Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

University of Padua

Department of information engineering

Master's degree in
Computer Engineering

Multimodal quality inspection in the bread production cycle

Supervisor:
Prof. Alberto Pretto

Student:
Giuseppe Labate
ID number:
2095665

Academic year 2025-2026
Graduation date 13/04/2026

“Che l’auto non ha preso il gatto e il camion è tornato a casa
Nel cuore della notte”

— Cit.Corsi

Abstract

This thesis presents an embedded multimodal computer-vision framework for quality monitoring in a real industrial bread-production process, designed to operate in an edge environment without network connectivity. The objective is to introduce minimally invasive technological tools into a real, weakly structured industrial environment, enabling both automated quality inspection and statistical analysis of the production process. The system was developed and validated in the *SMACT Competence Center*, as part of a collaborative project led by FlexSight Srl in the context of the Vis.IO project, developed for IRINOX S.p.A. under the IRISS framework (Ref. COR 22612178, CUP H19J24001010004 - funded by NextGenerationEU, M4C2I2.3). The framework was deployed at two acquisition sites characterized by different sensing modalities and inspection objectives. At Site 1, an NVIDIA Jetson Nano DK coupled with an Azure Kinect acquires RGB-D data at the output of the dough-processing stage. At Site 2, a Raspberry Pi 5 equipped with a Raspberry Pi Camera Module 3 Wide acquires RGB images of baked bread during the final handling stage.

In both sites, product localization is performed through a DEIMv2-based object-detection pipeline trained from automatically labelled data. The downstream analysis then differs according to the production stage. At Site 1, the detected dough balls are segmented and converted into local 3D point clouds in order to estimate product volume through different geometric reconstruction strategies. At Site 2, the detected bread loaves are cropped and analyzed with an anomaly-detection model to assess the quality of the final baked product.

The experimental results confirm the effectiveness of the proposed framework in both scenarios. For dough-ball inspection, the rotated convex-hull method yields the most accurate volume estimates, showing the closest agreement with the expected reference interval among the evaluated methods. For baked bread inspection, the anomaly-detection branch achieved a near-perfect image-level AUROC of 0.999958 under the adopted evaluation protocol, while also yielding very low anomaly scores on normal samples and meaningful responses on anomalous examples. The corresponding anomaly maps localize the most relevant irregular regions.

Overall, the thesis shows that computer vision, deep learning, and 3D sensing can be effectively integrated into minimally invasive embedded systems to support data-driven quality monitoring in real industrial food-production environments.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Problem statement Nowadays, it is increasingly important to make company production processes as efficient and effective as possible. This requirement is even more critical in the food industry, where any waste directly translates into food loss and, consequently, a waste of resources.
To mitigate this issue in a global context that demands a more responsible use of raw materials, modern technologies such as artificial intelligence can be integrated with traditional production processes.

1.1 Thesis goal

The objective of this thesis is to collect and analyse real production data in order to evaluate the performance of the bread manufacturing process, with particular emphasis on its two main stages: dough-ball production and final bread baking. To assess product quality, the proposed pipeline must first determine the position of dough units and loaves at the different stages of the process. To this end, **DEIMv2** [1], a state-of-the-art real-time deep learning model based on the **DETR (DEtection TRansformer)** architecture [6], is employed for object detection. Once the bounding boxes have been identified, downstream analysis is carried out differently at the two acquisition sites.

- **Site 1**: located at the output of the dough-processing stage. At this site, an **NVIDIA Jetson Nano** [7] is connected to a **Microsoft Azure Kinect** [8] sensor, which acquires RGB-D data. These data are subsequently processed to estimate the **volume of the dough balls**. Depth information is essential for this task because it enables a reliable three-dimensional reconstruction of the dough units, which in turn supports accurate volume estimation through point-cloud processing or depth-based shape fitting techniques.

- **Site 2**: located at the point where the bread is removed from the baking tray. At this site, a **Raspberry Pi 5** equipped with a **Raspberry Pi Camera** captures RGB images of the loaves. Because the illumination conditions are not constant, the quality of the baking process is not assessed through handcrafted colour and shape descriptors alone. Instead,

a dedicated **anomaly detection network** is applied to the loaf regions in order to identify potentially defective products. This approach provides a more robust assessment of bread quality, since it can learn complex patterns associated with satisfactory and unsatisfactory baking outcomes even under varying lighting conditions.

(a) Site 1: output area where dough balls are released and collected on the blue table

(b) Site 2: area in which the baked bread is removed from the tray

Figure 1.1: Overview of Site 1 and Site 2

Lastly, once all the data have been extracted and processed, **statistical distributions** of dough-ball volumes and anomaly scores for baked loaves are constructed. These distributions constitute the final analytical output of the system and provide a quantitative overview of production performance.

For each monitored variable, descriptive statistical indicators such as the mean, variance, and standard deviation are computed. The mean provides an estimate of the nominal production target, while the variance and standard deviation quantify dispersion and process variability. By analysing both the distribution curves and their associated statistical descriptors, it becomes possible to evaluate the consistency and repeatability of the production process over different time intervals and conditions. In particular, deviations in volume distributions may indicate instability in the forming stage, whereas systematic shifts in anomaly-score distributions may reveal variations in baking conditions or product quality.

The resulting statistical summary therefore represents a quantitative decision-support tool for quality evaluation. Rather than relying solely on qualitative visual inspection, the proposed framework enables objective measurement of process stability, variability, and deviation from expected operating conditions.

To further enhance the interpretability of the results, visualisations such as 3D point-clouds or meshes of dough-ball volumes (shown in Figure 5.10) and heatmaps of anomaly scores (illustrated in Figure 5.14) are generated to provide intuitive insight into the underlying data distributions. These visual aids facilitate the identification of shapes, trends, clusters, or outliers that may not be immediately apparent from numerical summaries alone.

In summary, the thesis aims to develop an integrated computer-vision and embedded-AI system for the real-time monitoring and performance evaluation of an industrial bread production process. By leveraging state-of-the-art object-detection and anomaly-detection techniques, the system extracts meaningful information from raw sensory data and ultimately provides a comprehensive statistical overview of production quality and consistency. In this context, the thesis demonstrates how computer-vision and embedded-AI systems can be integrated into an industrial environment to transform raw sensory data into actionable statistical insights, thereby bridging the gap between low-level image acquisition and high-level production-performance evaluation.

1.2 Internship

The thesis project was carried out during an internship at **FlexSight Srl** [9]. The internship was part of an industrial project led by FlexSight in collaboration with SMACT [10], contributing to the final demonstrator of the Vis.IO project, which was developed on behalf of IRINOX S.p.A. under the IRISS framework (Ref. COR 22612178, CUP H19J24001010004 - funded by NextGenerationEU, M4C2I2.3). During the internship period, the author was assigned responsibility for the whole project development, which was developed largely independently, with periodic collaboration and technical support from colleagues, when required.

A key aspect of the work involved **on-site interactions at the production facility** to gather requirements and clarify technical specifications. This included structured discussions with project supervisors to better understand project objectives, resolve ambiguities, and address feasibility constraints. Coordination with factory operators was also essential to determine which activities could be performed without interfering with production, to identify operational limitations, and to adapt specific procedures in order to enable proper data acquisition.

Extensive **information exchange** between supervisors, operators, and the project team significantly contributed to the design of the data acquisition and processing pipeline, as well as to the final outcome of the project. The planning phase, combined with careful coordination of operator activities, ensured that data could be collected accurately, safely, and efficiently.

1.2.1 Pipeline summary

Following the planning phase, the installation of the cameras and computing units was carried out to initiate the first operational stage: the **data acquisition phase**. To collect data, a motion detection algorithm based on background subtraction on RGB images was developed. The algorithm was designed to be highly configurable, with parameters controlling the ROI (Region of Interest), motion threshold, pixel-value threshold, exposure, and other acquisition settings. This design choice was motivated by the need to install and calibrate the devices directly on site, where storing all configuration parameters in a *.yaml* file made the setup procedure significantly faster and more practical.

Moreover, the two sites differed in terms of production line layout, lighting conditions, and camera placement. For this reason, a flexible algorithm with externally configurable parameters was required in order to adapt the acquisition pipeline to the specific conditions of each environment.

The project was structured into three main components:

- **Data collection using a motion detection algorithm**: this module was designed to acquire data by continuously monitoring the workstation areas (Figure 1.1). This phase consisted of an automated and fully autonomous data collection process conducted over a period of two weeks. During this interval, the system operated without human intervention, continuously analyzing the scene and capturing relevant frames based on motion-triggered events. The cameras were oriented toward the production lines and configured to acquire images only when a product passed beneath them. By defining appropriate motion thresholds and a Region of Interest (ROI), the dataset was built efficiently while minimizing redundant frames.

- **Object detection**: for this component, the **DEIMv2** [1] neural network was trained to detect bread loaves and dough balls using the dataset acquired in the previous phase. To support training, the dataset was automatically labeled using **Language Segment-Anything** [11], an open-source tool built on top of SAM2 and GroundingDINO that generates object masks and bounding boxes from textual prompts. After training, the DEIMv2 network was able to localize and classify the products within the images.

- **Quality analysis**: given the detected bounding boxes, two different downstream analyses were performed.

  - For **dough balls**, volume-related features were extracted and aggregated to compute statistical distributions. Dough volume was estimated by reconstructing the point cloud within each detected region, using RGB-D information acquired by the Kinect sensor [8]. After isolating the dough from the surrounding background, the local 2.5D geometry was processed to obtain a full 3D geometry and volume approximation of the product. The dough volume was then estimated using three different geometric methods over the entire test set, and the resulting volume distributions were analyzed

and compared. Further details on point-cloud processing, 3D completion, and volume estimation are provided in Chapter 4.

– For **baked loaves**, the detected loaf regions were analyzed through an anomaly detection approach rather than through handcrafted color or shape descriptors. In this case, the objective was to identify potentially non-conforming products and to derive anomaly-score distributions over time for process monitoring. To perform this task, the **EfficientAD** [5] model from anomalib [12] was adopted. EfficientAD was selected for its efficiency, speed, and practical ease of integration, making it suitable for real-time industrial quality assessment scenarios. Further details on the training data composition and on the evaluation procedure adopted at Site 2 are provided in Chapter 4.

1.2.2 Practical challenges and system robustness

During the internship period and the initial deployment of the project, several practical challenges emerged that required dedicated engineering solutions.

Robustness to Power Interruptions

Throughout the design phase, it was observed that power interruptions occurred frequently within the production facility. Such events resulted in unexpected shutdowns of the embedded devices, interrupting the acquisition pipeline and compromising data continuity. Since the data acquisition process was required to operate autonomously for a two-week period without human intervention, this issue represented a critical risk to the success of the experimental campaign.

To mitigate this problem, the entire software architecture was designed to be resilient to power outages. In particular, the acquisition pipeline was encapsulated within a system service configured to automatically restart upon system boot.[13]
A *system service* is a background process managed by the operating system's service manager (e.g., `systemd` in Linux-based systems). Services are designed to:

- Automatically start at system boot,
- Restart upon unexpected failure,
- Run independently of user login sessions,
- Provide logging and monitoring capabilities.

At both sites, a user-level `systemd` service was implemented to execute the image acquisition script at startup. The service invokes a dedicated bash launcher script responsible for:

- Navigating to the project directory,
- Activating the Python virtual environment,
- Launching the acquisition or processing script.

The service was configured with automatic restart policies (e.g., `Restart=always`) and a short restart delay to ensure that, upon restoration of electrical power, the system autonomously resumed operation without manual intervention.

This design choice significantly increased the reliability of the deployment and reduced maintenance overhead in the industrial environment.

Illumination Variability at Site 2

An additional challenge was related to the physical configuration of Site 2. The bread extraction station is located on a metallic and reflective table positioned in front of two large windows (as shown in Figure 1.1b).

This configuration introduced significant illumination variability throughout the day due to:

- Natural light intensity fluctuations,
- Direct sunlight reflections on the metallic surface,
- Shadows generated by operators during manual extraction.

