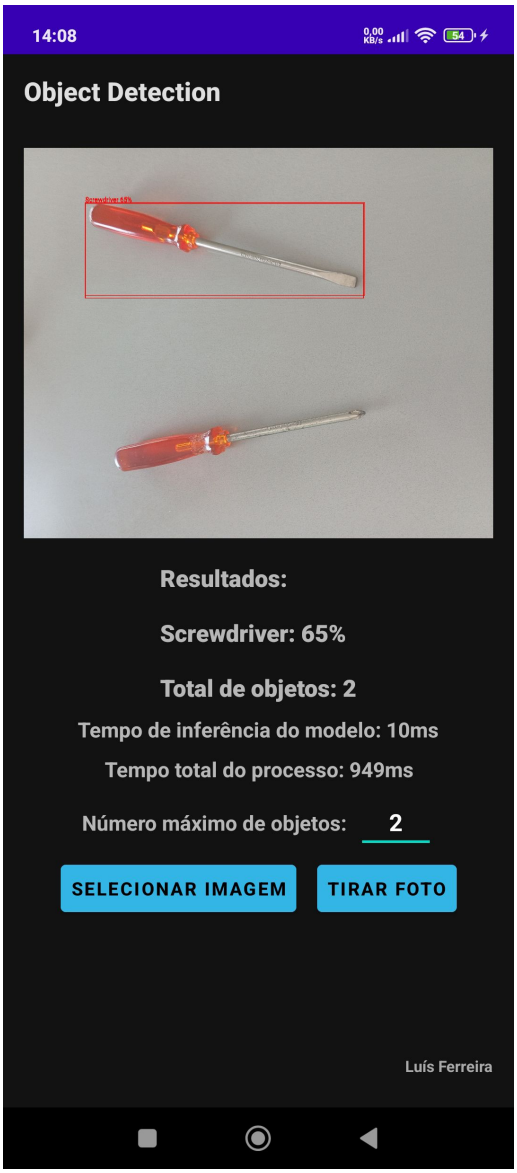

Figure 5.22: Detection of two 'Screwdriver' objects with the maximum detection limit set to 2.

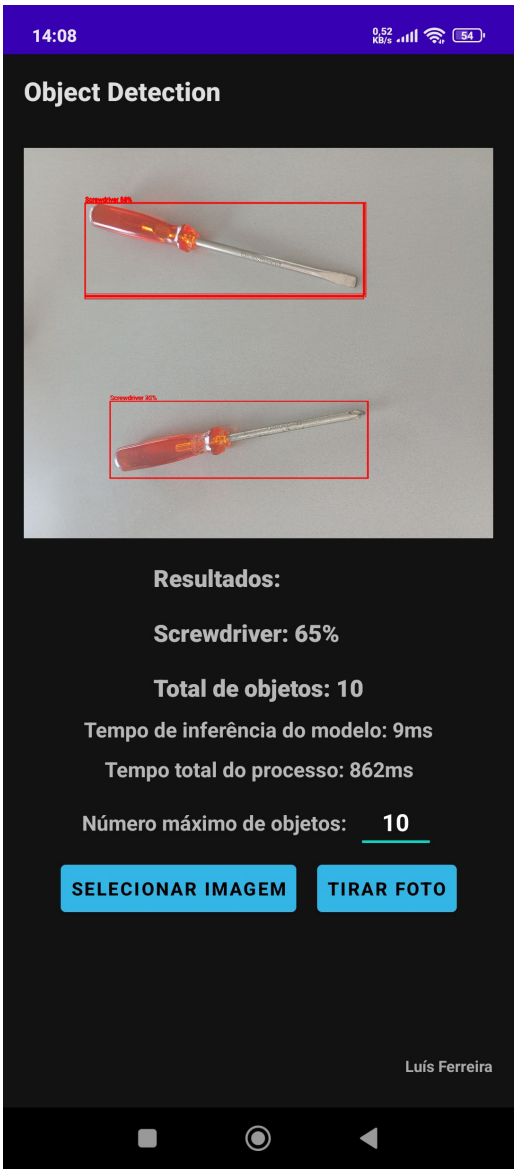

Figure 5.23: Detection of two 'Screwdriver' objects with the maximum detection limit set to 10.

To address such cases, the use of **Non-Maximum Suppression (NMS)** is essential.

Non-Maximum Suppression (NMS) is a post-processing technique used in object detection models to eliminate multiple overlapping bounding boxes that refer to the same object [55]. The algorithm works as follows:

- **Filter by confidence:** Discard detections with confidence scores below a certain threshold to remove weak or uncertain predictions.
- **Sort detections:** Rank the remaining detections in descending order according to their confidence scores.

- **Suppress overlaps:** Starting from the most confident detection, compare it with the others. If the overlap (measured using IoU) with any other detection is greater than a set threshold, the less confident detection is discarded.

This process ensures that only the most reliable and non-overlapping bounding boxes are retained, reducing false positives and making the model's output more accurate.

The result of applying NMS is shown in Figure 5.24, where only the correct and distinct detections remain, successfully removing the redundant overlapping predictions.

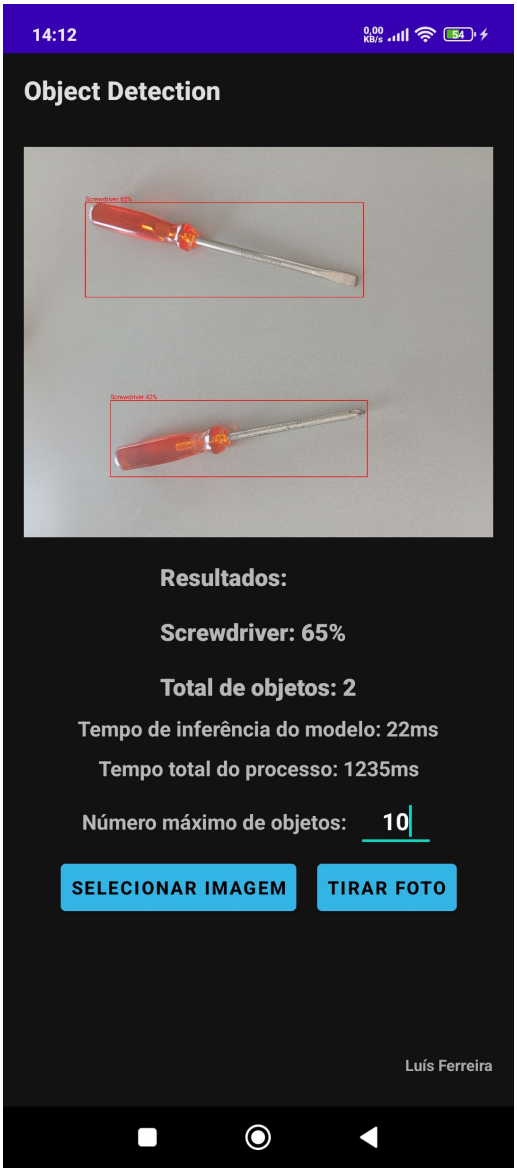

Figure 5.24: Detection of two 'Screwdriver' objects after applying NMS, removing redundant overlapping boxes.

Although NMS is essential for eliminating duplicate detections and improving overall detection quality, it can sometimes negatively impact the detection of closely positioned or overlapping

objects. When two real objects are situated very close to each other, their bounding boxes may have a high IoU value that exceeds the NMS threshold. In such cases, NMS might mistakenly suppress one of the valid detections, resulting in a false negative.

This trade-off requires careful tuning of the IoU threshold: setting a higher threshold allows more overlapping detections to be retained, which helps preserve distinct nearby objects but may increase duplicate detections. Conversely, a lower threshold reduces duplicates but risks suppressing valid detections of adjacent objects.

