algorithms

MDPI

Article

Edge AI for Industrial Visual Inspection: YOLOv8-Based Visual Conformity Detection Using Raspberry Pi

Marcelo T. Okano *, William Aparecido Celestino Lopes, Sergio Miele Ruggero, Oduvaldo Vendrametto and João Carlos Lopes Fernandes

Graduate Program in Production Engineering, Paulista University UNIP, São Paulo 04026-002, SP, Brazil; william.lopes12@aluno.unip.br (W.A.C.L.); smruggero@uol.com.br (S.M.R.); oduvaldove@gmail.com (O.V.); jlopesf@uol.com.br (J.C.L.F.)

* Correspondence: marcelo.okano1@docente.unip.br; Tel.: +55-11991096575

Abstract

This paper presents a lightweight and cost-effective computer vision solution for automated industrial inspection using You Only Look Once (YOLO) v8 models deployed on embedded systems. The YOLOv8 Nano model, trained for 200 epochs, achieved a precision of 0.932, an mAP@0.5 of 0.938, and an F1-score of 0.914, with an average inference time of ~470 ms on a Raspberry Pi 500, confirming its feasibility for real-time edge applications. The proposed system aims to replace physical jigs used for the dimensional verification of extruded polyamide tubes in the automotive sector. The YOLOv8 Nano and YOLOv8 Small models were trained on a Graphics Processing Unit (GPU) workstation and subsequently tested on a Central Processing Unit (CPU)-only Raspberry Pi 500 to evaluate their performance in constrained environments. The experimental results show that the Small model achieved higher accuracy (a precision of 0.951 and an mAP@0.5 of 0.941) but required a significantly longer inference time (~1315 ms), while the Nano model achieved faster execution (~470 ms) with stable metrics (precision of 0.932 and mAP@0.5 of 0.938), therefore making it more suitable for real-time applications. The system was validated using authentic images in an industrial setting, confirming its feasibility for edge artificial intelligence (AI) scenarios. These findings reinforce the feasibility of embedded AI in smart manufacturing, demonstrating that compact models can deliver reliable performance without requiring high-end computing infrastructure.

Keywords: YOLOv8; computer vision; Industry 4.0; Raspberry Pi; automated inspection; GPU

Academic Editor: Noman Khan

Received: 15 July 2025
Revised: 4 August 2025
Accepted: 7 August 2025
Published: 14 August 2025

Citation: Okano, M.T.; Lopes, W.A.C.; Ruggero, S.M.; Vendrametto, O.; Fernandes, J.C.L. Edge AI for Industrial Visual Inspection: YOLOv8-Based Visual Conformity Detection Using Raspberry Pi. *Algorithms* **2025**, *18*, 510. https://doi.org/10.3390/a18080510

1. Introduction

Digital transformation in the automotive industry has driven the adoption of emerging technologies aimed at increasing operational efficiency, product quality, and global competitiveness. One of the recurring challenges in this sector is ensuring that all manufactured components fully comply with stringent technical specifications. Among these components, fluid lines play a crucial role in vehicle systems, necessitating highly accurate visual inspection processes to ensure their integrity, functionality, and compliance with international standards.

Traditionally, the inspection of these pipeline assemblies is performed manually, a process subject to human variability, limited scale, and increased operating costs. In addition, the increasing complexity of products and the diversity of cable models make it

impractical to maintain purely visual or manual inspections on production lines operating under lean manufacturing and just-in-time production principles. In this context, the need arises for automated, intelligent, and adaptable solutions that can operate in real-time with high precision and low computational cost.

Computer vision, based on convolutional neural networks (CNNs), has established itself as one of the most promising technologies for automating industrial visual inspection. Specifically, the You Only Look Once (YOLO) family of algorithms offers an object detection approach that balances performance, accuracy, and speed and is widely adopted in scenarios that demand fast and precise responses. With the release of YOLOv8, new possibilities have opened up for industrial applications, especially with the YOLOv8 Nano (n) and YOLOv8 Small (s) variants, which feature architectures optimized for devices with limited computing power.

This paper proposes the development and application of a YOLOv8-based solution for the automated inspection of a set of pipes manufactured by an auto parts company, focusing on visual compliance identification. This study involves two main steps: (i) the training and validation of the YOLOv8n and YOLOv8s models using a Microsoft Windows 10 workstation (Redmond, WA, USA), Anaconda 3, and an NVIDIA RTX 4060 GPU (Santa Clara, CA, USA), and (ii) the implementation of the selected model on a Raspberry Pi 500 (Cambridge, UK), a low-cost and low-power device, which is responsible for real-time data inference on the production line.

The study is conducted following the methodological rigor of Design Science Research (DSR), which provides a structured framework for developing technological artifacts with a high degree of practical relevance and scientific rigor. DSR enables iterating between the phases of conception, development, demonstration, and evaluation, ensuring that the solution meets both industrial requirements and the principles of technological innovation.

Additionally, this paper discusses in detail the importance of hardware architecture for the successful implementation of AI-based solutions, highlighting the fundamental differences between CPUs and GPUs in the model training and inference processes. The RTX 4060 GPU (Santa Clara, CA, USA), for example, offers massively parallel performance that accelerates neural network training. At the same time, the Raspberry Pi 500 CPU (Cambridge, UK) poses challenges related to model optimization for efficient execution in a production environment.

With this approach, we hope not only to provide a practical and replicable solution for the automated inspection of fluid pipelines but also to contribute to the body of knowledge on the application of AI in restricted industrial environments, demonstrating how to overcome technical and operational limitations using robust methodologies and appropriate computational architectures. The relevance of this study extends to the possibility of adapting the solution to other visual inspection contexts, reinforcing its potential impact on the evolution of Industry 4.0.

Section 2 reviews related studies on visual inspection using deep learning and YOLO models in industrial contexts. Section 3 details the materials and methods, including the theoretical background, hardware and software architecture, dataset preparation, and model training. Section 4 presents and discusses the experimental results, including inference performance and validation on embedded systems. Finally, Section 5 concludes by summarizing the main findings, limitations, and directions for future research.

2. Related Works

Numerous studies have investigated the application of deep learning in industrial visual inspection, especially within the automotive sector. These studies can be grouped into three main categories: (i) traditional deep learning-based defect detection using YOLO,

(ii) recent enhancements to YOLO architectures, and (iii) embedded AI applications on edge devices.

In the first category, Yang et al. [1] applied a YOLO-based model to detect weld defects in steel pipes, achieving better performance than traditional architectures like Faster R-CNN. Their approach, although effective in terms of accuracy and inference speed, remained GPU-bound and did not explore edge deployment. Similarly, Mazzetto et al. [2] validated CNN-based real-time defect detection on an automotive assembly line, focusing on system integration. However, their study did not address deployment constraints typical of low-power hardware. Qi et al. [3] also proposed a general defect classification pipeline using YOLO, reinforcing the relevance of object detection models for non-conformity classification in manufacturing but without experimentation beyond server environments.

The second group involves more recent YOLO variants and architectural innovations. Huang et al. [4] proposed a YOLOv7 model enhanced with ECA attention and Alpha-IoU loss, improving precision while maintaining real-time capability. Although technically refined, the model remained computationally intensive. Zhang et al. [5] introduced PDS-YOLO, a lightweight model optimized for real-time detection on a Jetson Nano, addressing power and size constraints more directly. Nevertheless, even these studies focused on pipeline defects or infrastructure inspection and did not evaluate performance on CPU-only systems such as the Raspberry Pi (Cambridge, UK).

Despite these valuable contributions, to our knowledge, no prior study has implemented YOLOv8 for visual conformity detection of extruded polyamide tubes using a CPU-only Raspberry Pi. This gap is particularly relevant given the increasing demand for edge AI solutions in industrial quality control. Our study endeavors to fill this gap by validating the real-time feasibility of YOLOv8 Nano and Small models on low-power hardware, using an original dataset derived from an authentic automotive assembly process. We also demonstrate how accurate inference can be achieved with sub-500 ms latency, offering a practical and replicable alternative to GPU-reliant approaches.

3. Materials and Methods

3.1. Theoretical Background

3.1.1. Computer Vision and Convolutional Neural Networks in Industry 4.0

According to several recent studies, the widespread use of devices with cameras, such as cell phones, has significantly contributed to the exponential increase in daily image and video production and sharing. This scenario presents new challenges to the field of computer vision, requiring models to learn and adapt continuously and sequentially during the inference process, as completely reconfiguring models for each new batch of data would be a costly and inefficient process [6].

Computer vision, one of the main branches of artificial intelligence, has experienced accelerated advancement in recent years, driven by the rapid progress of deep learning techniques [7] and involving image classification, semantic segmentation, object detection, and super-resolution image reconstruction, with the rapid development of deep convolutional neural networks (CNNs) [6].

Convolutional neural networks (CNNs) stand out for their advanced capabilities for autonomous learning and information representation, enabling the efficient extraction of features directly from input data through the training of models aligned with practical applications. With the rapid advancement of deep learning techniques, CNN architectures have become increasingly sophisticated and varied, leading to the gradual replacement of conventional machine learning methods [6].

Over the years, convolutional neural networks (CNNs) have established themselves as a powerful and widely used approach in various computer vision applications, particularly in tasks such as image classification and object detection [8].

