

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITY OF MODENA AND REGGIO EMILIA

"Enzo Ferrari" Department of Engineering

Master's Degree in Artificial Intelligence Engineering (LM-32)

Real-Time Visual Anomaly Detection for Industrial Bottle Cap Inspection

Supervisor:
Prof. Roberto Vezzani

Candidate:
Mattia Isidoro
Student ID 207443

ACADEMIC YEAR 2025/2026

Contents

1 Introduction **1**

1.1 Industrial Quality Inspection 1

1.2 Problem Statement 1

1.3 Challenge and Motivation 2

1.4 Objectives 4

1.5 Contributions 5

1.6 Thesis Structure 6

2 Background **7**

2.1 Industrial computer Vision 7

2.2 Optical Inspection System 9

2.2.1 The Limit of Standard Optics 9

2.2.2 Optical Principle of Pericentric Lenses 10

2.2.3 Image Formation and Geometric mapping 12

2.3 Triggering and Image Synchronization 13

2.3.1 Precision Timing in High-Speed Lines 13

2.3.2 HW and SW Triggering 13

2.3.3 Critical Impact on Pericentric Geometry 13

2.3.4 System Synchronization Architecture 14

2.4 Deep Learning for Visual Inspection 15

2.4.1 From Hand-crafted Feature to Deep Representation 15

2.4.2 Hierarchical Feature Extraction 15

2.4.3 Transfer Learning in Industrial Environments 17

2.5 Anomaly Detection 18

2.5.1 Learning Paradigms: Supervised, Semi-Supervised, and Unsupervised . . . 18

2.5.2 The One-Class Learning Paradigm in Industry 19

2.5.3 Reconstruction-Based Methods 21

2.5.4 Feature-Embedding and Memory-Based Methods . . . 22
2.6 PatchCore . . . 23
2.6.1 PatchCore: Theoretical Foundations and Mathematical Formulation . . . 23
2.6.2 Position Within the Landscape of Anomaly Detection Methods . . . 24
2.6.3 Stage 1: Feature Extraction . . . 26
2.6.4 Stage 2: Memory Bank Construction . . . 29
2.6.5 Stage 3: Coreset Subsampling . . . 29
2.6.6 Stage 4: Anomaly Detection and Localization . . . 31
2.6.7 Normalization and Classification Threshold . . . 32

3 Related Works: State of the Art in Industrial Anomaly Detection 33
3.1 Reconstruction-Based Methods . . . 33
3.1.1 Autoencoders . . . 34
3.1.2 Variational Autoencoders . . . 34
3.1.3 GANomaly . . . 35
3.2 Feature-Embedding Methods . . . 39
3.2.1 PaDiM . . . 39
3.3 Normalizing-Flow-Based Methods . . . 41
3.3.1 FastFlow . . . 41
3.3.2 EfficientAD . . . 43
3.3.3 Reverse Distillation . . . 44
3.3.4 RD++ . . . 45
3.4 Quantitative Comparison on MVTec AD . . . 46

4 Industrial Case Study 48
4.1 Production Environment . . . 48
4.2 Product Description . . . 50
4.2.1 Large Red Cap . . . 51
4.2.2 Large Transparent Cap . . . 51
4.2.3 Small Transparent Cap . . . 52
4.2.4 Black cap . . . 53

4.3 Defect Description 54
4.3.1 Transparent Caps (Large and Small) 54
4.3.2 Large Red Cap 55
4.3.3 Black cap 56
4.4 Acquisition System 57
4.4.1 Pericentric Lens 57
4.4.2 Triggering and Illumination 62
4.5 Dataset 62
4.5.1 Dataset Augmentation 63
4.5.2 Limitations 63
4.5.3 Dataset Composition and Splits 63

5 Proposed Method **65**
5.1 Pipeline Overview 65
5.2 Two-Tier Configuration Architecture 68
5.3 Geometric Alignment 70
5.3.1 Slot Detection (Large Red Cap) 70
5.3.2 Template Matching via Marker Pair (Transparent Caps) 72
5.3.3 Template Matching for the Black Cap 73
5.4 Feature Extraction and Memory Bank Construction 75
5.5 Real-Time Inference 76
5.6 Implementation Details and Issues 81

6 Experimental Results **83**
6.1 Overview 83
6.2 Experimental Setup (Model and Backbone Comparison) 84
6.2.1 Dataset 84
6.2.2 Models and Backbones 84
6.2.3 Hardware and Software 85
6.3 Evaluation Metrics 86
6.3.1 Image-level AUROC 86

6.3.2 Image-level F1-Score . . . 86
6.3.3 Fit Time and Inference Latency . . . 86
6.4 Backbone Comparison (PatchCore) . . . 87
6.5 Model Family Comparison . . . 88
6.6 Rotation Sensitivity and Alignment Method . . . 90
6.6.1 Method . . . 90
6.6.2 Results . . . 91
6.7 Discussion . . . 92

7 Conclusions and Future Work **94**
7.1 Summary of the Thesis . . . 94
7.2 Achievement of the Objectives . . . 94
7.3 Limitations . . . 95
7.4 Future Work . . . 97
7.4.1 Real Production Deployment . . . 97
7.4.2 Model Compression for Deployment on Constrained Hardware . . . 97
7.4.3 Strengthening the Experimental Evidence . . . 98
7.5 Closing Remarks . . . 98

References **99**

List of Figures

1.1 Comparison between the technical design of the cap and physical samples (standard vs. defective). 2
1.2 Pericentric Lens 3
1.3 Project Pipeline 5

2.1 Primary hardware components 8
2.2 Entocentric lens 10
2.3 Pericentric lens scheme 11
2.4 Pericentric lens example 12
2.5 Trigger and Camera Sensor 14
2.6 Receptive Field through Layers 16
2.7 Patchore pipeline 25
2.8 feature extraction 27
2.9 Locally aware feature 28
2.10 Memory bank creation alg 30

3.1 PaDiM pipeline 40
3.2 FastFlow pipeline from [19] 42

4.1 Genric production line 49
4.2 Large Red Cap 51
4.3 Large Transparent Cap 52
4.4 Small Transparent Cap 52
4.5 Black cap 53
4.6 Absent seal 55
4.7 Partially displaced seal 55
4.8 Missing seal 56
4.9 Partially missing seal black cap 56

4.10 Examples of bottle caps acquired using the pericentric imaging system. 61

5.1 Overview of the proposed inspection pipeline. The diagram shows the one-time system initialization and runtime configuration phases (left), during which the rotation function, alignment parameters, and PatchCore model/memory bank for the active cap type are loaded, and the per-frame inference loop (right), in which each incoming image is processed and classified as `GOOD` or `REJECT`. 67

5.2 Red cap debug example 71

5.3 Debug visualization of marker detection and rotation estimation for the large transparent cap. The green circles mark the detected reference markers, the yellow dot the estimated cap centre, and the red/blue contours the detected cap boundary and the region considered for alignment. 73

5.4 REJECTED Transparent cap for seal defect 78

5.5 Screenshot of the structured log output, showing startup information and per-frame inspection results recorded in `inspection.log`. 79

5.6 GOOD transparent cap 80

5.7 REJECTED transparent cap 80

5.8 REJECTED red cap 81

5.9 GOOD red cap 81

Abstract

Automated visual inspection is increasingly used to replace or support manual quality control in industrial production lines, where consistency, speed, and reliability are critical. This thesis presents the design, implementation, and experimental evaluation of a real-time anomaly detection system for the automated inspection of bottle caps on an industrial line, developed in collaboration with Giacobazzi winery. The system is designed to inspect four geometrically and optically distinct cap variants at production-line speed, without interrupting the inspection process when individual frames cannot be processed.

The proposed pipeline combines a variant-specific geometric alignment stage, implemented through slot detection and template-matching strategies selected via a configurable rotation registry, with a PatchCore-based anomaly detection model. The model compares locally extracted patch features against a memory bank built from nominal training images. The software architecture separates global and cap-specific configuration through a two-tier design based on SQLite and YAML, allowing additional cap variants to be integrated without modifying the core inference pipeline.

Several anomaly detection configurations were evaluated, including different backbone architectures and baseline methods. PatchCore achieved the best overall performance and was therefore selected for the final system. The resulting pipeline was evaluated in terms of both detection performance and inference latency, showing that the proposed memory-bank-based one-class approach can satisfy the accuracy and real-time requirements of the target industrial inspection task.

1. Introduction

1.1 Industrial Quality Inspection

Quality inspection refers to a preventive process aimed at detecting potential defects in manufactured products before they reach customers. Defective products reaching customers can lead to economic losses, product recalls, and damage to a company's reputation. In the food industry, quality inspection is particularly important because packaging defects may compromise product safety, shelf life, and customer trust. Companies adopt various inspection and testing methods to ensure that their products meet the required quality standards and to prevent defective items from reaching the market.

In the past, quality inspection was mainly performed manually by humans. However, the increasing adoption of Industry 4.0 technologies, which has significantly accelerated manufacturing processes, has led companies to invest in automated quality inspection systems. Among these, visual inspection represents one of the most critical stages of the production process, as many defects can be identified directly from the appearance of the product.

The use of computer vision and artificial intelligence has significantly improved the effectiveness of automated inspection systems, enabling more accurate and consistent quality control while reducing human error and inspection costs.

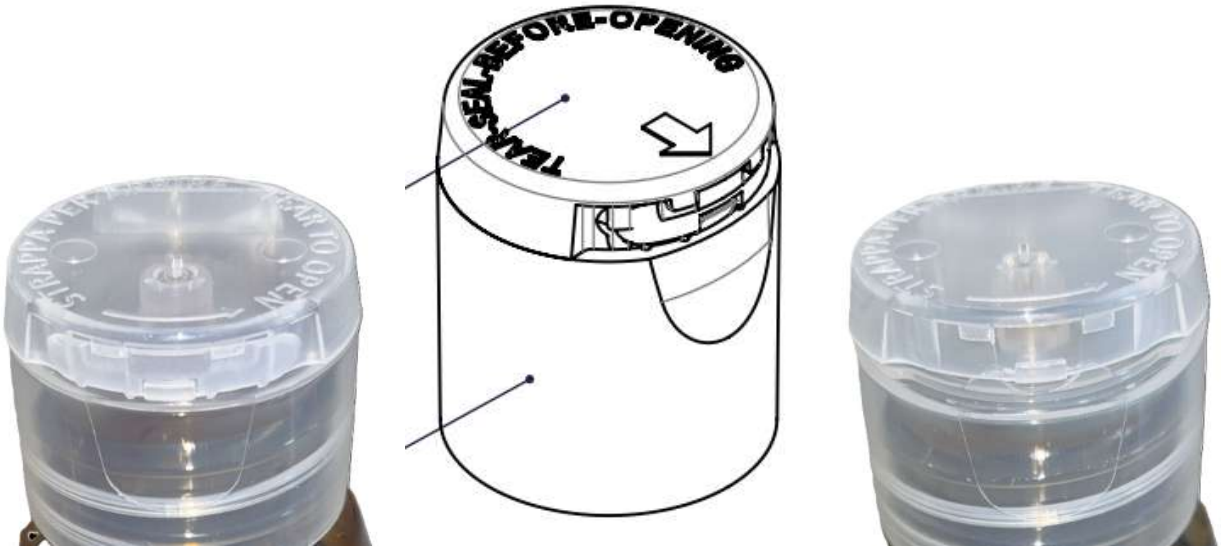
1.2 Problem Statement

In this work, the production line requires the inspection of vinegar bottle caps to detect missing or damaged safety seals. The line produces approximately one bottle per second, 3,600 bottles/hour. In particular, vinegar glaze bottles are automatically sealed by a capping machine, which places a plastic cap on each bottle.

Defects in the safety seal, such as partial breakages, Figure 1.1, or complete detachment, can either be present prior to assembly or occur dynamically due to the mechanical stress exerted during the capping process itself. Due to the line structure human operators can only inspect the caps before

they are used, removing visibly defective ones. So manual inspection cannot detect defects that occur during the capping process itself, allowing damaged products to continue along the production line. Moreover, employing operators for continuous inspection is expensive and time-consuming.

For these reasons, an automatic visual inspection system is proposed. With the objective of inspecting every bottle immediately after the capping process, ensuring that the safety seal is present and intact without slowing down the production line.



(a) Real cap with seal (b) CAD design of the cap (c) Real cap without seal

Figure 1.1: Comparison between the technical design of the cap and physical samples (standard vs. defective).

1.3 Challenge and Motivation

The development of an automatic inspection system for this application presents several technical and scientific challenges. Traditional supervised Deep Learning approaches, such as Convolutional Neural Networks (CNNs) for image classification or object detection algorithms, require large and balanced datasets. In practice, they need thousands of labeled images of both normal and defective samples to achieve good performance.

However, in a well-controlled industrial production line, defects are rare events. As a result, it is

very difficult, if not impossible, to collect a complete dataset that includes all the possible variations of broken or missing safety seals. In fact, defects can appear in many different and unpredictable ways.

So in our industrial scenario, collecting defective samples proved to be particularly difficult because production is optimized to minimize failures. For this reason, supervised learning was considered unsuitable, in response this work adopts **Unsupervised Anomaly Detection** techniques. These models are trained only on normal, defect-free samples, allowing them to learn the characteristics of a correct product. During inference, any significant deviation from this learned representation is detected and classified as an anomaly.

Another important challenge concerns the position and orientation of the safety seal, which is located on the side of the cap. Since the cap can be randomly rotated while moving along the production line, the inspection system must be able to observe the seal from every angle, requiring a 360° view.

A common industrial solution is to use multiple standard cameras placed around the product. Although effective, this approach increases the hardware cost, the complexity of the system, and the effort required for camera calibration. A more elegant solution is to use a **pericentric lens** (Figure 1.2), which can simultaneously capture both the top surface and the entire lateral surface of the cap in a single 2D image. This significantly simplifies the inspection setup while providing complete visual coverage of the safety seal.



Figure 1.2: Pericentric Lens

1.4 Objectives

The primary objective of this thesis is to develop and evaluate a deep learning-based visual inspection pipeline capable of detecting defects in the safety seals of vinegar bottle caps in real time.

The specific objectives of this work are:

1. Design an image acquisition pipeline (Figure 1.3) based on a pericentric vision system, studying its optical characteristics and defining a preprocessing strategy suitable for the inspection of cylindrical objects.

2. Collect and organize a novel industrial dataset of normal and defective bottle caps, ensuring data quality and preparing it for training and evaluation.

3. Implement a state-of-the-art anomaly detection model based on the **PatchCore** architecture using a pretrained backbone, leveraging its high performance in industrial visual inspection tasks. The model produces an anomaly score for defect classification and an anomaly map that highlights the regions where defects are detected.

4. Compare the proposed solution with other anomaly detection methods, such as **PaDiM** and **EfficientAD**, evaluating both detection performance (AUROC, Precision, Recall, and F1-score) and computational efficiency, including inference time, to satisfy the constraints of the production line.

5. Analyze the robustness of the proposed pipeline under realistic industrial conditions, discussing its strengths, limitations, and suitability for deployment on a production line.

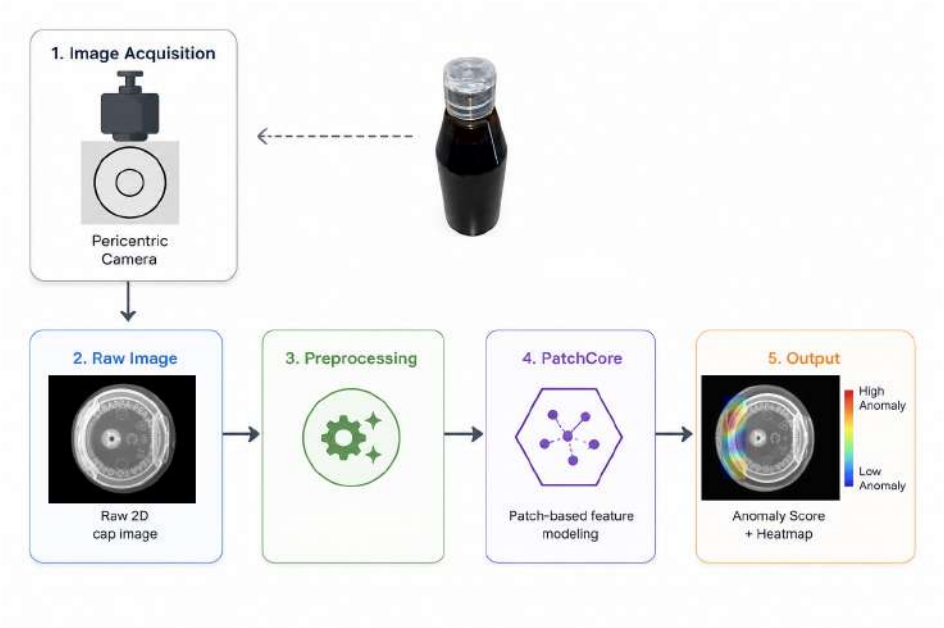


Figure 1.3: Project Pipeline

1.5 Contributions

The main contributions of this thesis can be summarized as follows:

1. The design of an end-to-end visual inspection pipeline for the automatic detection of defects in vinegar bottle cap safety seals, specifically developed to satisfy the requirements of an industrial production environment.

2. The creation of a novel industrial dataset acquired through a pericentric vision system, including both normal and defective samples, together with a preprocessing strategy tailored to the unique geometry of pericentric images.

3. An extensive experimental evaluation of state-of-the-art anomaly detection methods, providing a detailed comparison in terms of detection performance, computational efficiency, and suitability for real-time industrial deployment.

4. An analysis of the advantages and limitations of pericentric imaging for anomaly detection on cylindrical objects, highlighting the challenges introduced by this acquisition technique and its impact on deep learning-based inspection systems.

1.6 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2 – Background**: Introduces the theoretical foundations of industrial computer vision, with particular emphasis on pericentric imaging, deep learning for visual inspection, and anomaly detection. The chapter also provides a detailed description of the PatchCore algorithm and its underlying principles.

- **Chapter 3 – State of the Art**: Reviews the most relevant anomaly detection methods proposed in the literature for industrial visual inspection, highlighting their strengths, limitations, and applicability to real-world manufacturing scenarios.

- **Chapter 4 – Industrial Case Study**: Presents the industrial application addressed in this thesis, describing the production process, the acquisition system based on pericentric imaging, and the characteristics of the collected dataset.

- **Chapter 5 – Proposed Method**: Describes the proposed visual inspection pipeline, including the image preprocessing stage, the anomaly detection framework, and the overall system design adopted for the industrial application.

- **Chapter 6 – Experimental Results**: Presents the experimental evaluation of the proposed approach, including performance comparisons with other state-of-the-art methods, quantitative and qualitative analyses, and a discussion of the obtained results.

- **Chapter 7 – Conclusions and Future Work**: Summarizes the main findings of this work, discusses its limitations, and outlines possible directions for future research and industrial improvements.

2. Background

2.1 Industrial computer Vision

The paradigm of Industry 4.0 has heavily increased the concept of "Zero-Defect Manufacturing", an idea in which production processes are continuously monitored and optimized to eliminate waste, error and guarantee product quality. In this contest Industrial machine vision has emerged as a primary technology. Unlike human operators, who are susceptible to fatigue and physical limitations at high production speed, automated vision system ensure consistent, and objective inspection capabilities in real-time. A standard industrial computer vision application is not only a software algorithm, but a complex synergy of hardware and software working syncronously. The typical inspection pipeline can be splitted in four sequantial stages: Image Acquisition, Preprocessing, Feauta extraction and analiysys, decision Making. The first stage is fundamengtal for the success of the entire pipeline, becausa clearly no software algorithm is caoabale of detecting defect that are not physically visible in the captured image.

To have an ideal image acquisition, a machine vision system relies on four primary hardaware components [16]:

- Illumination System: Lighting is arguably the most critical variable in industrial vision. Its purpose is to maximize the contrast between the defect and the background while minimizing shadows, reflections, and ambient light interference. In high-speed manufacturing, strobe lighting is often synchronized with the camera to freeze the motion of the moving product.

- Optics (Lenses): The lens collects the light reflected from the object and focuses it onto the camera sensor. The choice of the lens determines the Field of View (FOV), Depth of Field (DOF), and the optical perspective. Specialized lenses are often required to inspect complex geometries without introducing severe perspective distortions.

- Sensor and Camera: The camera captures the focused light and converts it into a digital signal. Industrial cameras are generally categorized into line-scan (which build the image pixel-line by pixel-line as the object moves) and area-scan cameras (which capture a full 2D

matrix of pixels at once). For discrete objects like bottle caps, high-speed area-scan cameras combined with global shutters are typically employed to prevent motion blur.

- Processing Unit: The computational hardware responsible for executing the inspection algorithms. Depending on the latency requirements and the complexity of the AI models (such as Deep Neural Networks), the processing unit often incorporates dedicated hardware accelerators like GPUs (Graphics Processing Units) or edge-AI devices to ensure that the inference time remains well below the production cycle time.