Since the primary objective at Site 2 is visual quality assessment of baked bread through an anomaly detection model, uncontrolled illumination variations could directly affect the reliability of model predictions. In particular, variations in brightness and color temperature may introduce systematic biases in RGB-based inference.[14, 15]

To ensure consistency and minimize illumination-related variability, a high-intensity artificial lighting source was installed above the acquisition area (as shown in Figure 1.2).
The artificial illumination:

- Increases dominance over ambient light,
- Reduces the impact of daylight fluctuations,
- Improves the repeatability and stability of color measurements.

By stabilizing the lighting conditions, the system achieves more reliable visual inference, which is critical for robust anomaly detection and for detecting deviations in the baking process.[14, 15]

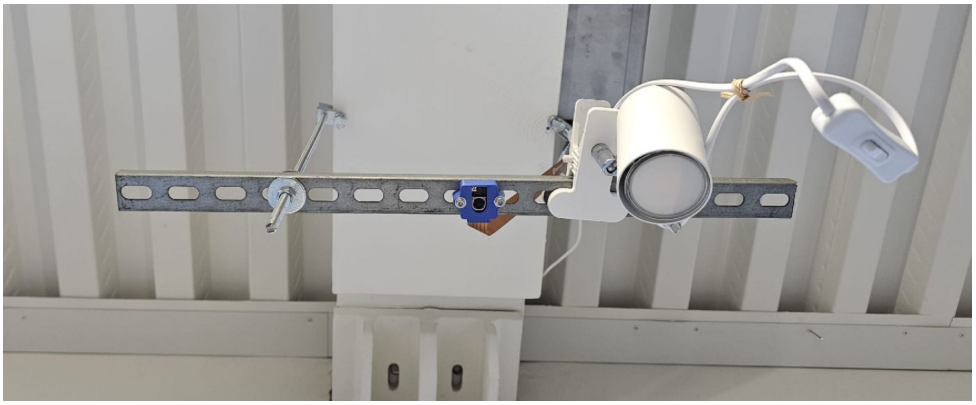


Figure 1.2: Light source placed next to the Raspberry Pi Camera to illuminate the scene and keep lighting as constant as possible

1.2.3 Companies

The internship was conducted in an industrial and research-oriented environment, enabling the practical application of computer engineering methodologies to real-world production scenarios. The activities included the analysis of technical requirements, the design and implementation of software solutions, and the integration of hardware and software components within embedded and computer vision systems.

Moreover, the internship context facilitated continuous interaction with engineers and researchers, contributing to the development of professional competencies such as structured problem-solving, project organization, system reliability design, and compliance with industrial standards and operational constraints.

FlexSight

FlexSight is an innovation-oriented company operating in the fields of computer vision, artificial intelligence, and robotics. Established in 2019, it originates from research activities carried out in artificial intelligence and robotic perception, and maintains strong links with the University of Padua research ecosystem [16]. Its work focuses on the development of advanced vision-based solutions for industrial and robotic applications, including high-resolution 3D reconstruction, neural rendering, autonomous perception, and intelligent sensing systems [9]. In this sense, FlexSight represents a concrete example of technology transfer from academic research to industrial practice, providing a context in which advanced perception and automation methodologies can be applied to real-world industrial problems.

Figure 1.3: FlexSight logo: an innovation-oriented company operating in the fields of computer vision, AI and robotics.

SMACT

SMACT is one of the eight Italian Competence Centers established within the national Industry 4.0 framework. It is a public-private partnership involving universities, research centres, and companies, with the goal of supporting the digital transformation of enterprises, particularly small and medium-sized ones, through technology transfer, training, demonstration activities, and innovation projects [17]. Originating in the North-Eastern Italian ecosystem, SMACT acts as a bridge between academic research and industrial practice by promoting the adoption of advanced digital technologies and by facilitating collaboration among research institutions, technology providers, and companies [10]. Within this context, the internship project can be framed as part of a broader initiative aimed at transferring computer engineering and artificial intelligence methodologies to real industrial production processes.

Figure 1.4: SMACT logo: one of the Italian Competence Centers for Industry 4.0, which provided the context for the internship project.

Chapter 2

Background Material

2.1 Computer Vision

Computer vision is a subfield of artificial intelligence (AI) that equips machines with the ability to process, analyze and interpret visual inputs such as images and videos. It uses machine learning to help computers and other systems derive meaningful information from visual data.[18]

Object detection and image segmentation are two fundamental tasks in the field of computer vision, widely used for scene understanding and automated visual analysis. While both aim at identifying relevant objects within an image, they differ significantly in terms of output representation, computational complexity, and application scope.

2.1.1 Object detection

Object detection [18] aims to localize and classify objects within digital images. In practice, it combines two complementary tasks:

- **Object localization** identifies the location of specific objects in an image by drawing bounding boxes around them.
- **Image classification** distinguishes the category to which objects belong.

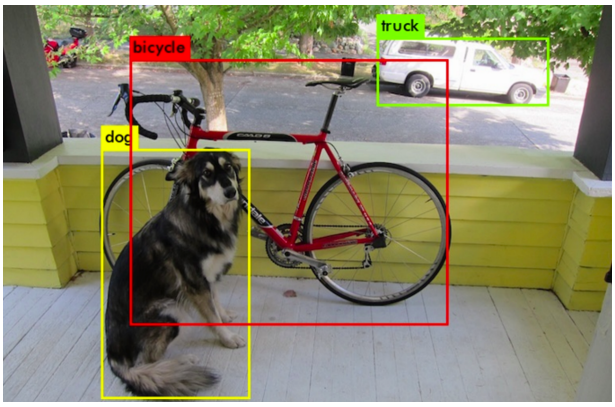


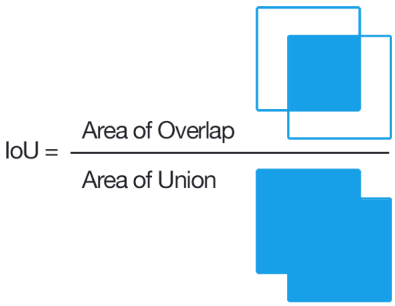
Figure 2.1: In the image, the detected objects are enclosed by bounding boxes and annotated with their corresponding class labels

IoU in object detection

One of the most widely used concepts in object detection is the **Intersection over Union (IoU)** metric [19]. IoU quantifies the overlap between a predicted bounding box and the corresponding ground-truth bounding box. It is defined as the ratio between the area of intersection and the area of union of the two bounding boxes:

$$\text{IoU} = \frac{\text{Area}(B_p \cap B_{gt})}{\text{Area}(B_p \cup B_{gt})} \tag{2.1}$$

where B_p denotes the predicted bounding box and B_{gt} the ground-truth bounding box. In object detection, IoU is used both to assess localization quality and to determine whether a predicted box is sufficiently aligned with the corresponding annotation. The IoU metric ranges from 0 to 1, where 1 indicates perfect overlap between the predicted and ground-truth bounding boxes, while 0 indicates no overlap.



(a) Visualization of the Intersection over Union (IoU) metric, illustrating the areas of intersection and union between a predicted bounding box and a ground-truth bounding box.

(b) The IoU metric quantifies the degree of overlap between the predicted bounding box (in green) and the ground-truth bounding box (in red).

Figure 2.2: The Intersection over Union (IoU) metric visualization

Non-Maximum Suppression (NMS)

In addition to evaluation, IoU is also used during post-processing stages of object detection pipelines. Modern detection architectures often generate multiple bounding box proposals for a single object. To eliminate redundant detections and retain only the most reliable prediction per object, a post-processing technique known as **Non-Maximum Suppression (NMS)** is applied. NMS operates by first sorting all predicted bounding boxes according to their confidence scores in descending order. The bounding box with the highest confidence is selected as a valid detection, and all other boxes with an Intersection over Union (IoU) greater than a predefined threshold with respect to the selected box are suppressed. This process is repeated iteratively on the remaining boxes until no further candidates are available. Formally, given a set of predicted bounding boxes B_i with associated confidence scores s_i, NMS retains a subset of boxes such that highly overlapping detections are removed while preserving the most confident predictions. The IoU threshold plays a crucial role in determining the strictness of suppression: lower thresholds lead to more aggressive filtering, whereas higher thresholds allow more overlapping detections to remain.

NMS significantly improves the clarity and interpretability of detection results by ensuring that each object instance is represented by a single bounding box.

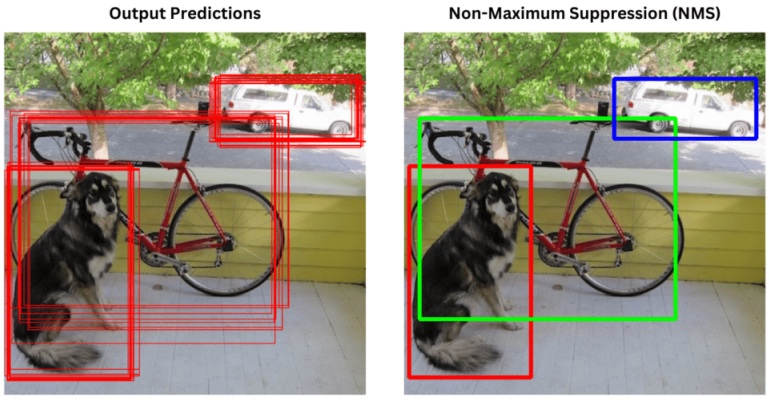


Figure 2.3: Non-Maximum Suppression (NMS) is a post-processing technique used in object detection to eliminate redundant bounding boxes and retain only the most confident prediction for each detected object.

Overall, object detection provides a coarse yet computationally efficient form of localization, making it particularly suitable for **real-time applications** and scenarios in which precise object boundaries are not strictly required. This balance between accuracy and computational cost makes it especially attractive for industrial environments characterized by constrained hardware resources.

2.1.2 Image segmentation

Image segmentation is the task of partitioning a digital image into meaningful regions at pixel level, so that each pixel is assigned to a class or to a specific object instance.[18]
Depending on the task formulation, segmentation can be divided into:

- **Semantic segmentation**: assigns each pixel in the image to a predefined category, without distinguishing between different object instances belonging to the same class.
- **Instance segmentation**: detects and segments each individual object instance in the image, producing a separate mask for each countable object (typically referred to as *things*), while usually not distinguishing amorphous background regions such as sky or ground (*stuff*).
- **Panoptic segmentation**: unifies semantic and instance segmentation by providing a complete scene understanding, where both individual object instances (*things*) and uncountable background regions (*stuff*) are segmented in a consistent and non-overlapping manner.[20]

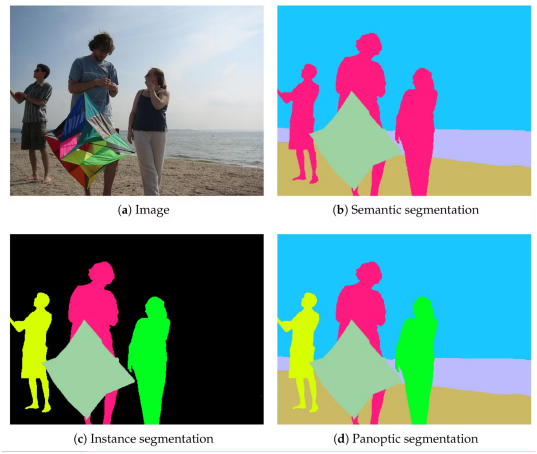


Figure 2.4: Different types of image segmentation

This fine-grained representation allows for a more accurate description of object shapes and boundaries than object detection, but it comes at the cost of higher computational complexity and increased sensitivity to noise, texture variability, and illumination changes.

Image segmentation can be addressed through both learning-based and classical image-processing approaches. Among the latter, clustering methods and background-subtraction techniques are often used to isolate regions on the basis of visual or geometric differences with respect to the surrounding scene.[21, 22]

From an evaluation perspective, segmentation is typically assessed through pixel-level overlap metrics. The most common ones include **Intersection over Union (IoU)**, the **Dice coefficient**, and task-specific extensions such as **mean IoU (mIoU)** for semantic segmentation, mask-based **Average Precision (AP)** for instance segmentation, and **Panoptic Quality (PQ)** for panoptic segmentation.