Figure 1 illustrates the typical architecture of a convolutional neural network (CNN), highlighting the main components involved in feature extraction and classification. This visual aid complements the following description of each component layer by layer.

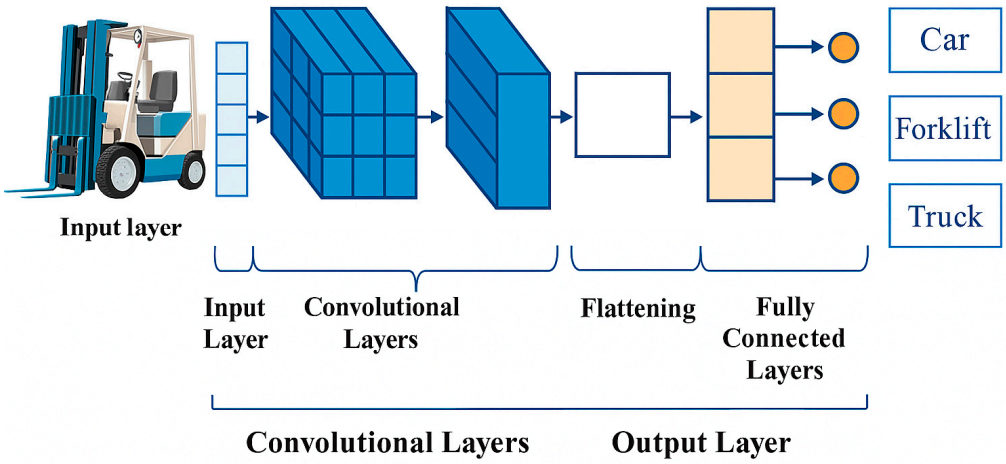

Figure 1. Typical CNN architecture for image classification.

According to [8], the main parts of a CNN are as follows:

- Convolution Layer: Uses kernels or filters to extract features from the input image. The kernel performs dot product operations on subregions of the image, producing a feature map. The displacement of the kernel is controlled by a parameter called stride, and the use of padding can preserve the original image's dimensions [8].
- Pooling Layer: Responsible for reducing the dimensionality of feature maps while retaining the most relevant information. The most common operations are max pooling (selects the most significant value in the region) and average pooling (calculates the average) [8].
- Flattening Layer: Converts the feature maps (two-dimensional) into a one-dimensional vector, which allows connecting the information to the dense layers (fully connected) to perform the final classification [8].
- Dense Layers (Fully Connected Layers): Perform the decision process based on the extracted and flattened features. These layers connect all the neurons in the previous layer to all those in the next layer, as in a traditional neural network [8].
- Activation Function: After each convolution or densely connected operation, activation functions such as the Rectified Linear Unit (ReLU) are applied to introduce non-linearities, thereby enhancing the network's modeling capability [8].
- In industry, CNNs are applied for fulfilling various functions such as the following:
- Automated Inspection of Surface Defects: CNNs are widely used to detect surface defects on industrial materials, including steel, fabrics, ceramics, photovoltaic panels, magnetic tiles, and LCD screens. These networks can extract relevant visual features directly from images captured during production, enabling fast and accurate decisions about product quality [9].
- Vision Systems for Quality Control: The combination of deep learning and computer vision has revolutionized industrial inspection systems, replacing manual methods and simple automation. This results in improved real-time quality control, with the ability to adapt to various lighting conditions and product pattern variation, for exam-

ple, the visual inspection of automotive surfaces, electronics, or textile items through systems with embedded cameras and CNNs trained to recognize anomalies [10].

- Integration with Industry 4.0: CNNs are considered essential technologies within the Industry 4.0 ecosystem, as they support intelligent automation, predictive maintenance, real-time process control, and customized manufacturing. With visual data collection and real-time analysis, CNNs facilitate decentralized and efficient decision-making. Applications in embedded devices, edge computing, and cyber–physical systems allow the use of CNNs even in industrial environments with limited computational resources [9].
- Visual Inspection in the Automotive Industry: In the automotive sector, CNNs are applied at several stages of production, from inspecting raw materials to verifying the final vehicle. They detect tiny defects, such as corrosion, paint flaws, cracks, or structural deformations, that escape human observation [10].
- Automated Inspection with CNNs in the Industrial Life Cycle: CNNs play a crucial role in various phases of the product life cycle, including visual inspection, assembly, process control, and logistics. The architecture of industrial vision systems integrates sensing modules, vision algorithms, and automated decision-making, optimizing production [7].
- Continuous Improvement in Complex Manufacturing Environments: With the advent of Industry 4.0, CNNs enable adaptive visual inspections that dynamically respond to changes in production processes. This significantly reduces response time and rework costs, promoting intelligent manufacturing based on visual data, and includes real-time inspections with embedded and edge sensor computing [10].
- Monitoring and Diagnostics in Critical Infrastructure Environments: CNNs are also applied in the automated inspection of overhead contact lines (OCLs) used in railways. Such applications demonstrate the potential of computer vision in highly critical industrial sectors and complex infrastructure [11].

3.1.2. YOLOv8 Architecture: Nano vs. Small

The You Only Look Once (YOLO) architecture is an innovative approach to real-time object detection, where the task is reformulated as a single regression problem that simultaneously predicts the classes and locations of objects in the input image using bounding boxes. Since its introduction by [12], the YOLO family of algorithms has undergone successive evolutions, culminating in the YOLOv8 version, released in 2023, which incorporates significant advances in accuracy, robustness, and real-time execution capacity with low computational cost [8].

According to [8], YOLOv8 maintains the unified detection philosophy that characterized previous versions, but with improvements such as the replacement of backbone components with lighter and more efficient architectures, such as C2f and CSPDarknet, in addition to improvements in the detection header (detection head), which is now anchor-free, making it easier to adapt to different datasets and resolutions. These modifications make YOLOv8 especially suitable for industrial applications such as surface defect inspection, smart manufacturing, and embedded devices in Industry 4.0 [8].

Furthermore, recent studies have highlighted that YOLOv8 offers remarkable performance in terms of precision, recall, and Mean Average Precision (mAP), outperforming traditional algorithms such as SSD, EfficientDet, and previous versions of the YOLO family [13]. Its processing speed can reach up to 150 frames per second, making it suitable for applications that require real-time response, such as autonomous vehicles, surveillance systems, and dynamic monitoring of industrial processes [14].

According to [15], the YOLOv8 architecture is also highly versatile. It can be trained with pre-trained models (pre-trained weights) or custom datasets, being compatible with libraries such as Ultralytics (Frederick, MD, USA) and platforms like Roboflow (Des Moines, IA, USA). The efficiency of training and inference, combined with its generalization capacity, contributes to its growing adoption in projects involving computer vision applied to industry and scientific research.

Although newer object detection models such as YOLOv9, YOLOv10, YOLOv11, and YOLOv12 have recently been introduced, the decision to employ YOLOv8n and YOLOv8s in this study is supported by practical considerations regarding model maturity, real-time performance, and deployment feasibility in edge computing environments.

YOLOv8 remains one of the most stable and thoroughly benchmarked versions in terms of detection accuracy (mAP), speed, and ease of deployment. Its architecture supports streamlined conversion to ONNX and CoreML formats, which is essential for inference on low-power devices like Raspberry Pi. Moreover, YOLOv8 offers a favorable trade-off between precision and computational load, fundamental in CPU-only environments where newer models may not perform efficiently.

While YOLOv9 introduced advanced architectural changes to improve accuracy and robustness, its deployment pipeline is still evolving, and it currently lacks widespread support and optimization for ARM-based edge devices. YOLOv10, YOLOv11, and YOLOv12 are even more recent, with a primary focus on server-side inference or GPU-accelerated platforms, making them less suited for real-time industrial applications constrained by hardware limitations.

Thus, YOLOv8 was chosen as a technically viable, well-supported, and field-tested solution aligned with the objectives of this study, which emphasizes low-latency visual inspection in industrial environments using affordable and energy-efficient edge devices.

The architectures of the You Only Look Once (YOLO) family have evolved significantly to meet both high-precision demands and execution requirements of devices with computational constraints. Among their optimized variants, YOLO Nano and YOLO Small stand out, both aimed at real-time applications, but with distinct complexity and performance profiles.

YOLO Nano has an ultra-compact architecture designed explicitly for inference in low-power embedded devices, including smart sensors, drones, and mobile applications. It is designed in a combination with network compression techniques, including depth-wise separable convolutions and parameter reduction modules, aiming to maintain performance on simple detection tasks even with severe hardware constraints [16]. It features less than 1 GFLOP and typically less than 2 million parameters, with a mAP performance of approximately 25–35% on standard datasets.

YOLO Small (e.g., YOLOv5s and YOLOv8s) represents the scaled-down version of modern YOLO architectures developed by Ultralytics. These variants maintain a richer structure, such as CSPNet or C2f in the backbone, when coupled with Feature Pyramid Networks (FPNs) or Path Aggregation Networks (PANs), which allow for greater depth and generalization capabilities [13,17]. Although they are compact (with about 7–11 million parameters and 5–10 GFLOPs), these versions maintain high accuracy (an mAP of 35–45%), making them ideal for real-time industrial applications such as visual quality inspection, intelligent monitoring, and autonomous robotics.

Table 1 presents a comparison chart between the Yolo Nano and YOLO Small versions.