Therefore, while NMS is crucial for reducing redundant detections, it may also inadvertently remove legitimate detections in scenarios involving overlapping or closely spaced objects.

Figures 5.25 and 5.26 illustrate examples where two objects overlap in the image. In Figure 5.25, two objects of class 'Wrench' are present, but the model fails to detect the overlapping objects correctly, highlighting the difficulty it faces in such scenarios. The detection results vary between identifying only one object, as shown in Figure 5.25, or missing all detections entirely, as depicted in Figure 5.26, where two objects of class 'Screwdriver' overlap.

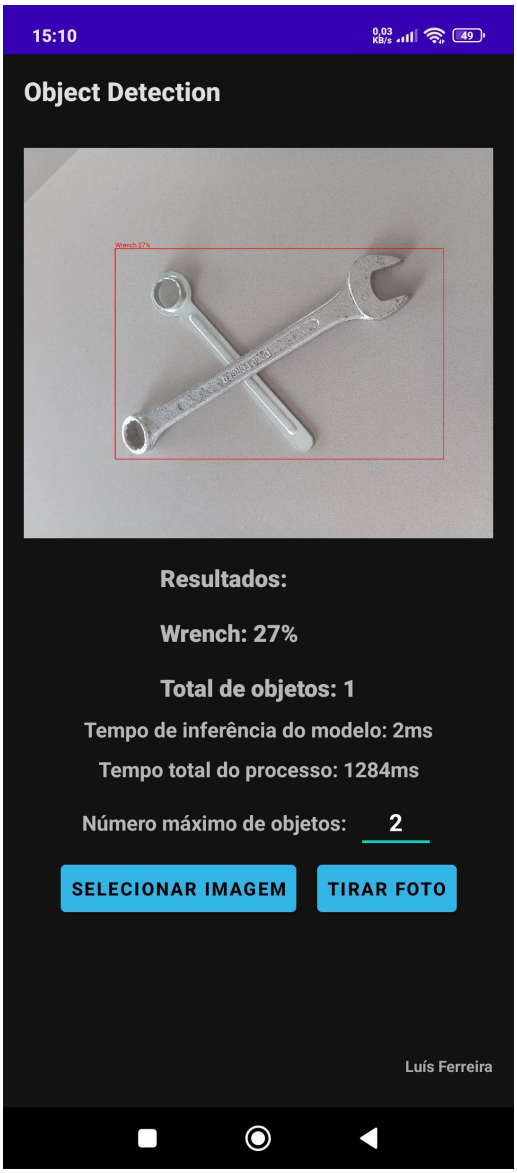

Figure 5.25: Two overlapping objects of class 'Wrench'.

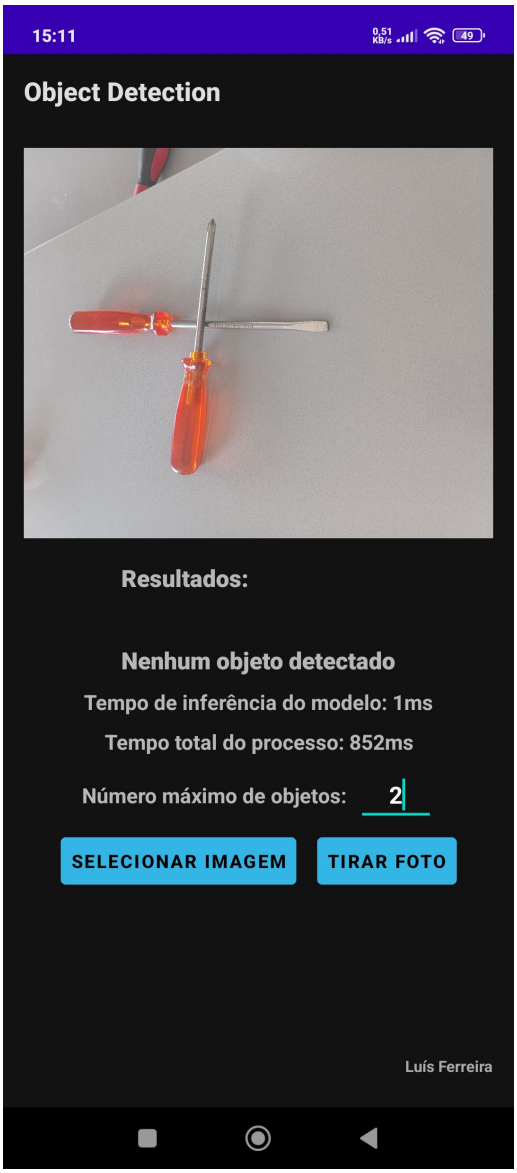

Figure 5.26: Two overlapping objects of class 'Screwdriver'.

Figures 5.27 and 5.28 provide further insight into how viewpoint variation can affect detection in closely positioned objects. Both images contain two 'Hammer' objects placed in close proximity. In Figure 5.27, the model successfully detects both hammers, despite their closeness, due to the favorable angle of view. However, in Figure 5.28, a slight change in the viewing angle results in the model detecting only one of the two objects. This highlights how spatial arrangement and perspective can influence NMS behavior, leading to inconsistent detection outcomes when objects are near each other.

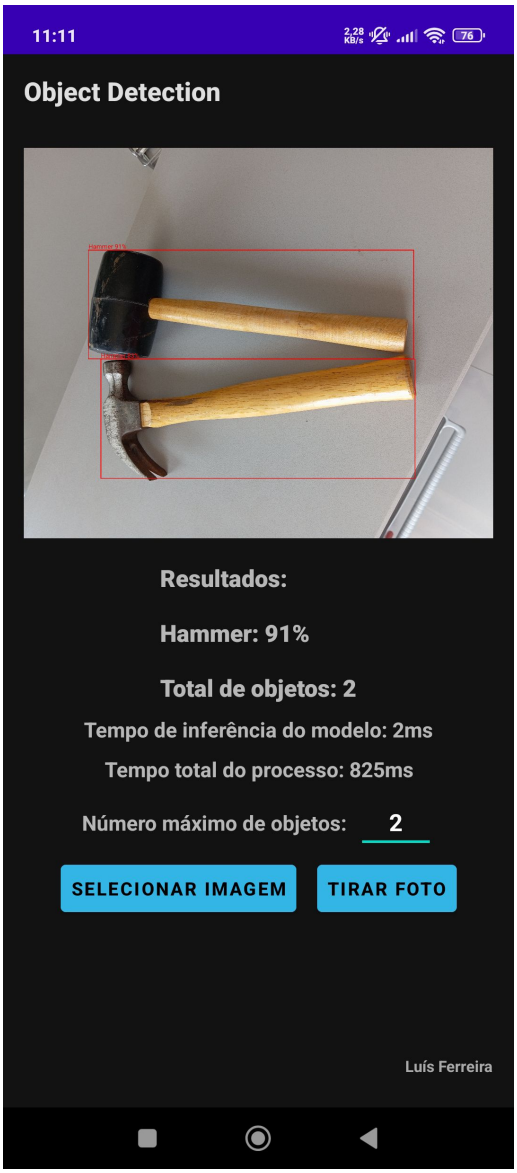

Figure 5.27: Two closely positioned 'Hammer' objects successfully detected.