These metrics are discussed in more detail in the evaluation metrics section, where the criteria adopted for object detection and segmentation are presented in a unified way.

2.1.3 Classical segmentation methods

Clustering

Clustering is a family of unsupervised methods aimed at grouping data samples according to **similarity** in a chosen feature space.[21] In image analysis, the samples typically correspond to pixels, superpixels, or local feature descriptors, while similarity may be defined in terms of color, intensity, texture, spatial position, or depth.[21]

Within computer vision, clustering can be exploited as a classical segmentation strategy.[21] The underlying idea is that pixels belonging to the same physical region often share common visual or geometric properties, and can therefore be grouped into homogeneous clusters. Once the clusters have been identified, they can be interpreted as candidate regions of interest or as a first approximation of the object masks.[21]

Among the simplest and most widely used approaches is **k-means clustering**, originally introduced by MacQueen,[23] which partitions the input data into k clusters by minimizing the

within-cluster variance. When applied to images, k-means can be performed directly on RGB values, grayscale intensities, depth values, or combinations of these features.[23, 21] This makes it particularly useful in scenarios where the target objects can be separated from the background through relatively simple low-level cues.[21]

In the present work, clustering is relevant because dough segmentation can benefit from the grouping of pixels or 3D points with similar visual or geometric properties. In the considered setup, color and depth cues can facilitate the separation between the dough and the surrounding scene. In this sense, clustering can be considered a useful classical tool for preprocessing, exploratory segmentation, or the extraction of candidate regions in controlled industrial environments.[21]

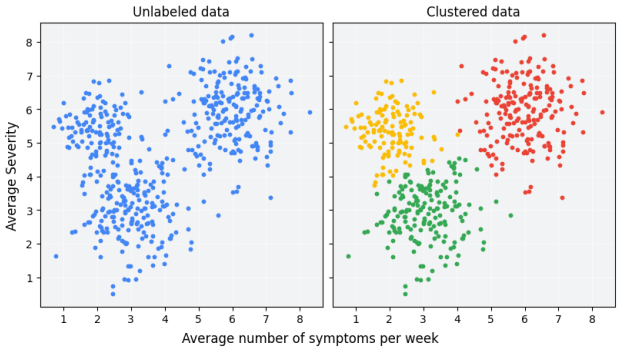


Figure 2.5: Visualization of k-means clustering applied to an image, where pixels are grouped into clusters based on their similarity and distance in the feature-space. Each cluster is represented by a different color, illustrating how the algorithm partitions the image into distinct regions.

Background subtraction

Background subtraction is another classical strategy for foreground extraction. The main idea is to maintain a model of the background and compare the current observation against it, so as to identify the regions that differ significantly from the static part of the scene.[22]

In vision-based applications, this approach is commonly used with static cameras to generate a foreground mask, that is, a binary image containing the pixels associated with the moving or changing objects in the scene.[22] More generally, the same principle can be extended to cases in which the background is known or can be estimated in advance, and the goal is to isolate the objects of interest by removing the supporting surface or the stationary scene structure.

In the present work, this idea is relevant in two complementary ways. First, background subtraction is used in a non segmentation context in the RGB stream as part of the motion-detection module for dataset acquisition. Second, a conceptually similar foreground-background separation is exploited on depth information to isolate the dough from the supporting table before the subsequent geometric analysis. Since the background (table) is static and can be reliably estimated in depth, this approach provides a computationally efficient way to segment the dough units without relying on complex learning-based models, which may be unnecessary in this controlled setting.

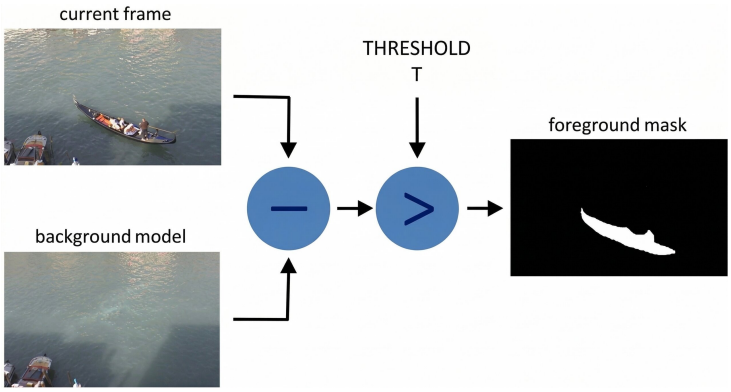


Figure 2.6: Background subtraction example scheme.

2.1.4 Evaluation metrics

The evaluation of computer vision models depends on the nature of the output produced by each task. Object detection predicts bounding boxes and class labels, whereas segmentation predicts pixel-level masks.[19, 20] For this reason, some metrics are shared across tasks, while others are specific to a particular output representation.

Metrics for object detection

In object detection, one of the fundamental geometric criteria, as previously said, is the **Intersection over Union (IoU)**. This metric is used to determine whether a prediction is considered correct under a predefined overlap threshold. Based on this matching criterion, **precision** and **recall** can be computed over the set of detections.[24, 25]

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

where

- TP (true positives) represents the number of bounding boxes correctly predicted as belonging to the class,
- FP (false positives) represents the number of bounding boxes incorrectly predicted as belonging to the class,
- FN (false negatives) represents the number of annotated objects that are missed by the detector.

Precision measures how many predicted detections are correct, while **recall** measures how many annotated objects are successfully retrieved. In object detection, these quantities are commonly aggregated through **Average Precision (AP)** and **mean Average Precision (mAP)**, which summarize the precision–recall trade-off across confidence thresholds and, depending on the benchmark, across multiple IoU thresholds.[25]

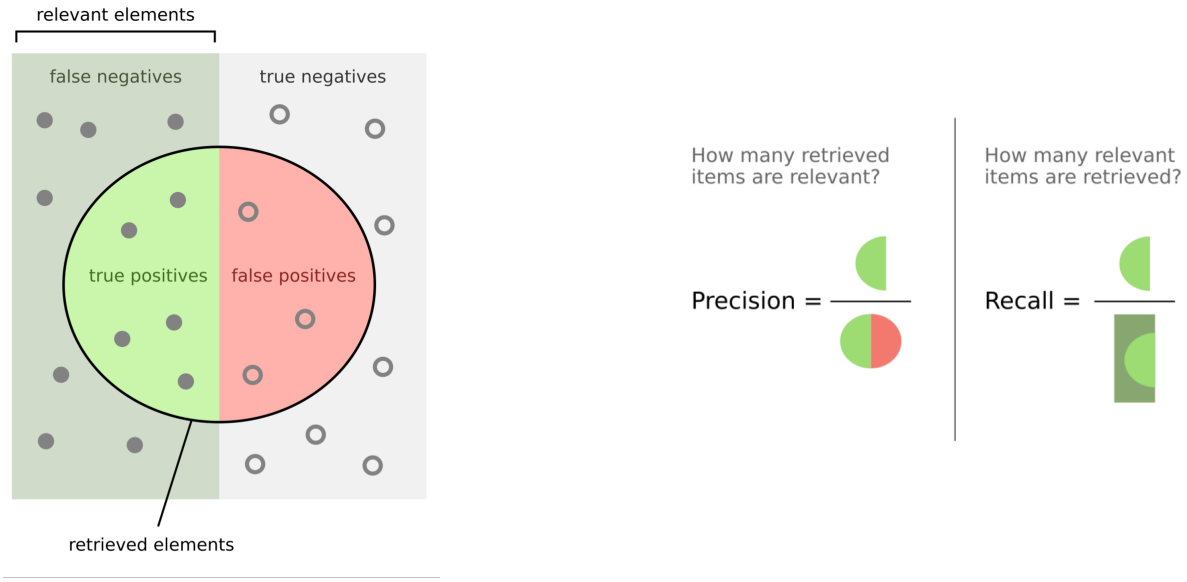


Figure 2.7: Precision and Recall visualization, illustrating the concepts of true positives, false positives, and false negatives predictions compared to ground-truth annotations.

Metrics for image segmentation

In segmentation tasks, evaluation is performed at pixel level by comparing the predicted mask with the ground-truth mask. For a given class, the **Intersection over Union (IoU)**, also referred to as the *Jaccard Index*, can be expressed as:[26]

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{2.4}$$

where A denotes the set of pixels predicted as belonging to the class and B denotes the set of ground-truth pixels for that class. An equivalent formulation in terms of classification outcomes is:

$$IoU = \frac{TP}{TP + FP + FN} \tag{2.5}$$

In semantic segmentation, IoU is often computed independently for each class and then averaged across classes, yielding the **mean Intersection over Union (mIoU)**.[27, 26] This metric is particularly informative in multi-class settings because it penalizes both over-segmentation and missed regions.

Figure 2.8: Visualization of the Intersection over Union (IoU) metric for image segmentation, illustrating the areas of true positives, false positives, and false negatives between a predicted segmentation mask and a ground-truth mask.

Other commonly adopted metrics include pixel-level **precision** and **recall**, whose harmonic mean is the **F1-score**, or equivalently the **Dice coefficient**:[24, 26]

$$F1 = Dice = \frac{2TP}{2TP + FP + FN} \tag{2.6}$$

In binary segmentation problems or anomaly-detection settings, the **Area Under the Receiver Operating Characteristic Curve (AUROC)** can also be used. AUROC evaluates the model's ability to discriminate between positive and negative pixels across different decision thresholds. However, in strongly imbalanced segmentation tasks, overlap-based metrics such as IoU or Dice are often considered more informative.[28, 26]

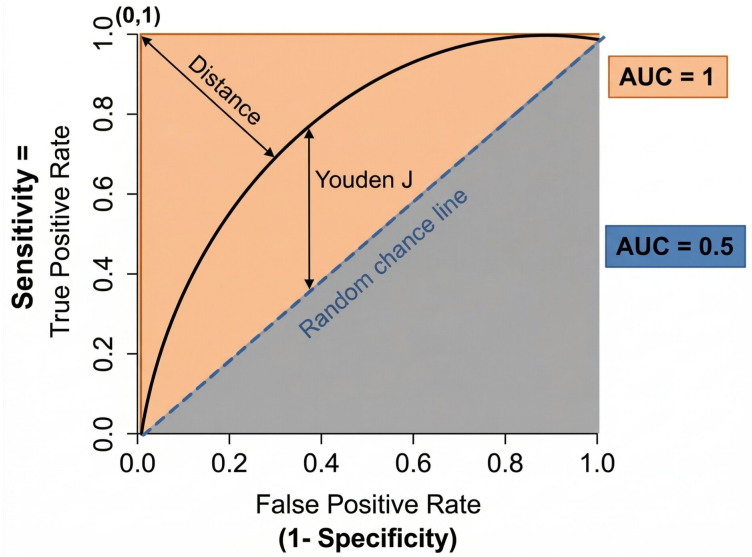


Figure 2.9: Visualization of the Area Under the Receiver Operating Characteristic Curve (AUROC) metric for image segmentation.

Metrics for instance and panoptic segmentation

For instance segmentation, COCO-style evaluation commonly reports **Average Precision (AP)** over mask IoU thresholds, since the task requires both correct localization and accurate mask prediction.[25]

Panoptic segmentation is commonly evaluated with **Panoptic Quality (PQ)**, a metric designed to jointly account for segmentation quality and recognition quality across both countable objects and background regions.[29]

Taken together, these evaluation metrics provide a quantitative basis for comparing different models and approaches across detection and segmentation tasks.[25, 26, 29]

2.2 Neural networks

Neural networks (NNs) are parametric function approximators inspired by the distributed information-processing paradigm of biological nervous systems. At a conceptual level, an NN is composed of interconnected layers of artificial neurons, where each neuron receives a set of inputs, combines them linearly through trainable weights and biases, and then applies a non-linear activation function. By stacking multiple layers, the network implements a nested composition of transformations that can model highly non-linear input–output relationships [30]. During training, the parameters are adjusted so as to minimize a task-specific loss function, thereby enabling the network to progressively improve its predictions from data [30].