Table 1. Comparison of YOLO Nano and YOLO Small.

Criterion	YOLO Nano	YOLO Small
Architecture	Optimized CNN with separable convolutions	CSPNet/C2f + FPN or PAN
Parameters	~1–2 million	~7–11 million
FLOPs	<1 GFLOP	5–10 GFLOPs
Accuracy (mAP)	25–35%	35–45%
Speed	Very high, ideal for edge	High, with real-time support
Applications	IoT, drones, mobile apps	Industry 4.0, autonomous vehicles, manufacturing
Compatible Frameworks	Limited support (YOLO Nano is academic research)	Broad support (Ultralytics, PyTorch 2.x, ONNX, CoreML)

3.1.3. Hardware Architectures for AI: CPU vs. GPU

A heterogeneous hardware architecture, composed of a CPU and a GPU, involves integrating these two processing units with distinct characteristics to optimize computational performance in specific applications, particularly those related to artificial intelligence (AI).

The Central Processing Unit (CPU) is responsible for the sequential execution of instructions, having a few cores with ample cache memory, which is ideal for handling small numbers of threads but with high logical complexity [18]. In contrast, the Graphics Processing Unit (GPU) features many cores with a massive parallel processing capacity, making it particularly well suited for intensive and repetitive operations, such as matrix calculations and deep neural network training [18–20].

According to [18], hybrid CPU–GPU platforms offer significantly higher performance in high-performance computing (HPC) applications compared to CPU-only platforms due to the parallel capacity and a high number of simultaneous threads available on the GPU. This combination allows complex calculations to be performed in an accelerated manner, considerably reducing the training time of deep learning models.

In [20], it is emphasized that hybrid architectures are crucial for the efficient training of convolutional neural networks (CNNs), particularly when evolutionary techniques, which require massively parallel processing, are employed. These architectures enable the distribution of computational tasks more effectively, allocating processes that require parallelization (such as processing large volumes of training data) to the GPU, while assigning more complex and less parallelizable tasks to the CPU.

Furthermore, [19] highlights the importance of heterogeneous CPU–GPU architectures in optimizing energy efficiency and performance in machine learning algorithms, indicating that a balanced configuration of these two units results in significant improvements in overall computational efficiency, which is essential in the face of the increasing complexity of modern AI applications.

The importance of CPU–GPU architectures for AI is as follows:

- Efficiency in Parallel Processing: it is essential for the fast and efficient execution of calculations required for deep learning, such as matrix multiplications and convolution operations [18,20].
- Reduction in Training Time: the joint use of CPUs and GPUs drastically reduces the training time of AI models, enabling faster development and tuning cycles [19].
- Energy Savings: they optimize energy consumption by leveraging the efficient parallel processing capabilities of GPUs, combined with CPU flow control, which is particularly essential in data centers and large-scale computing infrastructures [19].

Table 2 presents a comparison of the characteristics of CPUs and GPUs.

Table 2. CPU and GPU performance comparison.

Criterion	CPU	GPU
Performance	High in complex sequential tasks; low parallelization [18]	Very high in highly parallelizable tasks; ideal for deep learning and matrix computations [18,21]
Cost	Generally, a lower initial cost; good value for money for general tasks [19]	Higher initial cost; excellent cost–benefit in specialized applications requiring acceleration [22]
Energy Efficiency	Moderate energy efficiency due to less parallelization [23]	High energy efficiency in parallel applications that reduce total execution time [19,23]
Recommended Applications	Sequential applications with high logical complexity [18]	Parallelizable applications, deep learning, intensive numerical computation [20]

These aspects make hybrid CPU–GPU architectures essential for addressing the computational challenges posed by contemporary artificial intelligence applications, particularly in contexts that require processing large volumes of data and high precision in complex models.

3.2. Methodological Framework

3.2.1. Research Approach: Design Science Research (DSR)

This study was developed based on a real case applied to an automotive company located in Brazil, specialized in the development of piping systems for fluids and the extrusion of polyamide tubes used in the assembly of sets.

Design Science Research (DSR) was used, and its research process, according to [24], includes six main stages: the identification and motivation of the problem; definition of objectives for a possible solution; design and development of the proposed solution; practical demonstration; evaluation of the results obtained; and, finally, communication of the conclusions reached.

3.2.2. Problem Identification and Motivation

In the analyzed production line, internally extruded polyamide tubes are assembled with plastic connectors, and at the end of the assembly, they are positioned in customized aluminum jigs. These jigs are used to passively verify whether the components comply with the product design specifications (curvature dimensions, length, diameter, and connector positioning).

During the interview with the industrial director and the visits to the factory, it was possible to identify some problems related to the control devices or templates. One of the main challenges is the difficulty in handling and transporting these devices to the production line, as they are specifically designed for the manufacture of a particular type of piping. When not in use, they remain stored on shelves, which contributes to excessive space occupation in the production areas. Additionally, the high cost associated with manufacturing each customized template is highlighted.

The motivation for this problem is the replacement of current product control devices with a technological solution based on the technologies presented in the theoretical foundation.

3.2.3. Objectives and Success Criteria

For replacement purposes, the success criterion was defined as the ability of the AI-based system to identify the following:

- The general geometric shape of the tube (contour and curvature).

- The presence and correct position of the connectors.
- Visual fault detection, such as deformed, missing, or poorly fitted parts.

The objective is to develop a technological artifact for a multinational auto parts industry, which aims to replace physical templates used in the verification of extruded tubes with an automated inspection system based on computer vision using YOLO.

3.3. System Development

After analyzing the requirements and defining the objective, it was decided to propose the technological solution in two parts, the first part being the development of the identification and classification algorithm with YOLOv8 for training the model to be used and the second part being the development of the algorithm for the IoT device to be placed on the production line.

3.3.1. Dataset Collection and Annotation

A database was built with 2027 images of assembled pipes, captured under different lighting and positioning conditions, using an iPhone 15. The images were manually annotated with the Roboflow tool, indicating the following two classes:

- Conforming (class 1 with 800 images).
- Non-conforming (class 2 with 1227 images).

The dataset was split into 70% for training, 20% for validation, and 10% for testing. This proportion reflects the standard practice in deep learning for small-to-medium-sized datasets, ensuring sufficient data for model learning while reserving subsets for unbiased evaluation. Care was taken to preserve the class distribution across the three partitions, using stratified sampling to avoid imbalance between conforming and non-conforming samples.

A separate subset was extracted with a 96%/4%/2% split for the on-device testing of the Raspberry Pi 500 and was used exclusively to simulate real-time inference conditions and evaluate model responsiveness under constrained hardware. This split did not interfere with the training process or the main evaluation metrics reported.

Two representative examples were selected for the conforming class. Figure 2a shows an assembled tube classified as "conforming", and Figure 2b presents another example of the same class, captured under specific lighting and positioning conditions, highlighting the data variability.

(**a**) (**b**)

Figure 2. Examples of application of conforming class.

Figure 2a,b illustrate tubes that meet all dimensional and assembly specifications, including the correct curvature and tube length and the proper placement and fitting of both connectors. These are considered standard-compliant components, free of visible defects or assembly issues.

For the non-conforming class, Figure 3a illustrate the representative examples of objects classified as "non-conforming", captured under different lighting and positioning conditions.

(**a**) (**b**)

Figure 3. Examples of application of non-conforming class.

In contrast, Figure 3b show components that do not comply with specifications, including issues such as misaligned or missing connectors, incorrect curvature angles, deformed plastic fittings, or deviations in tube length. These defects are sometimes subtle and difficult to detect visually, especially under varying lighting conditions, which reinforce the need for automated visual inspection using AI-based systems.

Overall, 96% of the images were utilized for training, 4% for validation, and 2% for testing.

3.3.2. Training Environment and Parameters

The system implementation utilized the Ultralytics YOLOv8 library, with training conducted in a Python 3.10 environment using the Anaconda 24.9.2 environment manager, along with auxiliary libraries such as OpenCV, PyTorch, and Pandas. The training was performed on a computer with a Windows 10 operating system, an Intel i7-870 CPU (Santa Clara, CA, USA), 16 GB of RAM, SSD storage with 700 GB, and an NVIDIA RTX 4060 GPU (Santa Clara, CA, USA) (8 GB).

The training parameters were as follows:

- Number of epochs: 100;
- Image size: 640 × 640 px;
- Batch size: 8;
- Optimizer: SGD with a learning rate of 0.01;
- Early stopping: enabled after 10 epochs without mAP improvement.

3.3.3. YOLOv8 Model Implementation

The Raspberry Pi 500 Desktop was chosen as the IoT device to run the YOLOv8 Small (S) or Nano (N) models due to its CPU-only architecture, which enables the validation of computer vision models in environments with limited computing resources, thereby reflecting real-world field deployment scenarios. The decision to use a device without a GPU aims to test the viability of low-cost, low-power embedded solutions, promoting the

adoption of artificial intelligence in distributed applications where the use of dedicated graphics units is not technically or economically viable.

It is important to clarify that the Raspberry Pi 500 was not used during the training phase of the models. All training procedures were conducted on a GPU-powered workstation, as described in Section 3.3.2. The Raspberry Pi was used exclusively for on-line inference, serving as a lightweight and low-power target platform to validate the practical deployment of the pre-trained YOLOv8 models in real-world industrial environments.