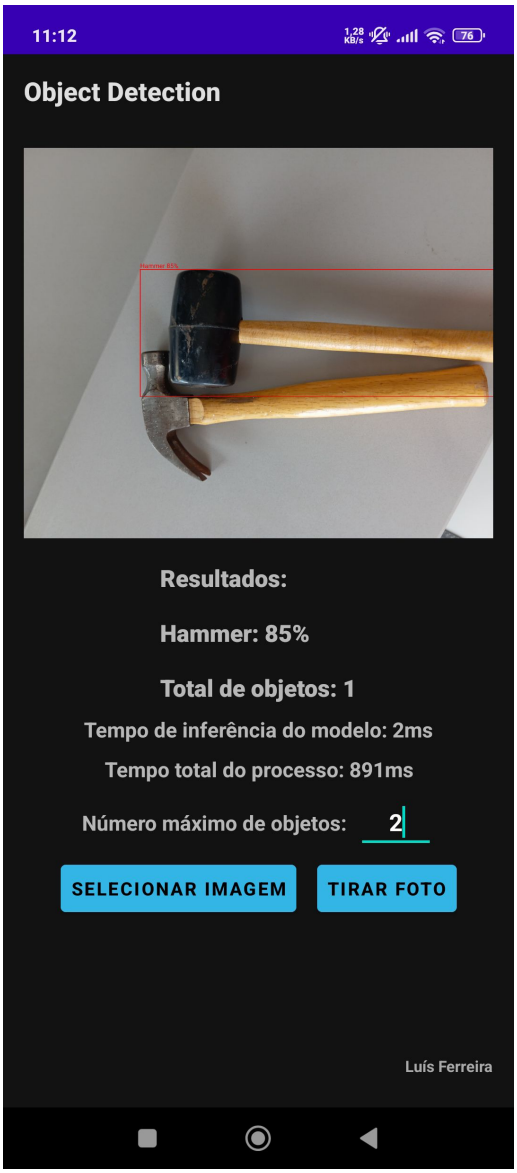

Figure 5.28: Two closely positioned 'Hammer' objects from a slightly different viewpoint.

Figure 5.29 illustrates the model attempting to detect multiple objects present in the same image, totaling three objects: two instances of the class 'Wrench' and one instance of the class 'Screwdriver'. The model successfully detected all three objects, achieving confidence scores of 65% and 70% for the 'Wrench' objects, and 36% for the 'Screwdriver' object.

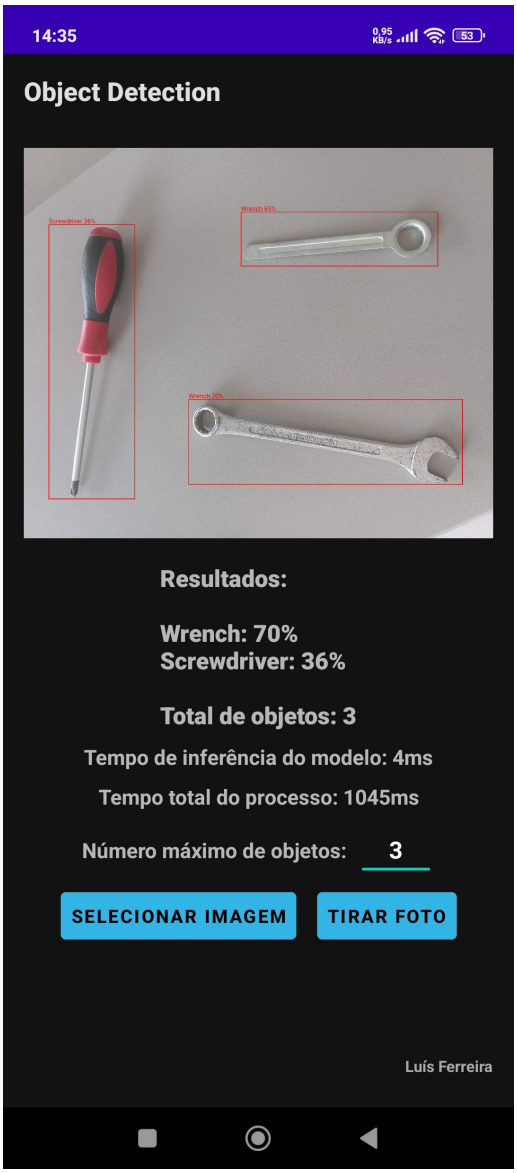

Figure 5.29: Detection of multiple objects in a single image: two 'Wrench' instances and one 'Screwdriver' instance.

Figure 5.30 illustrates the detection of two objects belonging to the 'Pallet' class. While both objects were correctly classified, the localization was only partially accurate, as one of the objects was incorrectly localized.

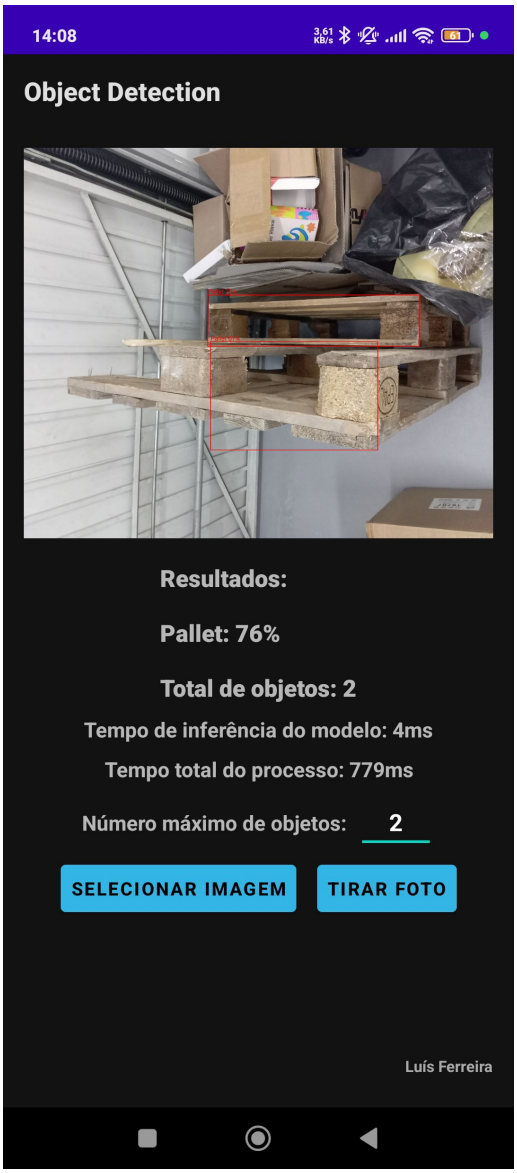

Figure 5.30: Detection of two objects from the 'Pallet' class.

Figures 5.31 and 5.32 illustrate the detection of the same two objects from the 'Pallet' class, positioned identically in both images. The only variable between the two cases is the camera angle from which the images were captured. This comparison highlights that the angle of image capture can significantly impact the model's output, influencing both the accuracy of object localization and the overall detection performance.