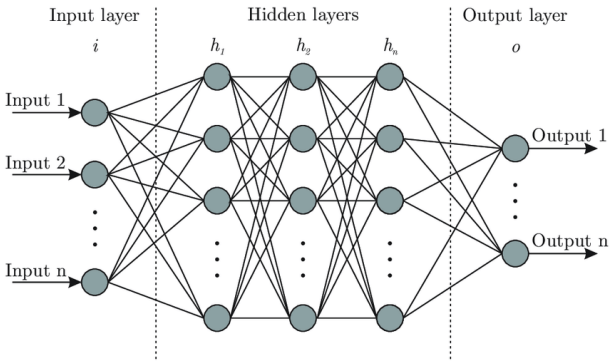


Figure 2.10: Schematic representation of a feedforward neural network with multiple hidden layers. Each layer applies a linear transformation followed by a non-linear activation, enabling progressively richer representations of the input.

The optimization of neural networks is typically performed through **gradient descent** and its stochastic variants. After computing the loss associated with a prediction, gradient descent updates the model parameters in the direction that most reduces the loss, with the size of each update controlled by the learning rate. Backpropagation provides the mechanism that makes this process computationally feasible: by applying the chain rule from the output layer back to the earlier layers, it efficiently computes the gradient of the loss with respect to each parameter in the network [31, 30]. Together, gradient-based optimization and backpropagation allow neural networks to learn complex functions directly from data.

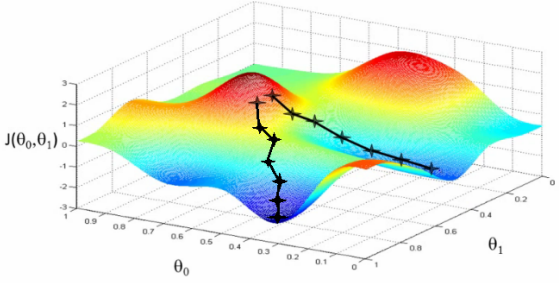


Figure 2.11: Illustration of gradient descent over a loss surface. At each iteration, the parameters are updated in the direction of the negative gradient so as to reduce the loss.

2.2.1 Deep Learning

While the term neural network refers broadly to a family of trainable models composed of interconnected artificial neurons, **deep learning** denotes the specific setting in which these models contain multiple representation-learning layers arranged in depth [30]. The practical importance of depth lies in its ability to learn hierarchical representations: lower layers tend to encode simpler patterns, whereas deeper layers combine them into progressively more abstract and task-relevant features [32, 30]. This property distinguishes deep learning from earlier machine learning pipelines, in which feature extraction was often designed manually before classification or regression.

By making end-to-end representation learning possible, deep learning allows the model to infer useful features directly from raw or weakly processed data rather than relying on handcrafted descriptors [30]. This shift has been one of the main reasons for the remarkable progress observed across computer vision, speech, and natural language processing. At the same time, the high representational capacity of deep models generally **requires large datasets**, substantial computational resources, and careful optimization procedures in order to achieve strong generalization performance [32]. A decisive milestone in this development was the success of deep convolutional networks in large-scale visual recognition, which accelerated both scientific interest and industrial adoption [33].

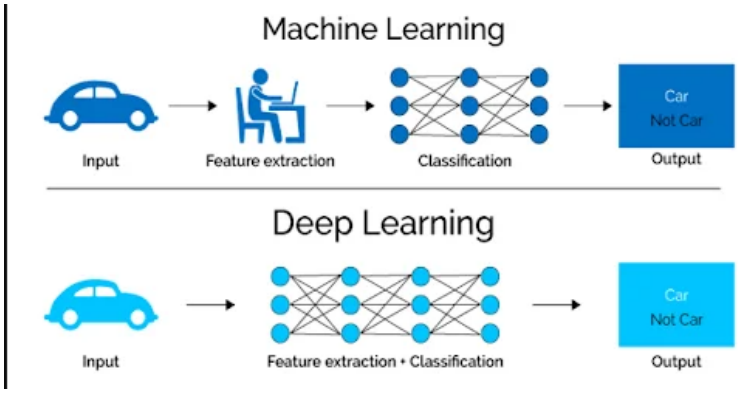


Figure 2.12: Schematic representation of a deep neural network, where feature representations are learned across multiple layers of abstraction and not manually engineered.

2.2.2 Convolutional Neural Networks

Among deep architectures for visual data, **Convolutional Neural Networks (CNNs)** have long represented the dominant paradigm. Their design is based on local receptive fields, **shared convolutional kernels**, and hierarchical feature extraction as for deep learning nets, which introduce useful inductive biases such as translation equivariance and improved parameter efficiency [34, 30]. In a convolutional layer, the same learnable kernel is applied across different spatial positions of the input image, producing feature maps that respond to recurring **local patterns**. This combination of local connectivity and parameter sharing drastically reduces the number of trainable parameters, making CNNs more efficient to optimize and, in many practical settings, less data-hungry than generic dense architectures for visual tasks [30, 34].

The use of kernels is particularly well suited to images because relevant visual structures, such as edges, corners, textures, and simple shapes, are inherently local. By learning detectors for these local patterns and reusing them across the whole image, CNNs can extract meaningful features while preserving spatial structure. This property generally improves generalization to unseen images and makes CNNs effective even when computational resources or dataset sizes are more limited than those required by less structured architectures [30, 34]. In practice, early layers tend to capture low-level structures, whereas deeper layers encode increasingly abstract semantic patterns. Additional operations such as spatial downsampling or pooling further enlarge the effective receptive field and support the progressive aggregation of information across layers [34]. This hierarchical organization explains the effectiveness of CNNs in tasks such as image classification, object detection, and semantic segmentation.

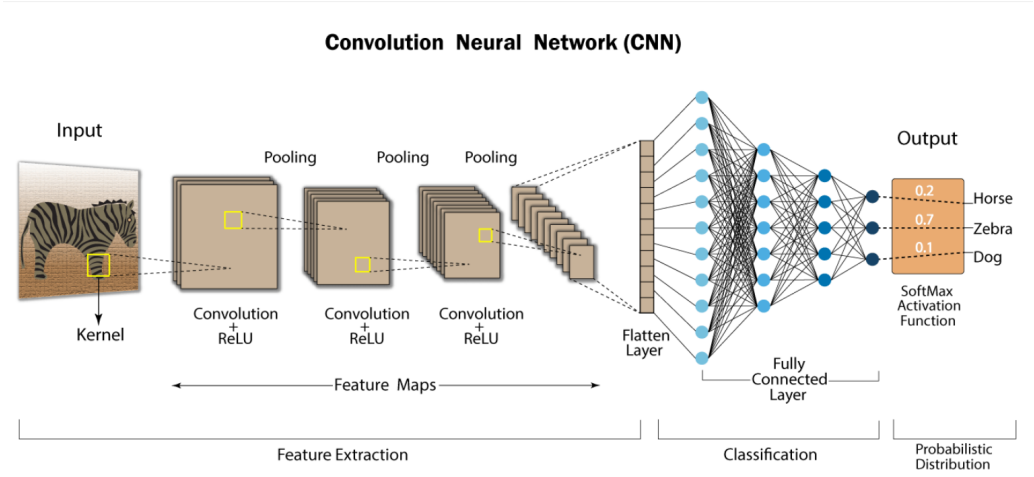


Figure 2.13: Typical convolutional neural network architecture for visual processing, in which convolutional and downsampling stages progressively extract hierarchical features from the input image.

In the context of this thesis, neural networks, deep learning, and CNNs provide the conceptual foundation for the architectures discussed in the following sections. Vision Transformers and RT-DETR will therefore be introduced as successive developments built upon, or reacting to, this baseline [35, 6].

2.3 Vision Transformers (ViT)

In recent years, **Vision Transformers (ViT)**[35], introduced by Dosovitskiy et al., have progressively reshaped the landscape of computer vision, offering an alternative to the long-standing dominance of convolutional neural networks (CNNs). Unlike the latter, which rely on local inductive biases such as translation equivariance and locality through sliding kernels, ViTs leverage the **self-attention mechanism** to capture global dependencies across the entire image from the very first layer. Originally inspired by Transformer architectures in natural language processing, ViTs process images as sequences of patches rather than as continuous grids.

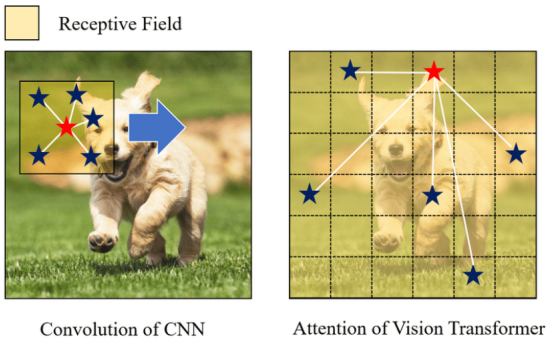
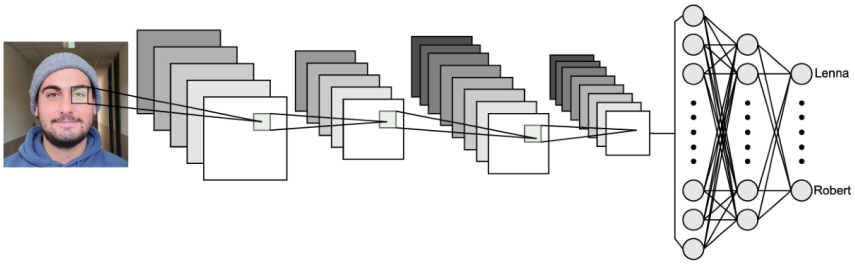
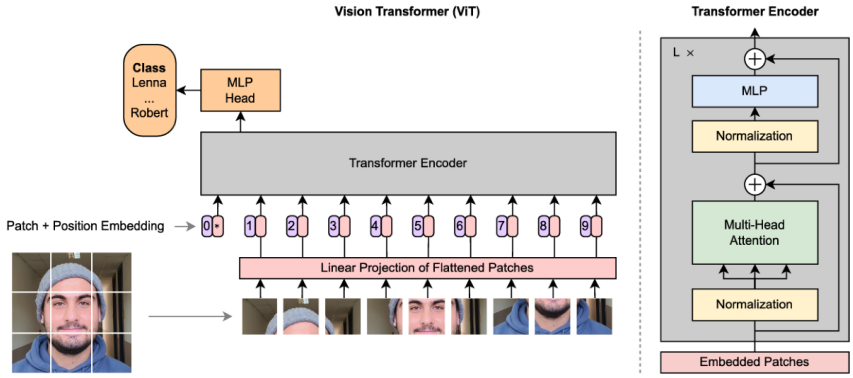


Figure 2.14: Receptive field comparison between CNNs and ViTs. While CNNs have a local receptive field that grows with depth, ViTs can capture global context from the very first layer due to their self-attention mechanism.

More specifically, an input image is divided into fixed-size patches, which are combined with the positional encodings to preserve the position information, then linearly projected into embedding vectors and treated as tokens. These tokens are processed through a Transformer encoder, where **self-attention mechanisms** allow the model to capture relationships and correlations between distant regions of the image. This ability to model **global context** is one of the main strengths of ViTs, especially when compared to CNNs, whose receptive field grows only progressively with depth.



(a) Common CNN architecture



(b) Vision Transformer architecture

Figure 2.15: Schematic comparison of CNN and ViT architectures. The CNN (a) processes the image through a hierarchy of convolutional layers with local receptive fields, while the ViT (b) treats the image as a sequence of patches and applies self-attention to capture global dependencies.

This architectural choice brings several advantages. When trained on large-scale datasets, ViTs can learn rich semantic representations and may outperform CNN-based approaches in high-capacity regimes [35]. At the same time, because vanilla ViTs incorporate weaker built-in

inductive biases than CNNs, they are generally less data-efficient and benefit substantially from large-scale pretraining before being transferred to downstream tasks [35].

However, these benefits come with non-negligible drawbacks. The self-attention operation has a **quadratic complexity** with respect to the number of tokens, which makes ViTs computationally demanding, especially for high-resolution inputs. This aspect has historically limited their use in real-time or resource-constrained environments.