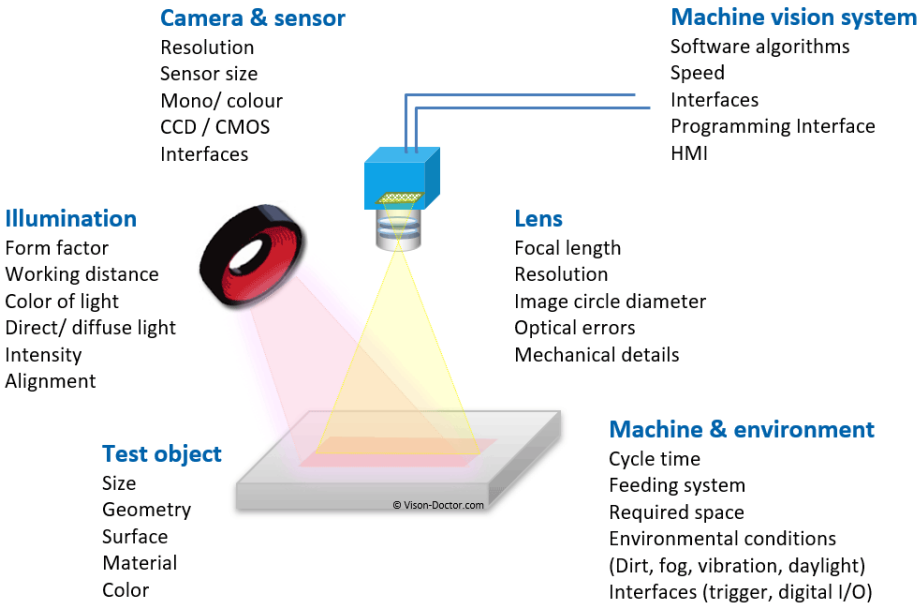
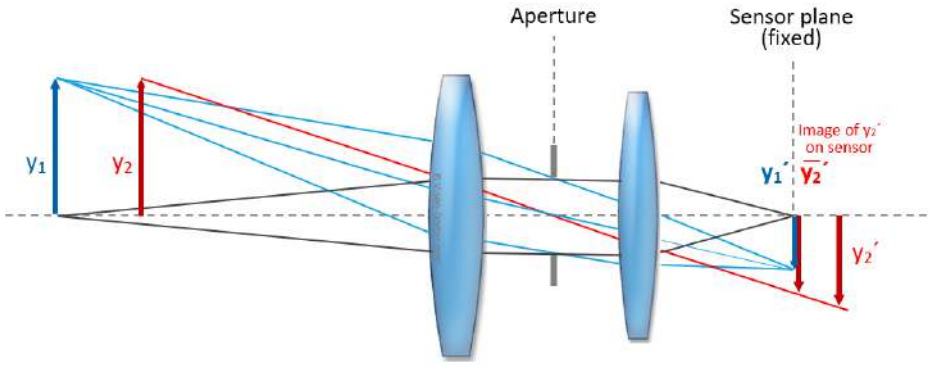


Figure 2.1: Primary hardware components

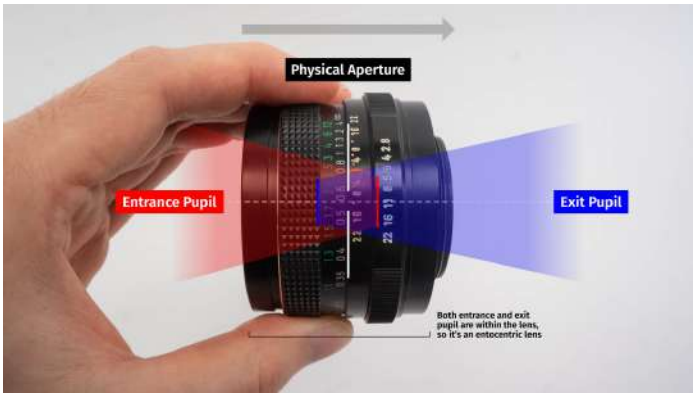
2.2 Optical Inspection System

2.2.1 The Limit of Standard Optics

In machine vision, most inspection systems use *entocentric* lenses, Figure2.2. In this type of optical system, the entrance pupil(optical centre) is located inside the lens, so the field of view increases with the distance from the lens. This creates the same perspective as human vision, where objects closer to the lens appear larger than objects farther away. While this approach is effective for inspecting flat surfaces, it has limitations when inspecting cylindrical objects such as bottle caps. In these cases, the camera can capture only the top surface, while the lateral surface, which contains critical features such as the safety seal, remains completely hidden due to occlusion. Traditional industrial approaches rely on either multi-camera arrays or mechanical actuators to rotate the object during inspection. However, both solutions have significant drawbacks. Multi-camera systems greatly increase the hardware cost and require complex software for multi-view calibration and image stitching. On the other hand, rotating the product introduces additional mechanical and software complexity, as the rotation must be precisely controlled and synchronized with image acquisition. This also reduces the throughput and flexibility of the production line.



(a) Entocentric lens scheme



(b) Real entocentric lens

Figure 2.2: Entocentric lens

2.2.2 Optical Principle of Pericentric Lenses

To address the challenge of 360-degree later inspection without moving parts or multiple cameras, this thesis employs a percinetric lens. The fundamental physical differences lies in the behaviour of the light rays and the position of the entrance pupil [12]. Unlike an entocentric lens, a pericentric lens provides an inverted perspective. Its optical design places the entrance pupil virtually in front of the lens, enabling the camera to observe the object from multiple viewing angles at the same time. Consequently, light reflected from both the top surface and the lateral surfaces reaches the sensor simultaneously, allowing a single image to capture features that would normally require multiple cameras or object rotation as shown in Figure 2.3. In practice, rays reflected from the side walls of

the object are directed toward the sensor under different viewing angles instead of being blocked by the object's geometry. As a result, the camera can image the object's circumference together with its top surface in a single exposure. The height of the visible lateral region depends on the lens design and the working distance, but a large portion of the object's side can typically be inspected without moving either the camera or the object.

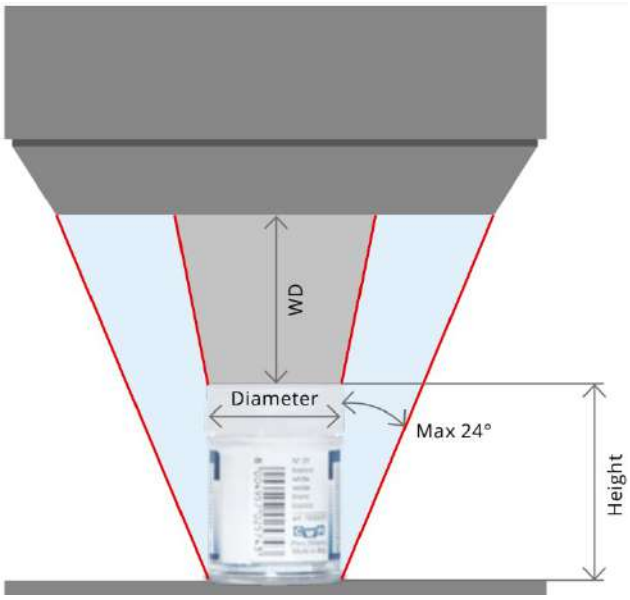


Figure 2.3: Pericentric lens scheme

2.2.3 Image Formation and Geometric mapping

The most important consequence of using a pericentric lens in a Deep Learning inspection system is the way the image is formed. Unlike an entocentric lens, a pericentric lens produces an inverted perspective: objects that are physically closer to the lens appear smaller, while objects farther away appear larger.

This projection can be seen as a non-linear mapping that transforms the three-dimensional cylindrical surface of the object into a two-dimensional image. As a result, different parts of the object are projected onto specific regions of the camera sensor [18]:

- **Top Surface:** projected onto the central circular region of the image.
- **Lateral Surface:** projected around the top surface as a concentric circular ring, where the cylindrical side wall is unfolded into a continuous annular region.

The width of this circular ring is related to the visible height of the object's lateral surface, as illustrated in Figure 2.4. Consequently, features such as the safety seal become visible in a single image without rotating the object. However, these features are affected by strong radial distortion. Depending on their angular position around the object, straight edges appear curved and may also be stretched or compressed.

Although this representation is not intuitive for human observation, it is well suited for Convolutional Neural Networks. CNNs can automatically learn robust spatial features from the distorted images, making them able to recognize defects despite the geometric deformation introduced by the pericentric projection.

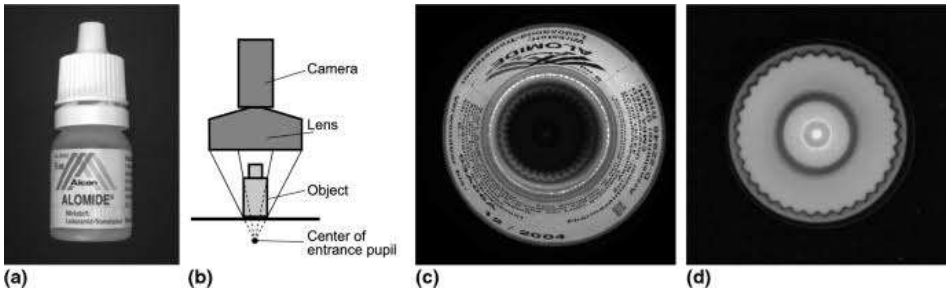


Figure 2.4: Pericentric lens example

2.3 Triggering and Image Synchronization

2.3.1 Precision Timing in High-Speed Lines

In continuous manufacturing environments, such as the vinegar glaze bottling line operating at a throughput of 3,600 units per hour, products do not stop for inspection. They move continuously along a conveyor belt. Consequently, capturing a perfectly sharp and well-positioned image requires precise synchronization between the physical arrival of the object, the camera exposure, and the illumination system. This synchronization is achieved through a mechanism known as a trigger.

2.3.2 HW and SW Triggering

In machine vision, image acquisition can be triggered by either software or hardware. A software trigger sends a command from a PC or a PLC to the camera through a communication protocol such as GigE Vision [13]. However, network delays and operating system latency can introduce timing variations, making software triggering unsuitable for high-speed inspection.

For this reason, industrial inspection systems usually use hardware triggering. A sensor, such as a photoelectric sensor or a laser sensor, is placed on the production line just before the inspection station. When the bottle passes in front of the sensor, it generates an electrical pulse that is sent directly to the camera. This allows the camera to capture an image with microsecond-level timing accuracy.

2.3.3 Critical Impact on Pericentric Geometry

While the use of triggering is standard in industrial vision systems, its timing precision becomes critical when combined with a pericentric lens. As described in the previous section, pericentric optics project the lateral walls of the cap into a well-defined circular shape.

If the trigger is activated even a fraction of a second too early or too late, the cap may not be perfectly aligned with the optical axis of the lens. This misalignment causes the projected circular ring in the 2D image to become distorted, asymmetric, and off-center. Since anomaly detection models rely on a consistent spatial distribution of nominal features to identify defects, such variations in the projection can significantly affect performance and lead to incorrect predictions.

For this reason, a highly deterministic hardware trigger is a key requirement for the stability and reliability of the subsequent deep learning pipeline.

2.3.4 System Synchronization Architecture

To summarize the physical acquisition constraints discussed in the previous section, Figure 2.5 illustrates the hardware synchronization architecture implemented in the inspection system. The acquisition process operates as a deterministic closed-loop sequence designed for **on-the-fly** inspection. As each bottle moves along the production line, a precision photoelectric sensor detects when it reaches the focal center of the pericentric camera lens. As soon as the bottle is detected, the sensor sends an electrical trigger pulse through a General-Purpose Input/Output (GPIO) connection directly to the camera. This hardware-based trigger bypasses operating system delays, ensuring precise and repeatable image acquisition. The trigger simultaneously starts the camera exposure and sends a synchronization signal to the lighting controller. The controller then generates a high-intensity strobe pulse that is perfectly synchronized with the camera exposure time. This strobe illumination optically **freezes** the motion of the bottle cap, completely eliminating motion blur while the product continues moving at full production speed without stopping or slowing down. Finally, the sharp and perfectly centered unwrapped image is transferred to the main processing unit through a high-speed data interface, where the deep learning pipeline starts the anomaly detection process. This precise hardware synchronization is a fundamental requirement for ensuring the spatial consistency of the acquired images, which is essential for the reliable operation of the subsequent PatchCore algorithm.

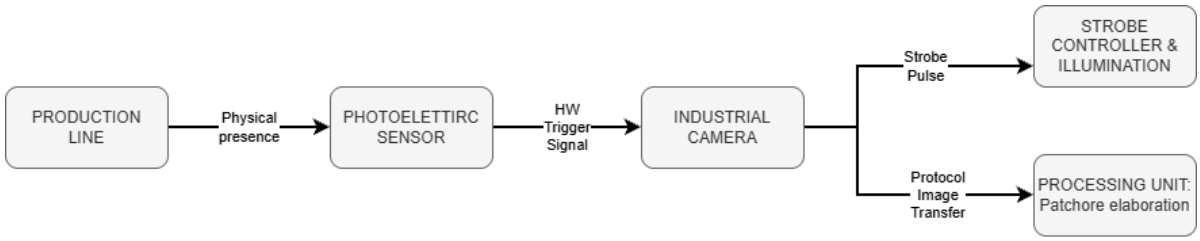


Figure 2.5: Trigger and Camera Sensor

2.4 Deep Learning for Visual Inspection

2.4.1 From Hand-crafted Feature to Deep Representation

Traditional industrial computer vision systems relied heavily on hand-crafted feature extractors, such as SIFT descriptors and edge detectors, combined with rule-based algorithms or shallow machine learning classifiers. While these approaches can be effective for simple and highly structured inspection tasks, they often struggle to handle complex and non-linear geometric distortions, such as those present in bottle seal images acquired with a pericentric lens.

The emergence of Deep Learning, and in particular **Convolutional Neural Networks (CNNs)**, represented a major breakthrough in the field. Instead of requiring engineers to manually define defect-related features, CNNs automatically learn hierarchical visual representations directly from raw image data. Through successive convolutional layers, non-linear activation functions, and pooling operations, CNNs can extract robust features even from heavily distorted pericentric images [14].

However, in this work the CNN is not used in the conventional way. Rather than being trained as a classifier to directly identify defects, it is employed as a feature extractor within the PatchCore anomaly detection framework. As discussed in the following sections, the CNN provides high-level visual features that PatchCore uses to model the appearance of normal samples and detect deviations from this learned representation.

2.4.2 Hierarchical Feature Extraction

To understand how state-of-the-art anomaly detection models operate, it is important to consider a CNN not only as a classifier but also as a hierarchical feature extractor.

As an image propagates through the network, the extracted representations become progressively more abstract and semantically meaningful. This process is closely related to the concept of the **receptive field**, which represents the region of the input image that influences the activation of a neuron. In general, a larger receptive field allows the network to capture more contextual information, leading to more abstract and semantic feature representations Figure 2.6.

1. **Shallow Layers (Early Stages):** These layers have a small receptive field and act as low-level

feature detectors. They capture basic visual patterns such as edges, corners, colors, and simple textures. They preserve a high spatial resolution but contain limited semantic information.

2. **Deep Layers (Late Stages):** These layers have a much larger receptive field, enabling them to capture high-level semantic information, such as the overall shape and structure of the safety seal. However, because of repeated pooling operations and strided convolutions, they have a much lower spatial resolution. As a result, they are very good at recognizing *what* an object is, but less accurate at determining *where* it is located.

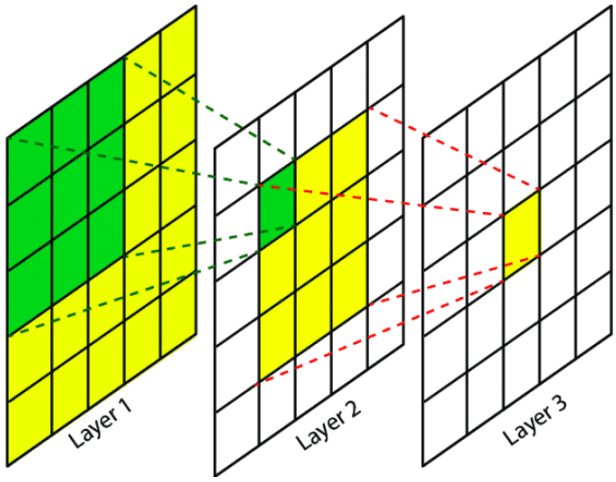


Figure 2.6: Receptive Field through Layers

2.4.3 Transfer Learning in Industrial Environments

Training a deep CNN from scratch requires a very large amount of labeled data. In an industrial manufacturing environment, such as the vinegar bottling line considered in this work, nominal (defect-free) samples are abundant, whereas defective samples are rare, diverse, and often unpredictable. As a result, training a network from scratch is impractical because of the severe class imbalance and the limited availability of defect examples.

To overcome this limitation, modern industrial computer vision widely adopts **Transfer Learning**. Instead of training a neural network from random initialization, a model that has already been trained on a large-scale dataset, such as *ImageNet*, is reused. ImageNet contains millions of images belonging to one thousand object categories, allowing the network to learn rich and general visual representations.

During pre-training, the CNN learns to detect visual patterns that are common across many different objects. The first layers typically learn universal low-level features, such as edges, corners, textures, and color transitions, while deeper layers progressively capture more complex shapes, object parts, and semantic structures. Although these features are learned for an image classification task, they are not specific to the ImageNet categories and can be transferred to many other vision problems. This observation is the key idea behind Transfer Learning: a network that has learned to extract meaningful visual features from a large and diverse dataset can also produce informative feature representations for images belonging to completely different domains. Therefore, instead of learning visual features from scratch, the pre-trained network can be reused as a powerful feature extractor, requiring much less task-specific data while achieving significantly better performance and faster convergence [6].

In the context of Unsupervised Anomaly Detection, the pre-trained network is not used to classify the image. Instead, its fully connected classification layers are discarded, and the convolutional backbone is frozen (its weights are not updated). The network is strictly utilized as a universal feature extractor. By passing an industrial image through this frozen network, it outputs a feature maps that provide a rich and detailed, descriptive representation of the product. This pre-trained representation forms the exact foundation upon which the PatchCore algorithm builds its memory bank.

2.5 Anomaly Detection

Anomaly Detection is a fundamental task in machine learning and computer vision. It aims to identify rare samples that differ significantly from the normal data distribution. Unlike traditional classification, where all classes are explicitly defined during training, anomaly detection focuses on learning the characteristics of normal data and identifying any sample that deviates from this learned representation.

In industrial visual inspection, the normal data correspond to defect-free products, whereas anomalies represent structural or cosmetic defects. In the application considered in this work, anomalies include bottle caps with broken, damaged, or missing safety seals.

The main challenge of anomaly detection is to accurately model the distribution of normal samples. Since defective products are often rare, highly variable, or even unavailable during training, the model must learn a robust representation of normality using only nominal data. During inference, any sample whose visual characteristics differ significantly from this learned representation is considered anomalous and is therefore flagged as a potential defect.

2.5.1 Learning Paradigms: Supervised, Semi-Supervised, and Unsupervised

The approach to solving an Anomaly Detection problem heavily depends on the availability of labeled data. Machine learning strategies for AD can be broadly categorized into three main paradigms:

1. **Supervised Anomaly Detection**: This approach requires a fully labeled dataset containing a substantial and balanced number of both normal and anomalous samples. While models trained this way (e.g., standard image classifiers) achieve high accuracy, this paradigm is highly impractical in real-world manufacturing. Anomalies are rare by definition, and it is impossible to collect samples of every conceivable way a product might fail or break.

2. **Semi-Supervised Anomaly Detection**: In this scenario, the training dataset consists mainly of labeled normal samples together with a small number of labeled anomalous examples. The model uses the available anomaly samples to better define the boundary between normal and abnormal data, improving its ability to detect defects compared with methods trained exclu-

sively on normal samples. Although the inclusion of labeled anomalies generally increases detection accuracy, the approach still has an important limitation. In industrial environments, defects are often highly diverse and continuously evolving. Since only a small subset of defect types is available during training, the model may overfit to these known anomalies and struggle to recognize previously unseen defects. Consequently, its generalization capability strongly depends on how representative the labeled anomalous samples are of the defects that will be encountered during deployment.

3. **Unsupervised Anomaly Detection (UAD)**: This approach assumes no access to anomalous labels during training. The model is tasked with finding the intrinsic structure of the data independently. In industrial applications, UAD is typically implemented as One-Class Classification (OCC), where the training set is strictly composed of normal, defect-free images. This is the most viable strategy for industrial quality inspection, as normal samples are abundant and inexpensive to collect.

2.5.2 The One-Class Learning Paradigm in Industry

As introduced in the previous sections, industrial quality inspection is characterized by a severe class imbalance. During normal production, the vast majority of manufactured products are defect-free, while defective samples occur only rarely. Furthermore, defects are often highly heterogeneous: the same component can fail in many different and unpredictable ways due to manufacturing variations, material degradation, mechanical damage, or unexpected process failures. For example, in the proposed bottle inspection system, a safety seal may be completely missing, partially detached, broken at different locations, folded, scratched, or deformed in ways that have never been observed before.

Collecting a sufficiently large and representative dataset containing every possible defect type is therefore practically impossible. Even if some defec tive samples are available, they usually represent only a very small subset of all anomalies that may occur during production. As a consequence, conventional supervised classification approaches, which require examples of every class during training, are generally unsuitable for industrial anomaly detection.

For these reasons, modern industrial inspection systems have largely adopted the **Unsupervised**

(One-Class) Anomaly Detection paradigm. Instead of learning to distinguish between normal and defective products, the model is trained exclusively using a dataset of nominal images,

$$D_{\text{train}} = \{x_1, x_2, \ldots, x_n\},$$

where every sample represents a correctly manufactured product. The objective is not to learn the appearance of defects, but rather to build an accurate representation of what a normal product looks like, we want the model to learn the perfection.

During training, the model learns the underlying distribution of normal data. Intuitively, this distribution represents the region of the feature space occupied by defect-free products. Since all training samples are visually similar, their extracted feature representations tend to cluster together, forming a compact distribution that describes normal operating conditions.

During inference, a previously unseen sample x_{test} is first transformed into the same feature space. The model then evaluates how well this new representation fits within the learned normal distribution. If the sample is sufficiently similar to the nominal data, it is classified as normal. Conversely, if its feature representation deviates significantly from the learned manifold, the sample is assigned a high anomaly score and is classified as defective.

One-Class Anomaly Detection offers an important practical advantage in industrial environments. Because the model learns only from normal products, it is not limited to recognizing defect types observed during training. Instead, it can detect any abnormal visual pattern that differs sufficiently from the learned representation of normality, including completely new and previously unseen defects. This capability makes one-class methods particularly suitable for automated quality inspection, where new defect types may appear unexpectedly throughout the lifetime of a production line.

Among the various one-class anomaly detection methods proposed in recent years, PatchCore has demonstrated state-of-the-art performance by combining powerful feature representations extracted from pre-trained convolutional neural networks with an efficient memory-based nearest-neighbor search. The following sections describe the principles behind this approach in detail.

2.5.3 Reconstruction-Based Methods

Historically, the most common approach to UAD has been reconstruction-based, heavily relying on Autoencoders (AEs) [2], Variational Autoencoders (VAEs) [9], and Generative Adversarial Networks (GANs) [10].The core assumption of these architectures is that a model trained exclusively on normal data, often forced to compress information through a lower-dimensional bottleneck, will successfully reconstruct nominal patterns but will fail to reconstruct anomalous regions. The anomaly score is commonly derived from the reconstruction error, for example using the L_2 distance between the input image x and the reconstructed image $\hat{x}$:

$$\text{Anomaly Score} = L_2 = \|x - \hat{x}\|_2 = \sqrt{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$$

While intuitive, reconstruction-based models face significant limitations in complex industrial environments. They often suffer from either poor image fidelity (resulting in noisy reconstructions and high false-positive rates) or the "identity shortcut," where an over-parameterized network generalizes too well and successfully reconstructs the defect itself, leading to severe false negatives.

Unlike conventional Autoencoders, Generative Adversarial Networks introduce a competitive learning framework composed of two neural networks: a Generator (G) and a Discriminator (D). The Generator is responsible for producing synthetic images or reconstructions that resemble normal training samples, while the Discriminator learns to distinguish between real normal images and generated ones [10]. During training, the Generator progressively improves its ability to create realistic samples, while the Discriminator becomes better at identifying artificial images. This adversarial competition forces the Generator to learn a more accurate representation of the visual distribution of normal data. In the context of anomaly detection, the Generator is trained only on defect-free products, meaning that the learned distribution should ideally correspond to the manifold of normal appearance. During inference, an anomalous image is expected to fall outside the learned normal distribution. The Generator should therefore have difficulty reproducing the anomalous characteristics, resulting in a larger difference between the input image and the generated output. Some GAN-based anomaly detection methods also exploit the latent space by searching for the closest latent representation capable of generating an image similar to the test sample. The distance between the original image and its generated counterpart is then used as an anomaly indicator.