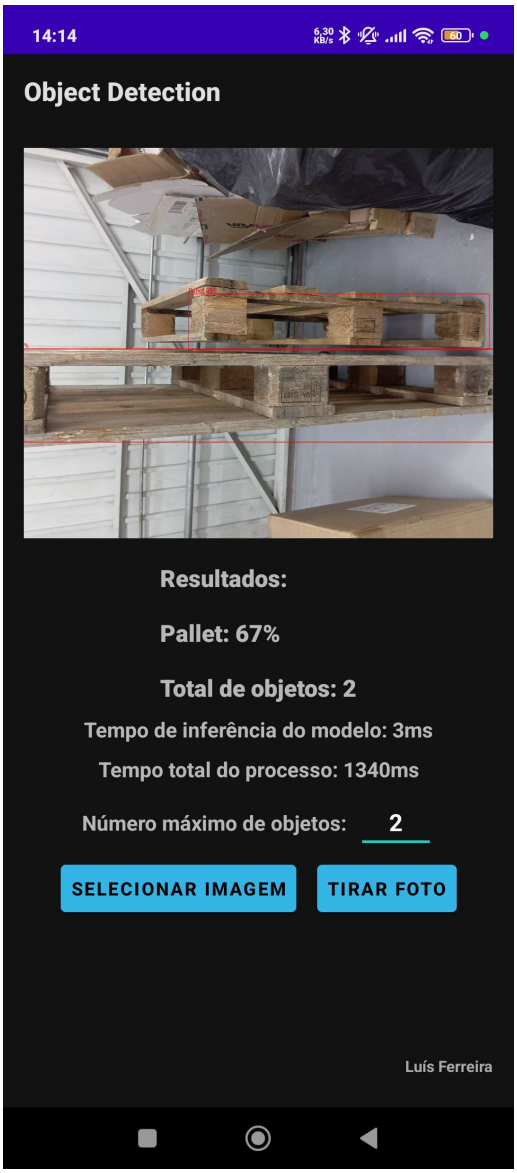

Figure 5.31: Detection of two 'Pallet' class objects captured from camera angle one.

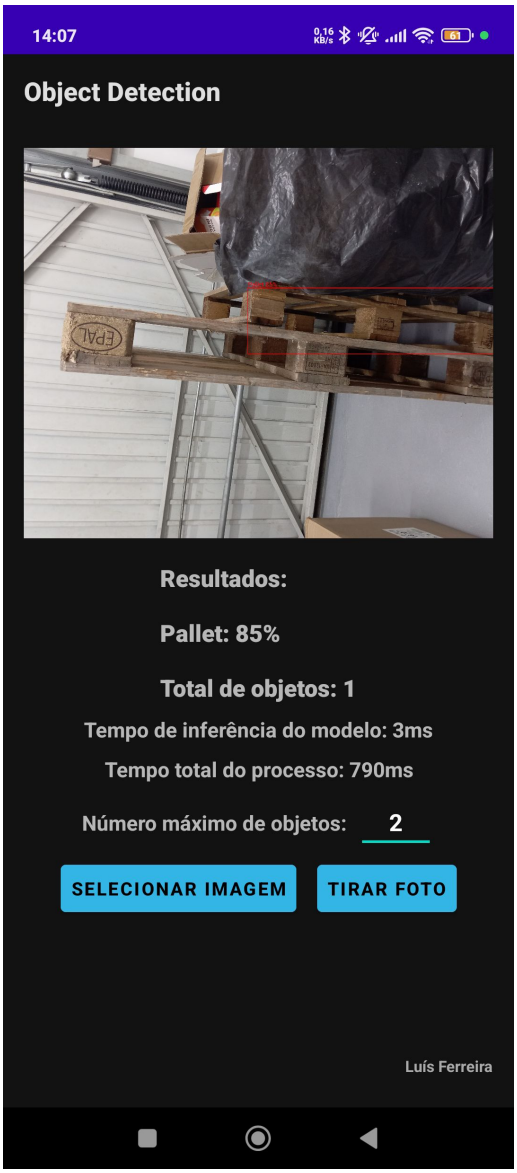

Figure 5.32: Detection of two 'Pallet' class objects captured from camera angle two.

Figures 5.33 and 5.34 present the results of an experiment designed to evaluate how camera movement affects object detection. In Figure 5.33, a slight camera movement was introduced, resulting in the detection of only one of the two 'Pallet' class objects. In Figure 5.34, where the movement was more pronounced, the model also detected only one object, but with lower confidence and poorer localization accuracy. Additionally, a false detection was registered over a large area, incorrectly identifying a non-existent object as belonging to the 'Hammer' class.

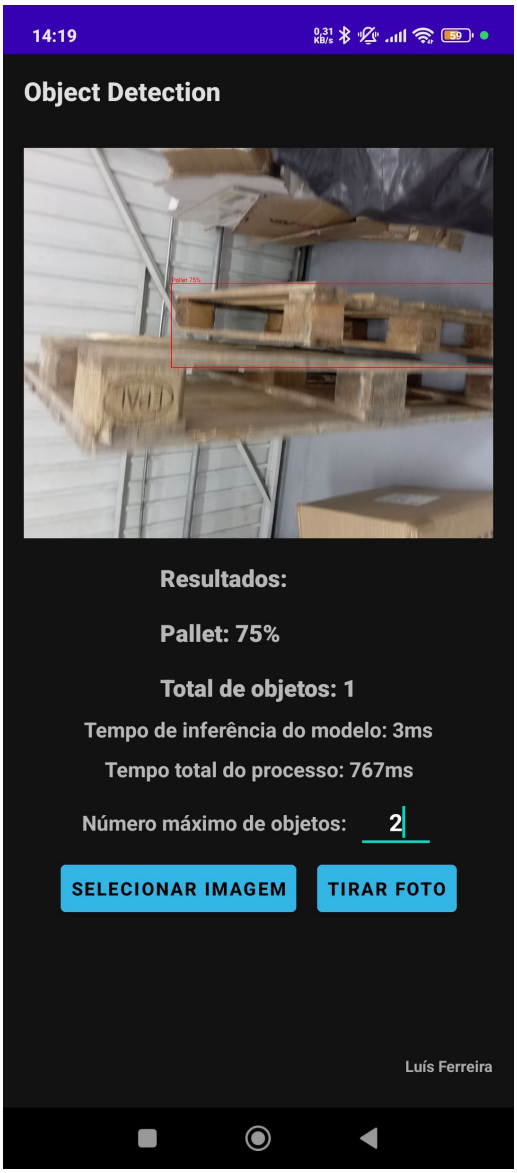

Figure 5.33: Detection under slight camera movement.

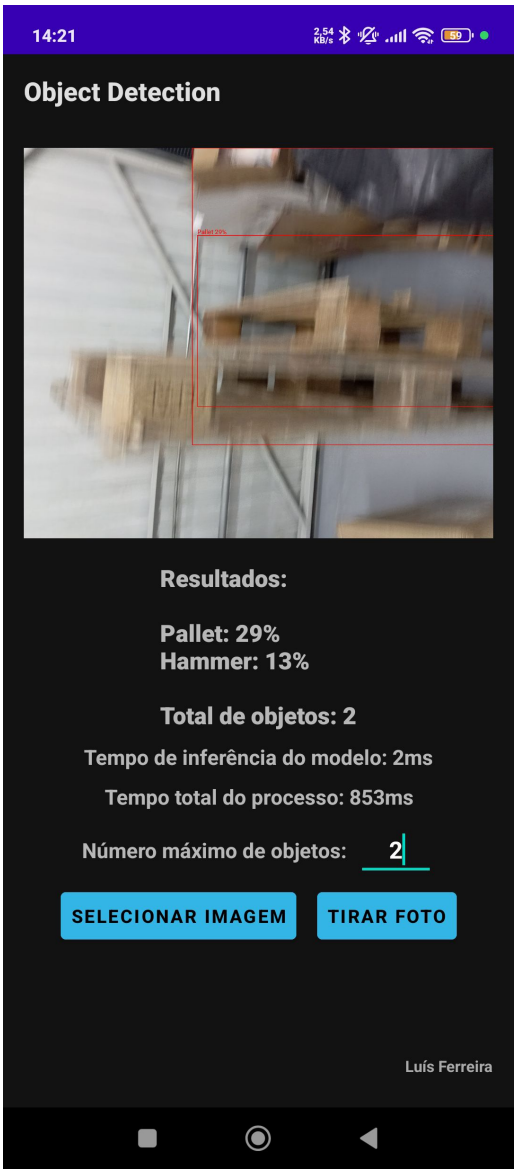

Figure 5.34: Detection under more intense camera movement.