Despite these limitations, ViTs have become a key component in many modern computer vision pipelines. In particular, they are widely used as backbone networks in object detection and segmentation frameworks (such as DINO [36], SAM [37] and DEIMv2 [1]), where their ability to extract both global context and fine-grained features proves especially valuable. This shift from local to global feature modeling represents one of the key paradigm changes introduced by Transformer-based architectures in computer vision.

2.4 RT-DETR

RT-DETR (Real-Time Detection Transformer) extends the DETR paradigm toward real-time object detection by preserving its end-to-end formulation while explicitly addressing the latency and computational limitations that hindered earlier transformer-based detectors [6]. As in DETR, object detection is formulated as a set prediction problem, so that the model directly outputs a fixed set of detections without relying on heuristic post-processing such as Non-Maximum Suppression (NMS). The main contribution of RT-DETR lies in making this formulation practical for real-time scenarios.

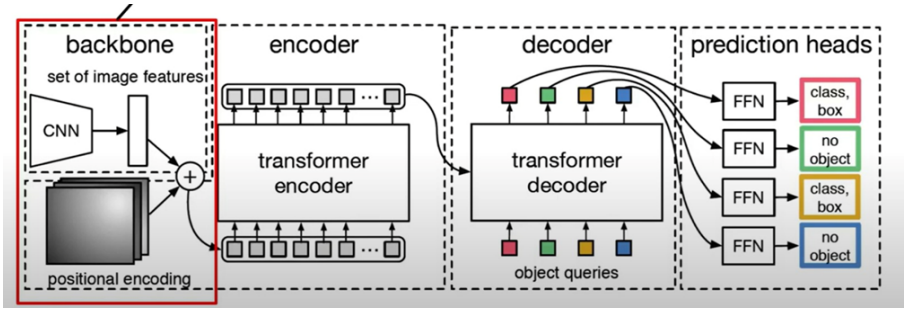


Figure 2.16: Original DETR architecture.

From an architectural perspective, RT-DETR retains the standard backbone–encoder–decoder structure of DETR-style detectors, with prediction heads for object classification and bounding box regression. In this pipeline, the encoder is responsible for processing and fusing the visual features extracted by the backbone, so as to build a more informative representation of the scene before object queries are introduced. The decoder then takes these encoded features together with a set of object queries and iteratively refines them into final detections, progressively improving class predictions and bounding box localization. However, RT-DETR introduces several refinements designed to improve the accuracy–speed trade-off. In particular, the model employs an **efficient hybrid encoder** to process multi-scale visual features more efficiently by decoupling

intra-scale interaction from cross-scale fusion [6]. This design reduces the computational burden of the encoder while still preserving the multi-scale information required for object detection.

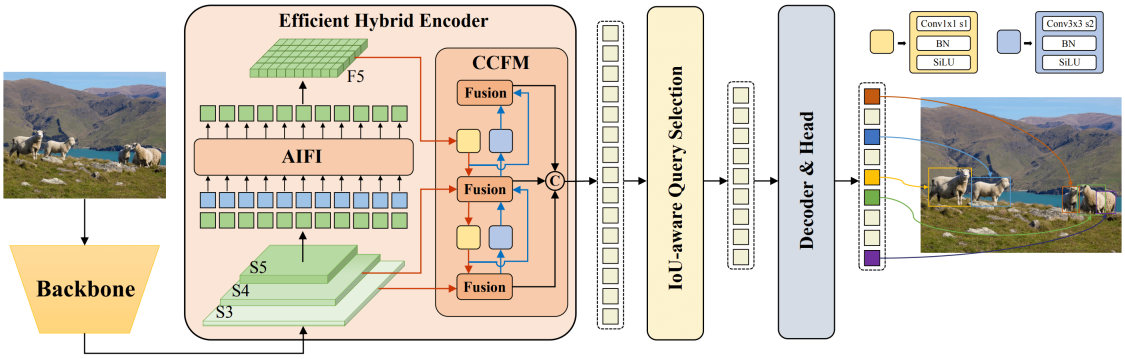


Figure 2.17: RT-DETR architecture.

Another distinctive component is the **uncertainty-minimal query selection** strategy proposed to initialize the decoder with higher-quality object queries [6]. Rather than forwarding a large set of generic queries, RT-DETR selects encoder features that are expected to be more reliable starting points for detection. Intuitively, the model favors features that are informative both for object classification and for localization, so that the decoder begins its refinement process from more promising candidate regions. This improves efficiency and contributes to stronger detection accuracy.

This query selection mechanism should not be interpreted as a simple hand-crafted thresholding rule, such as selecting all predictions above a fixed IoU value. Instead, it is part of the learned initialization process of the detector and is designed to provide the decoder with a compact set of high-quality queries. After this initialization, the decoder iteratively refines the selected queries into final predictions.

Furthermore, RT-DETR retains the one-to-one matching paradigm based on the **Hungarian algorithm**, ensuring that each predicted object is assigned to at most one ground-truth instance during training. This bipartite matching mechanism is central to DETR-style detectors because it removes duplicate assignments and enables an end-to-end detection pipeline without NMS [6].

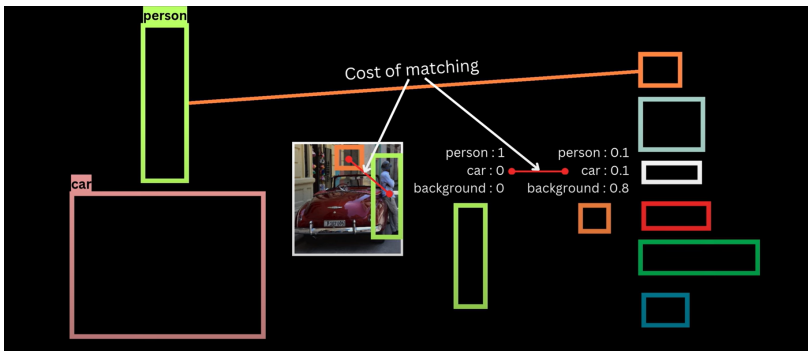


Figure 2.18: Hungarian matching in RT-DETR.

An additional practical advantage of RT-DETR is its **flexible speed tuning**: the inference speed can be adjusted by varying the number of decoder layers, allowing the model to adapt to different deployment constraints without retraining [6]. Overall, RT-DETR achieves a strong balance between detection accuracy and computational efficiency, making transformer-based object detection more compatible with real-time and resource-constrained applications. In the context of this thesis, RT-DETR is particularly relevant because it shows how transformer principles can be adapted from generic vision representation learning to an efficient end-to-end detector, while also providing the conceptual basis for subsequent models such as DEIMv2.

2.5 Related work

Before selecting the model to be deployed in the proposed inspection pipeline, a comparative literature study was conducted to identify the SOTA architecture that best matched the requirements of this thesis. The analysis focused on three candidate directions: a state-of-the-art real-time detector based on the DETR family (DEIMv2), an attention-centric evolution of the YOLO framework (YOLOv12), and a recent Vision-Transformer-based segmentation model (EoMT). The comparison was guided not only by reported benchmark accuracy, but also by computational cost, parameter efficiency, scalability across hardware regimes, and overall suitability for real-time deployment in the industrial scenarios considered in this work.

2.5.1 DEIMv2

DEIMv2 was considered the most direct candidate for this thesis because it explicitly targets the same design space of interest: high-accuracy *real-time* object detection under practical computational constraints [1]. Architecturally, DEIMv2 extends the RT-DETR family by combining an end-to-end transformer-based detector with **stronger backbone features**. It offers many variants that span a wide range of model sizes and computational budgets, making it a versatile choice for comparative evaluation. Specifically:

- For the mainstream variants (S, M, L, X), it adopts **DINOv3-pretrained** or distilled Vision Transformer backbones and introduces a ***Spatial Tuning Adapter* (STA)** to transform the single-scale ViT output into multi-scale features suitable for detection.
- For the ultra-lightweight variants (atto, femto, pico, nano), it instead relies on pruned **HGNetv2** backbones, thereby covering GPU, edge, and mobile deployment scenarios within the same design family [1].

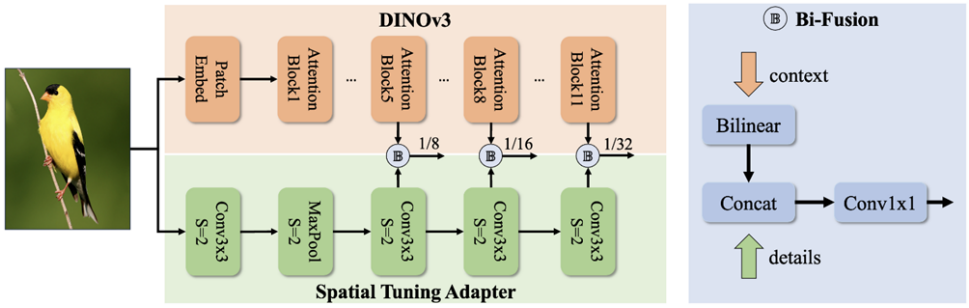


Figure 2.19: DEIMv2 architecture overview, illustrating the use of DINOv3 backbones and the Spatial Tuning Adapter (STA) to produce multi-scale features for detection [1].

From the perspective of the present application, the main strength of DEIMv2 lies in its ability to combine rich semantic representations with fine-grained local details by integrating DINOv3-based features into an RT-DETR-derived detection framework. This aspect is particularly attractive for the inspection tasks considered in the thesis, since the relevant visual cues may include both global shape information and localized appearance variations. Moreover, DEIMv2 preserves the end-to-end advantages inherited from RT-DETR, including NMS-free prediction, while improving the performance–cost trade-off through stronger features, a simplified decoder, and improved training strategies such as Copy-Blend augmentation [1]. The model is also highly scalable: the paper reports strong performance not only for the largest variants, but also for compact versions designed for resource-constrained deployment.

Variant	Backbone		Hidden Dimension			Layers		#Scales	#Query	#Params	GFLOPs	Latency	AP
	Model	Adapter	$d_{Back.}$	$d_{Enc.}$	$d_{Dec.}$	$\#_{Back.}$	$\#_{Dec.}$						
X	ViT-S+	STA	384	256	256	12	6	3	300	50.26	151.6	13.75	57.8
L	ViT-S	STA	384	256	256	12	4	3	300	32.18	96.32	10.47	56.0
M	ViT-T+	STA	256	256	256	12	4	3	300	18.11	52.20	8.80	53.0
S	ViT-T	STA	192	192	192	12	4	3	300	9.71	25.62	5.78	50.9
Nano	HGv2-B0	-	1024	128	128	5	3	2	300	3.57	6.86	2.32	43.0
Pico	HGv2-P	-	512	112	112	4	3	2	200	1.51	5.15	2.14	38.5
Femto	HGv2-F	-	256	96	96	3	3	2	150	0.96	1.67	1.91	31.0
Atto	HGv2-A	-	256	64	64	3	3	2	100	0.49	0.76	1.61	23.8

Figure 2.20: Summary of DEIMv2 results across model scales, highlighting the balance between detection accuracy, latency, and parameter efficiency [1].

The main limitation of DEIMv2 is that part of its performance advantage depends on the availability of powerful pretrained backbones, especially for the larger variants. In other words, the strongest configurations benefit from large-scale representation learning prior to the downstream detection task. Even so, the model remains particularly compelling within the present comparative study because it combines strong reported accuracy, broad scalability, and clear compatibility with real-time detection requirements [1, 6].

2.5.2 YOLOv12

YOLOv12 was selected as the main alternative detector because it represents a different and highly competitive design philosophy for real-time object detection [2]. Instead of building on the DETR lineage, YOLOv12 revisits the YOLO framework by introducing an ***attention-centric*** architecture. Its core idea is to integrate the modeling strength of attention mechanisms into

a real-time detector without sacrificing the low-latency behavior that has traditionally made YOLO models attractive in industrial settings. To do so, the paper [2] introduces three key elements:

- **Area Attention**: A novel attention mechanism that reduces the computational cost of attention by grouping spatial locations into areas, allowing the model to capture long-range dependencies with fewer computations (Figure 2.21).