However, despite their powerful image generation capabilities, GAN-based approaches present additional challenges. The adversarial training process is notoriously difficult to stabilize because the Generator and Discriminator must remain balanced throughout training. Problems such as mode collapse, where the Generator produces only a limited variety of outputs, can reduce the ability of the model to represent the full diversity of normal samples. Moreover, GANs are optimized to generate visually realistic images rather than to preserve exact pixel-level correspondence. In industrial inspection scenarios, where defects can be extremely small and localized, a visually plausible reconstruction may still contain inaccurate details or remove subtle anomalies. This makes the anomaly score less reliable, especially for fine-grained defects such as small cracks, scratches, or partial damages.

2.5.4 Feature-Embedding and Memory-Based Methods

To overcome the limitations of pixel-level reconstruction, state-of-the-art industrial UAD methods have shifted towards feature-embedding paradigms. Instead of attempting to redraw the image, these methods leverage the robust, semantic representations of pre-trained CNNs (as discussed in Section 2.4.3).These algorithms map normal images into a high-dimensional feature space, modeling normality using statistical or distance-based metrics on these embeddings. Notable sub-categories include:

1. **Normalizing Flows**: (FastFlow) which employ invertible neural networks to map the complex feature distribution into a tractable standard Gaussian distribution, computing exact likelihoods.

2. **Knowledge Distillation**: (EfficientAD) where a "Student" network is trained to mimic the feature outputs of a pre-trained "Teacher" network only on normal data. A divergence in their outputs during inference indicates an anomaly.

3. **Memory Bank / k-NN Approaches**: (e.g., PaDiM, PatchCore) which build a dictionary (or memory bank) of nominal features extracted from the training set. During inference, the anomaly score is computed by calculating the distance between the test image features and their nearest neighbors in the memory bank.

Memory-based methods currently dominate industrial benchmarks (MVTec AD dataset) due to their remarkable spatial accuracy and lack of complex adversarial training phases. This thesis specifically adopts and optimizes the PatchCore algorithm, which represents the current gold standard in this category.

2.6 PatchCore

2.6.1 PatchCore: Theoretical Foundations and Mathematical Formulation

This section presents the mathematical formulation of the PatchCore algorithm, proposed by Roth *et al.* (2022) in the paper *"Towards Total Recall in Industrial Anomaly Detection"* [15]. PatchCore is the anomaly detection method adopted as the core algorithm of the inspection system developed in this work.

The method is described following its logical workflow, which can be divided into four conceptually distinct stages. First, local feature embeddings are extracted from the input image using a pre-trained convolutional neural network (CNN). Next, these features are stored in a *memory bank* that represents the distribution of normal samples. To improve computational efficiency while preserving the diversity of the stored features, the memory bank is then reduced through a *coreset subsampling* procedure. Finally, during inference, the anomaly score is computed by comparing the feature representations of the test image with those stored in the memory bank. This comparison produces both an image-level anomaly score, indicating whether the inspected product is anomalous, and a pixel-level anomaly map, which localizes the defective regions within the image.

2.6.2 Position Within the Landscape of Anomaly Detection Methods

As discussed in the previous section, industrial visual anomaly detection methods can be broadly categorized into three main families: reconstruction-based methods, student-teacher methods, and memory-based feature embedding methods. PatchCore belongs to the last category and represents one of the most effective feature-based approaches proposed in recent years [15]. Unlike reconstruction-based methods, which detect anomalies through reconstruction errors, or student-teacher approaches, which rely on difference between the feature representations of two neural networks, PatchCore identifies anomalies by comparing deep feature embeddings extracted from normal and test images. During training, these embeddings are stored in a memory bank that models the distribution of normal data. During inference, the embeddings extracted from a test image are compared with those stored in the memory bank using a nearest-neighbor search in the feature space. Regions whose feature representations are sufficiently different from all stored normal embeddings are classified as anomalous. PatchCore can be considered as an evolution of earlier feature-based methods, particularly SPADE and PaDiM. While these approaches also exploit deep features extracted from pre-trained convolutional neural networks, PatchCore introduces two key innovations that significantly improve both detection performance and computational efficiency: the extraction of robust, locally aware local feature representations, and an efficient memory bank construction based on greedy coreset subsampling [15].

The PatchCore pipeline consists of four main stages Figure 2.7 . First, patch-level features are extracted from intermediate network layers, second tose patch are stored in a memory bank. third, a coreset subsampling strategy reduces the size of the memory bank while preserving its diversity. Finally, anomaly detection and localization are performed by comparing test features with the stored normal representations [15].

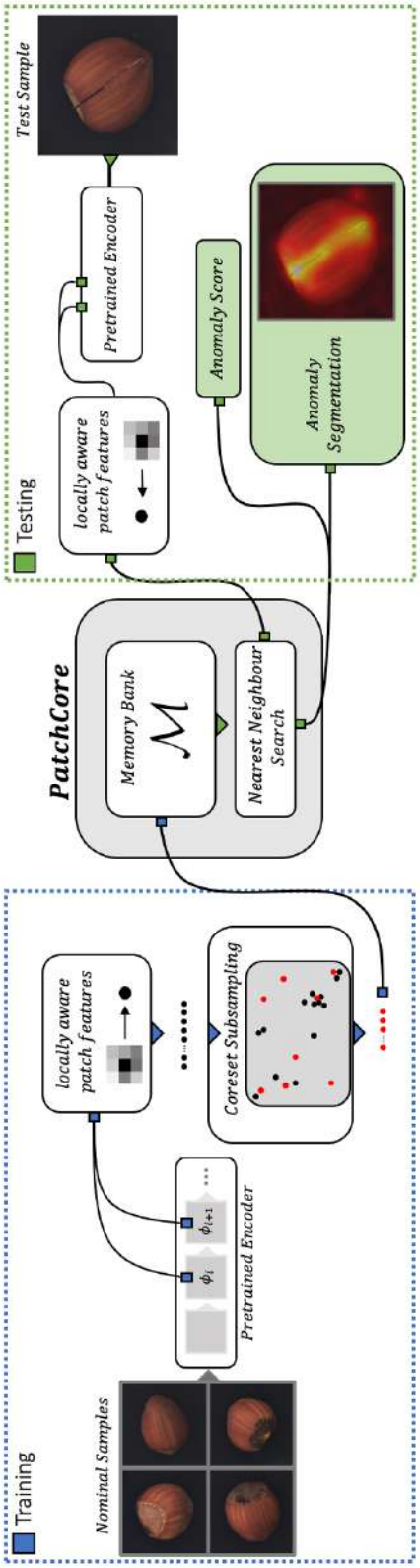


Figure 2.7: Patchore pipeline

2.6.3 Stage 1: Feature Extraction

The first stage of the algorithm consists of extracting visual features using a deep convolutional neural network pre-trained on a large-scale dataset, ImageNet. The network is used exclusively as a fixed feature extractor, meaning that all its parameters remain frozen during both training and inference. In the implementation adopted in this work, the backbone network is a WideResNet-50.

Given an input image x_i from a dataset $\mathcal{X}$, let $\phi_{i,j} = \phi_j(x_i)$ denote the feature map produced by the j-th level of the pretrained network ϕ. Following common convention for ResNet-like architectures, j indexes feature maps with $j \in \{1, 2, 3, 4\}$, indicating the final output of the respective spatial resolution blocks [15].

Using only the features from the last layer of the network, as done in earlier nearest-neighbour methods, has two main disadvantages. First, it loses local information, which is important when the type of anomaly at test time is unknown. Second, the deepest ImageNet features are mainly designed for natural image classification, making them less suitable for industrial anomaly detection tasks [15]. To address these issues, PatchCore builds a memory bank using patch-level features extracted from intermediate (mid-level) layers of the network. For ResNet-based architectures, these are typically layers $j \in 2, 3$ [15].

Assume that the feature map $\phi_{i,j} \in \mathbb{R}^{c^* \times h^* \times w^*}$ is a three-dimensional tensor with depth c^*, height h^*, and width w^*. We denote by $\phi_{i,j}(h, w) = \phi_j(x_i)_{h,w} \in \mathbb{R}^{c^*}$ the c^*-dimensional feature vector at the spatial position (h, w), where $h \in \{1, \ldots, h^*\}$ and $w \in \{1, \ldots, w^*\}$ as shown in Figure 2.8.

Each spatial location of a feature map corresponds to a receptive field in the original image; however, PatchCore does not use the raw feature vector at a single location directly. The following three subsections explain, how a single location is turned into a *locally-aware* feature, how the two layers are fused into one multi-scale representation, and how the full set of per-image features is finally collected.

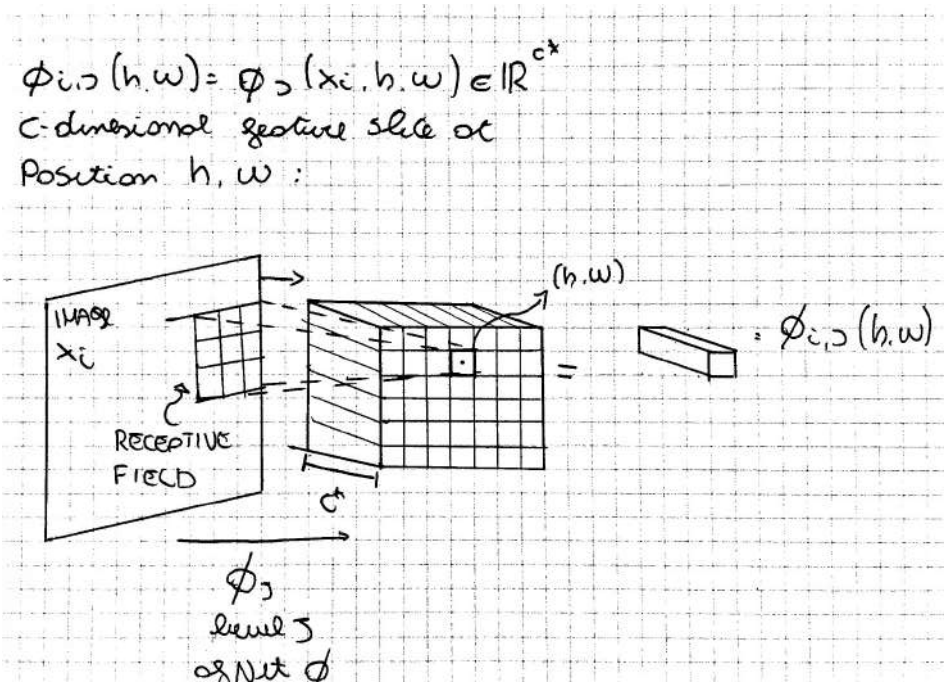


Figure 2.8: feature extraction

Locally-Aware Patch Features

For each spatial position (h, w) of layer j, the locally aware feature is computed by aggregating the feature vectors within a local neighborhood of size $p \times p$ as shown in Figure , using an adaptive average pooling operation [15]:

$$\phi_{i,j}\left(N_p^{(h,w)}\right) = f_{\text{agg}}\left(\left\{\phi_{i,j}(a,b) \mid (a,b) \in N_p^{(h,w)}\right\}\right),$$

where

$$N_p^{(h,w)} = \left\{(a,b) \,\middle|\, a \in \left[h - \left\lfloor \tfrac{p}{2} \right\rfloor, \ldots, h + \left\lfloor \tfrac{p}{2} \right\rfloor\right],\ b \in \left[w - \left\lfloor \tfrac{p}{2} \right\rfloor, \ldots, w + \left\lfloor \tfrac{p}{2} \right\rfloor\right]\right\}$$

denotes the square neighborhood of side length p centered at location (h, w), and f_{agg} is the aggregation function – adaptive average pooling in the case of PatchCore [15]. This local aggregation provides moderate robustness against small translations, image noise, and minor misalignments, without losing spatial resolution or the usability of the feature maps [15]. More importantly, it allows each feature embedding to describe not only an individual point of the feature map but also the visual context of the surrounding receptive field, which is more representative of the typical size

of industrial defects.

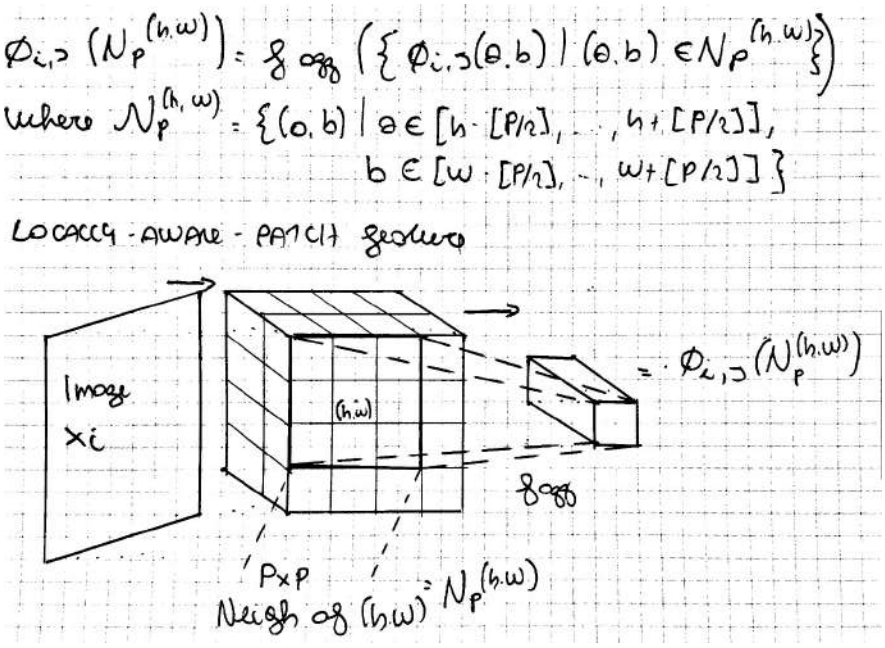


Figure 2.9: Locally aware feature

Multi-Scale Fusion

Features extracted from shallow layers preserve fine spatial details but contain limited semantic information, whereas deeper layers encode richer semantic representations at the cost of lower spatial resolution. To retain the generality of the features used as well as spatial resolution, PatchCore uses only two intermediate feature hierarchies, j and $j+1$ [15]. This is achieved by computing $\mathcal{P}_{s,p}(\phi_{i,j+1})$ and aggregating each of its elements with the corresponding patch feature at the lowest (i.e. highest-resolution) hierarchy level used, which is obtained by using bilinear upsampling $\mathcal{P}_{s,p}(\phi_{i,j+1})$ so that its spatial size matches that of $\mathcal{P}_{s,p}(\phi_{i,j})$ [15].

Once the two layers are fused in this way, the layer index j is no longer needed to identify a patch: each spatial position now holds a single combined embedding.

Patch Collection

For a feature map tensor $\phi_{i,j}$, its locally aware patch-feature collection $\mathcal{P}_{s,p}(\phi_{i,j})$ is defined, following the original paper, as [15]

$$\mathcal{P}_{s,p}(\phi_{i,j}) = \left\{ \phi_{i,j}\left(N_p^{(h,w)}\right) \;\middle|\; h, w \bmod s = 0, \;\; h < h^*, \;\; w < w^*, \;\; h, w \in \mathbb{N} \right\},$$

with the optional use of a striding parameter s, which the authors set to 1 except in dedicated ablation experiments [15]. Here h^* and w^* denote the height and width of the feature map grid, bounding the range of valid positions.

In particular $\mathcal{P}_{s,p}$ is defined per layer j: in practice it is evaluated once for j and once for $j+1$, and the two resulting collections are combined only afterwards via the multi-scale fusion step described above.

2.6.4 Stage 2: Memory Bank Construction

For all nominal training samples $x_i \in \mathcal{X}_N$, where $\mathcal{X}_N$ denotes the set of all defect-free images available at training time, the PatchCore memory bank $\mathcal{M}$ is then simply defined as [15]

$$\mathcal{M} = \bigcup_{x_i \in \mathcal{X}_N} \mathcal{P}_{s,p}(\phi_j(x_i)).$$

As the size of $\mathcal{X}_N$ grows, $\mathcal{M}$ becomes exceedingly large, and with it both the inference time required to evaluate novel test data and the storage required to hold the memory bank [15]. PatchCore addresses this issue introducing a great novelty: **greedy coreset subsampling.**

2.6.5 Stage 3: Coreset Subsampling

Conceptually, coreset selection aims to find a subset $\mathcal{S} \subset \mathcal{A}$ such that solutions to a problem posed over $\mathcal{A}$ can be approximated, more closely and more quickly, by solutions computed over $\mathcal{S}$ [15]. Because PatchCore relies on nearest-neighbour computations at inference time, a *minimax facility location* coreset is used, so as to guarantee approximately similar coverage of the coreset $\mathcal{M}_C$ in patch-level feature space compared to the original memory bank $\mathcal{M}$ [15]:

$$\mathcal{M}_C^* = \arg\min_{\mathcal{M}_C \subset \mathcal{M}} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_C} \|m - n\|_2.$$

The exact computation of $\mathcal{M}_C^*$ is NP-hard; so in [15] the authors propose an iterative greedy approximation. To further reduce coreset selection time, elements $m \in \mathcal{M}$ are first projected to a lower dimensionality through a random linear projection $\psi : \mathbb{R}^d \to \mathbb{R}^{d^*}$, with $d^* < d$, justified by the Johnson–Lindenstrauss lemma, which guarantees that pairwise distances are approximately preserved under such a projection [15]. The resulting greedy iteration, matching Algorithm 2.10 of the original paper, selects at each step the element m_i of the memory bank not yet in the coreset that is farthest from the coreset built so far [15]:

$$m_i = \arg\max_{m \in \mathcal{M} - \mathcal{M}_C} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2.$$

Algorithm 1: *PatchCore* memory bank.

Input: Pretrained ϕ, hierarchies j, nominal data $\mathcal{X}_N$, stride s, patchsize p, coreset target l, random linear projection ψ.
Output: Patch-level Memory bank $\mathcal{M}$.
Algorithm:
$\mathcal{M} \leftarrow \{\}$
for $x_i \in \mathcal{X}_N$ **do**
  $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{P}_{s,p}(\phi_j(x_i))$
end
/* Apply greedy coreset selection. */
$\mathcal{M}_C \leftarrow \{\}$
for $i \in [0, ..., l-1]$ **do**
  $m_i \leftarrow \arg\max_{m \in \mathcal{M} - \mathcal{M}_C} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2$
  $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$
end
$\mathcal{M} \leftarrow \mathcal{M}_C$

Figure 2.10: Memory bank creation alg

Intuitively, the goal of coreset subsampling is to keep only a small set of representative patch features while discarding redundant ones. Instead of storing many very similar feature vectors, the algorithm iteratively selects those that are the most distant from the representatives already chosen. As a result, the final coreset covers the original feature space as uniformly as possible,

preserving its diversity while significantly reducing the number of stored features and, consequently, the computational cost of nearest-neighbour search during inference.

The fraction of the original memory bank retained after subsampling is denoted ℓ; the notation *PatchCore-n%* is used to indicate that the original memory bank has been subsampled to $n\%$ of its original size [15]. In the present work, a subsampling ratio of 10% (*PatchCore-10%* in the notation of the original paper) has been adopted.

2.6.6 Stage 4: Anomaly Detection and Localization

With the nominal coreset-reduced patch-feature memory bank $\mathcal{M}_C$, the image-level anomaly score $s \in \mathbb{R}$ for a test image x^{test} is estimated as the maximum distance score s^* between the test patch-features in its patch collection $\mathcal{P}(x^{test}) = \mathcal{P}_{s,p}(\phi_j(x^{test}))$ and their respective nearest neighbour in $\mathcal{M}$ [15]:

$$m^{test,*},\ m^* = \underset{m^{test} \in \mathcal{P}(x^{test})}{\arg\max} \ \underset{m \in \mathcal{M}}{\arg\min} \|m^{test} - m\|_2, \qquad s^* = \|m^{test,*} - m^*\|_2.$$

To obtain the final score s, a scaling factor w is applied to s^* to account for the behaviour of neighbouring patches: if the memory bank features closest to the anomaly candidate ($m^{test,*}$ and m^*) are themselves far from their own neighbouring samples – and are therefore already a rare nominal occurrence – the anomaly score is increased [15]:

$$s = \left(1 - \frac{\exp\|m^{test,*} - m^*\|_2}{\sum_{m \in \mathcal{N}_b(m^*)} \exp\|m^{test,*} - m\|_2}\right) \cdot s^*,$$

with $\mathcal{N}_b(m^*)$ denoting the b nearest patch-features to m^* within $\mathcal{M}_C$. This re-weighting scheme was found by the authors to be more robust than using the maximum patch distance alone [15].

Given s, the pixel-level anomaly segmentation follows directly: the anomaly score computed for every patch through the arg max operation above can be re-aligned to its original spatial location, producing a low-resolution anomaly map [15]. To match the resolution of the input image, this map is upscaled via bilinear interpolation and further smoothed with a Gaussian filter of kernel width $\sigma = 4$ [15].

2.6.7 Normalization and Classification Threshold

The anomaly score s produced by the equations above does not, by itself, have a standardized range: its scale depends on the specific characteristics of the memory bank and, indirectly, on the training feature distribution. To obtain an interpretable score and an automatic classification threshold, the practical framework used in this work (the anomalib library) performs a calibration step on a validation set, typically comprising a min–max normalization of the score,

$$s_{norm} = \frac{s - s_{min}}{s_{max} - s_{min}},$$

together with an adaptive threshold τ, obtained by maximizing a classification metric (the F1-score) on the validation set, such that a sample is classified as anomalous if $s \geq \tau$. As discussed extensively in this work on the basis of an empirical observation made on the developed system, this calibration step requires the validation set to contain representative examples of both the nominal and the anomalous class: in their absence, the normalization statistics and the adaptive threshold are not computed reliably, with significant consequences on prediction quality at deployment time, even when the underlying memory bank has been built correctly. This aspect, while apparently secondary to the algorithmic formulation of the method presented above, proved in practice to be a critical element for the correct functioning of the system, and is discussed as a dedicated case study in the experimental validation section of this work.