Figures 5.35 and 5.36 present a new test involving occlusion applied to the Pallet class. Unlike the previous results shown in Figures 5.25 and 5.26, the outcomes in this case differ significantly.

In Figure 5.35, a human hand was placed in front of one of the Pallet objects in such a way that it did not cover the upper part but rather created a horizontal occlusion across the middle of the object, visually dividing it into two sections. The model detected only the lower part of the object up to the point where the hand appeared, failing to recognize the continuation of the object above the occlusion. The second pallet object in the image was not detected at all.

In Figure 5.36, a x-ato was placed across one of the pallet object, creating a thinner occlusion than the one produced by the hand in Figure 5.35. Although less obstructive, it still visually divided the object. In this case, the model successfully detected both Pallets objects. However, for the occluded object, the detection was once again incomplete, the model identified only the lower

portion beneath the x-ato, failing to detect the upper part. This behavior is consistent with the result observed in Figure 5.35, where the model struggled to recognize a continuous object when interrupted by a linear visual obstruction.

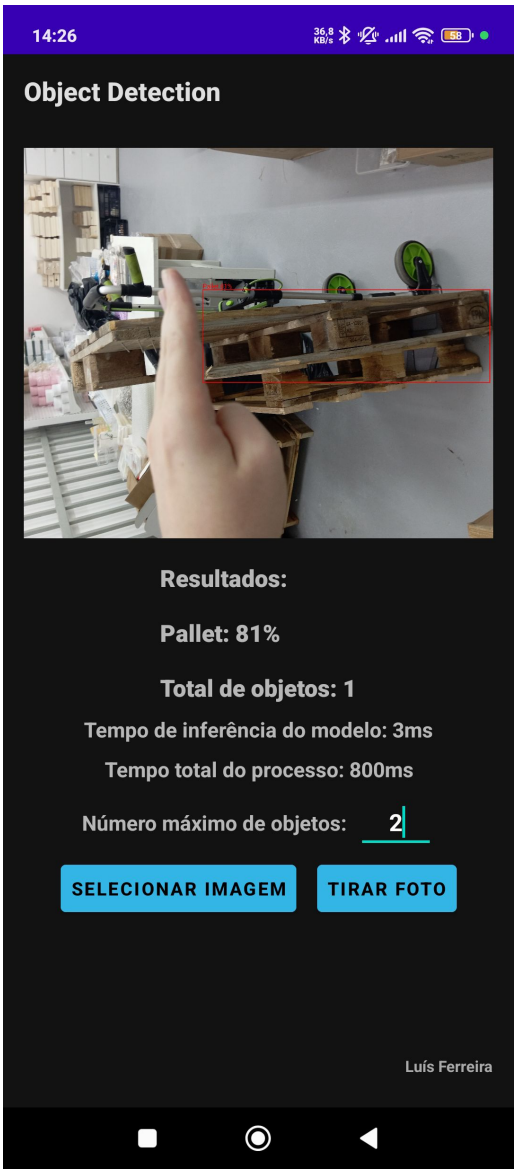

Figure 5.35: Occlusion test with a human hand placed in front of a *Pallet*.

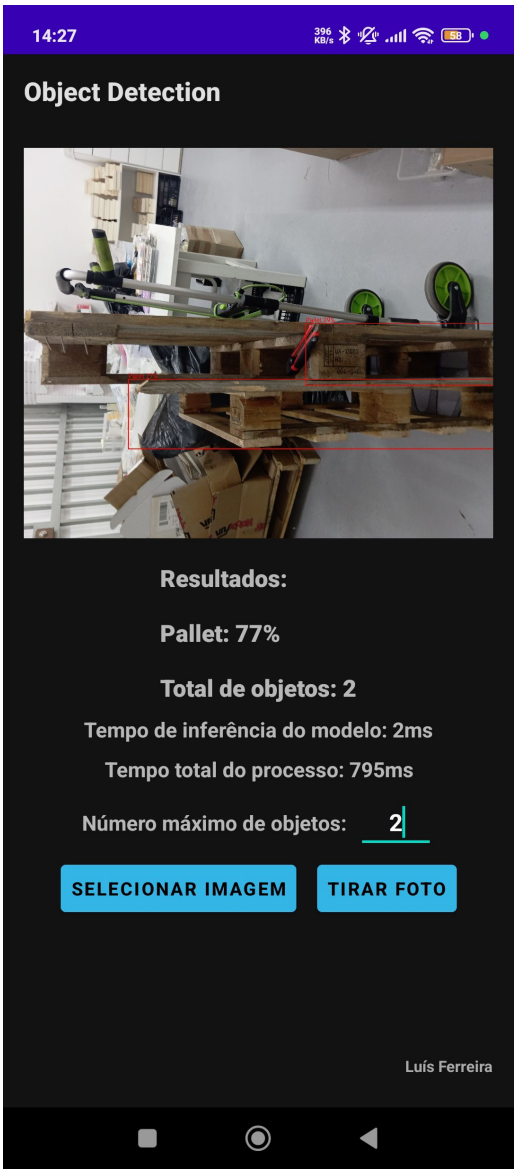

Figure 5.36: Occlusion test with a x-ato placed across a *Pallet*.

5.5 Summary

This chapter presented a comprehensive comparative analysis of the YOLOv11 and YOLOv12 models, focusing on their nano (n) and small (s) variants. The evaluation metrics show that both versions perform closely, with YOLOv11n achieving slightly higher precision (93.02%) and YOLOv12n slightly better mAP50-95 (69.00%). Among the small variants, YOLOv11s attained the highest F1-score (86.62%) while YOLOv12s reached the best mAP50 (87.60%). In terms of inference speed, YOLOv11n remains the fastest model with 1.7 ms, making it particularly suitable for real-time applications on edge devices.

The class-wise precision analysis indicates that all models perform excellently on the most represented classes such as 'Forklift' and 'Pallets', with precision rates around 95% or higher.

However, smaller objects like 'Wrench' and 'Hammer' remain challenging, showing lower precision between 69% and 77%, and a consistent misclassification of 'Wrench' as background across all variants.

Normalized confusion matrices further highlight that YOLOv11 variants generally show lower misclassification rates and demonstrate faster convergence and lower final losses, suggesting more stable training. YOLOv11n's low computational demands reaffirm its suitability for deployment on resource-constrained edge devices.

In practical testing, YOLOv11n was chosen for real-time deployment due to its balance of speed and accuracy. While effective in detecting target objects, classification performance can be affected by image context factors like background complexity and lighting conditions. Despite this, YOLOv11n remains the preferred choice for edge applications given its fast inference and efficiency.

Extensive testing on YOLOv11n revealed several key behaviors during real-world deployment, including repeated detections of the same object, sensitivity to background variation, and misclassification under poor lighting. Additional failure cases were observed in scenarios with overlapping or closely positioned objects, which affected detection precision and bounding box accuracy. These issues were mitigated through post-processing with Non-Maximum Suppression (NMS), which successfully removed redundant overlapping predictions but also introduced trade-offs in detecting closely spaced objects. Tests also showed that the angle from which images were captured influenced detection results, with different viewpoints impacting the model's ability to accurately localize objects. Furthermore, camera movement negatively affected performance, leading to lower confidence scores, poorer localization, and false detections. Such findings underscore the importance of post-processing and dataset diversity when deploying models in uncontrolled environments.