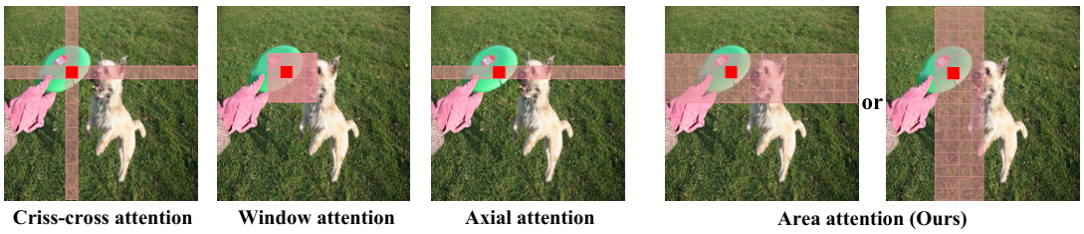


Figure 2.21: Area attention mechanism in YOLOv12 compared to traditional attention mechanisms.

- **R-ELAN**: A modified version of the ELAN architecture that improves optimization stability and convergence in deeper attention-based models, enabling YOLOv12 to scale effectively (Figure 2.22).

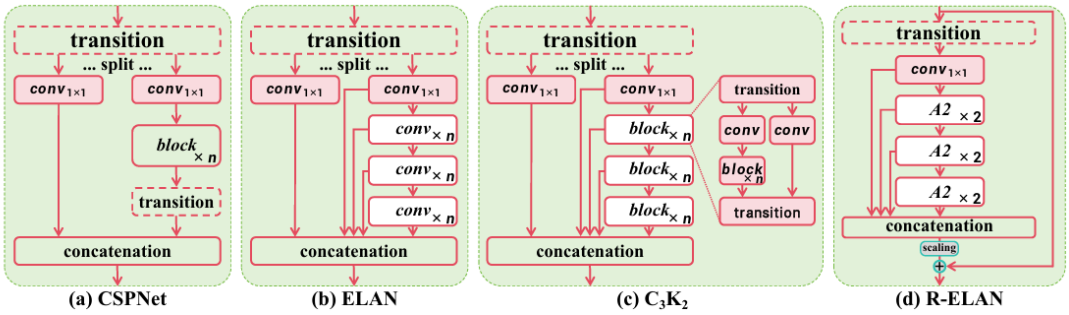


Figure 2.22: YOLOv12 R-ELAN architecture overview, illustrating the skip connection mechanism to improve optimization stability in deeper attention-based models [2].

- **FlashAttention**: An efficient implementation of attention that reduces memory-access overhead during inference, further contributing to YOLOv12's low latency.

The main appeal of YOLOv12 in the context of this thesis is its excellent latency–accuracy trade-off and its ability to achieve strong results without depending on additional large-scale pretraining strategies. This makes it an attractive candidate whenever extremely low inference time is the primary design criterion. In addition, YOLOv12 is conceptually important because it shows that attention mechanisms can be integrated effectively even within the traditionally convolution-centric YOLO family, thereby challenging the assumption that attention is necessarily too slow for real-time detection [2].

At the same time, YOLOv12 presents some practical limitations for the specific selection process carried out in this thesis. Its efficiency gains partly depend on hardware support for FlashAttention, which is not equally available across all deployment environments. Furthermore, according to the comparative evidence discussed in the DEIMv2 paper, DEIMv2 later

surpassed YOLOv12 in the performance–cost trade-off, achieving higher detection accuracy with fewer parameters and lower computational cost in corresponding regimes [1, 2], as showed in Figure 2.28. For this reason, YOLOv12 was regarded as a strong benchmark and a valuable reference point, but not as the final choice for deployment.

	$AP^{val}_{50:95}$ (%)	AP^{val}_{50}(%)	AP^{val}_{75} (%)	AP^{val}_{small} (%)	AP^{val}_{medium} (%)	AP^{val}_{large} (%)
YOLOv12-N	40.6	56.7	43.8	20.2	45.2	58.4
YOLOv12-S	48.0	65.0	51.8	29.8	53.2	65.6
YOLOv12-M	52.5	69.6	57.1	35.7	58.2	68.8
YOLOv12-L	53.7	70.7	58.5	36.9	59.5	69.9
YOLOv12-X	55.2	72.0	60.2	39.6	60.7	70.9

Figure 2.23: Summary of YOLOv12 results on the COCO benchmark, highlighting its strong balance between detection accuracy and inference efficiency across model scales [2].

2.5.3 EoMT

The third candidate direction explored in the literature study was not a detector, but a recent Vision-Transformer-based segmentation model, namely the Encoder-only Mask Transformer (EoMT), introduced in *Your ViT is Secretly an Image Segmentation Model* [3]. This work is particularly relevant because it addresses a dense prediction problem that is closely related to object localization, while doing so with a design philosophy aligned with the transformer-based trend discussed in the previous sections. The central idea of EoMT is that, given sufficiently large ViTs and sufficiently strong pretraining, many task-specific segmentation components become less necessary. For this reason, the model removes the convolutional adapter, the pixel decoder, and the separate transformer decoder typically used in ViT-based segmentation pipelines, and replaces them with a simpler encoder-only design augmented by learnable queries and a lightweight mask prediction module [3].

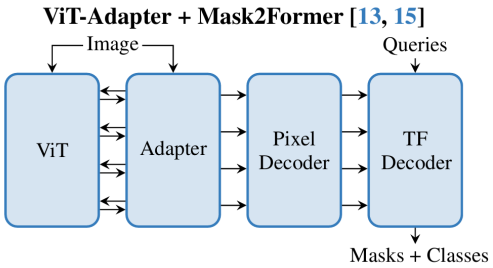


Figure 2.24: Typical ViT-based segmentation pipeline

This approach offers several attractive properties. First, it substantially simplifies the segmentation architecture while retaining competitive accuracy. Second, by progressively removing masked attention during training through a *mask annealing* strategy, EoMT can enable faster inference than more complex ViT-based segmentation systems. Third, because it stays close to the plain ViT architecture, it can directly benefit from future improvements in large-scale pretraining and transformer optimization [3]. From a research standpoint, EoMT was therefore an appealing candidate because it suggested a potentially elegant route toward precise object delineation.

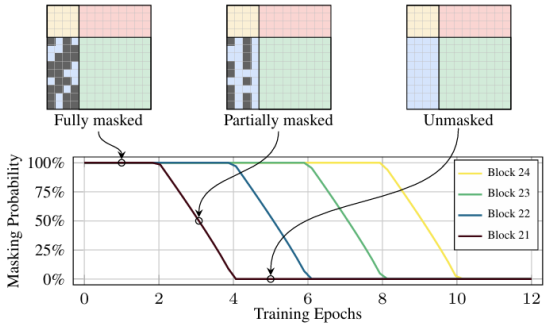


Figure 2.25: EoMT mask annealing strategy, which progressively removes masked attention during training.

However, the comparative study also highlighted an important mismatch with the main requirements of this thesis. The proposed pipeline ultimately requires robust and efficient *object localization* on embedded or edge-oriented hardware, including resource-constrained deployment targets such as Raspberry Pi and Jetson Nano platforms. By contrast, EoMT is designed for segmentation benchmarks and derives much of its strength from large pretrained ViTs and high-throughput inference settings [3]. Although segmentation can provide richer pixel-level information, that additional output is not strictly necessary for the core deployment objective considered here, namely fast localization of dough balls and bread loaves prior to subsequent processing stages. Moreover, EoMT is more naturally suited to scenarios in which large ViTs and higher-performance hardware are available, whereas this thesis requires an efficient real-time detector that can be deployed in practical industrial edge environments with limited computational resources. Consequently, EoMT was considered highly interesting from a methodological perspective, but less aligned than DEIMv2 with the operational constraints, deployment targets, and output requirements of the thesis.

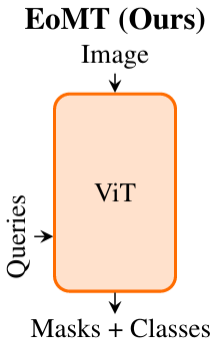


Figure 2.26: Progressive removal of task-specific segmentation components in EoMT [3].

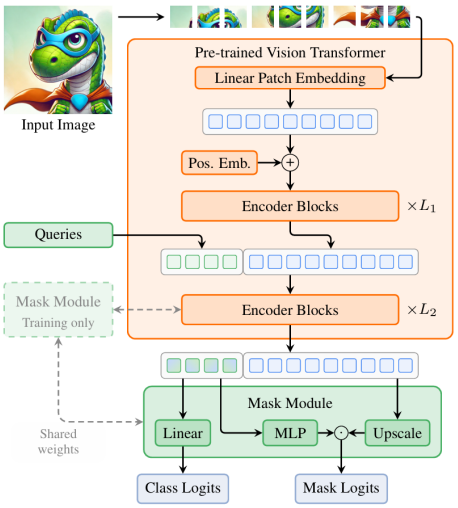
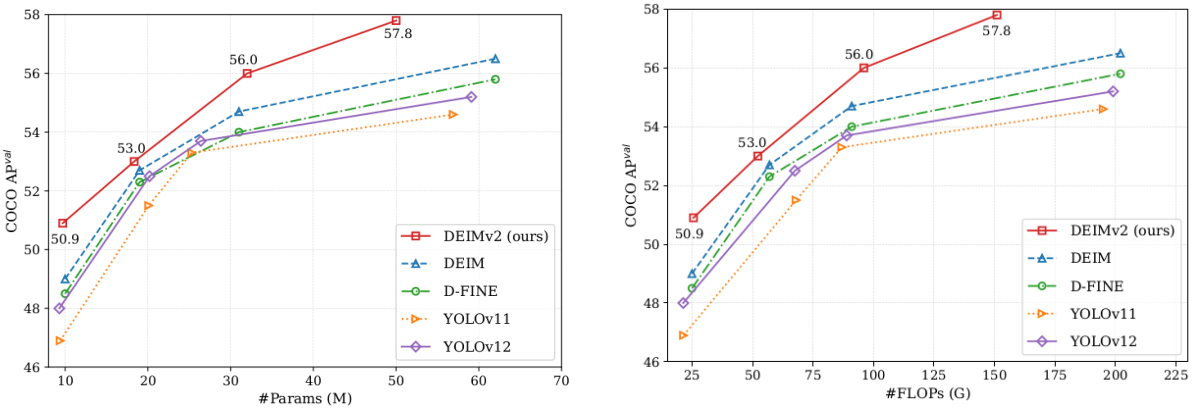


Figure 2.27: EoMT architecture, which removes the convolutional adapter, pixel decoder, and separate transformer decoder, and replaces them with a simpler encoder-only design augmented by learnable queries and a lightweight mask prediction module [3]

2.5.4 Final considerations

The comparative analysis of the three candidate families revealed a common trend: modern vision systems are increasingly moving beyond purely convolutional designs and are exploiting transformer-based representations in progressively more effective ways. Nevertheless, the final choice could not be based on architectural novelty alone. It had to reflect the concrete needs of the thesis, namely accurate and real-time localization of products in industrial environments, compatibility with edge-oriented deployment, and a favorable balance between predictive performance and computational cost.

Within this framework, DEIMv2 emerged as the most suitable solution. Compared with YOLOv12, it offered a more favorable overall performance–cost trade-off according to the evidence reported in the literature, combining higher detection accuracy with competitive or lower parameter and FLOP budgets in the corresponding model regimes, as illustrated in Figure 2.28 [1, 2]. In addition, its integration of DINOv3-based representations and multi-scale adaptation made it particularly appealing for scenarios in which both local visual details and more global semantic structure may contribute to robust object detection. Furthermore, DEIMv2's variety of model sizes and its compatibility with real-time detection requirements made it a versatile choice for the different deployment scenarios considered in the thesis, ranging from GPU-based inference to edge-oriented applications [1, 6]. By contrast, part of YOLOv12's efficiency advantage depends on hardware support for FlashAttention, which is not uniformly available across all deployment environments. In this respect, DEIMv2 appears more flexible and robust as a practical choice for heterogeneous hardware settings.



(a) Accuracy versus number of parameters for DEIMv2 and YOLOv12 across model scales [1, 2].