3. Related Works: State of the Art in Industrial Anomaly Detection

The design choices behind the anomaly detection pipeline developed in this thesis, and in particular the adoption of PatchCore as the core algorithm, are best understood in the context of the broader landscape of unsupervised visual anomaly detection methods. This chapter reviews the main families of approaches that have shaped the field over the past years: reconstruction-based methods , feature-embedding and memory-based methods, normalizing-flow-based methods, and a selection of more recent architectures that have been proposed after PatchCore, several of which explicitly aim to address its limitations . The chapter closes with a comparative discussion that positions all reviewed methods along the axes most relevant to an industrial deployment: detection accuracy, inference speed, training requirements, and overall suitability for a production-line setting such as the one addressed in this work.

Where a method shares significant conceptual overlap with PatchCore – already introduced in full mathematical detail in Chapter 2 – this chapter avoids repeating that derivation, and instead focuses on how each method differs from it, and why those differences matter (or do not) for the specific industrial scenario considered in this thesis.

3.1 Reconstruction-Based Methods

Reconstruction-based methods were the first approach widely used for unsupervised anomaly detection. They are also a good starting point for reviewing the field because they are simple to understand and introduced many of the ideas used by later methods.

The main idea behind these methods is simple. A model is trained only on normal images and is forced to compress the input information, usually through a low-dimensional bottleneck. As a result, it learns to reconstruct normal images well but struggles to reconstruct images containing anomalies, since it has never seen them during training. The anomaly score is then computed from the difference between the original image and its reconstruction.

3.1.1 Autoencoders

The simplest instantiation of this idea is the standard convolutional Autoencoder (AE) [2]. An AE is composed of two sub-networks: an encoder E_θ, which maps an input image $x \in \mathbb{R}^{H\times W\times C}$ to a compact latent representation $z = E_\theta(x) \in \mathbb{R}^d$, with d typically much smaller than the dimensionality of x; and a decoder D_ϕ, which attempts to reconstruct the original image from this latent code, $\hat{x} = D_\phi(z)$. The two networks are trained toghetr, by minimizing a pixel-wise reconstruction loss over the training set of nominal images $\mathcal{X}_N$:

$$\mathcal{L}_{rec} = \frac{1}{|\mathcal{X}_N|} \sum_{x\in\mathcal{X}_N} \|x - D_\phi(E_\theta(x))\|_2^2.$$

At inference time, the anomaly score for a test image x^{test} is computed directly as the reconstruction error,

$$s(x^{test}) = \|x^{test} - D_\phi(E_\theta(x^{test}))\|_2^2,$$

While conceptually elegant, the bottleneck-based AE formulation suffers from two well-documented failure modes in practice. The first is under-reconstruction: if the bottleneck is too aggressive, the network fails to reconstruct fine, legitimate details of normal images, producing a reconstruction error that is high everywhere, not only on anomalous regions, and therefore uninformative. The second, and more insidious, is the so-called *identity shortcut*: if the network has enough capacity, or if the anomaly occupies only a small fraction of the image, the decoder may learn to reconstruct the anomalous region just as faithfully as the rest of the image, effectively "copying through" the defect rather than suppressing it. In this second failure mode the reconstruction error at the location of the actual defect can be as low as, or lower than, the error in genuinely normal regions, resulting in a severe false-negative.

3.1.2 Variational Autoencoders

Variational Autoencoders (VAEs) [9] extend the standard Autoencoder (AE) by introducing a probabilistic latent space. Instead of encoding an input image into a single latent vector, the encoder estimates the mean μ and standard deviation σ of a Gaussian distribution. A latent vector is then

sampled from this distribution using the reparameterization trick and passed to the decoder to reconstruct the input image.

The model is trained by maximizing the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{ELBO} = \mathbb{E}_{q_\theta(z|x)} \left[\log p_\phi(x|z) \right] - D_{KL} \left(q_\theta(z|x) \parallel p(z) \right),$$

where the first term represents the reconstruction loss, which encourages the decoder to generate an output similar to the original input. The second term is the Kullback–Leibler (KL) divergence, which acts as a regularization term by forcing the learned latent distribution to remain close to a predefined prior distribution, usually a standard Gaussian distribution $\mathcal{N}(0, I)$.

Compared with standard autoencoders, this probabilistic regularization produces a smoother and more structured latent space. This allows the decoder to generate meaningful samples even from latent vectors that were not directly observed during training. In the context of anomaly detection, the reconstruction error remains the main anomaly score, but the KL divergence, or other quantities derived from the probabilistic latent representation, can also provide additional information about how different a test sample is from the learned distribution of normal data.

Despite these advantages, VAE-based anomaly detectors still suffer from some limitations similar to those observed in standard autoencoders. In particular, they may reconstruct anomalous regions too accurately or generate blurry outputs due to the strong regularization imposed on the latent space. Therefore, their performance on industrial anomaly detection benchmarks, such as MVTec AD, is generally lower than that achieved by more recent feature-embedding methods, such as PatchCore [15].

3.1.3 GANomaly

Generative Adversarial Networks (GANs) [11] introduced a different learning paradigm for generative models. Unlike autoencoders, which learn by minimizing the difference between an input image and its reconstruction, GANs learn through a competition between two neural networks: a generator G and a discriminator D. The generator aims to produce synthetic samples that are similar to real data, while the discriminator tries to distinguish between real and generated samples. This adversarial process can be formulated as a minimax optimization problem:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)} \left[\log D(x)\right] + \mathbb{E}_{z \sim p_z(z)} \left[\log(1 - D(G(z)))\right],$$

where x represents real samples drawn from the data distribution p_{data}, while z is a latent vector sampled from a prior distribution p_z. During training, the generator learns to transform latent vectors into realistic samples, while the discriminator improves its ability to distinguish real images from generated ones.

In the context of anomaly detection, GANs are trained only using normal samples. The main assumption is that the generator learns the distribution of normal data and therefore becomes capable of representing only normal patterns. Consequently, when an anomalous sample is provided during inference, the generator should not be able to reproduce it correctly because such patterns were not observed during training.

Based on this idea, Akcay et al. proposed GANomaly [1], a semi-supervised anomaly detection framework that combines adversarial training with an encoder–decoder–encoder architecture. The main difference with traditional GAN-based approaches is that GANomaly does not require an expensive optimization procedure to map an image back into the latent space. Instead, it jointly learns the image and latent representations during training.

The proposed generator is composed of three main components: an encoder G_E, a decoder G_D, and a second encoder E. Given an input image x, the first encoder extracts a compact latent representation:

$$z = G_E(x),$$

where z represents the bottleneck features containing the most relevant information about the input image. The decoder then reconstructs the image from this latent representation:

$$\hat{x} = G_D(z) = G_D(G_E(x)).$$

The reconstructed image is finally passed through a second encoder network:

$$\hat{z} = E(\hat{x}),$$

obtaining a second latent representation that can be compared with the original latent vector z.

The main idea behind GANomaly is that, when the network is trained only on normal images, the latent representation of a normal sample and the latent representation of its reconstruction should remain close. On the contrary, when an anomalous image is provided, the generator has not learned the corresponding abnormal patterns and the reconstruction will not preserve the same latent information. Therefore, the distance between z and $\hat{z}$ becomes a measure of abnormality.

The training objective combines three different loss functions. The first one is the adversarial loss, which encourages the generated image to appear similar to real normal samples. In GANomaly, feature matching is used instead of the classical discriminator output loss, improving training stability:

$$\mathcal{L}_{adv} = \mathbb{E}_{x \sim p_X} \left[\| f(x) - f(G(x)) \|_2 \right],$$

where $f(\cdot)$ represents the feature representation extracted from an intermediate layer of the discriminator network.

The second component is the contextual reconstruction loss. This loss ensures that the generated image preserves the main characteristics of the input image:

$$\mathcal{L}_{con} = \mathbb{E}_{x \sim p_X} \left[\| x - G(x) \|_1 \right].$$

The third and most important component for anomaly detection is the encoder loss. It minimizes the distance between the latent representation of the input image and the latent representation of the reconstructed image:

$$\mathcal{L}_{enc} = \mathbb{E}_{x \sim p_X} \left[\| G_E(x) - E(G(x)) \|_2 \right].$$

The final training objective of GANomaly is obtained by combining the three losses:

$$\mathcal{L} = w_{adv} \mathcal{L}_{adv} + w_{con} \mathcal{L}_{con} + w_{enc} \mathcal{L}_{enc},$$

where w_{adv}, w_{con}, and w_{enc} are weighting parameters that control the contribution of each term.

During inference, the anomaly score is computed using the latent-space discrepancy:

$$A(x) = \| G_E(x) - E(G(x)) \|_1 .$$

A low value of $A(x)$ indicates that the input image is consistent with the learned distribution of normal samples, while a high value suggests the presence of an anomaly.

Although GANomaly improves previous GAN-based approaches by jointly learning image and latent spaces, GAN-based anomaly detection methods still present some limitations. Their effectiveness strongly depends on the ability of the generator to correctly learn the distribution of normal data. If the generator becomes too powerful, it may also reconstruct unseen anomalous patterns, reducing the difference between normal and abnormal samples. Moreover, adversarial training can be unstable and difficult to optimize.

For industrial inspection tasks, these limitations are particularly relevant. Defects are often small and localized, occupying only a limited region of the image. Since GAN-based methods are designed to generate globally realistic images, they may fail to preserve and detect small local differences. In applications such as safety seal inspection on bottle caps, where anomalies can correspond to small tears or partial detachments, accurately capturing these fine-grained defects remains a challenging problem.

3.2 Feature-Embedding Methods

The limitations of reconstruction-based methods – particularly their unreliability on small, subtle defects, and the identity-shortcut failure mode – motivated a shift, over the past several years, towards methods that do not attempt to regenerate the input image at all. Instead, feature-embedding methods leverage the representations produced by convolutional neural networks pre-trained on large, generic image datasets (most commonly ImageNet), and model normality directly in this pre-trained feature space, either statistically or via explicit memory of nominal feature examples. As discussed in Section 2.5.3, this shift is motivated by the observation that pre-trained CNN features already encode rich, general-purpose visual information without requiring any domain-specific adaptation, making them an effective foundation for cold-start anomaly detection.

3.2.1 PaDiM

PaDiM (Patch Distribution Modeling) [7] is one of the first and most influential feature-embedding methods for industrial anomaly detection. It is also one of the methods most closely related to PatchCore, since both use intermediate CNN features and PaDiM is one of the main baseline methods used in the original PatchCore paper [15], Figure 3.1 shows the pipeline.

Like PatchCore, PaDiM first extracts feature maps from a pre-trained CNN. The input image is divided into a regular grid according to the spatial resolution of the feature maps. For each spatial position (h, w), PaDiM collects the corresponding feature vectors from all normal training images. These features, usually obtained by concatenating features from multiple CNN layers, are modeled using a multivariate Gaussian distribution.

For each position (h, w), the mean vector and covariance matrix are estimated as

$$\mu_{(h,w)} = \frac{1}{N}\sum_{i=1}^{N} \phi_{(h,w)}(x_i), \qquad \Sigma_{(h,w)} = \frac{1}{N-1}\sum_{i=1}^{N} \left(\phi_{(h,w)}(x_i) - \mu_{(h,w)}\right)\left(\phi_{(h,w)}(x_i) - \mu_{(h,w)}\right)^{\top} + \epsilon I,$$

where ϵI is a small regularization term added to ensure that the covariance matrix is invertible.

During inference, the feature vector extracted from the test image at position (h, w) is compared with the Gaussian distribution learned for the same position. The anomaly score is computed using the Mahalanobis distance:

$$d_{(h,w)}(x^{test}) = \sqrt{\left(\phi_{(h,w)}(x^{test}) - \mu_{(h,w)}\right)^{\top} \Sigma^{-1}_{(h,w)} \left(\phi_{(h,w)}(x^{test}) - \mu_{(h,w)}\right)}.$$

Repeating this computation for every spatial position produces a pixel-level anomaly map, similarly to PatchCore.

The main difference between PaDiM and PatchCore is how they model the normal feature distribution. PaDiM assumes that the features at each spatial position follow a Gaussian distribution. Once the mean and covariance are estimated during training, they remain fixed and are used for all test images.

PatchCore follows a different strategy. Instead of fitting a Gaussian distribution, it stores the normal training features in a memory bank and compares each test feature with its nearest neighbours. This non-parametric approach is more flexible because it does not assume a specific distribution for the normal features. As discussed in the original PatchCore paper [15], this is especially useful when the normal data contain different appearance variations that cannot be represented well by a single Gaussian distribution.

Another limitation of PaDiM is that the Gaussian model is learned independently for each spatial position. As a result, both the training and test images must have the same spatial alignment. This makes PaDiM more sensitive to object misalignment than PatchCore, whose nearest-neighbour search provides greater robustness to small spatial variations.

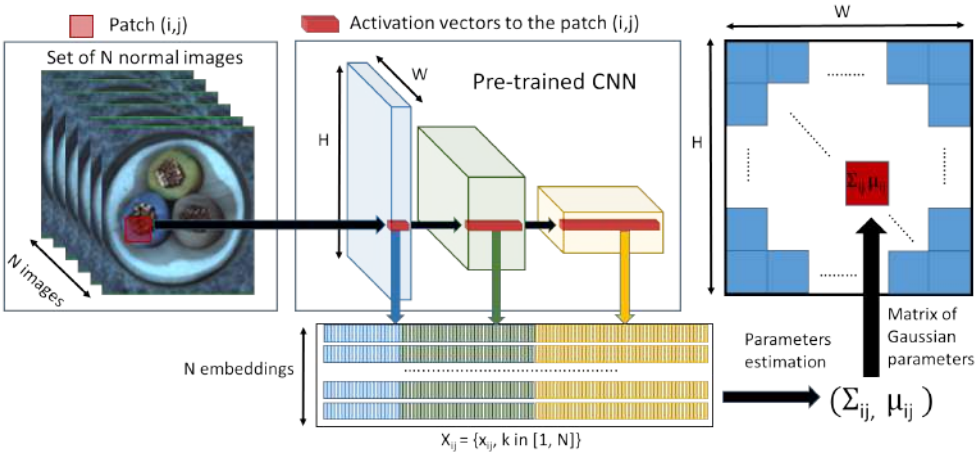


Figure 3.1: PaDiM pipeline

3.3 Normalizing-Flow-Based Methods

Normalizing-flow-based methods represent another family of anomaly detection approaches. Unlike PaDiM, which assumes a Gaussian distribution, or PatchCore, which stores training features in a memory bank, these methods learn the probability distribution of normal features directly.

A normalizing flow consists of a sequence of invertible and differentiable transformations,

$$f = f_K \circ \cdots \circ f_1,$$

that map a complex data distribution into a simple one, usually a standard Gaussian distribution. Since every transformation is invertible, it is possible to compute the exact probability of a sample using the change-of-variables formula:

$$\log p_X(x) = \log p_Z(f(x)) + \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial x_{k-1}} \right|,$$

where the first term is the probability of the transformed sample in the latent space, while the second term compensates for the change introduced by each transformation through the determinant of its Jacobian matrix.

During training, the normalizing flow is optimized using only normal training samples. The objective is to learn the probability distribution of normal features extracted from a pre-trained CNN backbone. During inference, each test feature is assigned a likelihood score. Features that are similar to the normal training data receive a high likelihood, whereas anomalous features receive a low likelihood and are therefore identified as anomalies.

3.3.1 FastFlow

FastFlow [19] applies the normalizing-flow framework to industrial anomaly detection while also providing pixel-level localization. Unlike earlier flow-based methods that processed an entire image as a single feature vector, FastFlow operates directly on the two-dimensional feature maps extracted by a pre-trained CNN, as shown in Figure 3.2.

To preserve the spatial structure of these feature maps, FastFlow uses two-dimensional convolutional coupling layers inside the normalizing flow. This allows the likelihood to be computed

independently at every spatial position instead of only once for the whole image.

As a result, FastFlow produces both an image-level anomaly score and a pixel-level anomaly map. Regions with low likelihood correspond to features that do not match the learned distribution of normal samples and are therefore identified as anomalous. This is conceptually similar to the anomaly maps produced by PaDiM and PatchCore. However, instead of using the Mahalanobis distance or a nearest-neighbour search, FastFlow detects anomalies by evaluating the likelihood estimated by the normalizing flow.

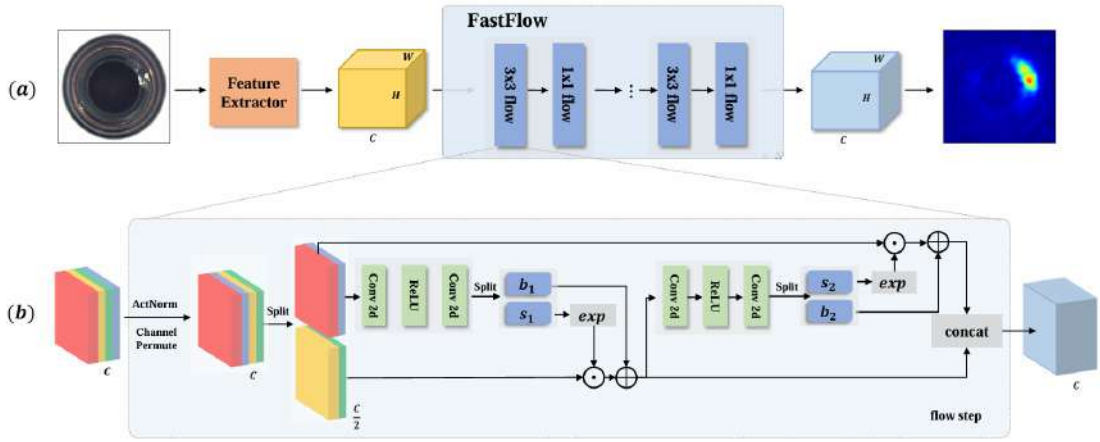


Figure 2: (a) the whole pipeline for unsupervised anomaly detection and localization in our method, which consists of a feature extractor and our FastFlow model. We can use an arbitrary network as the feature extractor such as CNN or vision transformer. FastFlow is alternatly stacked by the "3×3" and "1×1" flow. (b) one flow step for our FastFlow, the "Conv 2d" can be 3×3 or 1×1 convolution layer for 3×3 or 1×1 flow, respectively.

Figure 3.2: FastFlow pipeline from [19]

3.3.2 EfficientAD

EfficientAD [3] belongs to the family of knowledge-distillation-based anomaly detection methods. Knowledge Distillation (KD) was originally introduced as a technique to compress large neural networks into smaller and faster models. The main idea is to transfer the knowledge learned by a large and powerful network (the *Teacher*) to a smaller network (the *Student*).

In the anomaly detection field, this concept is particularly useful because the Student is trained only on normal images. Therefore, it learns to reproduce the behaviour of the Teacher only for the normal data distribution. When an anomalous image is provided during inference, the Student is expected to fail because it has never learned how the Teacher represents abnormal patterns. The difference between the Teacher and Student outputs can therefore be used as an anomaly indicator.

EfficientAD was specifically designed to overcome some limitations of previous feature-embedding methods such as PatchCore. Although PatchCore achieves excellent detection accuracy, it relies on a large pre-trained backbone network and on a memory bank containing many normal feature embeddings. During inference, these features must be compared using nearest-neighbour search, which can increase the computational cost. This can become a limitation in industrial applications requiring very high inspection speed or deployment on devices with limited computational resources.

The main architecture of EfficientAD follows a student–teacher paradigm. A large Teacher network extracts feature representations from normal training images, while a smaller Student network learns to reproduce these representations. During training, both networks receive the same normal input image x, and the Student is optimized to minimize the difference between its output and the Teacher output:

$$\mathcal{L}_{distill} = \|T(x) - S(x)\|_2^2 ,$$

where $T(x)$ represents the feature representation produced by the Teacher network and $S(x)$ represents the feature representation generated by the Student network.

Because the Student is trained only using defect-free samples, it learns a good approximation of the Teacher behaviour only inside the normal data distribution. During inference, if a test image contains an anomaly, the Teacher can still extract meaningful features, while the Student is unable

to reproduce them correctly. This produces a larger feature discrepancy:

$$A(x^{test}) = \left\| T(x^{test}) - S(x^{test}) \right\|_2^2 .$$

This discrepancy is used as the anomaly score. By computing this difference at different spatial positions of the feature map, EfficientAD can also generate a pixel-level anomaly map showing the location of possible defects.

A key difference compared with traditional knowledge distillation approaches is that EfficientAD does not rely on a very large backbone during inference. Instead, it introduces a lightweight Patch Description Network (PDN), a small feature extractor specifically designed for fast industrial inspection. The PDN produces local feature descriptors from image patches while keeping the computational cost very low. This design allows EfficientAD to replace the expensive nearest-neighbour search used by memory-bank methods with a single forward pass through a small network.

EfficientAD also introduces an additional autoencoder-based branch to detect *logical anomalies*. While the student–teacher branch is effective for local appearance defects, such as scratches or small surface damages, some anomalies are related to incorrect global structures, for example missing or incorrectly arranged components. The autoencoder branch provides a global representation of the image and helps detect these types of inconsistencies.

The total EfficientAD architecture therefore combines two complementary signals: the local discrepancy between Teacher and Student features, which detects local appearance changes, and the autoencoder branch, which captures global structural differences. This combination allows EfficientAD to achieve high detection accuracy while maintaining extremely low inference latency, making it particularly suitable for real-time industrial inspection systems.

3.3.3 Reverse Distillation

Reverse Distillation [8] also uses a student-teacher architecture, but with a different design from conventional approaches, including EfficientAD discussed bedfore. Instead of training the student to directly mimic the teacher's feature extraction process, the student is designed as a *decoder* that reconstructs the teacher's feature representations from a compressed latent representation.

More specifically, the frozen, pre-trained teacher encoder T extracts feature maps from the input image at multiple layers. These features are compressed by a trainable bottleneck into a low-dimensional embedding. The student decoder S then learns to reconstruct the original teacher feature maps from this compressed embedding using only normal training images.

During inference, the anomaly score is computed by measuring the difference between the teacher's feature maps and the student's reconstructed feature maps at each layer. These differences are then combined across multiple layers to produce the final anomaly score, similarly to the multi-scale feature aggregation used by PatchCore.

The main problem behind this design is that a conventional student network may become too good at copying the teacher and eventually generalize this behavior to anomalous images as well. This problem is similar to the identity mapping issue discussed for reconstruction-based methods, but it occurs in the feature space instead of the image space. By forcing the student to reconstruct the teacher's features from a compressed bottleneck, Reverse Distillation makes this shortcut much harder. As a result, the student is encouraged to learn the distribution of normal feature representations rather than simply copying the teacher's output.

3.3.4 RD++

RD++ [17] is an improved version of the Reverse Distillation method described above. The authors observed two main limitations of the original approach. First, although Reverse Distillation reduces shortcut learning more effectively than conventional student–teacher methods, the reconstructed normal features could still be made more compact and consistent. Second, the model has no explicit mechanism to prevent anomalous information from passing through the bottleneck, relying only on the reverse architecture to avoid shortcut learning.