Although YOLOv12 models did not outperform YOLOv11 in this study, they show promising results and potential for future improvement. If equipped with advanced hardware features such as FlashAttention, YOLOv12 could become a strong candidate for industrial object detection in deployment scenarios.

In conclusion, YOLOv11 offers a better trade-off between detection accuracy and computational efficiency, with YOLOv11n standing out as the ideal model for edge-based industrial object detection.

Chapter 6

Conclusion and Future Work

This Chapter presents the conclusions drawn from the development of this Dissertation. Subsequently, it outlines potential directions for future work that may enhance or expand upon the findings presented.

6.1 Conclusion

The theme of this Dissertation focused on efficient object detection at the edge, aiming to develop a robust and accurate model capable of identifying various objects in industrial environments through image analysis. The proposed solution was optimized for deployment on edge devices, striking a balance between detection accuracy and computational efficiency to enable real-time performance in resource-constrained scenarios.

Through the study of the state of the art, it was observed that the YOLO architecture stands out as the most suitable for deployment on low-resource hardware due to its balance between speed and accuracy. Furthermore, this research highlighted three fundamental components for developing an effective edge-based object detection model: the quality and diversity of the dataset, the choice of the model architecture, and the selection and tuning of training hyperparameters. Additionally, the use of comprehensive evaluation metrics proved essential for assessing the model's performance in realistic scenarios. In this dissertation, the YOLOv11 and YOLOv12 models were employed, specifically using their 'n' (nano) and 's' (small) variants, in order to evaluate performance trade-offs between computational efficiency and detection accuracy.Through the evaluation of YOLOv11 and YOLOv12 variants, several key conclusions were drawn.

The results indicate that all models achieved competitive performance across various evaluation metrics. YOLOv11n stood out for having the fastest inference time at just 1.7 milliseconds, making it particularly well-suited for real-time applications. YOLOv11s provided the highest F1-score (86.62%), suggesting a strong balance between precision and recall. Meanwhile, YOLOv12 models showed slightly better performance in terms of mAP50-95, with YOLOv12s reaching 69.66%. Although YOLOv12 incorporates attention mechanisms aimed at improving detection performance, it did not achieve a significant advantage over YOLOv11 in this study.

YOLOv11n model was successfully deployed on an Android device using TensorFlow Lite. The deployment validated the practical feasibility of the model but also revealed certain limitations in real-world conditions, such as inconsistent detections under varied lighting or background scenarios. These results underscore the need for more diverse training data and potential fine-tuning for specific environments to enhance generalization.

In conclusion, this work contributed to the advancement of lightweight and efficient object detection models for industrial applications, particularly on edge devices. It also provided a strong foundation for future research, emphasizing the importance of model choice, dataset quality, and deployment strategies tailored to real-world constraints.

6.2 Future Work

This Dissertation opens several directions for future exploration and improvement. Future work could focus on systematically tuning the model hyperparameters to explore a wider range of configurations and evaluate their impact on detection performance.

Another important aspect concerns the dataset. Expanding the number of images, improving their quality, and ensuring that the backgrounds reflect realistic industrial environments could significantly enhance the models' ability to generalize and perform well on previously unknown data. Such improvements would likely contribute to the development of more robust and reliable detection systems for real-world industrial applications.

In addition to supervised learning, future research could explore unsupervised learning techniques. These approaches may enable models to adapt more flexibly to new or unseen object categories without requiring manual labeling, potentially outperforming the supervised models evaluated in this dissertation in scenarios involving unknown classes.

A further enhancement would be to transition from static image analysis to video-based detection. This would allow the system to identify and track objects in motion, which is more aligned with practical use cases in dynamic industrial environments. Incorporating temporal information could also improve detection accuracy and reduce false positives.

Moreover, while the models were tested on a mobile device in this study, future work could involve deploying the trained models on lower-resource edge devices, such as a Raspberry Pi. This would provide a more rigorous evaluation of the models' performance and real-time capabilities in constrained hardware environments, further validating their applicability for industrial edge-based deployments.

Finally, another promising direction would be to experiment with alternative lightweight model architectures, beyond the YOLO family, such as MobileNet, EfficientDet, or NanoDet. Comparing different architectures could reveal trade-offs in accuracy, speed, and hardware efficiency, and help identify the most suitable model for specific industrial use cases.

References

[1] Android Developers. *Get started with Kotlin on Android*. Accessed: 21-04-2025. n.d. URL: https://developer.android.com/kotlin/first.

[2] Aref Bahi et al. "Object detection in low light environments". In: vol. 246. C. Cited by: 0; All Open Access, Gold Open Access. 2024, pp. 3381–3389. DOI: 10.1016/j.procs.2024.09.219. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85213350029&doi=10.1016%2fj.procs.2024.09.219&partnerID=40&md5=e0b4f8b0bb1aa81b0d33e80be5cd4cad.

[3] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (2020). Cited by: 4513; All Open Access, Green Open Access, pp. 82–115. DOI: 10.1016/j.inffus.2019.12.012. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077515399&doi=10.1016%2fj.inffus.2019.12.012&partnerID=40&md5=720e37936410af916e3efe40346dbeed.

[4] P. Bharat Siva Varma et al. "Enhancing Robust Object Detection in Weather-Impacted Environments using Deep Learning Techniques". In: Cited by: 0. 2024, pp. 599–604. DOI: 10.1109/ICSSAS64001.2024.10760295. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85213342917&doi=10.1109%2fICSSAS64001.2024.10760295&partnerID=40&md5=fea5b173f114ab1c31c42c718b20cc15.

[5] Ahmed Boussihmed et al. "A TinyML model for sidewalk obstacle detection: aiding the blind and visually impaired people". In: *Multimedia Tools and Applications* (2024). Cited by: 0. DOI: 10.1007/s11042-024-20070-9. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85202940434&doi=10.1007%2fs11042-024-20070-9&partnerID=40&md5=9b5f64aead0280456a6589dff08f0bf8.

[6] Yuepeng Cai et al. "Automated marine oil spill detection algorithm based on single-image generative adversarial network and YOLO-v8 under small samples". In: *Marine Pollution Bulletin* 203 (2024). Cited by: 3. DOI: 10.1016/j.marpolbul.2024.116475. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85193443252&doi=10.1016%2fj.marpolbul.2024.116475&partnerID=40&md5=0ce39f2e20aed3b4a6e4b37823b2a48c.

[7] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 213–229. ISBN: 978-3-030-58452-8.

[8] Mahavir Chandaliya et al. "UAV-Based Powerline Problem Inspection and Classification using Machine Learning Approaches". In: Cited by: 0. 2024, pp. 52–59. DOI: 10.1109/BigDataService62917.2024.00014. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85209628366&doi=10.1109%2fBigDataService62917.2024.00014&partnerID=40&md5=76323f7ee775e5aae505874119f8222b.

[9] Jonas Conrad et al. "Deep learning-based error recognition in manual cable assembly using synthetic training data". In: vol. 128. Cited by: 0; All Open Access, Gold Open Access. 2024, pp. 239–244. DOI: 10.1016/j.procir.2024.04.005. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85208802436&doi=10.1016%2fj.procir.2024.04.005&partnerID=40&md5=7572fabb6da2c5e7992fab171736f0

[10] Dissertation. *combined-tools Dataset*. https://universe.roboflow.com/dissertation-lfsrr/combined-tools-kwpxb. Open Source Dataset. visited on 2025-04-03. Feb. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/combined-tools-kwpxb.

[11] Dissertation. *fv Dataset*. https://universe.roboflow.com/dissertation-lfsrr/fv-lklpt-as1qu. Open Source Dataset. visited on 2025-04-03. Feb. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/fv-lklpt-as1qu.