(b) Accuracy versus computational cost for DEIMv2 and YOLOv12 across model scales [1, 2].

Figure 2.28: Comparison between DEIMv2 and YOLOv12 in terms of accuracy, parameter count, and computational cost, highlighting the stronger overall efficiency of DEIMv2 across the considered model scales [1, 2].

The comparison with EoMT requires a different interpretation, because EoMT addresses image segmentation rather than object detection. For this reason, the decisive argument was not that DEIMv2 is universally superior on a common benchmark, but that it is better aligned with the objective of the thesis. In the proposed pipeline, the essential requirement is to localize each dough ball or loaf quickly and reliably, so that subsequent modules can estimate geometric or appearance-based quantities. For this thesis, object detection is therefore the more appropriate

output representation: a full segmentation model would provide richer pixel-level information, but at the cost of denser prediction, greater architectural complexity, and weaker compatibility with low-latency embedded deployment. Moreover, the advantages of EoMT are most evident when large pretrained ViTs and high-performance hardware are available, whereas the thesis prioritizes an efficient real-time detector that can be deployed in practical industrial settings [3].

For these reasons, DEIMv2 was selected as the detection model adopted in this thesis. It combines state-of-the-art real-time detection performance, strong scalability across model sizes, and a transformer-based design that remains compatible with the broader evolution of modern vision architectures. At the same time, the study of YOLOv12 and EoMT remained valuable: YOLOv12 highlighted the growing viability of attention-centric real-time detectors, while EoMT showed that large pretrained ViTs can increasingly absorb task-specific dense prediction components. Together, these works provided the conceptual background against which DEIMv2 was identified as the most appropriate choice for this thesis.

2.6 Anomaly Detection

Anomaly detection is the task of identifying samples that deviate significantly from the expected distribution of normal data. In computer vision, this usually means detecting images or regions whose appearance is inconsistent with the visual patterns observed during training. In industrial inspection, anomaly detection is particularly relevant because defective samples are often rare, heterogeneous, and difficult to collect in a representative way.[4]

Depending on the formulation, visual anomaly detection may operate at different levels of granularity. Some methods produce an **image-level anomaly score**, indicating whether an image is normal or abnormal as a whole, while others also provide a **pixel-level anomaly map** that localizes suspicious regions. This distinction is important in industrial settings, where both the decision itself and the interpretability of the detected defect can be relevant.[4]

A common scenario in industrial practice is that abundant normal samples are available, whereas anomalous examples are scarce or completely unavailable. For this reason, many modern anomaly detection methods are designed to learn only the distribution of normal data and to treat deviations from that distribution as potential anomalies.[4] This setting is especially attractive when defects are difficult to enumerate in advance or when collecting defective products would be impractical.

From a methodological perspective, deep-learning-based anomaly detection methods can be grouped into several broad families, including reconstruction-based approaches, feature-embedding methods, teacher–student models, memory-bank methods, and hybrid approaches.[4] Although these methods differ in architecture and supervision signals, they share the same central objective: building a representation of normality that makes abnormal patterns stand out at inference time.

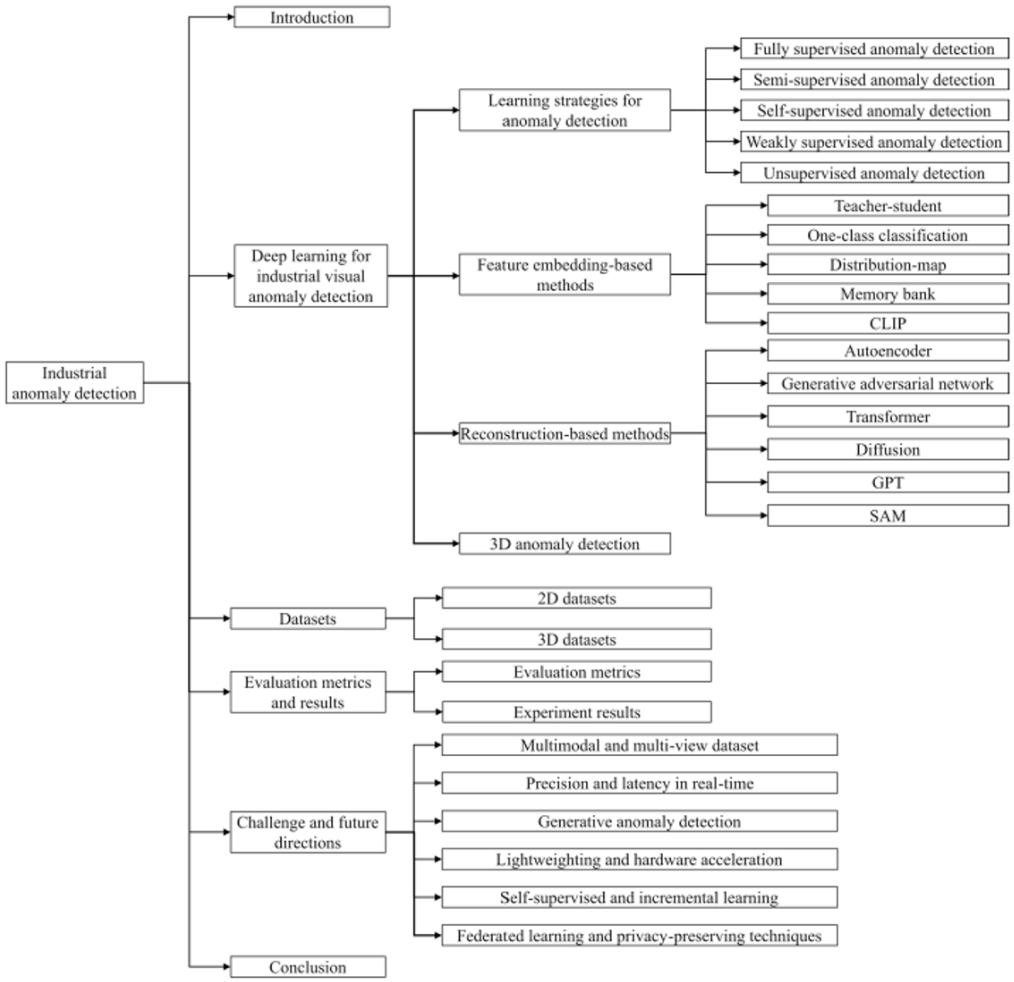


Figure 2.29: Overview of the main methodological families in industrial visual anomaly detection.[4]

2.6.1 EfficientAD

Among the anomaly detection methods considered for this thesis, **EfficientAD** was particularly attractive because it was explicitly designed to achieve a favorable balance between detection accuracy and very low inference latency.[5] This property is especially important in industrial applications, where anomaly detection must often be integrated into real-time or near-real-time processing pipelines.

At a high level, EfficientAD follows a **teacher–student paradigm**. A teacher network extracts reference features from normal images, while a student network is trained to reproduce those features on anomaly-free data.[5] During inference, a large discrepancy between teacher and student representations indicates that the input may contain an anomalous pattern. In addition to this feature-prediction mechanism, EfficientAD incorporates an autoencoder branch to better capture more global or logical anomalies that cannot be explained only through local feature mismatches.[5]

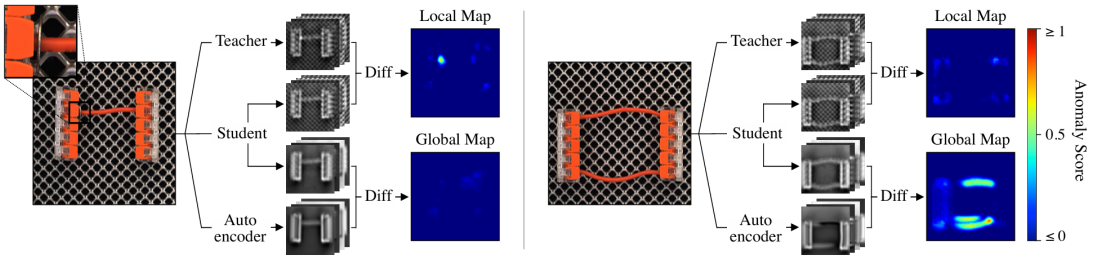


Figure 2.30: EfficientAD architecture combining teacher–student feature comparison and an autoencoder branch to produce anomaly maps and an image-level anomaly score.[5]

The main reason for selecting EfficientAD over other anomaly detection methods was therefore practical as well as methodological. First, it is designed for **high-speed inference**, which makes it suitable for deployment scenarios in which computational resources and response time are both important constraints.[5] Second, it is available within the **anomalib** framework, which provides a unified and relatively straightforward implementation environment for state-of-the-art anomaly detection models.[12] In this sense, EfficientAD offered a combination of efficiency, strong reported performance, and ease of integration that made it particularly convenient for the present work.

Another important aspect is that EfficientAD is well suited to settings in which only normal training samples are available.[5, 4] This makes it particularly relevant for industrial inspection problems in which anomalous data are scarce, poorly characterized, or unavailable during model development. For the purposes of this thesis, this characteristic was especially valuable, because the anomaly detection task had to be framed around learning a reliable representation of normal bread appearance and then measuring deviations from it at inference time.

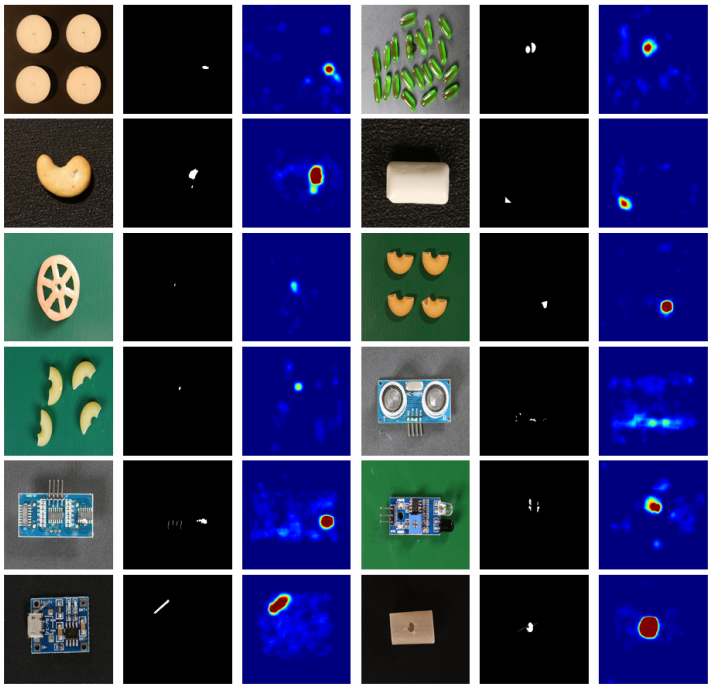


Figure 2.31: Examples of EfficientAD anomaly maps. The anomalies are highlighted in red, while the normal regions are shown in blue.