To address these limitations, RD++ introduces two additional training objectives. The first encourages the reconstructed features to be closer to the distribution of normal features extracted by the teacher. Instead of only minimizing the reconstruction error, it helps the student learn a more compact representation of normal data.

The second objective improves the model's ability to reject anomalies. During training, artificial anomalous features are created by adding structured noise to normal features. The student is then trained to reconstruct these perturbed features as well, making the model more robust to abnormal

patterns.

At the time of its publication, RD++ reported the highest image-level AUROC on the MVTec AD benchmark among the methods compared in the original paper, including PatchCore, while running approximately six times faster than PatchCore and requiring only around 4GB of inference memory, since it relies on a single, fixed-size forward pass through the student network rather than on a nearest-neighbour search over an external memory bank [17].

3.4 Quantitative Comparison on MVTec AD

To complement the qualitative discussion previuosly made, Table 3.1 reports quantitative image-level AUROC figures on the MVTec AD benchmark for a subset of the methods reviewed in this chapter, collected directly from their respective original publications (or, where indicated, from independent reproductions reported in later papers) rather than estimated.

It is important to notify that these figures are not directly comparable in a strict sense: different papers evaluate under slightly different conditions (backbone architecture, input image resolution, and in some cases a different split or pre-processing of the MVTec AD dataset itself), and MVTec AD, despite being the de facto standard benchmark in this literature, remains a comparatively clean, well-controlled academic dataset whose absolute AUROC figures, often at or above 99% for the strongest methods and should not be read as directly predictive of performance on the industrial dataset developed in this thesis.

Two figures in Table 3.1 deserve a direct comment, as they are the most solidly grounded in a single, consistent source and are also the two methods most directly relevant to the remainder of this thesis. PaDiM's 95.3% and PatchCore's 99.0–99.1% are both taken from Table 1 of the original PatchCore publication [15], evaluated under identical experimental conditions by the same authors, and can therefore be compared to one another with full confidence, a reduction in error rate of more than half, as already discussed PaDim dedicated section. The remaining figures, collected from separate publications with their own experimental setups, should be treated as indicative of each method's general competitiveness on this benchmark.

The specific subset of methods actually trained and evaluated on the cap datasets developed in this work is presented separately in the next chapter, together with the corresponding experimental

Table 3.1: Image-level AUROC (%) on the MVTec AD benchmark, as reported in the original publications.

Method	**Image AUROC (MVTec AD)**	**Source**
PaDiM	97.8%	[15]
FastFlow	~98.9%	[19]
PatchCore (25% coreset)	99.1%	[15]
PatchCore (1% coreset)	99.0%	[15]
Reverse Distillation (RD)	~98.5%	[8]
EfficientAD	~99.1%	[4]

methodology and results obtained under the real acquisition and latency constraints of the production line described in Chapter 5.

4. Industrial Case Study

The previous two chapters introduced the theoretical foundations of anomaly detection and reviewed the main related methods. This chapter focuses on the specific industrial context in which the system developed in this thesis was deployed. The aim is not to justify the choice of a particular algorithm, but to describe the environment in which the system had to operate. In particular, the chapter presents the production line, the product being inspected, the types of defects that must be detected, the hardware used for image acquisition, and the dataset collected to train and evaluate the model. Due to confidentiality constraints, some details about the production environment cannot be provided.

4.1 Production Environment

The system described in this thesis was developed for a real bottling line for vinegar in plastic bottles with plastic tamper-evident caps. The line operates at a nominal speed of approximately one bottle per second, corresponding to about 3,600 bottles per hour. This production speed is an important constraint for the entire inspection system, affecting both the choice of the anomaly detection method discussed in Chapter 2 and the latency optimizations presented in Chapter 3.

The bottles continuously move along a conveyor belt and are not stopped or slowed down during inspection. The system must therefore operate entirely *on-the-fly*, acquiring and analyzing each cap while the bottle is moving. This requirement prevents the use of inspection methods that would require the product to be stopped, rotated, or physically repositioned. For this reason, a pericentric acquisition system was used. This optical system allows the entire circumference of the cap to be observed from a single fixed viewpoint and with a single image, without requiring any movement of the product.

Before reaching the inspection station, each bottle passes through a capping machine, which applies the plastic safety cap using mechanical pressure. This step is important because it is the main source of the defect that the inspection system is designed to detect. The mechanical stress generated during capping can cause the tamper-evident seal to become partially or completely

detached. This defect is generally not present, or is much less frequent, before the capping process.

The position of the capping machine also determines where the inspection system must be placed in the production line. Since the defect can be introduced during the capping process, it can only be reliably detected after the bottle has been capped. An inspection performed before this step, for example directly on the caps before assembly, would therefore not be sufficient for this application.

As in many high-throughput manufacturing processes, there is no manual inspection of every product after the automated system. Human quality control, when performed, is limited to occasional checks and does not provide a systematic inspection of each bottle. The automated system described in this thesis is therefore the main systematic quality control for this specific type of defect. This requirement also influenced the design of the system, leading to a conservative approach in which false rejections are preferred to false acceptances when the anomaly score is close to the decision threshold.

For privacy reasons, the real images of the production line cannot be shown. Therefore, Figure 4.1 shows a generic representation of a similar production line environment. The bottles move continuously along the conveyor belt and are guided by metal structures that keep them along a defined path. These structures can also be used as mounting points for cameras and other inspection equipment like some actuators.



Figure 4.1: Genric production line

4.2 Product Description

The inspection system described in this thesis was not developed for a single, fixed product, but for a small family of four physically distinct cap variants,applied to different bottle formats. All four variants share the same functional design intent – a tamper-evident safety seal, moulded into the cap and intended to tear or detach visibly upon first opening, printed with the words *"TEAR-SEAL-BEFORE-OPENING"* or *"WARRANTY SEAL"* and a directional arrow indicating the tear direction – but differ substantially in material, transparency, and the specific geometric features available on their surface. Table 4.1 summarizes the four variants; each is described individually in the subsections that follow. How each of these reference features is detected and used to computationally align the cap is a matter of methodology, and is treated separately in Chapter 5.

This product diversity do not affect system's design: as we will see in Chapter 5, managing four optically different products without duplicating the entire inspection pipeline for each one was a primary costraint of the system.

Table 4.1: Summary of the four cap variants inspected by the system.

Variant	**Material**	**Reference feature**	**Section**
Large red cap	Opaque, coloured plastic	Moulded slot near the seal + arrow	4.2.1
Large transparent cap	Clear plastic	Pair of small circular reference marks	4.2.2
Small transparent cap	Clear plastic	Pair of small circular reference marks	4.2.3
Small black cap	Dark plastic	Printed directional arrow only	4.2.4

4.2.1 Large Red Cap

The first variant is a large, fully opaque red cap Figure 4.2. Unlike the other three variants, it does not have small circular reference marks on its surface. Instead, next to the safety seal, there is a moulded slot , which appears as a darker region under the system's illumination. This slot is the only rotational reference feature of the cap and is used for the geometric alignment described in Chapter 5

Since the cap is opaque, its internal structure is not visible. The acquisition system only captures its external surface and moulded features, including the safety seal, the slot, and the surrounding tamper-evident ring.



Figure 4.2: Large Red Cap

4.2.2 Large Transparent Cap

The second variant is a large, transparent cap Figure 4.3 . Since the cap is transparent, several internal features are visible in the acquired image. In particular, the cap has two small circular reference marks, positioned symmetrically around its centre. These marks are used as the rotational reference for the geometric alignment described in Chapter 5.

The image also shows other features through the transparent material, including the safety-seal print, the moulded structural ribs, and the tear-seal mechanism along the cap edge. All these features are visible in the same image.

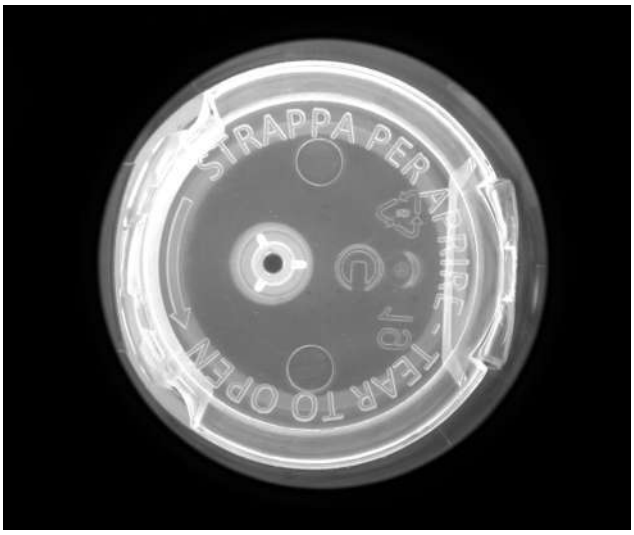


Figure 4.3: Large Transparent Cap

4.2.3 Small Transparent Cap

The third variant is similar to the large transparent cap described previously, Figure 4.4. It is also clear and has two small circular reference marks moulded symmetrically around its centre. However, it is physically smaller and is imaged at a different scale. In particular, the distance between the cap border and the circular marks is different from that of the large transparent cap.

Although the two caps look similar, the inspection system treats them as two different products. In fact, the small transparent cap has its own anomaly detection model and its own dedicated image region, separate from those used for the large transparent cap.

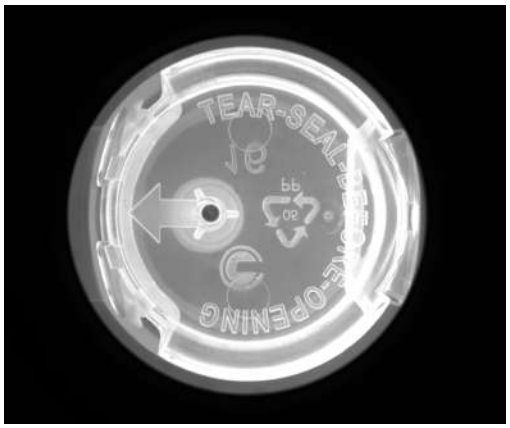


Figure 4.4: Small Transparent Cap

4.2.4 Black cap

The fourth variant is a dark-coloured, opaque cap, Figure 4.5. Unlike the other three variants, it does not have any small circular reference marks or other moulded reference features. Its only rotational reference is the printed directional arrow included in the tamper-evident seal instructions, which is present on all four cap variants.

This also makes the dark opaque cap the most difficult variant to handle during preprocessing, since the system cannot rely on clear geometric reference marks for alignment. The printed arrow must therefore be used as the main reference feature for determining the cap orientation.



Figure 4.5: Black cap

4.3 Defect Description

Across all four cap variants described above, the system is designed to detect only one type of failure: an anomaly in the safety seal. Other defects, such as cosmetic marks, material impurities, or printing defects, are not considered in this inspection task. As discussed above, the main problem is the possibility of a bottle being sold without a proper safety seal, which could cause serious problems for the factory.

However, the visual appearance of the seal anomaly can vary between the four variants. This is mainly due to differences in cap geometry and material, as well as differences in the defect samples collected for each variant so far.

4.3.1 Transparent Caps (Large and Small)

For the two transparent variants (Sections 4.2.2 and 4.2.3), two main manifestations of the seal defect are considered:

- **Absent seal**: the tear-tab is completely missing from the cap. This is the most severe failure, as the bottle has no tamper-evident protection. However, it is also the most difficult defect to detect, since the missing tear-tab occupies only a small portion of the acquired image. Figure 4.6 shows an example of a cap with a completely missing seal. As can be seen, the defective cap is visually very similar to a normal cap, since the missing seal affects only a small portion of the acquired image.
- **Partially displaced seal**: The tear-tab is present but shifted from its correct position. As shown in Figure 4.7, this defect is more visible and easier to detect because it affects a larger area of the acquired image.

Both cases are considered part of the same anomaly category and are therefore evaluated together. A suitable anomaly-score threshold is selected to provide a good balance between the two defect types, ensuring that the system can detect both the clearly visible absent-seal cases and the more subtle partially displaced cases.

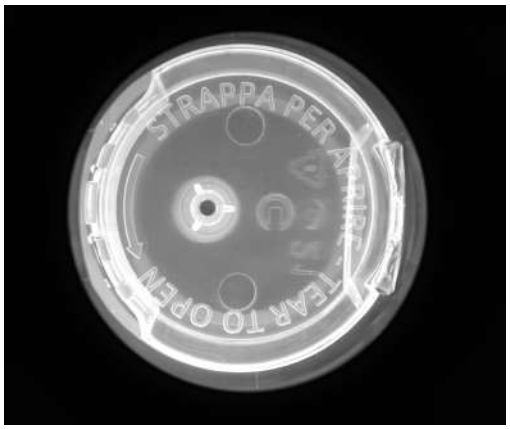


Figure 4.6: Absent seal

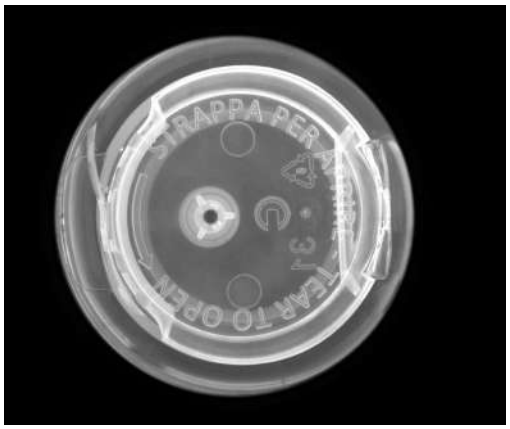


Figure 4.7: Partially displaced seal

4.3.2 Large Red Cap

For the large red cap (Section 4.2.1), the considered defect is the complete absence of the tear-tab. This is the easiest defect to detect among the four variants as shown in Figure 4.8. When the seal is missing, the difference from a normal cap is clear and easy to identify. Moreover, the seal has a very different colour from the red cap, appearing black in the acquired image. This strong colour contrast makes the affected region particularly visible and helps the system detect the defect reliably.



Figure 4.8: Missing seal

4.3.3 Black cap

For the black cap without reference marks, the defect dataset contains both complete seal absence and partial seal detachment, as for the transparent caps. In the first case, the tear-tab is completely missing, while in the second case it remains partially attached to the cap Figure 4.9.

Both cases are considered part of the same anomaly category and are evaluated together using a suitable anomaly-score threshold.



Figure 4.9: Partially missing seal black cap

4.4 Acquisition System

The image acquisition system was designed to meet four main requirements derived from the production environment described in Section 4.1. First, the entire circumference of the cap must be visible, since the tamper-evident seal can have any angular position after the capping and conveying process. Second, the images must be acquired while the bottle is moving, without stopping or reorienting it. Third, the acquisition must avoid motion blur caused by the movement of the bottles along the production line. Finally, the time required to acquire and process each image must be low enough to meet the line speed of one bottle per second.

4.4.1 Pericentric Lens

Meeting the first two requirements together – full circumferential coverage and image acquisition without moving the bottle – makes the two most common industrial solutions impractical for this application. The first solution is to use multiple cameras positioned around the product. This can provide complete coverage, but it increases the hardware cost and requires the cameras to be accurately calibrated with respect to each other. The images from the different viewpoints must also be aligned or combined. The second solution is to use a conventional camera together with a mechanical rotation system. This approach introduces additional moving parts and synchronization requirements and, more importantly, requires the bottle to be stopped or rotated during inspection. This would reduce the achievable production speed and is not compatible with the continuous operation of the line.

For these reasons, the acquisition system uses a single *pericentric lens*. To understand its advantage, it is useful to first consider a conventional *entocentric lens*. In a standard camera, the entrance pupil is located inside the optical system and the image is approximately described by a perspective projection. In a simplified pinhole model, the relationship between the size of an object H, its distance from the camera Z, and its image size h is

$$h = f\frac{H}{Z}, \tag{4.1}$$

where f is the focal length of the lens. This means that objects closer to the camera appear

larger, while objects farther away appear smaller. More importantly for the present application, a conventional camera observes the object mainly from a single viewing direction. When it is placed in front of a cylindrical cap, parts of the lateral surface are hidden by the geometry of the object itself. Therefore, a single entocentric camera cannot normally provide a complete 360-degree view of the lateral surface.

A pericentric lens uses a different optical configuration. Its entrance pupil is effectively located in front of the physical lens, on the object side. This changes the direction from which light rays can enter the optical system. Instead of observing the cylindrical object from a single dominant viewpoint, the lens can collect light coming from different angular positions around the lateral surface at the same time. The geometry of the optical system therefore allows the side walls of the object to be projected onto the sensor together with the top surface.

For a cylindrical cap, this produces a characteristic image. The top surface is projected mainly in the central part of the image, while the lateral surface is projected around it as an annular, or ring-shaped, region. If the angular position around the cap is described by θ, a complete lateral surface corresponds to

$$0 \leq \theta < 2\pi, \tag{4.2}$$

which represents a full 360-degree rotation around the cylinder. The pericentric lens maps this complete angular range onto a single sensor image. Consequently, the entire lateral surface of the cap can be acquired in one exposure, without rotating the bottle.

This property is particularly important for the tamper-evident seal. Since the seal can have an arbitrary angular position after the capping and conveying processes, its position cannot be known in advance. With a conventional camera, this would require either multiple viewpoints or physical rotation of the bottle. With the pericentric system, all angular positions are captured simultaneously. The seal is therefore visible in the acquired image regardless of its orientation.

The main difference between the two optical systems can therefore be summarized by the position of the entrance pupil and the resulting field of view. An entocentric lens provides a conventional perspective projection, while a pericentric lens is designed to include the lateral surfaces of an object within its field of view. For approximately cylindrical objects, this makes it possible to obtain a complete view of the side surface using a single camera and a single exposure.

The pericentric lens therefore satisfies the first two acquisition requirements without requiring additional cameras or moving components. The complete 360-degree coverage is obtained directly from the optical geometry of the lens, rather than through mechanical rotation or computational reconstruction of multiple images.

However, this advantage also introduces a specific geometric distortion. The cylindrical lateral surface is projected into a radial region of the image rather than being represented using a conventional perspective projection. As a result, distances, shapes, and the apparent scale of features are not uniform across the image. In particular, the tamper-evident seal can appear stretched or compressed depending on its position within the projected annular region.

These geometric effects are important for the subsequent image-processing pipeline. The mapping between the original cylindrical surface and the acquired image, together with the distortion introduced by the pericentric projection, is therefore considered when defining the preprocessing and region-of-interest operations described in Chapter 5.

OPTO PC23020XC Pericentric Lens

For the inspection system, the selected optical component is the *PC23020XC*, a compact pericentric lens manufactured by Opto Engineering and designed for use with $2/3''$ image sensors. The choice of this specific model was mainly determined by the dimensions of the cap and by the available space around the inspection station. The lens is part of the PC series, which is specifically designed for simultaneous top and 360-degree lateral inspection using a single camera.

The PC23020XC provides an image circle of 6.8 mm and is designed for sensors with a minimum format of $2/3''$. For the supported optical configuration, the manufacturer specifies an inspectable field of view ranging from approximately 10.0×6.7 mm to 35.0×26.5 mm, where the first value represents the diameter of the inspected object and the second its maximum visible height. This range is particularly suitable for the dimensions of the cap considered in this work.

An important aspect of the PC23020XC is that the field of view is not described by a single rectangular area, as in a conventional imaging lens. Instead, it is defined by the diameter of the object and by the height of its lateral surface that can be observed simultaneously. The two quantities are related to the working distance. According to the manufacturer, the maximum field of view of 35.0×26.5 mm is specified for a working distance of zero. When the working distance is increased

while keeping the object diameter constant, the maximum observable height decreases.

This relationship is important in the practical installation of the system. The lens cannot simply be positioned at an arbitrary distance from the conveyor: the working distance must be selected together with the diameter and height of the cap so that the complete region containing the tamper-evident seal remains inside the lateral field of view. For the smallest specified object size, the PC23020XC provides a working distance of 46.6 mm, while the maximum object size is specified at a working distance of 0 mm.

The optical aperture is another relevant parameter. The PC23020XC has a nominal F-number starting from $f/1.4$, with the aperture adjustable through a variable iris. The F-number is defined as

$$N = \frac{f}{D}, \tag{4.3}$$

where f is the effective focal length and D is the diameter of the entrance pupil. A lower F-number allows more light to reach the sensor, which is useful for short exposure times, while closing the iris reduces the amount of light and generally increases the depth of field. The possibility of adjusting the iris is therefore useful for balancing illumination, exposure time, and depth of field during system setup.

The choice of the $2/3''$ sensor is also relevant to the optical configuration. The sensor must be large enough to cover the image circle produced by the lens. At the same time, the available sensor resolution determines how many pixels are used to represent the projected lateral surface. If the physical diameter of the inspected region is denoted by D_o and its corresponding image diameter by D_i, an approximate lateral scale can be expressed as

$$M = \frac{D_i}{D_o}, \tag{4.4}$$

where M is the magnification. The corresponding physical size represented by one pixel can then be approximated as

$$s_{\mathrm{px}} = \frac{p}{M}, \tag{4.5}$$

where p is the physical pixel pitch of the sensor. This relationship is important because the

minimum defect size that can be detected is ultimately limited by the spatial resolution of the acquired image.

The PC23020XC is also designed as an ultra-compact solution. Its mechanical length is approximately 212 mm, with a front diameter of 84 mm and a mass of about 1.03 kg. The lens uses a standard C-mount connection. These characteristics are relevant for the mechanical integration of the inspection station, since the lens must be positioned above the conveyor while leaving enough free space for the bottles to pass underneath it.

Another important characteristic of the PC23020XC is its single-camera acquisition principle. The manufacturer specifically highlights that the continuous image produced by the pericentric lens does not require image matching between multiple views. This simplifies the acquisition pipeline because the complete lateral surface is already contained in a single image. In the context of this thesis, this is particularly useful because the subsequent anomaly detection pipeline can operate directly on the acquired image after the required preprocessing and region-of-interest extraction.

The selected lens therefore provides a suitable compromise between optical coverage, spatial resolution, mechanical integration, and acquisition simplicity. Its field of view covers the dimensions required by the cap, while its pericentric geometry allows the entire lateral surface to be captured during the continuous motion of the production line. The remaining challenge is then to transform this non-conventional optical representation into a suitable input for the anomaly detection model. The preprocessing operations used for this purpose are described in the following sections.



(a) red cap (b) black cap (c) transparent cap

Figure 4.10: Examples of bottle caps acquired using the pericentric imaging system.

4.4.2 Triggering and Illumination

The last two requirements, avoiding motion blur and keeping a low processing time, are achieved using a hardware-triggered acquisition system.

A photoelectric sensor is placed just before the camera. When a bottle passes in front of the sensor, it generates an electrical signal that is sent directly to the camera through a GPIO connection. This hardware trigger starts the camera exposure and also sends a signal to the strobe light controller.