[12] Dissertation. *optobot Dataset*. https://universe.roboflow.com/dissertation-lfsrr/optobot-klcob. Open Source Dataset. visited on 2025-04-03. Mar. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/optobot-klcob.

[13] Dissertation. *pallet detect Dataset*. https://universe.roboflow.com/dissertation-lfsrr/pallet-detect-kr02r-c1wsn. Open Source Dataset. visited on 2025-04-03. Feb. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/pallet-detect-kr02r-c1wsn.

[14] Dissertation. *plier Dataset*. https://universe.roboflow.com/dissertation-lfsrr/plier-gxmpy. Open Source Dataset. visited on 2025-04-03. Mar. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/plier-gxmpy.

[15] Dissertation. *Screw driver detection Dataset*. https://universe.roboflow.com/dissertation-lfsrr/screw-driver-detection-jdj3r. Open Source Dataset. visited on 2025-04-03. Mar. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/screw-driver-detection-jdj3r.

[16] Dissertation. *Wrench Dataset*. https://universe.roboflow.com/dissertation-lfsrr/wrench-qlhgx-fmz4c. Open Source Dataset. visited on 2025-04-03. Mar. 2025. URL: https://universe.roboflow.com/dissertation-lfsrr/wrench-qlhgx-fmz4c.

[17] Niklas Donges. *What Is Transfer Learning? A Guide for Deep Learning*. https://builtin.com/data-science/transfer-learning. Accessed: 2025-05-28. 2024.

[18] Forklift. *Forklift Dataset*. https://universe.roboflow.com/forklift-cotid/forklift-apoxx. Open Source Dataset. visited on 2025-04-03. Sept. 2023. URL: https://universe.roboflow.com/forklift-cotid/forklift-apoxx.

[19] GeeksforGeeks. *Python AI - GeeksforGeeks*. Accessed: 21-04-2025. n.d. URL: https://www.geeksforgeeks.org/python-ai/.

[20] Google. *Colaboratory*. https://colab.google. Accessed: 18-04-2025. n.d.

[21] Ehtesham Hassan, Yasser Khalil, and Imtiaz Ahmad. "Learning Feature Fusion in Deep Learning-Based Object Detector". In: *Journal of Engineering* 2020 (May 2020), pp. 1–11. DOI: 10.1155/2020/7286187.

[22] Mohammadreza Hassanzadehtalouki et al. “Development of a Slug Detection and Localization System for a Pest Control Robot in Organic Horticulture”. In: *Journal of Crop Health* 76.6 (2024). Cited by: 0, pp. 1529–1539. DOI: 10.1007/s10343-024-01031-6. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85205050672&doi=10.1007%2fs10343-024-01031-6&partnerID=40&md5=3dd06f5828cf94736c5e94baa479b7a6.

[23] Bowen Jiao et al. “RS-YOLO: An efficient object detection algorithm for road scenes”. In: *Digital Signal Processing: A Review Journal* 157 (2025). Cited by: 0. DOI: 10.1016/j.dsp.2024.104889. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85210543995&doi=10.1016%2fj.dsp.2024.104889&partnerID=40&md5=5d591146bb77d143f1c29b547bda17a6.

[24] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[25] Sorayus Kittisorayut et al. “A Machine Learning Algorithms for Drug Identification: Enhancing Medication Safety in Thailand”. In: Cited by: 0. 2024, pp. 47–52. DOI: 10.1109/JCSSE61278.2024.10613653. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85201405513&doi=10.1109%2fJCSSE61278.2024.10613653&partnerID=40&md5=9fcd6f26383075ebb0cacbfb5f7d7db8.

[26] Brenden M. Lake et al. “Building machines that learn and think like people”. In: *Behavioral and Brain Sciences* 40 (2017). Cited by: 1358; All Open Access, Green Open Access. DOI: 10.1017/S0140525X16001837. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84996868095&doi=10.1017%2fS0140525X16001837&partnerID=40&md5=832c562ef0aa364c55edd9bee453a867.

[27] Jun Li et al. “A safety wearing helmet detection method using deep leaning approach”. In: *Journal of Optics (India)* 53.2 (2024). Cited by: 4, pp. 1163–1169. DOI: 10.1007/s12596-023-01282-y. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85164174504&doi=10.1007%2fs12596-023-01282-y&partnerID=40&md5=8ce9ef70ee1a0148a239600bdfd94713.

[28] P. Likhitha et al. “SightSpeak Object detection and speech generation for visually challenged people.” In: Cited by: 0. 2024. DOI: 10.1109/ICCCNT61001.2024.10725655. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85212864175&doi=10.1109%2fICCCNT61001.2024.10725655&partnerID=40&md5=1be08854977f7e4372913d40fe200838.

[29] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV]. URL: https://arxiv.org/abs/2103.14030.

[30] Vasyl Lytvyn, Nazar Oleksiv, and Mariia Nazarkevych. “Tanks detection and recognition based on custom dataset model”. In: Cited by: 0. 2024, pp. 583–586. DOI: 10.1109/SIST61555.2024.10629455. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85202806385&doi=10.1109%2fSIST61555.2024.10629455&partnerID=40&md5=ab2558691e7f80f8fda165c59964d1e0.

[31] K. Mallikharjuna Rao et al. “Computer Vision-Based Self-inflicted Violence Detection in High-Rise Environments Using Deep Learning”. In: *Lecture Notes in Electrical Engineering* 1247 LNEE (2024). Cited by: 0, pp. 69–83. DOI: 10.1007/978-981-97-6714-4_6. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85212931431&doi=10.1007%2f978-981-97-6714-4_6&partnerID=40&md5=5b5c4bc35c85d94484f39446504b080c.

[32] Anke Meyer-Baese and Volker Schmid. "Chapter 7 - Foundations of Neural Networks". In: *Pattern Recognition and Signal Analysis in Medical Imaging (Second Edition)*. Ed. by Anke Meyer-Baese and Volker Schmid. Second Edition. Oxford: Academic Press, 2014, pp. 197–243. ISBN: 978-0-12-409545-8. DOI: https://doi.org/10.1016/B978-0-12-409545-8.00007-8. URL: https://www.sciencedirect.com/science/article/pii/B9780124095458000078.

[33] New Horizons. *Learn AI with Python*. Accessed: 21-04-2025. n.d. URL: https://www.newhorizons.com/resources/blog/learn-ai-with-python.

[34] Dung Nguyen, Van-Dung Hoang, and Van-Tuong-Lan Le. "V-DETR: Pure Transformer for End-to-End Object Detection". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 14796 LNAI (2024). Cited by: 0, pp. 120–131. DOI: 10.1007/978-981-97-4985-0_10. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85210075029&doi=10.1007%2f978-981-97-4985-0_10&partnerID=40&md5=c78cc79fd1d82e80bbac46cc76a46b5a.

[35] Rafael Padilla, Sergio Netto, and Eduardo da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms". In: July 2020. DOI: 10.1109/IWSSIP48289.2020.

[36] G.K. Patra et al. "Chapter 6 - Deep learning methods for scientific and industrial research". In: *Deep Learning*. Ed. by Venu Govindaraju, Arni S.R. Srinivasa Rao, and C.R. Rao. Vol. 48. Handbook of Statistics. Elsevier, 2023, pp. 107–168. DOI: https://doi.org/10.1016/bs.host.2022.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0169716122000578.

[37] Sergio A. Pérez-Zarate et al. "Automated Car Damage Assessment Using Computer Vision: Insurance Company Use Case". In: *Applied Sciences (Switzerland)* 14.20 (2024). Cited by: 0; All Open Access, Gold Open Access. DOI: 10.3390/app14209560. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85207332150&doi=10.3390%2fapp14209560&partnerID=40&md5=8336727a5c721328b449656b9a7a857f.