3.3.4. Embedded System Deployment on Raspberry Pi 500

The Raspberry Pi 500 Desktop features a compact, integrated architecture designed to deliver a desktop experience with sufficient computing power for educational and light industrial applications. Its hardware consists of a 64-bit Broadcom BCM2712 quad-core ARM Cortex-A76 (San Jose, CA, USA) processor operating at 2.4 GHz, accompanied by up to 8 GB of LPDDR4X RAM (Boise, ID, USA), as well as integrated interfaces such as USB 3.0, dual HDMI, gigabit Ethernet, and Wi-Fi and Bluetooth connectivity, making it suitable for embedded IoT applications.

One of the technological innovations of this study lies in the choice of the Raspberry Pi 500 as the deployment platform for the AI-based visual inspection system. Unlike mobile devices that often require image format conversions, custom app development environments, or adaptation layers to support object detection frameworks, the Raspberry Pi 500 runs a full Linux-based operating system. This native Linux environment allows seamless development and execution of Python-based routines and facilitates the direct use of the Ultralytics YOLO framework without additional transformations or dependencies. By leveraging the command-line interface and direct access to system resources, developers can optimize inference processes, manage datasets, and implement real-time processing workflows in an efficient and reproducible manner. This flexibility, combined with the affordability and portability of the Raspberry Pi 500, makes it a highly suitable edge computing solution for industrial applications that demand fast prototyping and deployment without the complexity of mobile integration pipelines.

In the execution environment configured for this application, the YOLOv8 Nano or Small model was used, developed in Python 3.10.16 with support from the Anaconda distribution, which allowed the efficient management of libraries and virtual environments. Real-time detection was made possible by using a USB webcam connected directly to the device, using libraries such as cv2 (OpenCV) for video capture and Ultralytics for inference of the YOLOv8 model.

The choice of this execution environment for a CPU, without the use of a dedicated GPU, reinforces the objective of validating the applicability of lightweight computer vision models on low-cost embedded platforms, simulating realistic operational conditions in industrial, educational, or smart city contexts.

To enhance the understanding of the system's operation, Figure 4 presents a schematic overview of the embedded visual inspection architecture. The process starts with image acquisition via webcam, followed by object detection using the YOLOv8 model deployed on the Raspberry Pi 500. Detected objects are then analyzed to verify compliance based on predefined geometric criteria, and the final classification is outputted for decision-making or feedback into the production line.

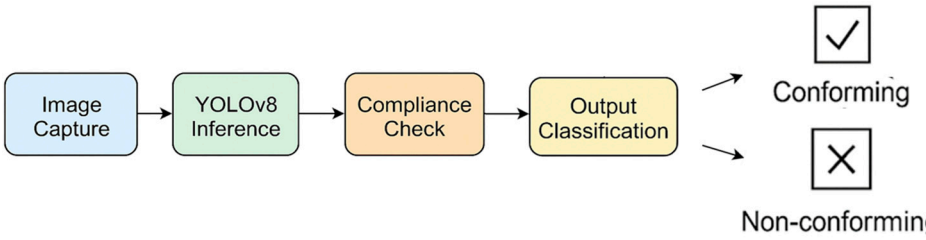

Figure 4. Schematic pipeline of embedded visual inspection architecture using YOLOv8 on Raspberry Pi 500.

3.4. Evaluation and Testing

3.4.1. Inference Testing on GPUs and CPUs

After training, the model was integrated into a Python program and tested with several images of compliant and non-compliant products that were not in the original dataset. The system processed the image and provided the verification result in less than 1 s:

- "Conforming" for parts conforming to the standard;
- "Non-conforming" with visual highlighting of non-conforming elements.

The results were automatically recorded in a database for traceability and auditing.

3.4.2. Evaluation and Performance Metrics

The performance metrics adopted per class highlight explanations that are critical to understanding the model's performance for each class, especially in datasets with multiple-object categories [25].

To evaluate the performance of the proposed YOLOv8-based detection system, several standard metrics of object detection were used. These metrics help analyze the accuracy, reliability, and generalization capability of the models under test, particularly in constrained edge environments.

- Accuracy measures the proportion of correct predictions among all predictions made by the model.
- Precision and Recall are defined using the following terms:
  - True Positives (TP): the number of correctly predicted positive detections.
  - False Positives (FP): the number of incorrectly positive predictions (i.e., predicted as positive but actually negative).
  - False Negatives (FN): the number of missed detections (i.e., predicted as negative but actually positive).

The formulas for calculating precision, recall, and F1-score are presented below.

- Precision (Equation (1)) quantifies the proportion of correctly predicted positive detections among all positive predictions:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (1)$$

- Recall (Equation (2)) measures the ability of the model to detect all relevant objects:

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (2)$$

- F1-score (Equation (3)) is the harmonic mean between Precision and Recall, balancing both measures:

$$\text{F1-score} = 2 * (\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}) \quad (3)$$

- Mean Average Precision (mAP) is the average of the APs obtained for all classes and IoU thresholds.
  - mAP@0.5:0.95: the average precision across multiple IoU thresholds from 0.5 to 0.95, used in the COCO benchmark.
  - mAP@0.5: AP calculated at an IoU threshold of 0.5.

These metrics provide a comprehensive view of model performance, especially in industrial applications where both high precision and high recall are critical to avoid false detections and missed defects.

3.4.3. Practical Demonstration in Industrial Scenario

The results and evaluations are presented in Section 4—Results and Discussion—of this article. The validation was conducted by an industrial director and a production manager from the researched industry.

The conclusions, as well as the intention of this article, are stated in Section 5 of this article, to disseminate the findings of this research to the academic community.

Noteworthily, the validation focused on static image inference rather than continuous video stream analysis. Each test image was processed individually to evaluate classification performance, inference time, and reliability under constrained computational conditions. Although real-time streaming with continuous frames was not the objective in this phase, the architecture was designed to support frame-by-frame analysis using the webcam input, which is suitable for many industrial scenarios that involve low-to-moderate image acquisition rates. Future research may include quantitative benchmarking in the continuous operation mode, including FPS and system latency measurements.

4. Results and Discussion

4.1. YOLOv8 Model Selection (Small x Nano)

The YOLOv8 Nano and Small models were carefully selected for testing based on the computational constraints of the final IoT device, which would exclusively use Central Processing Units (CPUs) without the support of Graphics Processing Units (GPUs). The device in question was the Raspberry Pi 500, a representative embedded platform for real-world applications in edge computing environments, widely used in Industry 4.0 due to its low energy consumption, affordable cost, and ease of integration with sensors and actuators.

As discussed by [8,16], IoT devices and embedded industrial applications often operate under severe computational resource and power constraints, requiring optimized models that can guarantee acceptable performance under restricted conditions. In this context, the choice of a compact and efficient model, such as YOLOv8 Nano, was particularly strategic, aiming to achieve compatibility with the Raspberry Pi 500 ARM architecture and the ability to perform inferences locally, without requiring external servers or GPU acceleration.

These limitations posed a significant challenge, since the most advanced YOLO models (such as YOLO Medium or Large) generally required high computational power, which was mainly available with high-performance GPUs. As indicated in [19], energy efficiency and performance in devices based exclusively on CPUs became decisive factors in the practical adoption of these technologies in Industry 4.0. Thus, the experimentation with the Raspberry Pi 500 reflected a realistic approach oriented towards the practical applicability of YOLO models in embedded industrial environments.

4.2. Performance of YOLOv8 Nano and Small Models

Testing was performed using the YOLOv8 Nano and YOLOv8 Small models, each trained for 100 and 200 epochs, respectively, on the NVIDIA RTX 4060 GPU. The perfor-

mance evaluation, however, considered the feasibility of running the inference later on an ARM CPU of the Raspberry Pi 500.

The analysis of the training curves shows behaviors consistent with theoretical expectations for the YOLOv8 architectures in their Nano and Small variants. Figure 5, Figure 6, Figure 7, and Figure 8 correspond to the training curves for YOLOv8 Nano with 100 epochs, YOLOv8 Nano with 200 epochs, YOLOv8 Small with 100 epochs, and YOLOv8 Small with 100 epochs, respectively.