The lighting system uses a high-intensity strobe light that produces a very short flash during the camera exposure. The short flash duration effectively freezes the movement of the bottle. Although the bottle continues moving at full production speed, the acquired image appears as if the bottle were stationary, with negligible motion blur.

Using a hardware trigger also makes the timing of the acquisition more predictable than a software-based trigger. The camera and the strobe are therefore synchronized directly with the position of the bottle on the production line.

This approach prevents motion-related problems directly during image acquisition instead of trying to correct them later with image-processing algorithms. As a result, the anomaly detection model receives sharp images even though the bottles are continuously moving along the production line.

4.5 Dataset

The number of distinct real source images available at the time of writing is limited, particularly for the anomalous class, which is rare in a well-controlled production line (Section 4.1). Table 4.2 reports the number of real source images collected from the production line before augmentation.

Table 4.2: Real, independently acquired source images used as the basis for dataset construction.

Class	**Real source images**
Nominal (GOOD)	6
Defective (REJECT)	2–3

4.5.1 Dataset Augmentation

Due to the small number of real source images, the dataset was expanded using a synthetic augmentation procedure. Approximately 100–200 nominal images and 20 defective images were generated from the original samples. Each image was obtained by applying a small random rotation, with a maximum angle of approximately $\pm 1.5^\circ$ to $\pm 2.5^\circ$, depending on the cap variant. The rotation was performed around the geometric centre of the cap, estimated using the same procedure adopted by the alignment method described in Chapter 5.

The augmentation was designed to reproduce the small rotational variations that may remain after the alignment stage. The alignment brings each cap to a common reference orientation, but small residual errors may remain due to the limited accuracy of marker or template detection. Therefore, the augmentation focuses on small post-alignment rotations rather than the full range of possible cap orientations.

4.5.2 Limitations

This augmentation procedure only models rotational variability. Other sources of variation, such as changes in lighting, surface conditions, dirt, wear, and batch-to-batch differences, are not represented because the augmented images are generated from a limited number of real samples. Therefore, collecting a larger and more diverse set of real images under different production conditions remains an important direction for future work.

4.5.3 Dataset Composition and Splits

Table 4.3 summarizes the resulting dataset composition after augmentation. Nominal images are primarily used for training, consistent with the one-class formulation of PatchCore (Chapter 2), while a held-out subset of nominal images, together with the defective images, is reserved for validation and test purposes. Following directly from the calibration issue already identified and discussed , both the nominal and the defective images are explicitly assigned to the validation/test configuration of the training pipeline, rather than relying on an automatically generated validation split containing nominal images only.

As discussed in Section 4.3.1, the defective images are not homogeneous across cap variants:

Table 4.3: Dataset composition after augmentation.

Class	**Total images (post-augmentation)**	**Role**
Nominal (GOOD)	~100–200	Training, with a held-out subset for validation/test
Defective(REJECT)	~20	Validation/test only (not used for training)

for the two transparent variants, they include both fully absent and partially displaced seal instances, while for the red cap and the black cap, they correspond exclusively to fully absent seal instances (Section 4.3).

5. Proposed Method

5.1 Pipeline Overview

The previous chapter described the industrial context in which the system was developed. This chapter presents the software pipeline used to process the acquired images and inspect the four different cap variants described in Section 4.2.

Two main requirements guided the design of the system. First, the same pipeline must support all four cap variants without requiring a separate implementation for each one. Second, the system must operate continuously at the production-line speed, which is approximately one bottle per second.

Figure 5.1 shows the overall architecture of the proposed system. The pipeline can be divided into three main phases: system initialization, runtime configuration, and the inference loop.

During the **system initialization**, `main.py` starts the application, loads the general configuration, initializes the database connection, and prepares the resources required by the inspection system.

The **runtime configuration** is then loaded according to the selected cap variant. The rotation registry identifies the appropriate alignment procedure, while the corresponding configuration profile provides the parameters required for the inspection. The system also loads the PatchCore model and its memory bank associated with the selected cap type. This approach allows the same software pipeline to be used for all variants, with only the variant-specific parameters and resources changing. This loading step, implemented in `load_model()` and invoked once within `main()` at startup, ensures that the trained model and its memory bank are held in memory for the entire runtime session, rather than being reloaded for every incoming frame.

Once the system is initialized, it enters the **inference loop**. For each acquired frame, the image is first aligned using the rotation method associated with the current cap variant. The alignment rotates the complete image to a common reference orientation without cropping the frame.

The aligned image is then passed to the PatchCore model. The model extracts image features and compares them with the corresponding memory bank to obtain an anomaly score and an anomaly map. The anomaly map covers the complete input image.

The region of interest (ROI) associated with the selected cap variant is used only after inference. It is applied to the anomaly map to focus the score calculation on the relevant part of the cap. The resulting score is compared with the calibrated threshold described in Section 2.6.7, producing the final `GOOD` or `REJECT` decision. The result and the corresponding visualization are then saved and logged.

An important design choice is that the variant-specific operations are handled through configuration rather than separate implementations. In particular, the alignment function, model and memory bank, ROI, and other inspection parameters are selected according to the cap type. Therefore, adding a new cap variant mainly requires a new configuration and the corresponding alignment procedure, without changing the main inference pipeline.

The separation between initialization and inference also allows the resources required by the system to be loaded only once. During normal operation, each new frame can therefore be processed using the already loaded configuration and model resources.

Finally, the system was designed for continuous operation on the production line. Errors related to individual frames are handled at the frame level so that a problem with one inspection does not stop the processing of subsequent bottles. The implementation details of this mechanism are discussed in Section 5.6.

The remainder of this chapter describes the main components of the pipeline in more detail. Section 5.2 presents the configuration architecture and rotation registry. Section 5.3 describes the geometric alignment procedures used for the four cap variants. Section 5.4 covers feature extraction and PatchCore memory bank construction. Section 5.5 describes the inference and decision process, while Section 5.6 discusses the main implementation details relevant to reproducibility.

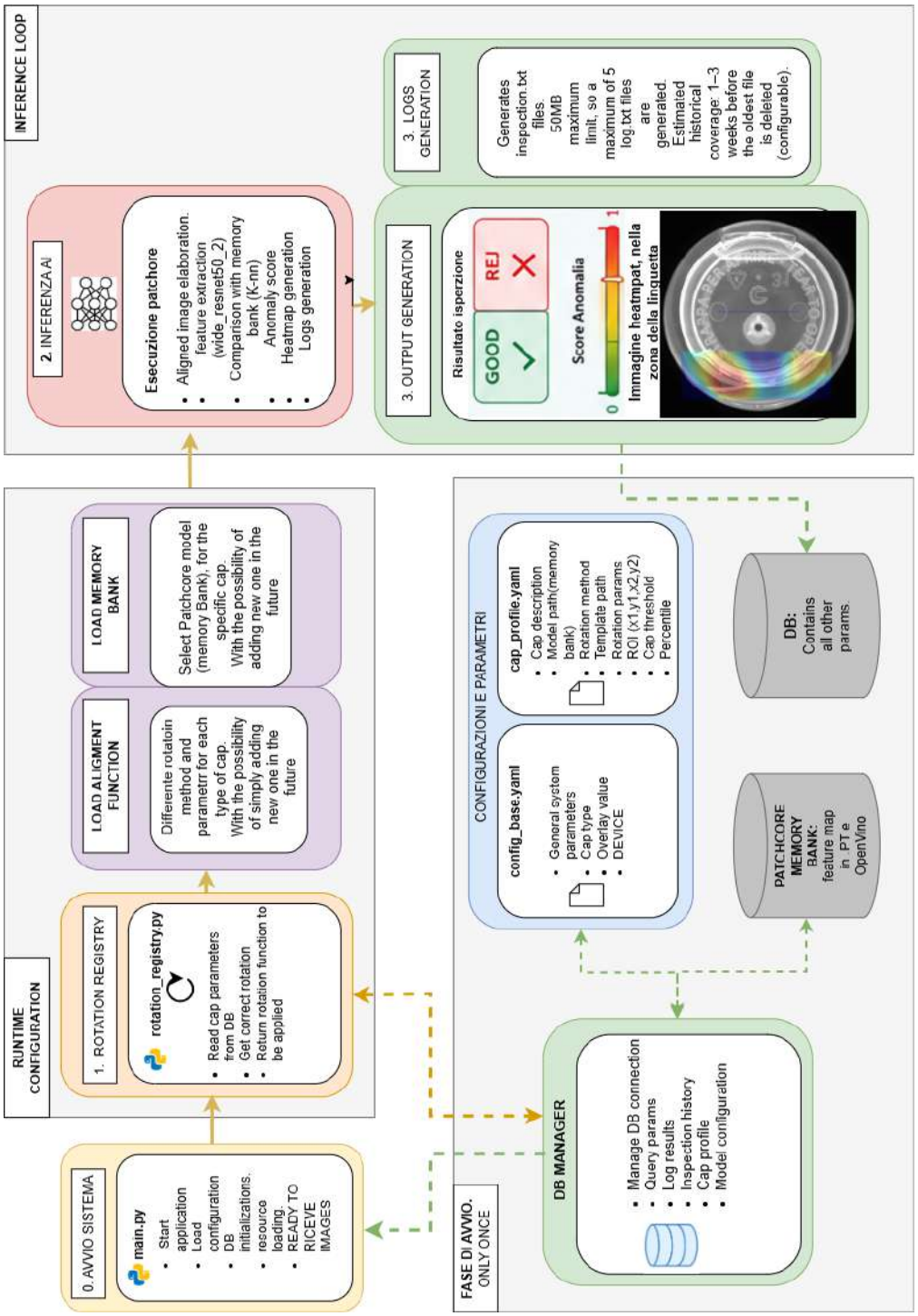


Figure 5.1: Overview of the proposed inspection pipeline. The diagram shows the one-time system initialization and runtime configuration phases (left), during which the rotation function, alignment parameters, and PatchCore model/memory bank for the active cap type are loaded, and the per-frame inference loop (right), in which each incoming image is processed and classified as `GOOD` or `REJECT`.

5.2 Two-Tier Configuration Architecture

To support the four cap variants without duplicating the main pipeline, the system separates global parameters from cap-specific parameters. This is implemented through a two-tier configuration architecture.

Global parameters, such as the input and output directories, the default computation device, the active cap type, overlay parameters, and the buffer polling interval, are stored in a SQLite database (`giacobazzi.db`) and managed by the `ConfigDB` class. When the database is initialized, the values defined in `config_base.yaml` are used to populate only missing entries through an `INSERT OR IGNORE` operation. Existing database values are therefore preserved across restarts. If the database cannot be accessed, the system can fall back to the YAML configuration file.

Cap-specific parameters are stored separately in a YAML file `cap_profile.yaml`. Each profile is identified by the cap type and contains the resources and parameters required for that variant, including the PatchCore model path, the alignment method, its parameters, the region of interest (ROI), the anomaly threshold, and the percentile used for score calculation. The active profile is selected through the global `TAPPO_TYPE` parameter and is loaded at system startup. Changing the cap type therefore requires restarting the application, which is suitable for the intended production workflow.

This separation allows global settings and cap-specific settings to be managed independently. It also avoids placing variant-specific conditions directly in the main processing loop. Before being used, the cap profile is checked for the presence of the required parameters and for the validity of the ROI coordinates. Invalid configurations therefore produce an explicit error during startup instead of causing an error later during image processing.

The connection between the cap profiles and the alignment algorithms is provided by a rotation registry. The value of `ROTATION_METHOD` stored in each profile is used to select the corresponding alignment function:

Code 5.1: Common alignment interface and registry dispatch (adapted from `rotation_registry.py`).

```python
# Every registered rotation function shares the signature:
#     _rotate_<method>(frame_bgr: np.ndarray, profile: dict) -> np.ndarray

ROTATION_REGISTRY = {
    "template_matching": _rotate_template_matching,
    "slot_detection": _rotate_slot_detection,
    "template_matching_small": _rotate_template_matching_small,
    "template_matching_black": _rotate_template_matching_black,
}

def get_rotation_fn(rotation_method: str):
    """Fail-fast dispatch: unregistered methods raise immediately,
    at startup, instead of failing on the first inspected frame."""
    if rotation_method not in ROTATION_REGISTRY:
        disponibili = ", ".join(sorted(ROTATION_REGISTRY.keys()))
        raise ValueError(
            f"ROTATION_METHOD='{rotation_method}' not registered. "
            f"Available methods: {disponibili}"
        )
    return ROTATION_REGISTRY[rotation_method]
```

At startup, `main.py` calls `get_rotation_fn(profile["ROTATION_METHOD"])` once for the active cap type and keeps a reference to the returned function for the rest of the run. The main inference loop therefore does not need to know how many alignment methods exist or how they are implemented. Adding a new cap variant with a new alignment strategy only requires implementing a function with the shared signature and adding it to `ROTATION_REGISTRY`; `main.py` remains unchanged. This is an application of the registry (or dispatch-table) pattern and follows the open/closed principle: new cap types can be added without modifying the code that uses the registry.

A second dictionary, `ASSET_LOADERS`, follows the same pattern for methods that require additional assets. For example, `template_matching` and `template_matching_small` require a reference template image. These assets are loaded once at startup through `load_assets_for_profile()` rather than being read from disk for every processed frame. The loaded assets are stored in

`profile["_assets"]` and can then be accessed directly by the corresponding alignment function. Methods that do not require additional assets, such as `slot_detection`, use an empty loader so that every registered method has a corresponding entry in both dictionaries.

Adding a new cap variant that requires a new alignment strategy means writing one function matching this signature and registering it, without modifying the main inference loop or any of the existing alignment methods.

5.3 Geometric Alignment

Because PatchCore compares the local features of an inspected image against a memory bank built from images in a fixed reference orientation (Section 2.6.4), every acquired frame must be rotated to a common angular reference before feature extraction. Since the tamper-evident seal can be in an arbitrary angular position after capping (Section 4.1), this alignment step is a prerequisite for the anomaly detection stage to be meaningful at all, not merely a preprocessing convenience.

Four rotation methods are registered (Section 5.2), implementing three geometrically distinct alignment strategies – template matching is shared, with different parameters, between the two transparent cap variants.

5.3.1 Slot Detection (Large Red Cap)

The opaque cap has no transparent internal features, so alignment relies on the moulded slot next to the safety seal, which appears as a dark, elongated region under the system's illumination. The frame is first converted to grayscale, since the slot is distinguished from its surroundings by intensity rather than colour. The cap boundary is then isolated from the background by thresholding and morphological closing, and its enclosing circle gives an estimate of the cap centre and radius. Within an annular search zone around this centre, dark blobs are extracted and filtered by three rotation-invariant criteria – area, elongation (from the minimum-area bounding rectangle), and distance from the cap centre relative to the estimated radius – since the search zone typically also contains a much larger, irregularly shaped dark region corresponding to background visible through the tamper-evident ring, which shape alone would not reliably exclude:

Code 5.2: Three-criteria blob filter for slot candidates (adapted from `slot_detection.py`).

```
ok = (p["SLOT_AREA_MIN"] <= area <= p["SLOT_AREA_MAX"] and
      p["SLOT_ELONG_MIN"] <= elong <= p["SLOT_ELONG_MAX"] and
      p["SLOT_DIST_RATIO_MIN"] <= dist_ratio <= p["SLOT_DIST_RATIO_MAX"])
if ok:
    candidati.append((i, area, dist, cx_s, cy_s))
```

The angle of the surviving candidate blob relative to the cap centre gives the correction angle applied via an affine warp. If no blob satisfies all three criteria, the function returns the unrotated frame together with a failure flag without interrupting the cycle. Figure 5.2 shows an example of the debug visualization used to verify the correctness of the detected rotation angle for the large red cap. The directional arrow(that will be used for the black cap) is also present on this cap and could be used for template matching. However, the molded slot provides a simpler solution: its angular position can be directly computed from the vector between the cap centre and the detected blob centroid. This requires only one connected-component analysis, whereas template matching must evaluate multiple candidate rotation angles.

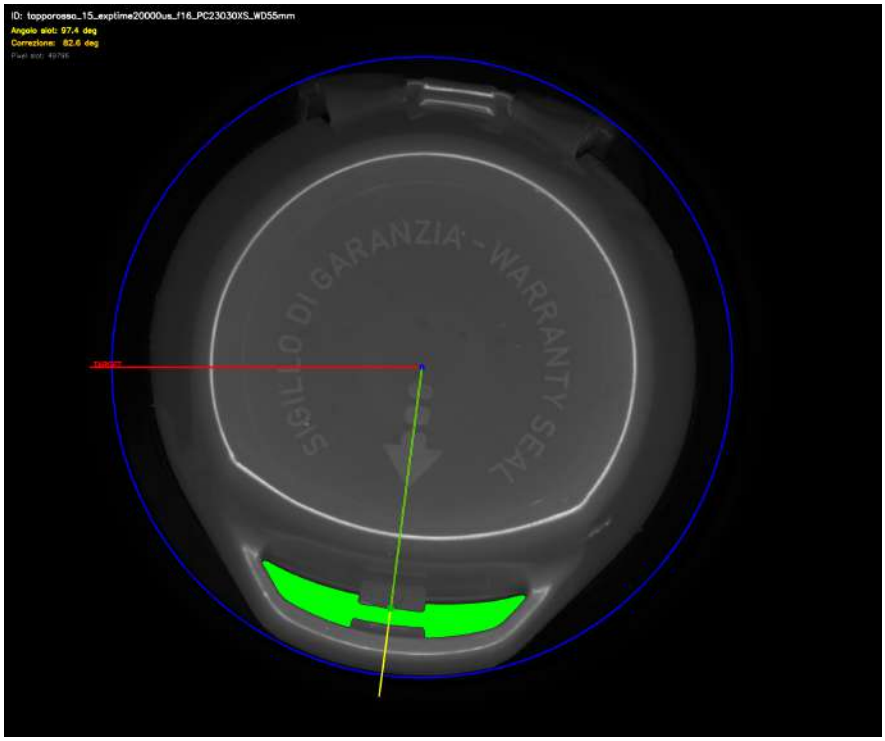


Figure 5.2: Red cap debug example

5.3.2 Template Matching via Marker Pair (Transparent Caps)

Both transparent variants contain two small circular reference marks moulded symmetrically around the cap centre and visible through the material. These marks are detected using the Hough circle transform, with progressively relaxed parameter sets to improve robustness when the initial detection fails. Candidate circle pairs are evaluated based on their consistency with the estimated cap centre and expected distance between the markers. The best pair defines a coarse rotation angle from the line connecting the two markers.

Since this line introduces a 180° ambiguity, both the original and the 180°-rotated orientations are compared against a small reference template using normalized cross-correlation on a restricted crop. The orientation with the higher matching score is selected:

Code 5.3: 180° disambiguation via normalized cross-correlation (adapted from `alignment.py`).

```
score_upright = cv2.minMaxLoc(
    cv2.matchTemplate(crop_upright, template, cv2.TM_CCOEFF_NORMED)
)[1]
score_flipped = cv2.minMaxLoc(
    cv2.matchTemplate(crop_flipped, template, cv2.TM_CCOEFF_NORMED)
)[1]

final_angle = base_rotation
if score_flipped > score_upright:
    final_angle += 180
```

Only the final rotation is applied to the full-resolution colour image. The small transparent cap uses the same procedure with variant-specific radius and distance parameters to account for its smaller scale. It is nevertheless treated as a separate profile, with its own trained model and region of interest, since the two transparent variants are not assumed to be interchangeable.

Figure 5.3 shows an example of the debug visualization used to verify marker detection and the estimated rotation for the large transparent cap.

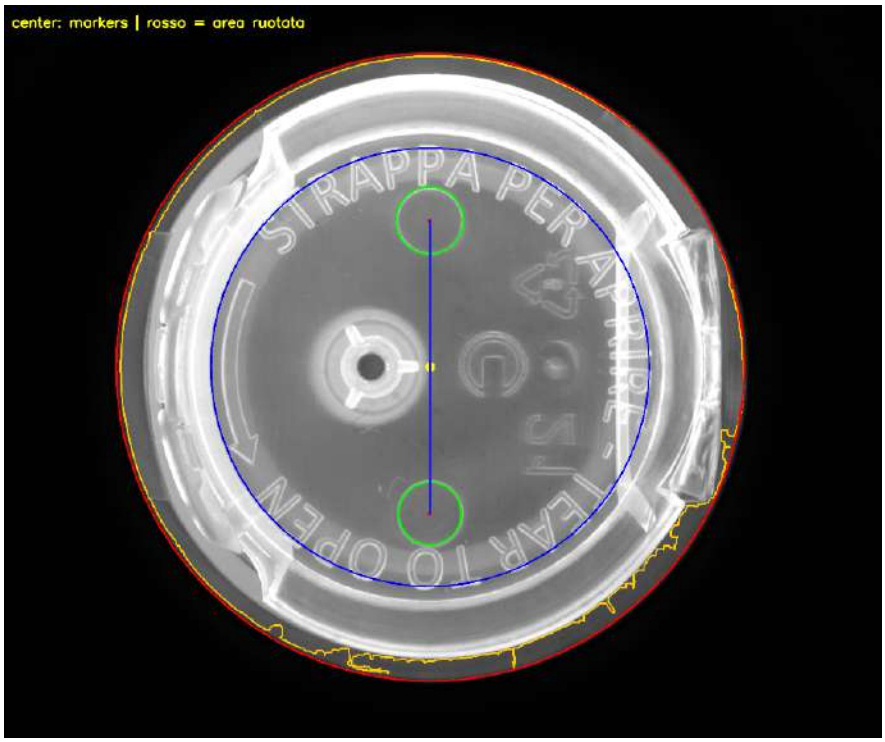


Figure 5.3: Debug visualization of marker detection and rotation estimation for the large transparent cap. The green circles mark the detected reference markers, the yellow dot the estimated cap centre, and the red/blue contours the detected cap boundary and the region considered for alignment.

5.3.3 Template Matching for the Black Cap

The fourth variant has no moulded reference marks; its only orientation cue is the printed directional arrow present on the safety-seal instructions. Since the marker-pair approach cannot be used, alignment searches for the rotation angle that best matches a template of the arrow.

An exhaustive full-resolution search is computationally expensive, requiring approximately 16,000 ms per frame. This is far above the line's target of about one bottle per second. Three optimizations reduce the processing time to approximately 170 ms:

1. **Restricted annular crop.** The search is limited to a square crop containing the annular region where the arrow can appear at any rotation angle, rather than the full cap image.

2. **Coarse search at reduced scale.** The full 360° range is first scanned using a 10° angular step on the crop and template downscaled to 20% of their original size. This provides an approximate best angle and match location.

3. **Fine search with a restricted output window.** A narrow range around the coarse angle (default ±6° with a 1° step) is then evaluated at full resolution. Instead of rotating the entire crop for each candidate angle, only a small window around the coarse-stage match location is processed.