[38] David M. W. Powers. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. 2020. arXiv: 2010.16061 [cs.LG]. URL: https://arxiv.org/abs/2010.16061.

[39] Akshay Prakash, Ashin Babu, and A.G. Hari Narayanan. "Novel Approaches to Automatic License Plate Reading (ALPR)". In: *Signals and Communication Technology* Part F2556 (2024). Cited by: 0, pp. 413–421. DOI: 10.1007/978-3-031-47942-7_35. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85191443040&doi=10.1007%2f978-3-031-47942-7_35&partnerID=40&md5=29d350fb2bd2b068d0a0f44bae8f987c.

[40] Andrea Pretto et al. "A novel low-cost visual ear tag based identification system for precision beef cattle livestock farming". In: *Information Processing in Agriculture* 11.1 (2024). Cited by: 6; All Open Access, Gold Open Access, Green Open Access, pp. 117–126. DOI: 10.1016/j.inpa.2022.10.003. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85149851456&doi=10.1016%2fj.inpa.2022.10.003&partnerID=40&md5=369c84e424e40d6831926dfe0a357b4f.

[41] Hamed Raoofi et al. "Deep Learning Method to Detect Missing Welds for Joist Assembly Line". In: *Applied System Innovation* 7.1 (2024). Cited by: 1; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3390/asi7010016. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85185911281&doi=10.3390%2fasi7010016&partnerID=40&md5=a2f10b469dfd5b294a6f54b7de17ac5f.

[42] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[43] Hamid Rezatofighi et al. "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 658–666. DOI: 10.1109/CVPR.2019.00075.

[44] Ranjan Sapkota et al. "Immature Green Apple Detection and Sizing in Commercial Orchards Using YOLOv8 and Shape Fitting Techniques". In: *IEEE Access* 12 (2024). Cited by: 10; All Open Access, Gold Open Access, Green Open Access, pp. 43436–43452. DOI: 10.1109/ACCESS.2024.3378261. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85188461432&doi=10.1109%2fACCESS.2024.3378261&partnerID=40&md5=993aa7ee1eece869cc741a7b76667f17.

[45] Serokell. *A Guide to F1 Score*. Accessed: 2024-12-30. 2024. URL: https://serokell.io/blog/a-guide-to-f1-score.

[46] Mohammad Shahin et al. "Deploying Computer-Based Vision to Enhance Safety in Industrial Environment". In: *Lecture Notes in Mechanical Engineering* (2024). Cited by: 5, pp. 503–509. DOI: 10.1007/978-3-031-38165-2_59. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85172727919&doi=10.1007%2f978-3-031-38165-2_59&partnerID=40&md5=cb85a5101da8826493ff8bbebcd3f8

[47] Osvaldo Simeone. "A Very Brief Introduction to Machine Learning with Applications to Communication Systems". In: *IEEE Transactions on Cognitive Communications and Networking* 4.4 (2018). Cited by: 355; All Open Access, Bronze Open Access, pp. 648–664. DOI: 10.1109/TCCN.2018.2881442. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85057154880&doi=10.1109%2fTCCN.2018.2881442&partnerID=40&md5=78a0a1e845749e0993ccc7a4334b0579.

[48] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. ISBN: 049508252X.

[49] Jia He Tan and Ching Pang Goh. "Enhancing Child Safety: Computer Vision-Based Accident Detection for Infants and Toddlers". In: Cited by: 0. 2024, pp. 179–183. DOI: 10.1109/ICDXA61007.2024.10470712. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85190517804&doi=10.1109%2fICDXA61007.2024.10470712&partnerID=40&md5=9d8f2ab0d5a4534dea87fc20633f5b1e.

[50] Yasmeen Tandon, Brian Bartholmai, and Chi Wan Koo. "Putting artificial intelligence (AI) on the spot: Machine learning evaluation of pulmonary nodules". In: *Journal of Thoracic Disease* 12 (Nov. 2020), pp. 6954–6965. DOI: 10.21037/jtd-2019-cptn-03.

[51] K.L. Thara et al. "Deep Learning Techniques for Real-Time Object Detection and Classification". In: *Smart Innovation, Systems and Technologies* 392 SIST (2024). Cited by: 0, pp. 307–318. DOI: 10.1007/978-981-97-3690-4_23. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85206369899&doi=10.1007%2f978-981-97-3690-4_23&partnerID=40&md5=80646610fc487dcdf4b65130bf0519

[52] Yunjie Tian, Qixiang Ye, and David Doermann. *YOLOv12: Attention-Centric Real-Time Object Detectors*. 2025. URL: https://github.com/sunsmarterjie/yolov12.

[53] Yunjie Tian, Qixiang Ye, and David Doermann. "YOLOv12: Attention-Centric Real-Time Object Detectors". In: *arXiv preprint arXiv:2502.12524* (2025).

[54] *Title*. Lecture Notes of Computer Vision course, Faculty of Engineering of University of Porto, 2024.

[55] Ultralytics. *Non-Maximum Suppression (NMS) - Ultralytics Glossary*. https://www.ultralytics.com/pt/glossary/non-maximum-suppression-nms. Accessed: 2025-06-03. 2024.

[56] Ultralytics. *What is DFL Loss in YOLOv8?* Accessed: 20-04-2015. 2024. URL: https://yolov8.org/what-is-dfl-loss-in-yolov8/.

[57] Esteban Uri. *pub-yolo-android: Minimalist way to integrate YOLO in Android*. GitHub repository. Accessed on June 2, 2025. 2025. URL: https://github.com/estebanuri/pub-yolo-android.

[58] Abhimanyu Verma et al. "Humerus Bone Fracture Detection Utilizing YOLOv4 Algorithm: A Deep Learning Approach". In: Cited by: 0. 2024, pp. 1191–1196. DOI: 10.1109/ICDT61202.2024.10489429. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85191241665&doi=10.1109%2fICDT61202.2024.10489429&partnerID=40&md5=325087a3b925f471d9c9d183cfe68a6a.

[59] Zhengyu Wang et al. "Multi-category sorting of plastic waste using Swin Transformer: A vision-based approach". In: *Journal of Environmental Management* 370 (2024). Cited by: 0. DOI: 10.1016/j.jenvman.2024.122742. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85205700632&doi=10.1016%2fj.jenvman.2024.122742&partnerID=40&md5=d5dceb98e873501e80c3e157bb06e2e6.

[60] Wikipedia contributors. *Transfer learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-May-2025]. 2025. URL: https://en.wikipedia.org/wiki/Transfer_learning.

[61] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[62] Zhiyuan Yang, Yuanyuan Shen, and Yanfei Shen. "Football referee gesture recognition algorithm based on YOLOv8s". In: *Frontiers in Computational Neuroscience* 18 (2024). Cited by: 2; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3389/fncom.2024.1341234. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85186465940&doi=10.3389%2ffncom.2024.1341234&partnerID=40&md5=802bb40bf8499043363f73633f83c5a3.

[63] Ming Zhang et al. "Detection of Mulberry Leaf Diseases in Natural Environments Based on Improved YOLOv8". In: *Forests* 15.7 (2024). Cited by: 0; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3390/f15071188. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85199592732&doi=10.3390%2ff15071188&partnerID=40&md5=d1a34693b54ac5bf5b3a512dcbbc7bea.

[64] Xizhou Zhu et al. "Deformable DETR: Deformable transformers for end-to-end object detection". In: *arXiv preprint arXiv:2010.04159* (2020). Vision-DETR builds upon the transformer-based object detection framework introduced here.