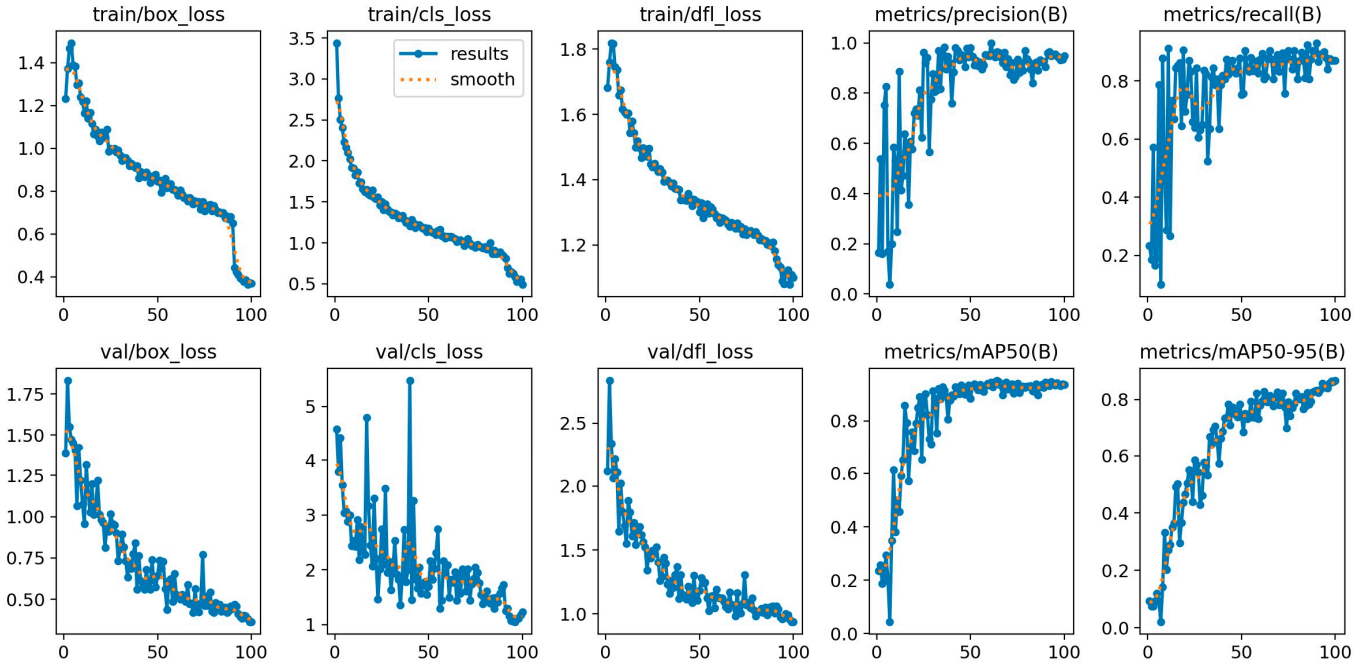

Figure 5. Training curves for YOLOv8 Nano with 100 epochs.

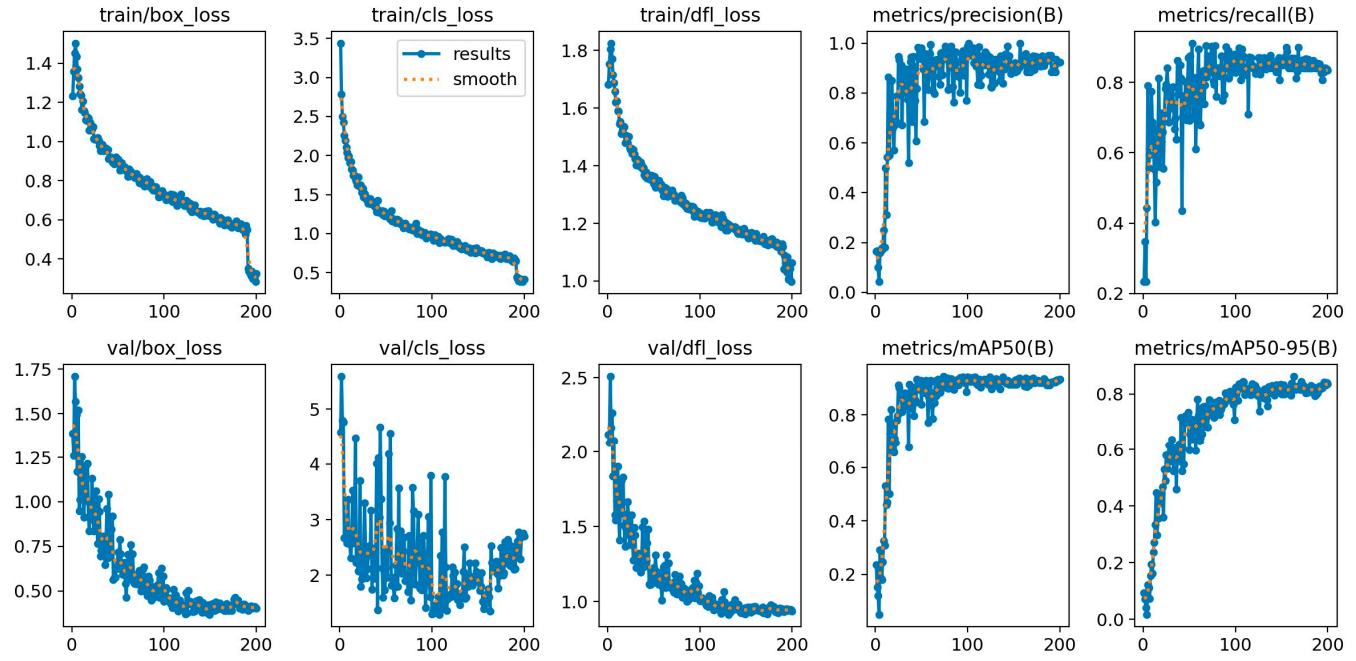

Figure 6. Training curves for YOLOv8 Nano with 200 epochs.

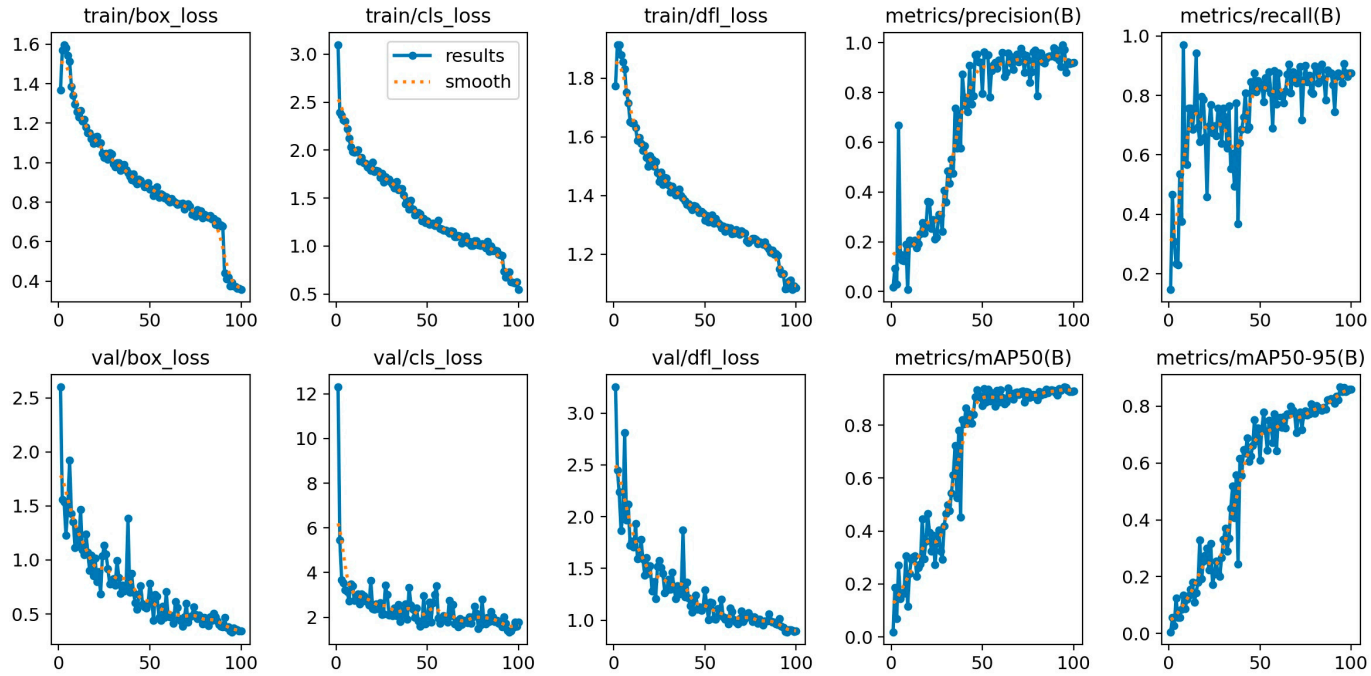

Figure 7. Training curves for YOLOv8 Small with 100 epochs.