The following pseudocode summarizes the procedure:

Code 5.4: 180° disambiguation via normalized cross-correlation.

```
Input: restricted crop C, arrow template T, cap centre (cx, cy)

# Stage 1: coarse search on downscaled crop
Cs <- resize(C, 0.2);  Ts <- resize(T, 0.2)
best_angle <- 0;  best_score <- -inf
for theta = -180 to 180 step 10:
    R <- rotate(Cs, theta, centre=(cx, cy) * 0.2)
    score, loc <- matchTemplate(R, Ts)
    if score > best_score:
        best_score, best_angle, best_loc <- score, theta, loc

# Stage 2: fine search, output restricted to a small window around best_loc
W <- window of size (template + margin) centred on best_loc  # full resolution
for theta = best_angle - 6 to best_angle + 6 step 1:
    Rw <- rotate(C, theta, centre=(cx, cy)), cropped to W only
    score <- matchTemplate(Rw, T)
    if score > best_score:
        best_score, best_angle <- score, theta

return best_angle, best_score
```

The final angle is applied once to the original colour image using a full-resolution affine warp. Thus, all iterative searches operate on reduced crops or small windows, while the full image is transformed only once. A minimum match-score threshold is also used to reject unreliable alignment results; if the best score is below the threshold, the original orientation is retained.

5.4 Feature Extraction and Memory Bank Construction

Once a frame has been aligned (Section 5.3), it is passed to the PatchCore pipeline whose theoretical foundations were introduced in Section 2.6. The implementation relies on the `anomalib` library (version 2.3.3), with a WideResNet-50 backbone pretrained on ImageNet, consistent with the transfer-learning rationale discussed in Section 2.4.3.

During training, the datamodule is configured with three image sets: a set of nominal images used to populate the memory bank (train/good), a set of defective images (test/abnormal), and a separate held-out set of nominal images (test/normal) – the composition already anticipated for the dataset in Section 4.5.3. Memory bank construction follows the two stages described in Sections 2.6.4 and 2.6.5: locally aware patch features are extracted from intermediate backbone layers for every nominal training image, and the resulting embedding set is reduced via greedy coreset subsampling, with the sampling ratio (0.1) controlling the trade-off between memory-bank size and inference latency versus representational fidelity to the full nominal feature distribution. Because coreset sampling is not deterministic between runs by default, a global random seed is set explicitly before constructing the datamodule, so that the same nominal images always yield the same subsampled memory bank across repeated training runs – a prerequisite for the experiments in Chapter 6 to be reproducible.

The trained model is exported for deployment in multiple backend formats – native PyTorch, and OpenVINO with several precision/compression settings (FP32, FP16, INT8 post-training quantization).

The model operates on 256×256 inputs. `anomalib`'s default `PatchCore` pre-processor resizes each image to 256×256 and applies ImageNet normalization using the per-channel mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$ for the red, green, and blue channels, respectively:

$$x' = \frac{x - \mu}{\sigma}$$

No center crop is applied, since it is not explicitly configured in the training script; the default `anomalib` pre-processing for PatchCore performs resizing and normalization only. The normalization is consistent with the ImageNet-pretrained WideResNet-50 backbone, whose weights were learned using inputs normalized with these statistics. Using the same normalization during feature

extraction therefore preserves the input distribution expected by the pretrained backbone. This differs from the 224×224 center-cropped configuration used to reproduce the original PatchCore paper's results, which `anomalib` supports but does not apply by default.

At inference time, the configured model path is checked once at startup to identify the backend (`torch` or `openvino`). This identifier is then reused both to initialize the corresponding inferencer and to prepare the input frame. For the Torch backend, the image is passed directly to `anomalib`, which handles the required preprocessing internally. For OpenVINO, `main.py` explicitly resizes the frame to 256×256, converts it to channel-first format, and scales pixel values to $[0, 1]$ before passing it to the model. The same backend identifier is therefore used throughout the inference pipeline, avoiding repeated checks of the model path. The same normalization statistics and resize target are applied consistently across training and both inference backends, ensuring the input distribution seen by the backbone remains unchanged between training and deployment.

5.5 Real-Time Inference

The inference loop polls a buffer directory for newly written frame files, processes them one at a time in filename order, and removes each frame from the buffer once processing has completed. Each iteration delegates the actual per-frame work to a dedicated processing function, whose stages mirror the pipeline of Section 5.1:

- The frame is aligned using the rotation function resolved once at startup from the active profile (Section 5.3).
- The aligned frame is converted to the tensor layout required by the resolved model backend (Section 5.4) and passed to the inferencer, producing a raw anomaly map.
- The anomaly map is min–max normalized and resized to the original frame resolution. The region of interest is not a fixed set of coordinates shared across variants: because the four cap types differ in physical size and are captured within the same fixed camera frame, each cap-type profile defines its own region, corresponding to the pixel extent the cap occupies in that frame. Restricting score aggregation to this region excludes areas of the frame that do not belong to the cap itself – background, bottle neck margins, or other structures visible in the

acquired image, where an elevated anomaly score would be meaningless for the inspection task. Rather than aggregating the anomaly score over the whole frame, scoring is restricted to the cap's region of interest and reduced to a single value using a configurable percentile of the pixel-level scores within that region, rather than a simple maximum:

Code 5.5: Region-restricted, percentile-based anomaly score (adapted from `compute_cap_score`).

```
def compute_cap_score(heatmap_full, cfg):
    y1, y2 = cfg["TAPPO_Y1"], cfg["TAPPO_Y2"]
    x1, x2 = cfg["TAPPO_X1"], cfg["TAPPO_X2"]
    percentile = cfg["PERCENTILE_TAPPO"]

    cap_roi = heatmap_full[y1:y2, x1:x2]
    return float(np.percentile(cap_roi, percentile))
```

A percentile-based aggregation is less sensitive to isolated noisy pixels in the anomaly map than the raw maximum would be, while still being dominated by the highest-scoring region when a genuine defect is present. The percentile value differs for each cap type and is configurable through the corresponding profile, rather than fixed across variants.

- The resulting score is compared against the profile's threshold to produce a `GOOD`/`REJECT` verdict, and an annotated output image – the original cap region blended with a colour-mapped heatmap overlay – is written to disk. Output retention follows a configurable policy: `REJECT` frames are always saved, while `GOOD` frames are saved only if an explicit flag is enabled, since saving every nominal frame at line speed would otherwise accumulate a large volume of images with little diagnostic value. Figure 5.4 shows an example of a REJECTED cap with a seal defect.

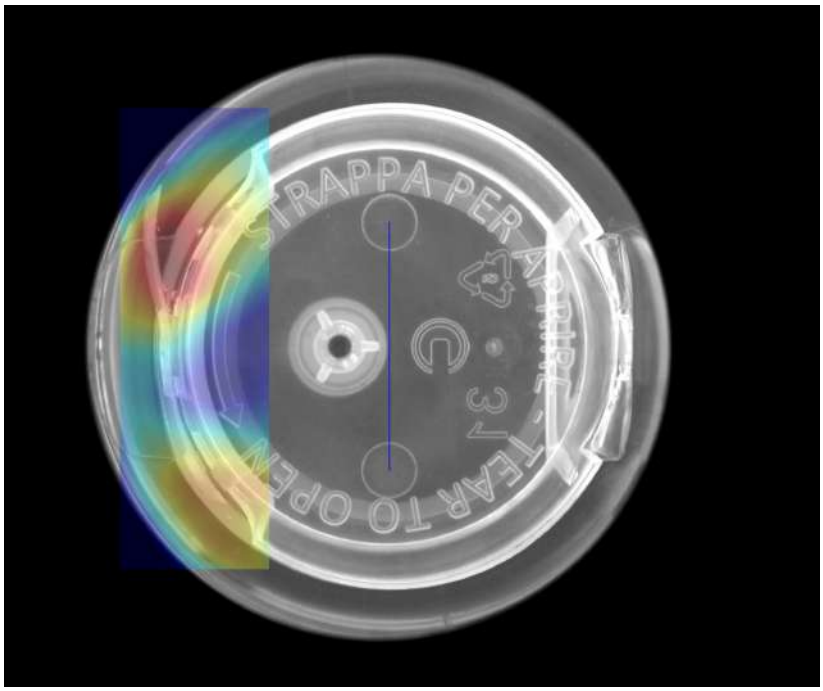


Figure 5.4: REJECTED Transparent cap for seal defect

Robustness during continuous operation is handled at the level of the main loop. If an exception occurs while processing a frame, for example due to a failed alignment, an invalid tensor, or an I/O error, the exception is logged together with the corresponding filename and the frame is moved to a dedicated quarantine directory. The loop then continues with the next pending frame. This prevents a single problematic frame from interrupting the inspection of subsequent bottles, while preserving the failed input for later analysis.

A separate exception handler is used when removing an input frame after successful processing. Since the inspection result has already been produced and saved, any failure during this cleanup step is treated as a warning rather than a fatal error. This allows the inspection loop to continue even in the presence of transient or external file-system issues.

All relevant events, including startup, per-frame results, processing latency, quarantine actions, and warnings, are recorded using a structured logger. The logger writes both to the console and to rotating log files stored in a dedicated `logs/` directory (`logs/inspection.log`), with a maximum size of 10 MB per file and up to five backup files (approximately 50 MB of retained history in total), providing a persistent diagnostic history independently of the console output.

Figure 5.5 shows an example of the generated log file, including startup messages and per-frame

inspection results.

```
2026-07-21 16:19:06 [INFO] STARTING AI INSPECTION MODEL — REAL TIME
2026-07-21 16:19:06 [DEBUG] Global parameters loaded from DB: {'BUFFER_DIR': 'buffer_telecamera', 'OUTPUT_DIR':
'risultati_linea', 'DEVICE': 'cpu', 'TAPPO_TYPE': 'tappo_trasparente_grande', 'OVERLAY_ORIGINALE': 0.7,
'OVERLAY_HEATMAP': 0.3, 'PAUSA_BUFFER_VUOTO': 0.05}
2026-07-21 16:19:06 [INFO] Active cap type (TAPPO_TYPE): 'tappo_trasparente_grande'
2026-07-21 16:19:06 [DEBUG] Profile loaded for 'tappo_trasparente_grande': {'descrizione': 'Tappo trasparente grande,
allineamento via template matching', 'MODEL_PATH':
'model/tappo_trasparente_grande/full_set/torch/weights/torch/model.pt', 'DEVICE': 'cpu', 'ROTATION_METHOD':
'template_matching', 'TEMPLATE_PATH': 'orientation/template_matching.jpg', 'ROTATION_PARAMS': {}, 'TAPPO_Y1': 310,
'TAPPO_Y2': 1700, 'TAPPO_X1': 350, 'TAPPO_X2': 800, 'SOGLIA_TAPPO': 0.19, 'PERCENTILE_TAPPO': 99}
2026-07-21 16:19:06 [INFO] Rotation method 'template_matching' ready.
2026-07-21 16:19:06 [INFO] Loading model from model/tappo_trasparente_grande/full_set/torch/weights/torch/model.pt
2026-07-21 16:19:06 [DEBUG] Model path      : model/tappo_trasparente_grande/full_set/torch/weights/torch/model.pt
2026-07-21 16:19:06 [DEBUG] Model backend   : torch
2026-07-21 16:19:06 [DEBUG] Requested device: 'cpu' -> resolved to: 'cpu'
2026-07-21 16:19:06 [DEBUG] PyTorch CUDA    : False
2026-07-21 16:19:08 [INFO] PATCHCORE MODEL READY IN 1869.21 ms — WAITING FOR IMAGES
2026-07-21 16:19:08 [INFO] tappotrasparente_10_exptime20000us_f16_PC23030XS_WD53mm_aug037_a-2.76.bmp -> REJECT
CAP_DEFECT | cap score: 0.349 | latency: 480.6 ms
2026-07-21 16:19:08 [INFO]      Average over 1 frames: 480.6 ms
2026-07-21 16:19:09 [INFO] tappotrasparente_10_exptime20000us_f16_PC23030XS_WD53mm_aug038_a-1.07.bmp -> REJECT
CAP_DEFECT | cap score: 0.351 | latency: 467.8 ms
2026-07-21 16:19:09 [INFO]      Average over 2 frames: 474.2 ms
2026-07-21 16:19:09 [INFO] tappotrasparente_12_exptime20000us_f16_PC23030XS_WD58mm_aug025_a+1.88_GOOD.bmp -> GOOD  |
cap score: 0.138 | latency: 422.6 ms
2026-07-21 16:19:09 [INFO]      Average over 3 frames: 457.0 ms
2026-07-21 16:19:10 [INFO] tappotrasparente_12_exptime20000us_f16_PC23030XS_WD58mm_aug026_a+2.18_GOOD.bmp -> GOOD  |
cap score: 0.136 | latency: 483.2 ms
2026-07-21 16:19:10 [INFO]      Average over 4 frames: 463.5 ms
```

Figure 5.5: Screenshot of the structured log output, showing startup information and per-frame inspection results recorded in `inspection.log`.

The following figures show some examples of results for both GOOD and REJECTED caps:

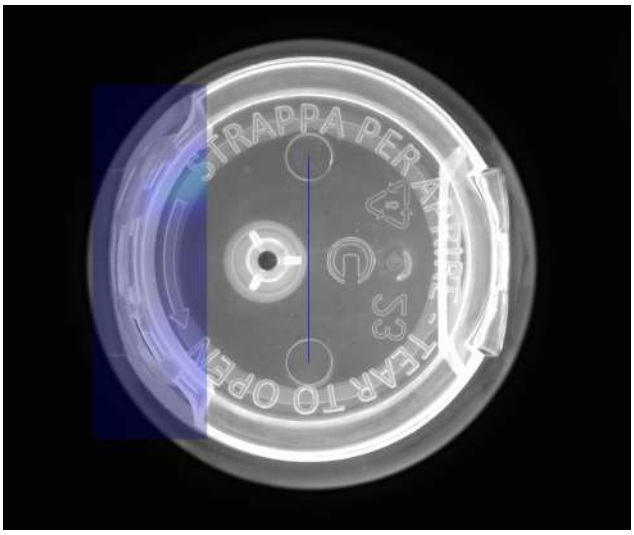


Figure 5.6: GOOD transparent cap

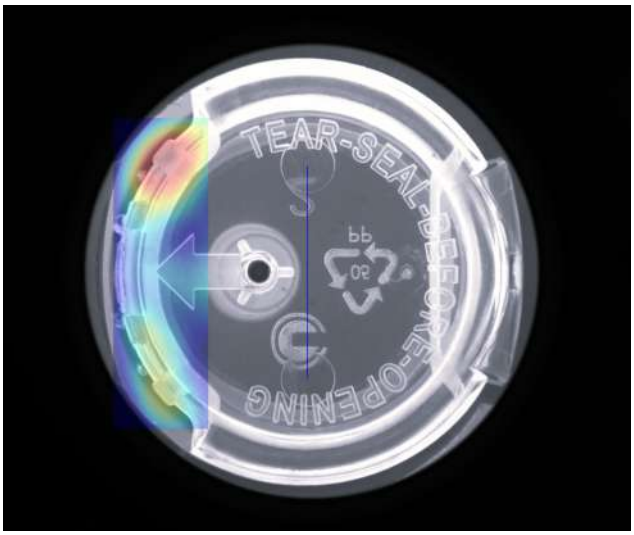


Figure 5.7: REJECTED transparent cap

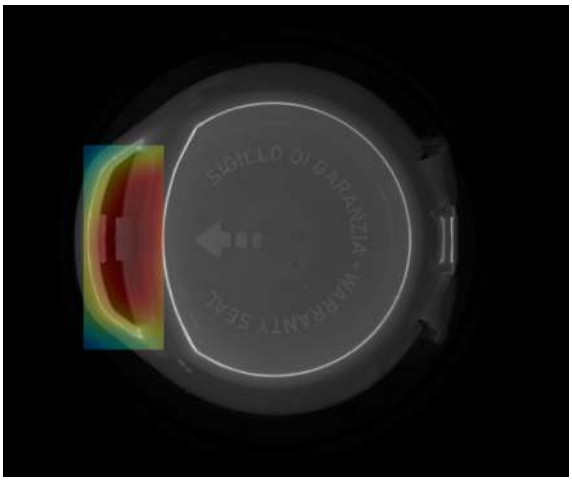


Figure 5.8: REJECTED red cap

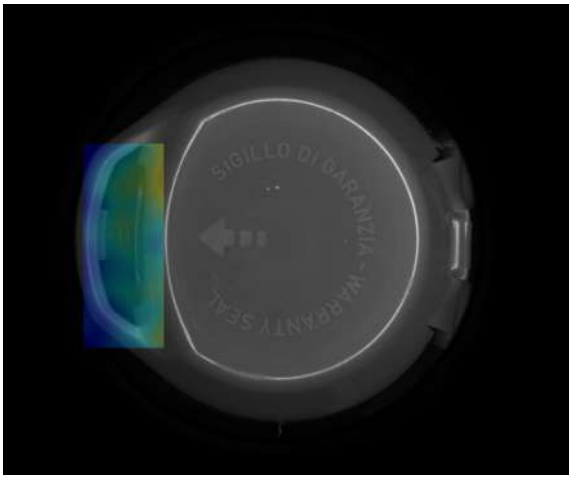


Figure 5.9: GOOD red cap

5.6 Implementation Details and Issues

Three implementation issues are worth discussing explicitly. They concern subtle failure modes or inefficiencies that did not cause visible errors during development and were identified only later.

Image format handling. Frames are received as BMP files, as required by the camera acquisition system. In an earlier version of the pipeline, output images were also saved in BMP format. The output handling was later modified to save annotated frames as PNG files instead, reducing storage requirements while preserving image quality.

PatchCore score normalization. As anticipated in Sections 2.6.7 and 4.5.3, `anomalib`'s min-

max calibration of the anomaly score requires the validation split used during training to contain both nominal and anomalous samples. Without anomalous samples, the normalization statistics cannot be properly estimated. In an earlier version of the training configuration, no defective-image directory was provided to the datamodule. As a result, no anomalous samples were available during the calibration step, causing every inference-time pixel-level anomaly score to be equal to 1.0, regardless of the input. The configuration was subsequently corrected by providing the defective-image directory and explicitly assigning both nominal and defective images to the validation/test splits, as described in Section 4.5.3.

Deterministic coreset sampling. Because greedy coreset subsampling (Section 5.4) has a stochastic component, two training runs on identical data can, without explicit seeding, produce different memory banks and therefore different anomaly scores for the same evaluation image. A global random seed is set at the start of the training script for this reason, ensuring that comparisons across backbones and parameter settings in Chapter 6 are attributable to the variable under study rather than to uncontrolled sampling variability. Empirically, different seed values were found to produce memory banks of varying quality, with some seeds yielding slightly better held-out performance than others. The seed adopted throughout this work, 42, was selected based on these preliminary trials and consistently provided excellent performance. However, no exhaustive search over seed values was performed, so no claim is made that 42 is globally optimal. Its primary purpose is to ensure reproducibility and enable fair, consistent comparisons across the experiments.

6. Experimental Results

6.1 Overview

The system presented in Chapter 5 was designed and calibrated on a real industrial dataset of bottle caps (Section 4.5). However, this dataset is not suitable for the quantitative model and backbone comparison carried out in this chapter. Each cap variant is based on only six real nominal and two to three real defective source images (Table 4.2). Even with rotational augmentation, the resulting test sets are too small and too easy to separate to provide statistically meaningful differences between models. A preliminary comparison on the industrial dataset confirmed this: almost every configuration achieved near-ceiling AUROC/F1 values, making it difficult to distinguish between model families and backbones.

For this reason, the model and backbone comparison in this chapter (Sections 6.2 to 6.5) is instead performed on the controlled external benchmark MVTec AD [5]. This dataset is used only as a proxy for general-purpose model behaviour under conditions with more data. This choice requires some clarification, since the MVTec AD object categories are visually unrelated to bottle caps. The comparison is therefore not intended to draw conclusions about the industrial cap-inspection task, and it does not use the production-specific concepts introduced in Chapter 5. In particular, the region-of-interest scoring of `compute_cap_score`, `SOGLIA_TAPPO`, and the per-cap-profile calibration have no equivalent on MVTec AD and are not used in this part of the chapter. The purpose of this experiment is simply to answer a question that the industrial dataset cannot answer on its own: *given enough data, how do the considered model families and backbones compare in general image-level anomaly detection?* This question was already raised, but left open, when the same model families and backbones were reviewed theoretically in Section 3.4.

A second, independent experiment, reported in Section 6.6, returns to the industrial dataset. Its purpose is different: rather than comparing models, it evaluates a specific design choice made in Chapter 5, in particular the use of a per-cap-profile, geometry-specific rotation alignment method (`slot_detection.py` for `tappo_rosso` and marker-based template matching for the transparent-cap profiles) instead of a generic, off-the-shelf image-registration method. This experiment evaluates

two points:

- (i) whether PatchCore is sensitive to rotation
- (ii) whether a generic keypoint-based alignment method would have been sufficient as an alternative to the personalised approach used in the system.

The chapter is organised as follows: Section 6.2 describes the experimental setup for the MVTec AD comparison; Section 6.3 defines the evaluation metrics used; Section 6.4 compares PatchCore backbones; Section 6.5 compares model families; Section 6.6 presents the rotation-sensitivity and alignment-method experiment on the industrial dataset; Section 6.6.2 discusses qualitative results; and Section 6.7 summarises the findings and their implications for Chapter 7.

6.2 Experimental Setup (Model and Backbone Comparison)

The experiments in Sections 6.4 and 6.5 were run with a dedicated experimental script, kept intentionally separate from the production training script (`src/training.py`, Chapter 5). This separation reflects a difference in purpose: the production script generates the models actually deployed on the four cap profiles, while the experimental script exists solely to produce the comparative results discussed in this chapter and is not part of the deployed system.

6.2.1 Dataset

Experiments were performed on the `transistor` category of MVTec AD [5], which provides several hundred nominal training images and a larger and more diverse test set than the industrial cap dataset. The validation split ratio was set to 0.2.

6.2.2 Models and Backbones

The following model families, already reviewed in Section 3.4, were evaluated:

- **PatchCore** [15], with three backbones: `wide_resnet50_2`, `resnet50`, and `resnet18` (all using `layer2` and `layer3`, coreset sampling ratio 0.1, 9 nearest neighbours);
- **PaDiM** [7], with backbone `wide_resnet50_2` (`layer1`, `layer2`, `layer3`);

- **FastFlow** [19], with backbone `wide_resnet50_2`;
- **Reverse Distillation** [8], with backbone `wide_resnet50_2`.

Two method families discussed in Section 3.4, EfficientAD and RD++, were not included. EfficientAD requires an external ImageNet directory for its "hard feature loss" and a training regime that departs from the one-shot/short-training setting shared by the other methods; it was therefore excluded rather than adapted, to keep the comparison methodologically consistent. RD++ is not available as a standalone class in `anomalib==2.3.3`, which only exposes the base Reverse Distillation model used here.