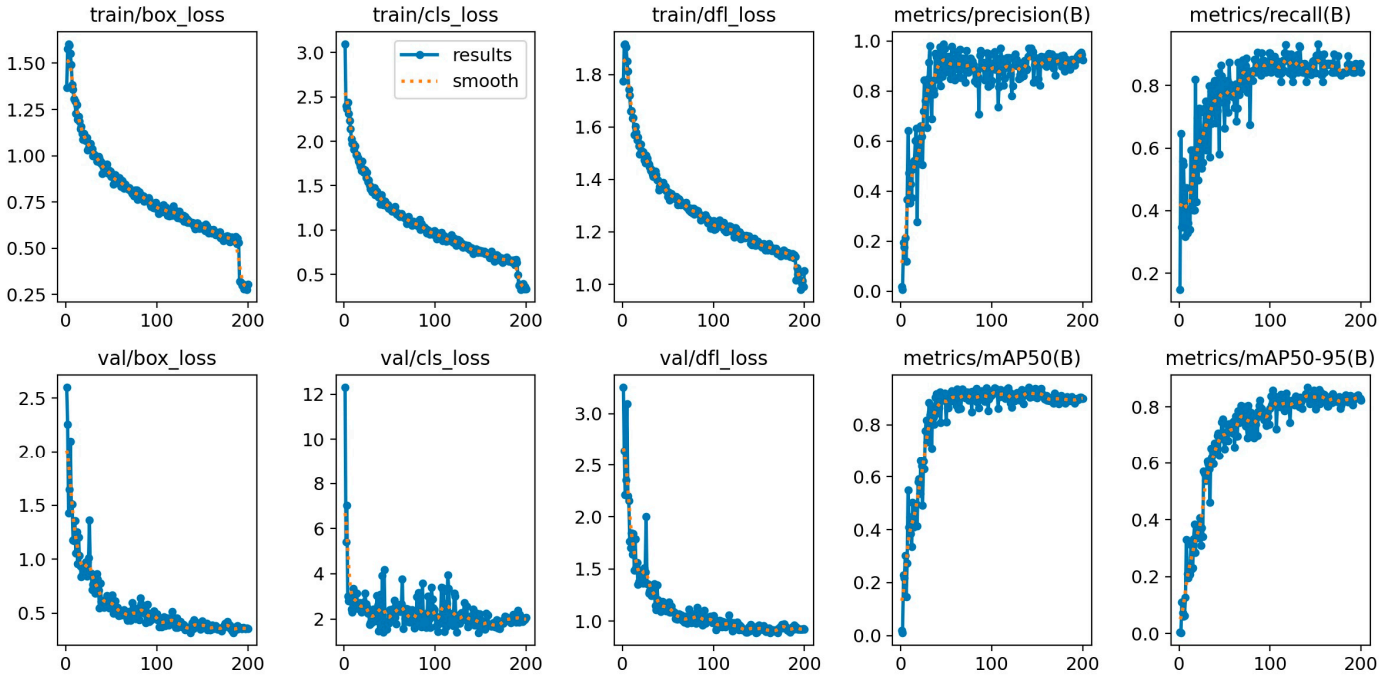

Figure 8. Training curves for YOLOv8 Small with 200 epochs.

Figures 5–8 show the training curves of YOLOv8 Nano and Small models with 100 and 200 epochs, respectively. In each plot, the horizontal axis represents the number of training epochs, while the vertical axis shows the values of performance metrics, such as training loss, validation loss, precision, recall, and mAP. These curves allow the analysis of learning behavior, convergence, and possible overfitting.

Losses were observed in the early stages of training, followed by a gradual convergence. This evolution is expected because, as highlighted in [16], compact models such as YOLO Nano tend to converge more quickly due to their fewer parameters. The stabilization of the loss and the maintenance of validation metrics (precision, recall, and mAP)

indicate that the model was learning robust representations without showing strong signs of overfitting in this 100-epoch configuration.

This evolution confirms that the model continues to refine its weight with more iterations, which may slightly improve generalization ability. However, the occasional oscillations suggest that, near the end of training, the incremental gain is small. This aligns with the literature, which indicates that for lightweight models, extending training beyond a certain point can lead to marginal gains without severe overfitting—a desirable behavior for applications on resource-constrained devices, such as the Raspberry Pi 500.

For YOLOv8 Small trained for 100 epochs, the curves demonstrate a rapid reduction in losses, followed by a convergence phase where the validation metrics (precision, recall, mAP@0.5, and mAP @[0.5:0.95]) reach high and stable values. According to [8,17], models with greater complexity and a larger number of parameters, such as YOLOv8 Small, also exhibit a rapid improvement in the early stages of training. The stability in the validation metrics indicates that the model effectively learned the patterns present in the data and was able to generalize well without showing significant overfitting in this configuration.

In the case of YOLOv8 Small trained for 200 epochs, the loss curves demonstrated a longer stabilization; however, a slight degradation of the validation metrics was observed with training for 200 epochs compared to training for 100 epochs. This slight deterioration suggests the possibility of overfitting, which is often observed in more complex models when the number of epochs is increased beyond the optimal point of convergence. According to [15], although longer training can better adjust the weights of the training set, this does not always translate into a proportional improvement in generalization, especially in environments where the data have limited variability. Thus, training with 100 epochs can be considered a more efficient breakeven point for YOLOv8 Small, considering the application scenario of IoT devices with CPUs, such as the Raspberry Pi 500.

Both models demonstrated rapid loss reduction in the early stages of training, which is expected for optimized architectures. The YOLOv8 Nano, with its lightweight structure, achieved relatively rapid stabilization, corroborating the findings of [16].

Extending training to 200 epochs showed limited benefits for Nano. It indicated a slight drop in performance for Small, suggesting that there is a saturation point beyond which gains are marginal, or signs of overfitting may occur [15].

As emphasized in the theoretical discussion, the choice of models is dependent on the need to implement the solution using the Raspberry Pi 500—an IoT device that operates exclusively with a CPU. More compact and optimized models (such as the YOLOv8 Nano) are especially advantageous in this context because they offer an excellent balance between predictive performance and the reduced consumption of computational resources [8,19].

The other results obtained are presented in Table 3.

Table 3. Results obtained in training of models.

Model	Epochs	Precision (P)	Recall (R)	mAP@0.5	mAP @[0.5:0.95]	F1-Score	Inference Time (ms/Image)
YOLOv8 Small	100	0.951	0.904	0.941	0.851	0.907	~13–14 ms
YOLOv8 Small	200	0.925	0.871	0.922	0.847	0.911	~13–14 ms
YOLOv8 Nano	100	0.922	0.841	0.934	0.865	0.925	~13–14 ms
YOLOv8 Nano	200	0.932	0.862	0.938	0.868	0.914	~13–14 ms

The study results demonstrated that both models achieved high performance, with Precision and Recall above 90% for YOLOv8 Small and slightly lower but consistent values for YOLOv8 Nano. Surprisingly, the Nano model outperformed Small in the mAP @[0.5:0.95] metrics with 200 epochs (0.868 vs. 0.847), demonstrating greater generalization

in detections across different IoU thresholds. This corroborates the findings of Lin et al. [16], who highlight the structural efficiency of YOLO Nano in resource-constrained scenarios, achieving an optimal balance between performance and computational efficiency.

Small models, despite having a larger number of parameters (~11 million versus ~3 million for Nano), achieved slightly higher absolute precision over 100 epochs. This suggests that, although heavier, Small is advantageous for environments where greater processing capacity is available, which is not the case for the Raspberry Pi 500. The drop in performance observed over 200 epochs (Precision: from 0.951 to 0.925) indicates the possibility of overfitting, as warned by Ahmed et al. [15], which validates the choice of fewer epochs for more complex models.

The YOLOv8 Nano model demonstrated performance growth between 100 and 200 epochs, without compromising generalization, proving to be more robust in extended training, a behavior consistent with lightweight models described in [17] and ideal for low-power devices. This is further reinforced by the F1-score analysis: while YOLOv8 Small at 100 epochs achieved the highest F1-score (0.925), the YOLOv8 Nano at 200 epochs reached a competitive F1-score of 0.911, surpassing Small's performance at 200 epochs (0.914), despite having a third of the parameters. This reinforces Nano's robustness and adaptability, especially when training is extended in environments with limited computational resources.

The YOLOv8 Nano model demonstrated an unexpectedly strong generalization capability, even when evaluated on external images not included in the training or validation sets. This behavior can be explained by a combination of architectural, training, and dataset-related factors.

From an architectural standpoint, YOLOv8 Nano is optimized for edge inference and employs a reduced number of convolutional layers, depth-wise separable convolutions, and C2f blocks (cross-stage partial connections with fewer channels), which reduce redundancy and overfitting risks while maintaining sufficient representation power. Its lightweight backbone and narrower head layers enforce a form of regularization by limiting model complexity and encouraging general feature abstraction rather than over-specialization.

In addition, the model benefits from built-in enhancements introduced in YOLOv8, such as anchor-free detection, loss function improvements (e.g., CIoU loss), and dynamic label assignment strategies, which collectively improve convergence stability and robustness in small datasets.

Regarding the dataset, although the total number of images was modest (2027 annotated instances), care was taken to ensure intra-class variability, with examples captured under different lighting conditions, angles, distances, and tube orientations. Furthermore, the training pipeline employed data augmentation strategies such as horizontal and vertical flipping, brightness and contrast shifts, random cropping, and affine transformations. These techniques helped expose the model to a broader distribution of visual conditions, increasing its resilience to unseen data.

Finally, the use of stratified sampling during dataset splitting preserved the balance between conforming and non-conforming classes across all partitions, preventing bias during training and validation. All these factors combined likely contributed to the observed generalization performance of YOLOv8 Nano, even under CPU-only edge deployment conditions.

Overfitting Behavior of YOLOv8 Small

The slight degradation observed in the validation metrics of the YOLOv8 Small model after 200 epochs, compared to its performance at 100 epochs, suggests the occurrence of overfitting. This happens when the model begins to memorize the training data rather

than learning generalizable features, thus reducing its effectiveness when tested on new, unseen images.

Reference [15] emphasizes that such behavior is common in high-capacity models, especially when trained on datasets with limited variability. In this study, the drop in precision (from 0.951 to 0.925) and in mAP@0.5 (from 0.941 to 0.922) despite additional training indicates a saturation point. Extending training time may no longer contribute to generalization and may reduce robustness in more variable scenarios.

While a quantitative decrease was observed in the metrics, further signs of overfitting can be inferred from the model's inconsistent confidence levels on external test images. For instance, the YOLOv8 Small model showed variability in prediction confidence for non-compliant parts, with diminishing benefits between 100 and 200 epochs.

Although this study does not include visual samples highlighting such effects, the presented data support the hypothesis of limited generalization beyond a certain training threshold. Future studies could include qualitative error analysis using external samples to identify the typical manifestations of overfitting in bounding box predictions.

Furthermore, the average inference time for both models was similar when tested in a GPU environment (~13–14 ms per image). Still, the expectation is that Nano will maintain a significant advantage when running on an ARM CPU, as described in [19], due to its smaller number of layers and lower memory load, facilitating its direct application for the Raspberry Pi 500 with reduced energy consumption.

The practical application of these models in the automated visual inspection of components in an industrial environment reinforces the importance of selecting network architectures that are compatible with the available computing ecosystem. The use of YOLOv8 Nano on the Raspberry Pi 500 becomes a viable and efficient solution, offering high detection performance without relying on high-cost computing infrastructures, which is strategic for smart factories and decentralized Industry 4.0 operations.

This study makes a significant scientific contribution by demonstrating that compact YOLOv8 models can effectively replace traditional hardware inspection tools. Unlike prior studies that focused solely on accuracy, this study evaluates the real-time feasibility of CPU-only devices, providing a replicable and cost-effective alternative for Industry 4.0 applications.

4.3. IoT Performance—Pi 500

After training and validating the YOLOv8 Nano and Small models on a GPU-based workstation, inference tests were performed directly on the Raspberry Pi 500, using only its ARM Cortex-A76 CPU, without graphics acceleration. This configuration reflects the study's practical objective: to validate the use of AI-based computer vision solutions on low-cost embedded devices that are compatible with Industry 4.0 requirements.

Eight test images external to the original dataset were used, representing scenarios of compliant and non-compliant parts. The YOLOv8 Nano and Small models were evaluated after 100 and 200 epochs of training. Table 4 presents a summary of the classification results and inference times obtained with the Pi 500.

The YOLOv8 Nano model showed significantly faster inference times (~470 ms per image) and consistent reliability across most images. Its confidence increased slightly between 100 and 200 epochs, and despite some sensitivity to more visually variable cases (e.g., ok1.jpg), it delivered satisfactory results for real-time embedded applications.

The YOLOv8 Small model demonstrated higher confidence scores on average but required over 1300 ms per inference, making it less suitable for applications with real-time constraints on CPU-only devices. The longer inference times restrict its practical deployment in industrial edge environments without GPU acceleration.

Table 4. CPU inference for Raspberry Pi 500.

Model	Epochs	Image	Detected Class	mAP@0.5	Reliability	Inference Time
YOLOv8 Nano	100	ok.jpg	Conforming	0.934	0.91	~470 ms
YOLOv8 Nano	100	nok.jpg	Non-conforming	0.934	0.88	~460 ms
YOLOv8 Nano	100	ok1.jpg	Conforming	0.934	0.41 (low)	~470 ms
YOLOv8 Nano	100	nok1.jpg	Non-conforming	0.934	0.63	~470 ms
YOLOv8 Nano	100	ok2.jpg	Conforming	0.934	0.71	~460 ms
YOLOv8 Nano	100	nok2.jpg	Non-conforming	0.934	0.83	~470 ms
YOLOv8 Nano	100	ok3.jpg	Conforming	0.934	0.86	~460 ms
YOLOv8 Nano	100	nok3.jpg	Non-conforming	0.934	0.89	~470 ms
YOLOv8 Nano	200	ok.jpg	Conforming	0.938	0.91	~470 ms
YOLOv8 Nano	200	nok.jpg	Non-conforming	0.938	0.88	~460 ms
YOLOv8 Nano	200	ok1.jpg	Conforming	0.938	0.62	~470 ms
YOLOv8 Nano	200	nok1.jpg	Non-conforming	0.938	0.85	~470 ms
YOLOv8 Nano	200	ok2.jpg	Conforming	0.938	0.71	~460 ms
YOLOv8 Nano	200	nok2.jpg	Non-conforming	0.938	0.89	~470 ms
YOLOv8 Nano	200	ok3.jpg	Conforming	0.938	0.92	~460 ms
YOLOv8 Nano	200	nok3.jpg	Non-conforming	0.938	0.94	~470 ms
YOLOv8 Small	100	ok.jpg	Conforming	0.941	0.98	~1337 ms
YOLOv8 Small	100	nok.jpg	Non-conforming	0.941	0.87	~1315 ms
YOLOv8 Small	100	ok1.jpg	Conforming	0.941	0.61	~1319 ms
YOLOv8 Small	100	nok1.jpg	Non-conforming	0.941	0.64	~1322 ms
YOLOv8 Small	100	ok2.jpg	Conforming	0.941	0.89	~1337 ms
YOLOv8 Small	100	nok2.jpg	Non-conforming	0.941	0.81	~1316 ms
YOLOv8 Small	100	ok3.jpg	Conforming	0.941	0.72	~1321 ms
YOLOv8 Small	100	nok3.jpg	Non-conforming	0.941	0.74	~1313 ms
YOLOv8 Small	200	ok.jpg	Conforming	0.922	0.95	~1339 ms
YOLOv8 Small	200	nok.jpg	Non-conforming	0.922	0.82	~1326 ms
YOLOv8 Small	200	ok1.jpg	Conforming	0.922	0.67	~1314 ms
YOLOv8 Small	200	nok1.jpg	Non-conforming	0.922	0.68	~1305 ms
YOLOv8 Small	200	ok2.jpg	Conforming	0.922	0.83	~1327 ms
YOLOv8 Small	200	nok2.jpg	Non-conforming	0.922	0.80	~1319 ms
YOLOv8 Small	200	ok3.jpg	Conforming	0.922	0.69	~1334 ms
YOLOv8 Small	200	nok3.jpg	Non-conforming	0.922	0.71	~1329 ms

Performance gains from increasing epochs were modest for both models, with signs of overfitting becoming more noticeable in the Small model. In contrast, Nano maintained generalization capability, reinforcing its suitability for edge AI implementations where energy efficiency and low latency are critical.

Although the external evaluation involved only eight images, the goal of this stage was to conduct a proof-of-concept validation using the Raspberry Pi 500, focusing on real-time inference feasibility in constrained environments. The limited sample size reflects the difficulty of obtaining diverse, annotated, real-world industrial images under consistent conditions. Nevertheless, the dataset used in training incorporated variations in lighting, orientation, and connector positioning to promote generalization.

Future iterations of this study will address this limitation by expanding the test set, incorporating images from multiple production batches, and performing k-fold cross-validation to obtain statistically robust performance metrics. Additionally, the authors are investigating the use of synthetic image generation and domain adaptation techniques to simulate variability and augment the dataset without requiring large-scale manual labeling.

4.4. Performance Comparison: GPU vs. CPU

To evaluate the practical feasibility of the embedded implementation, the inference times obtained in the tests performed on the GPU (during the training and validation

phase) and on the CPU (during the execution with the Raspberry Pi 500) were compared. Table 5 summarizes the average times per image in each scenario.

Table 5. Comparison of average inference times (GPU vs. CPU—Raspberry Pi 500).

Model	Epochs	Inference Time (GPU)	Inference Time (CPU—Pi 500)	Approximate Difference
YOLOv8 Nano	100	~13 ms	~470 ms	~36× slower
YOLOv8 Nano	200	~13 ms	~470 ms	~36× slower
YOLOv8 Small	100	~13 ms	~1315 ms	~101× slower
YOLOv8 Small	200	~13 ms	~1315 ms	~101× slower

The performance difference between the GPU and CPU was significant for both models, as expected. However, the Nano model demonstrated a more favorable CPU execution time, 36 times slower than the GPU execution time, while the Small model was over 100 times slower.

In hardware-constrained embedded environments, such as the Raspberry Pi 500, this difference directly impacts real-time applicability. The YOLOv8 Nano can process approximately two images per second, which is sufficient for many industrial visual inspection applications at moderate cadence.

YOLOv8 Small, on the other hand, has severe limitations pertaining to real-time response, making it more suitable for applications on GPU-powered servers or hardware-accelerated industrial workstations.

While the methodology employed relies on established deep learning architectures such as YOLOv8, the novelty of this study lies in the end-to-end integration of an AI-based visual inspection system optimized explicitly for real-time inference of edge devices without GPU support. The proposed solution addresses a critical need in the automotive manufacturing industry, where cost, space, and energy constraints hinder the adoption of traditional GPU-based quality control systems. Furthermore, the manually curated dataset and adaptation of the inference pipeline for a Raspberry Pi CPU-based environment represent a practical and replicable contribution for Industry 4.0 applications. Future research will aim to incorporate lightweight attention mechanisms and neural architecture search to further optimize performance under resource-constrained conditions.

5. Conclusions

This study aimed to develop and evaluate an automated visual inspection solution based on computer vision and artificial intelligence, capable of replacing physical inspection jigs used in the dimensional verification of extruded polyamide tubes, intended for application in fluid conduction systems in the automotive sector. It involved training YOLOv8 Nano and Small models, as well as their implementation and testing on a low-cost embedded device, the Raspberry Pi 500, to validate their technical and practical feasibility in industrial environments with computational constraints.