For PatchCore and PaDiM, training is a single-pass memory-bank/statistics construction step with no epochs in the conventional sense. For FastFlow and Reverse Distillation, a maximum of 50 epochs was used with early stopping. Training stopped automatically when the validation AUROC no longer improved significantly (patience 20, minimum delta 0.0005). Gradient clipping (clip_val = 1.0, norm-based) was applied to both backpropagation-trained families as a mitigation against the training instability discussed in Section 6.5.

6.2.3 Hardware and Software

All experiments were run with `anomalib==2.3.3` on a laptop equipped with an Intel(R) Core(TM) Ultra 7 155H processor and 32 GB of RAM. The system runs a 64-bit operating system on an x64-based processor. The experiments were performed using the CPU.

Fit time and inference latency reported in this chapter were measured on the same laptop. During these runs, the machine was connected to mains power but under other concurrent load from other processes. Therefore, these measurements should be considered indicative rather than reliable absolute performance figures. This limitation is taken into account when reporting the corresponding results in Table 6.1.

6.3 Evaluation Metrics

6.3.1 Image-level AUROC

The image-level Area Under the Receiver Operating Characteristic curve summarises, over all possible decision thresholds, how well the model's anomaly score separates nominal from defective items at the whole-image level. A value of 1.0 indicates that a threshold exists that perfectly separates the two classes in the test set; a value of 0.5 indicates no better than random guessing; values substantially *below* 0.5 indicate that the score is systematically inverted (defective items score *lower* than nominal ones), which is discussed as a distinct failure mode in Section 6.6. Computed with `anomalib.metrics.AUROC` on the fields `pred_score` and `gt_label`.

6.3.2 Image-level F1-Score

The harmonic mean of precision and recall at the image level, computed at a specific operating threshold. In the experiments of Sections 6.4 and 6.5, the threshold is chosen automatically by the model's `PostProcessor` (an adaptive threshold calibrated on the validation split to maximise F1). In Section 6.6, F1 is instead reported both at the fixed `SOGLIA_TAPPO` threshold actually used in production for the relevant cap profile, and at a per-condition optimal threshold (see that section for details).

6.3.3 Fit Time and Inference Latency

Fit time is the wall-clock time (seconds) required to fit the model on the training split; for PatchCore/PaDiM this is the single-pass feature extraction and memory-bank/statistics construction, for FastFlow/Reverse Distillation the full backpropagation training loop including early stopping. Inference latency (ms/image) is measured using the same Torch export path (`ExportType.TORCH`) and `TorchInferencer` used in production (Section 5.6), averaged over real test images after a short warm-up discard, deliberately distinct from the time measured by `engine.test()` (which includes metric computation and Lightning overhead and is not representative of real inference cost). See the hardware caveat in Section 6.2.

Pixel-level metrics (pixel AUROC, AUPRO) are not reported in this chapter: the evaluator used for these experiments (`build_evaluator()` in `run_experiments.py`) was deliberately simplified to compute only the two image-level metrics above, since pixel-level ground truth is not available for the industrial dataset and was judged unnecessary complexity for the MVTec comparison, whose purpose here is limited to image-level model/backbone ranking (Section 6.1).

6.4 Backbone Comparison (PatchCore)

Table 6.1 isolates the effect of the feature-extraction backbone on PatchCore, comparing `wide_resnet50_2`, `resnet50`, and `resnet18` on the MVTec AD `transistor` category, all other hyperparameters held fixed (Section 6.2).

Table 6.1: Model and backbone comparison on MVTec AD (`transistor`). Fit time and ms/image are indicative only; see the hardware caveat in Section 6.2.

Model	Backbone	Image AUROC	Image F1	Fit time(s)	Inference
PatchCore	wide_resnet50_2	**0.9948**	**0.9524**	624.97	414.97
PatchCore	resnet50	0.9909	0.9032	548.14	374.97
PatchCore	resnet18	0.9792	0.8966	495.82	**147.78**
PaDiM	wide_resnet50_2	0.9109	0.8775	**260.50**	344.40
FastFlow	wide_resnet50_2	0.9486	0.8333	23173.89	167.75
Reverse Distillation	wide_resnet50_2	0.9642	0.9180	7465.99	318.43

Restricting attention to the three PatchCore rows: image AUROC and F1 both decrease monotonically as the backbone narrows, from `wide_resnet50_2` (0.9948 / 0.9524) to `resnet50` (0.9909 / 0.9032) to `resnet18` (0.9792 / 0.8966). This is consistent with the original PatchCore ablation [15], where `wide_resnet50_2`'s larger per-layer channel width is shown to capture richer local features than narrower backbones. It is also consistent with a qualitative observation made during production-side testing on the industrial `tappo_trasparente_grande` variant (Section 4.3). In this setting, inference is performed on the aligned region of interest (ROI) containing the cap, rather than on the full production image. Under the same ROI-based preprocessing and evaluation procedure,

resnet50 and resnet18 produced some misclassifications that wide_resnet50_2 did not. For this profile, defects can include a partially displaced seal rather than a fully absent one, resulting in a subtler visual discrepancy that may be better captured by a backbone with a richer feature representation. The MVTec AD comparison provides additional quantitative evidence that could not be obtained reliably from the industrial dataset alone: the observed ordering is also maintained on a substantially larger and more diverse dataset, suggesting that the differences are not simply an artefact of the limited size of the industrial test set.

6.5 Model Family Comparison

The remaining rows of Table 6.1 compare the four model families introduced in Section 6.2. PatchCore (wide_resnet50_2) achieves the highest image-level AUROC (0.9948), followed by Reverse Distillation (0.9642), FastFlow (0.9486), and PaDiM (0.9109).

Beyond predictive performance, the fitting time shows a substantial difference between the model families. PatchCore and PaDiM, which do not require iterative gradient-based optimisation for fitting their nominal feature representations, require less than eleven minutes in these experiments. FastFlow and Reverse Distillation, in contrast, require substantially longer training times: approximately 23,174 seconds (≈6.4 hours) for FastFlow and 7,466 seconds (≈2.1 hours) for Reverse Distillation. FastFlow therefore required roughly three times the fitting time of Reverse Distillation while achieving a lower AUROC.

Nevertheless, the difference between the training times is substantial and is practically relevant for the intended application. In particular, retraining may need to be repeated as new cap profiles are introduced, making fitting time an additional consideration alongside anomaly-detection performance.

Inference latency is also directly relevant to deployment, since the production line operates at roughly one bottle per second, imposing an effective per-image budget of approximately 1000ms (Section 7.4). All six configurations in Table 6.1 comfortably satisfy this budget, with measured latencies ranging from 147.78ms (PatchCore resnet18) to 414.97ms (PatchCore wide_resnet50_2). Notably, wide_resnet50_2 – the backbone identified in Section 6.4 as the strongest performer on both image AUROC and F1 – is also the slowest of the three PatchCore backbones, though still

well within the available budget. This is consistent with the average inference latency observed on the deployed production system itself (Chapter 5), which sits at a comparable order of magnitude. The margin to the 1000ms budget is therefore large enough, even under the indicative (rather than rigorously controlled) timing conditions of these experiments (Section 6.2), that latency does not act as a binding constraint on the choice of `wide_resnet50_2` as the preferred backbone: the accuracy gain it provides over narrower backbones is obtained without meaningfully compromising the system's ability to keep pace with the production line.

6.6 Rotation Sensitivity and Alignment Method

The production system (Chapter 5) aligns each incoming frame to a canonical orientation before scoring it, using a method specific to each cap profile's geometry: `slot_detection.py` (detection of a slot/notch feature) for `tappo_rosso`, and marker-based template matching (`alignment.py`) for the transparent-cap profiles. This section empirically evaluates whether the per-profile alignment design is necessary by addressing two questions:

- (i) is PatchCore sensitive to image rotation, as expected from the rotation sensitivity of the CNN backbone?
- (ii) would a *generic*, non-profile-specific alignment method provide a sufficiently effective substitute for the personalised alignment procedure?

6.6.1 Method

A single PatchCore `wide_resnet50_2` model (the best-performing configuration identified in Section 6.4) was fitted once on the standard `tappo_trasparente_grande` training set (Section 4.5) and evaluated, without refitting, on three variants of the same test set:

- **baseline**: the test set as normally evaluated (residual misalignment only, as produced by the dataset's own augmentation pipeline);
- **no_alignment**: each test image rotated by a large, fixed angle (one of 15°, 30°, 60°, 90°, 120°, 180°, 270°), with *no* correction applied – simulating a failed or skipped alignment step;
- **generic_alignment**: the same rotated images, corrected using a generic, non-profile-specific method: ORB keypoint detection, brute-force Hamming matching against a nominal reference image, and a partial affine transform estimated via RANSAC (`cv2.estimateAffinePartial2D`), implemented in `src/alignment_generic.py`. This method uses no prior knowledge of the cap's physical features (no slot or marker detection), unlike either production alignment method.

Results across the seven rotation angles are aggregated (mean ± standard deviation) rather than reported individually, since the per-condition breakdown is available in the accompanying result files but does not change the conclusions drawn here.

6.6.2 Results

Table 6.2: Rotation robustness on `tappo_trasparente_grande`, PatchCore `wide_resnet50_2`. Image AUROC reported as mean ± standard deviation across the seven tested angles.

Condition	Image AUROC
No alignment	0.548 ± 0.118
Generic (ORB) alignment	0.692 ± 0.117

Under the production pipeline – geometric alignment specific to the cap profile (`slot_detection.py` or marker-based template matching, as appropriate) applied to the same rotated test images, correcting them back to the canonical orientation – this configuration achieves 100% image AUROC and 100% image F1: every nominal and defective image is correctly separated. This condition is not included in Table 6.2 because, consistent with the small-dataset ceiling effect discussed in Section 6.1, a perfect score on this limited test set is expected from the separability of the data itself and is not, on its own, discriminative evidence about alignment quality; it is reported here only as the reference point the other two conditions are measured against.

Two findings emerge. First, PatchCore is clearly *not* rotation-invariant: removing alignment entirely drops mean AUROC to 0.548, close to the 0.5 value expected of a non-discriminative score, down from the near-perfect performance obtained under the production alignment described above. A standard CNN backbone is not inherently rotation-invariant: convolutional layers are approximately translation-equivariant, but a rotated version of an otherwise identical image is not guaranteed to produce the same feature representation. Since PatchCore's memory bank is built exclusively from nominal images in their canonical (aligned) orientation, the patch-level features extracted from a rotated image are, in general, less well represented in that memory bank, which increases nearest-neighbour distances and, consequently, the resulting anomaly score – regardless of whether the image is actually defective. PatchCore's patch-level nearest-neighbour matching

therefore compares local features against a memory bank built from correctly-oriented patches, and a substantial rotation moves those local features out of the distribution the memory bank represents.

Second, and more directly relevant to the design choice being tested, the generic ORB-based alignment method only partially recovers the lost performance: mean AUROC rises to 0.692, still well short of the near-perfect performance obtained under the production alignment.

Together, these two results support the design choice made in Chapter 5: a generic, off-the-shelf alignment method is not, on this evidence, an adequate substitute for the profile-specific alignment actually deployed, even though PatchCore's sensitivity to rotation error is severe enough that *some* form of accurate alignment is clearly required.

6.7 Discussion

This chapter deliberately separates two different experimental questions, run on two different datasets, and the conclusions should be read accordingly.

On MVTec AD (Sections 6.4–6.5), used purely as a controlled, sufficiently-sized proxy, PatchCore with `wide_resnet50_2` was the strongest configuration by image AUROC/F1, with a clear and monotonic backbone ordering (`wide_resnet50_2` > `resnet50` > `resnet18`) consistent with theoretical expectations from Section 3.4 and with qualitative production-side observations on the transparent-cap profile. Reverse Distillation was competitive in accuracy but, together with FastFlow, substantially more expensive to fit – a difference large enough to be practically relevant regardless of the uncontrolled hardware conditions of this particular measurement. These results should not be read as a claim about accuracy on the industrial cap-inspection task itself (Section 6.1): their role is to validate, on data abundant enough to be informative, model and backbone choices that the industrial dataset alone is too small to validate directly.

On the industrial dataset (Section 6.6), the rotation-sensitivity experiment supports two specific claims that bear directly on a design decision made in Chapter 5: PatchCore requires accurate rotational alignment to function, and a generic, non-profile-specific alignment method is not sufficient to provide it. This is direct evidence in favour of the per-cap-profile alignment methods (`slot_detection.py`, marker-based template matching) actually implemented and developted.

Taken together, the two halves of this chapter support the same overall picture already stated

in Section 6.1: the deployed system (Chapter 5) rests on choices – model family, backbone, and alignment method – that are individually well-supported by controlled experimental evidence, even though the industrial dataset itself remains too small to provide a direct, statistically robust accuracy estimate for the deployed configuration as a whole. Validating that remaining question on an expanded, real production dataset is the necessary next step proposed in Chapter 7.

7. Conclusions and Future Work

7.1 Summary of the Thesis

This thesis presented the design, implementation, and experimental evaluation of a deep learning-based visual inspection pipeline for detecting defects in the safety seals of vinegar bottle caps, developed in collaboration with Giacobazzi. The system covers four distinct cap variants (tappo rosso, tappo trasparente grande, tappo trasparente piccolo, tappo nero), each handled through a dedicated calibration profile, and combines a pericentric image acquisition setup with a PatchCore-based anomaly detection model, per-profile geometric rotation alignment, and a region-of-interest scoring function (`compute_cap_score`) calibrated against a two-tier configuration system (SQLite `ConfigDB` and `cap_profile.yaml`).

The work was structured around two complementary lines of experimental evidence. On a controlled external benchmark (MVTec AD, `transistor` category), Chapter 6 compared PatchCore against PaDiM, FastFlow, and Reverse Distillation, and compared PatchCore across three backbones of increasing capacity, in order to validate model and backbone choices that the industrial dataset alone is too small to validate directly. On the industrial dataset itself, a dedicated experiment tested PatchCore's sensitivity to rotation and the adequacy of a generic, non-profile-specific alignment method as a substitute for the per-profile geometric alignment actually deployed.

7.2 Achievement of the Objectives

Section 1.1 set out five specific objectives for this work. This section reviews each in turn.

- **Objective 1 – Image acquisition pipeline.** A pericentric acquisition system was designed and characterised in Section 4.1 and motivated directly by the constraints of the production line (continuous line motion, no possibility of stopping or rotating the bottle) and compared explicitly against the two conventional alternatives – multi-camera setups and mechanical rotation – both discarded as impractical for this line speed. The resulting optical distortion introduced by the pericentric projection was characterised and carried forward into the

preprocessing and region-of-interest design of Chapter 5. This objective was fully achieved.

- **Objective 2 – Novel industrial dataset.**A dataset of normal and defective bottle caps was collected and organised across the four profiles (Section 4.5). This objective was achieved, but the number of real source images per profile is small (Table 4.2). Therefore, the dataset has limited value for statistically meaningful evaluation (Section 6.1).

- **Objective 3 – PatchCore-based anomaly detection.** A PatchCore model with a pretrained backbone was implemented and integrated into the production pipeline, producing both an anomaly score (via `compute_cap_score`) and an anomaly map for defect localisation. This objective was fully achieved and is the core contribution of the deployed system (Chapter 5).

- **Objective 4 – Comparison with other anomaly detection methods.** This objective was achieved with one deviation from the original plan stated in Section 1.1: EfficientAD, initially proposed as a comparison target, was excluded because it requires an external ImageNet directory and a training regime incompatible with the one-shot/short-training setting shared by the other methods evaluated (Section 6.2). PaDiM, FastFlow, and Reverse Distillation were compared against PatchCore instead, on both detection performance and computational cost (Sections 6.4–6.5), which covers the intent of the original objective even though the specific method set changed.

- **Objective 5 – Robustness analysis under realistic industrial conditions.** The rotation-sensitivity experiment of Section 6.6 directly addresses this objective, showing both that accurate alignment is a hard requirement for PatchCore and that a generic alignment method is not an adequate substitute for the profile-specific one deployed. This objective was achieved for the rotation dimension specifically; broader robustness questions (illumination variation, cap wear, sensor drift over time) were not tested and are discussed below as limitations.

7.3 Limitations

The following limitations were identified over the course of this work and are stated here explicitly, consistent with the framing already adopted in Chapter 6:

- **Small industrial dataset.** Each cap variant is built from only six real nominal and two to three real defective source photographs (Section 6.1), which produces a ceiling effect on that dataset and prevented it from being used for the quantitative model/backbone comparison in this thesis.

- **Single external benchmark category.** The MVTec AD comparison (Sections 6.4–6.5) was carried out on a single category (`transistor`), after `bottle` was discarded for producing saturated results. The resulting ranking of models and backbones is informative but has not been confirmed across a wider range of object categories.

- **No repeated-seed evaluation.** All experiments in Chapter 6 report a single run per configuration; no confidence intervals or seed variability are reported, which is a relevant caveat given how close some of the compared configurations are (e.g. PatchCore `wide_resnet50_2` vs. Reverse Distillation, Table 6.1).

- **Rotation experiment scope.** The rotation-sensitivity and alignment-method experiment (Section 6.6) was reported only for `tappo_trasparente_grande`; the equivalent experiment on `tappo_rosso` was inconclusive, plausibly for the same small-dataset ceiling effect discussed throughout the thesis, and was not included.

- **Uncontrolled timing measurements.** Fit time and inference latency reported in Chapter 6 were measured on a laptop connected to mains power but under other concurrent load, and are indicative only, not reliable absolute figures (Section 6.2).

- **No real production deployment.** The system was developed and validated in an offline/prototype setting; it has not been installed or tested on an actual production line, and questions such as long-term robustness to sensor drift, cap design changes, or line-speed variability under real operating conditions remain open.

7.4 Future Work

7.4.1 Real Production Deployment

The most direct next step is deploying the system on an actual production line at throughput conditions matching the target of roughly one bottle per second, i.e. a latency budget of about 1000ms, rather than validating it offline as done throughout this thesis. This would serve two purposes at once: it would provide the large, continuously-growing dataset needed to overcome the small-sample ceiling effect discussed in Section 7.3, and it would expose failure modes that a controlled, offline evaluation cannot – for instance gradual sensor drift, physical wear of the acquisition setup, or changes in ambient lighting over long operating periods.

7.4.2 Model Compression for Deployment on Constrained Hardware

The current implementation already exports the trained model to multiple backend formats, including OpenVINO with FP32, FP16, and INT8 post-training quantization, but this thesis did not quantitatively evaluate the trade-off between these precision settings. Investigating this trade-off is a natural next step for two reasons. While inference latency on the development machine used in this thesis was already well within the target budget (Section 6.3), this does not extend to the lower-cost or power-constrained hardware a production deployment might ultimately require, where the memory and latency benefits of reduced-precision inference would matter more directly. First, reduced-precision inference (FP16, and especially INT8) is typically associated with substantially lower memory footprint and inference latency, which could allow the system to run on cheaper or more power-constrained hardware than the development machine used in this thesis – a relevant consideration for an industrial deployment where per-line hardware cost matters. Second, anomaly detection scores are sensitive to small numerical perturbations in a way classification logits are not always assumed to be, so it is not safe to assume accuracy is preserved under quantization without measuring it directly: a systematic comparison of image AUROC/F1 against inference latency across FP32, FP16, and INT8 for the deployed PatchCore configuration would need to be run on both the MVTec benchmark and the industrial dataset before recommending a specific precision setting for production use.

7.4.3 Strengthening the Experimental Evidence

Beyond deployment and compression, three concrete extensions would directly address the limitations listed in Section 7.3: repeating the MVTec AD model/backbone comparison across additional object categories to test how consistent the observed ranking is; running each configuration across multiple random seeds to report variance rather than a single point estimate; and directly quantifying the residual rotation error left by the generic ORB-based alignment method in degrees, as already flagged in Section 6.6, rather than inferring it indirectly from the AUROC gap.

7.5 Closing Remarks

This thesis has covered the full path from image acquisition to deployed anomaly detection model, and, in Chapter 6, from a general model/backbone comparison down to a targeted validation of a specific design choice made in Chapter 5. The single largest step remaining before the system could be trusted in an unattended production role is the one identified throughout this chapter: moving from offline validation to a real deployment, on real production data, at real scale.

Bibliography

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. "Ganomaly: Semi-supervised anomaly detection via adversarial training". In: *Asian conference on computer vision*. Springer. 2018, pp. 622–637.

[2] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *Machine learning for data science handbook: data mining and knowledge discovery handbook* (2023), pp. 353–374.

[3] Kilian Batzner, Lars Heckler, and Rebecca König. *EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies*. 2024. arXiv: 2303.14535 [cs.CV]. URL: https://arxiv.org/abs/2303.14535.

[4] Kilian Batzner, Lars Heckler, and Rebecca König. "Efficientad: Accurate visual anomaly detection at millisecond-level latencies". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2024, pp. 128–138.

[5] Paul Bergmann et al. "MVTec AD—A comprehensive real-world dataset for unsupervised anomaly detection". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2019, pp. 9584–9592.

[6] Haoyang Cao et al. "Feasibility and transferability of transfer learning: A mathematical framework". In: *arXiv preprint arXiv:2301.11542* (2023).

[7] Thomas Defard et al. *PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization*. 2020. arXiv: 2011.08785 [cs.CV]. URL: https://arxiv.org/abs/2011.08785.

[8] Hanqiu Deng and Xingyu Li. *Anomaly Detection via Reverse Distillation from One-Class Embedding*. 2022. arXiv: 2201.10703 [cs.CV]. URL: https://arxiv.org/abs/2201.10703.

[9] P Kingma Diederik and Welling Max. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[10] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[11] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[12] Joe Gugliotti. “Pericentric Lenses Explained”. In: *Quality* 63.12 (2024), pp. 36–36.

[13] Ehsan Norouznezhad et al. “A high resolution smart camera with GigE Vision extension for surveillance applications”. In: *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE. 2008, pp. 1–8.

[14] Keiron O’shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[15] Karsten Roth et al. “Towards total recall in industrial anomaly detection”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 14318–14328.

[16] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine vision algorithms and applications*. John Wiley & Sons, 2018.

[17] Tran Dinh Tien et al. “Revisiting Reverse Distillation for Anomaly Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 24511–24520.

[18] Markus Ulrich and Carsten Steger. “A camera model for cameras with hypercentric lenses and some example applications: M. Ulrich, C. Steger”. In: *Machine Vision and Applications* 30.6 (2019), pp. 1013–1028.

[19] Jiawei Yu et al. *FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows*. 2021. arXiv: 2111.07677 [cs.CV]. URL: https://arxiv.org/abs/2111.07677.