The results obtained demonstrated that both models achieved good performance in terms of metrics such as precision, recall, and mAP, both during GPU training and CPU inference. The YOLOv8 Small model demonstrated greater accuracy in images with visual variability; however, it had a significantly higher inference time than YOLOv8 Nano, making it less suitable for real-time execution on devices without a GPU. The YOLOv8 Nano model exhibited acceptable response times and performance, making it more suitable for embedded industrial applications, especially in Industry 4.0 contexts that require efficient and cost-effective solutions.

The study objective was fully achieved: a functional artifact was developed and evaluated both in the laboratory and on a real IoT platform, demonstrating its practical feasibility. Replacing the physical templates with an automated inspection system using YOLOv8 proved to be feasible, with results demonstrating reduced operating costs and greater flexibility in adapting to different tube models.

Although the number of external test images was limited, the results obtained were sufficient to support the proof-of-concept validation of real-time AI inspection using CPU-only devices. The experimental setup was designed to reflect realistic conditions found in industrial environments. Broader tests, including a more diverse and larger dataset, are planned for future work to reinforce the statistical robustness of the proposed solution.

However, some limitations were observed. First, the database used, although composed of real images, presented restrictions in terms of scenario variations, which may have influenced the drop in performance of the Nano model in less standardized photos. In addition, the system was validated with only fourteen images outside the training set, which suggests the need for a broader and continuous evaluation. Finally, CPU inference, although viable, still presents challenges related to scalability and performance optimization for high-speed industrial applications.

In future research, it is recommended to expand the dataset with a greater diversity of images, including variations in lighting, angles, and types of tubes. The authors acknowledge that the validation and test subsets consist of a relatively small number of samples (approximately 20% and 10% of the dataset, respectively). While this split follows standard practices in deep learning for small datasets, the limited size may introduce some statistical uncertainty in the reported performance metrics. Future work will consider applying k-fold cross-validation or expanding the dataset through additional image acquisition or domain adaptation techniques. These improvements will help better assess the generalization ability of the model and mitigate the risk of overfitting to a small test set. Furthermore, model compression and quantization strategies should be investigated to further reduce the CPU inference time, and the use of intermediate platforms, such as the NVIDIA Jetson Nano or Coral TPU, should be explored for performance comparisons with the Raspberry Pi 500.

This study is expected to make a significant contribution to the practical application of AI in industrial product inspection, demonstrating that it is possible to achieve a good balance between cost, performance, and accuracy through well-designed technological solutions tailored to the operational realities of smart manufacturing.

Author Contributions: Conceptualization, M.T.O. and W.A.C.L.; methodology, O.V.; software, M.T.O.; validation, O.V., S.M.R., and J.C.L.F.; formal analysis, S.M.R.; investigation, M.T.O.; resources, M.T.O.; data curation, S.M.R.; writing—original draft preparation, M.T.O.; writing—review and editing, S.M.R. and W.A.C.L.; visualization, W.A.C.L.; supervision, M.T.O.; project administration, M.T.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data will be made available upon request.

Acknowledgments: The authors thank the Coordination of Superior Studies—CAPES for the scholarship.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
GPU	Graphics Processing Unit

CNN	Convolutional Neural Network
YOLO	You Only Look Once
DSR	Design Science Research
ReLU	Rectified Linear Unit
mAP	mean Average Precision
FLOP	Floating Point Operation
IoT	Internet of Things

References

1. Yang, D.; Cui, Y.; Yu, Z.; Yuan, H. Deep learning based steel pipe weld defect detection. *Appl. Artif. Intell.* **2021**, *35*, 1237–1249. [CrossRef]
2. Mazzetto, M.; Teixeira, M.; Rodrigues, É.O.; Casanova, D. Deep learning models for visual inspection on automotive assembling line. *arXiv* **2020**, arXiv:2007.01857. [CrossRef]
3. Qi, Z.; Ding, L.; Li, X.; Hu, J.; Lyu, B.; Xiang, A. Detecting and classifying defective products in images using YOLO. *arXiv* **2024**, arXiv:2412.16935.
4. Huang, H.; Zhu, K. Automotive parts defect detection based on YOLOv7. *Electronics* **2024**, *13*, 1817. [CrossRef]
5. Zhang, K.; Qin, L.; Zhu, L. PDS-YOLO: A Real-Time Detection Algorithm for Pipeline Defect Detection. *Electronics* **2025**, *14*, 208. [CrossRef]
6. Zhao, X.; Wang, L.; Zhang, Y.; Han, X.; Deveci, M.; Parmar, M. A review of convolutional neural networks in computer vision. *Artif. Intell. Rev.* **2024**, *57*, 99. [CrossRef]
7. Zhou, L.; Zhang, L.; Konz, N. Computer vision techniques in manufacturing. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *53*, 105–117. [CrossRef]
8. Anumol, C.S. Advancements in CNN architectures for computer vision: A comprehensive review. In Proceedings of the 2023 Annual International Conference on Emerging Research Areas: International Conference on Intelligent Systems (AICERA/ICIS), Kanjirapally, India, 16–18 November 2023; IEEE: New York, NY, USA, 2023; pp. 1–7.
9. Khanam, R.; Hussain, M.; Hill, R.; Allen, P. A comprehensive review of convolutional neural networks for defect detection in industrial applications. *IEEE Access* **2024**, *12*, 94250–94295. [CrossRef]
10. Islam, M.R.; Zamil, M.Z.H.; Rayed, M.E.; Kabir, M.M.; Mridha, M.F.; Nishimura, S.; Shin, J. Deep Learning and Computer Vision Techniques for Enhanced Quality Control in Manufacturing Processes. *IEEE Access* **2024**, *12*, 121449–121479. [CrossRef]
11. Yu, L.; Gao, S.; Zhang, D.; Kang, G.; Zhan, D.; Roberts, C. A survey on automatic inspections of overhead contact lines by computer vision. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 10104–10125. [CrossRef]
12. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
13. Hussain, M. YOLO-v1 to YOLO-v8: The Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines* **2023**, *11*, 677. [CrossRef]
14. Karthika, B.; Venkatesan, R.; Dharssinee, M.; Sujarani, R.; Reshma, V. Object Detection Using YOLO-V8. In Proceedings of the 15th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kamand, India, 24–28 June 2024; IEEE: New York, NY, USA, 2024.
15. Ahmed, T.; Maaz, A.; Mahmood, D.; Abideen, Z.U.; Arshad, U.; Ali, R.H. The yolov8 edge: Harnessing custom datasets for superior real-time detection. In Proceedings of the 2023 18th International Conference on Emerging Technologies (ICET), Peshawar, Pakistan, 6–7 November 2023; IEEE: New York, NY, USA, 2023.
16. Lin, J.; Chen, Y.; Huang, S. YOLO-Nano: A Highly Compact You Only Look Once Convolutional Neural Network for Object Detection. *arXiv* **2020**, arXiv:2004.07676. [CrossRef]
17. Jocher, G. YOLOv5 by Ultralytics Documentation. Ultralytics. 2023. Available online: https://docs.ultralytics.com (accessed on 10 July 2025).
18. Chandrashekar, B.N.; Geetha, V.; Shastry, K.A.; Manjunath, B.A. Performance Model of HPC Application on CPU-GPU Platform. In Proceedings of the IEEE 2nd Mysore Sub Section International Conference (MysuruCon), Mysuru, India, 16–17 October 2022.
19. Kaur, R.; Mohammadi, F. Power Estimation and Comparison of Heterogeneous CPU-GPU Processors. In Proceedings of the IEEE 25th Electronics Packaging Technology Conference (EPTC), Singapore, 5–8 December 2023.
20. Venkat, R.A.; Oussalem, Z.; Bhattacharya, A.K. Training Convolutional Neural Networks with Differential Evolution using Concurrent Task Apportioning on Hybrid CPU-GPU Architectures. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June 2021–1 July 2021.

21. Kimm, H.; Paik, I.; Kimm, H. Performance Comparison of TPU, GPU, CPU on Google Colaboratory over Distributed Deep Learning. In Proceedings of the IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 20–23 December 2021.
22. Wang, Y.; Li, Y.; Guo, J.; Fan, Y.; Chen, L.; Zhang, B.; Wang, W.; Zhao, Y.; Zhang, J. Cost-effective computing power provisioning for video streams in a Computing Power Network with mixed CPU and GPU. In Proceedings of the 2023 Asia Communications and Photonics Conference (ACP/POEM), Wuhan, China, 4–7 November 2023.
23. Jay, M.; Ostapenco, V.; Lefèvre, L.; Trystram, D.; OOrgerie, A.C.; Fichel, B. An experimental comparison of software-based power meters: Focus on CPU and GPU. In Proceedings of the CCGrid 2023—23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, Bangalore, India, 1–4 May 2023.
24. Peffers, K.; Tuunanen, T.; Rothenberger, M.A.; Chatterjee, S. A design science research methodology for information systems research. *J. Manag. Inf. Syst.* **2007**, *24*, 45–77. [CrossRef]
25. Antunes, S.N.; Okano, M.T.; Nääs, I.d.A.; Lopes, W.A.C.; Aguiar, F.P.L.; Vendrametto, O.; Fernandes, J.C.L.; Fernandes, M.E. Model Development for Identifying Aromatic Herbs Using Object Detection Algorithm. *AgriEngineering* **2024**, *6*, 1924–1936. [CrossRef]