2.7 Training Deep Learning Models

Training a deep learning model consists of iteratively updating its parameters in order to minimize a predefined loss function on the available data.[30] In supervised and weakly supervised settings, this optimization process is not evaluated solely on the data used for parameter updates, but also on held-out data that provide an estimate of how well the model generalizes to unseen samples.[30]

For this reason, both the organization of the dataset and the interpretation of training dynamics are central to the development of reliable models. In particular, a clear separation between training, validation, and test data is essential for model selection, while the notions of overfitting and underfitting are fundamental for understanding whether a model is learning meaningful patterns or failing to generalize beyond the training set.[30]

2.7.1 Dataset Partitioning

A standard practice in machine learning is to divide the available data into at least three subsets: a training set, a validation set, and a test set.[30] The training set is used to update the model parameters during optimization. The validation set is instead used to monitor generalization during development and to support decisions such as hyperparameter tuning, model selection, and stopping time. Finally, the test set is reserved for the final evaluation of the model and should not influence training decisions.[30]

The rationale behind this separation is that good performance on the training set alone is not sufficient: the model must also perform well on unseen data. The validation set therefore provides an intermediate estimate of generalization quality while the model is being developed, whereas the test set is used only after the design choices have been fixed.[30] In practical applications, the exact proportions assigned to these subsets depend on the size and structure of the dataset, and there is no universally optimal split.[38] However, the underlying principle remains the same: parameter learning should be performed on one subset, model selection on another, and final assessment on a third independent subset. This approach helps to prevent overfitting to the training data and ensures that the reported performance metrics reflect the model's ability to generalize to new, unseen examples.[30]

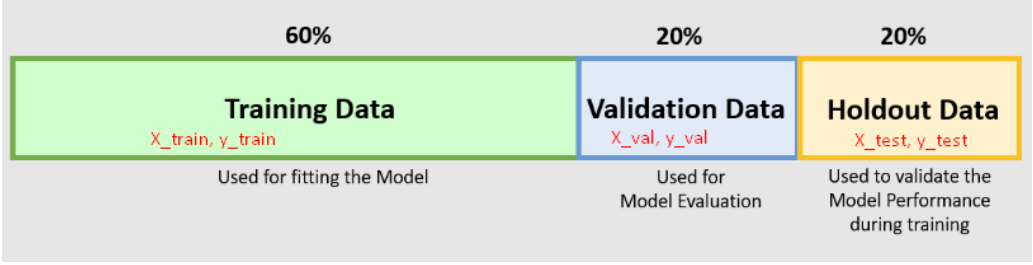


Figure 2.32: Illustration of the standard partition of a dataset into training, validation, and test subsets.

2.7.2 Overfitting, Underfitting, and Early Stopping

Two of the central concepts in machine learning are **underfitting** and **overfitting**.[30] Underfitting occurs when a model is unable to achieve sufficiently low error even on the training set. This usually indicates that the model is too simple, insufficiently trained, or poorly configured for the complexity of the task. Overfitting, by contrast, occurs when the model performs very well on the training data but poorly on unseen data, meaning that it has adapted too strongly to idiosyncrasies of the training set rather than learning patterns that generalize.[30]

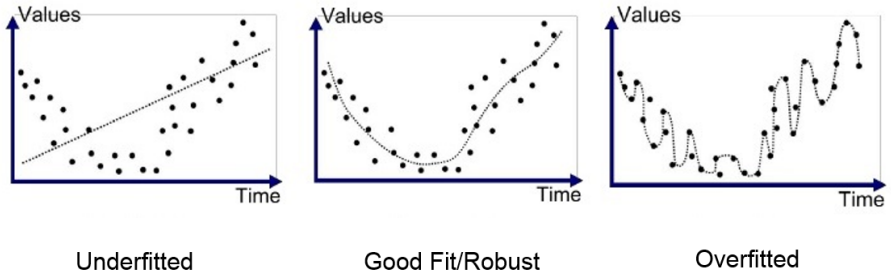


Figure 2.33: Example of underfitting and overfitting in a regression task. The left plot shows an underfitting model that fails to capture the underlying trend, the middle plot shows a well-fitted model that generalizes well, and the right plot shows an overfitting model that captures noise in the training data.

These two phenomena can often be identified through the evolution of training and validation curves. In an underfitting regime, both training and validation errors remain relatively high. In an overfitting regime, the training error keeps decreasing whereas the validation error stagnates or starts increasing. Monitoring these trends is therefore essential for understanding whether additional training is beneficial or detrimental.[30, 39]

One of the most widely used strategies for limiting overfitting is **early stopping**. During optimization, the training loss may continue to decrease even after the model has already reached its best generalization performance. When the validation error stops improving and begins to worsen, this is a signal that the model is starting to adapt too closely to the training data. Early stopping addresses this issue by interrupting training at the point of best validation performance and retaining the corresponding model parameters.[30, 39]

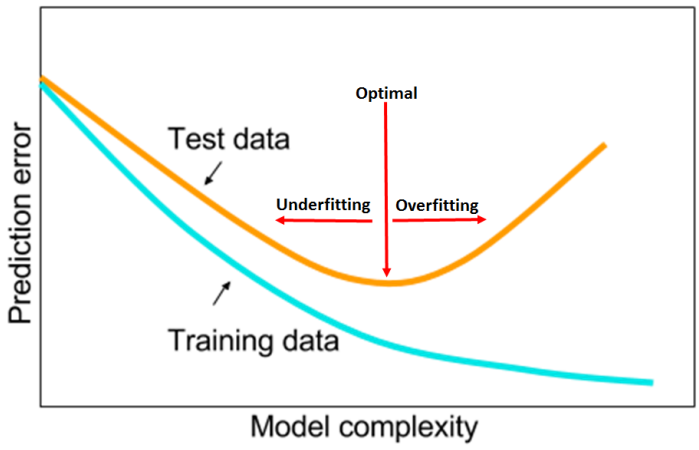


Figure 2.34: Typical training and validation behaviors in underfitting and overfitting regimes.

Several strategies can be adopted to mitigate these problems. Underfitting may be addressed by increasing model capacity, improving feature quality, extending training, or refining optimization settings. Overfitting can instead be reduced through regularization techniques, data augmentation, early stopping, or, when appropriate, by simplifying the model.[30] In practice, achieving good generalization requires balancing model capacity and training duration so that the model captures meaningful structure in the data without memorizing the training set.

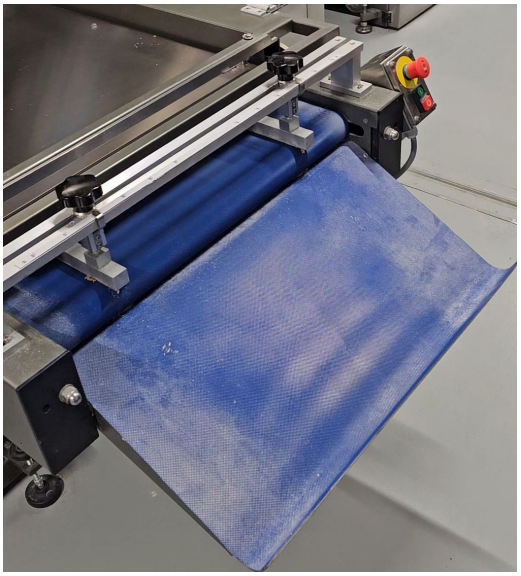
Chapter 3

Data acquisition system

3.1 Site 1: dough ball production and data acquisition setup

Site 1 represents the first deployment environment of the proposed vision-based monitoring system. It corresponds to the production stage in which the dough exits the industrial forming machine in the form of individual cylindrical dough balls. The objective at this stage is twofold: (i) to acquire a representative RGB-D dataset for offline training of the object detection network, and (ii) to deploy the trained DEIMv2 model online in order to detect the bounding boxes corresponding to the dough balls and support volumetric estimation.

(a) Automated machine pipeline at Site 1



(b) Output table where the dough balls arrive

Figure 3.1: Overview of Site 1 and close-up of the machine output area

Industrial Context and Process Description

At Site 1, the dough mixture is processed by an automated forming machine that portions and shapes the material into approximately cylindrical dough balls. These dough balls exit the machine sequentially and move along a conveyor belt until they reach the output table shown in Figure 3.1b. Variability in size and shape may occur due to fluctuations in the upstream mixing and forming processes. For quality-control purposes, it is therefore important to monitor geometric properties such as volume in a non-contact manner.

To address this requirement, a vision-based acquisition setup was installed above the machine output area, as illustrated in Figure 3.1 and Figure 3.2. The hardware configuration consists of an *NVIDIA Jetson Nano 2GB Developer Kit* [7] for edge computation and an *Azure Microsoft Kinect* RGB-D sensor [8] for synchronized color and depth acquisition. Further details on the sensing hardware and on the Time-of-Flight principle are provided in the dedicated subsections of this chapter.

(a) Jetson Nano and Azure Kinect used in the Site 1 setup

(b) Azure Kinect mounted above the machine output area

Figure 3.2: Hardware configuration at Site 1, showing the Jetson Nano and the Azure Kinect mounted above the machine output area for RGB-D acquisition of the dough balls shown in Figure 3.1b

Offline Data Acquisition

During the offline phase, the system operates in data acquisition mode. In this configuration, the Azure Kinect continuously captures synchronized RGB and depth frames. A motion detection algorithm is executed on the Jetson Nano to identify pixel variations in the scene corresponding to the passage of dough balls under the camera.

The motion detection module serves exclusively as a trigger mechanism for dataset collection. When motion is detected within a predefined and configurable region of interest (ROI),

the corresponding RGB-D frame is stored. In this setup, the ROI was defined so as to cover the entire machine output area (shown in Figure 3.1b).

This approach ensures that:

- only relevant frames containing dough balls are acquired;
- redundant background movement is discarded;
- the dataset remains balanced and computationally manageable.

Figure 3.3: Example of a configurable region of interest (ROI) superimposed on a background-subtraction mask. Although this example refers to the Site 2 workstation shown in Figure 3.13, it illustrates the same motion-triggering principle adopted for data acquisition at both sites. The white region corresponds to the detected foreground, while the black region represents the background.

The collected dataset is subsequently annotated through an automatic labelling strategy based on Language Segment-Anything [11], and the resulting annotations are then used to train the DEIMv2 object detection network in an offline environment. The details of the labelling procedure and training setup are presented in Chapter 4.

Figure 3.4: Example of the motion detection module at Site 1, showing a dough ball passing through the monitored area and activating the trigger used to store the corresponding RGB-D frame for dataset creation.

Online Inference and Volume-Oriented Processing

After training, the system is deployed online at Site 1 using the same hardware configuration, namely the Jetson Nano and the Azure Kinect. In the online phase, the trained DEIMv2 network

detects the dough balls directly in the RGB stream, while the aligned depth data provide the spatial information required for volumetric analysis.

The detected regions are therefore converted into local 3D representations and further processed to estimate dough volume. The detailed processing pipeline, including point-cloud generation, background removal, 3D completion, and final volume estimation, is described in Chapter 4.

System Objectives at Site 1

The implementation at Site 1 serves as a proof-of-concept for an edge-based RGB-D inspection system capable of:

- automated dataset acquisition in industrial conditions;
- real-time object detection on embedded hardware;
- 3D reconstruction from RGB-D data;
- Non-contact volumetric estimation of deformable food products.

The modular design adopted at Site 1 ensures that the same sensing and computational infrastructure can be reused across both offline training and online inference phases. More generally, the same acquisition philosophy based on edge computing, motion-triggered capture, and downstream visual analysis is also maintained at Site 2, thereby minimizing hardware complexity and deployment costs across the overall framework.

3.1.1 Nvidia Jetson Nano DK

To perform the computations at Site 1, namely the acquisition of the dough dataset and the collection of volume measurements, the **NVIDIA Jetson Nano Developer Kit 2GB** [7] was adopted.

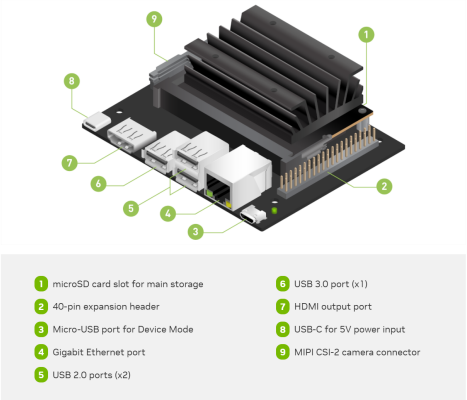


Figure 3.5: NVIDIA Jetson Nano 2GB Developer Kit connectivity.

This hardware platform is an embedded computing board designed for edge artificial intelligence applications. The device is developed by NVIDIA and belongs to the Jetson family of embedded systems specifically tailored for GPU-accelerated workloads.

The Jetson Nano 2GB is based on the NVIDIA Tegra X1 System-on-Chip (SoC), which integrates a quad-core ARM Cortex-A57 64-bit CPU and a 128-core Maxwell GPU capable of delivering up to 472 GFLOPS of FP16 compute performance. The board is equipped with 2GB of 64-bit LPDDR4 memory, which is **shared between CPU and GPU**, enabling efficient execution of deep learning inference workloads under constrained memory conditions [40].

From a hardware perspective, the development kit provides several connectivity and expansion interfaces, as shown in Figure 3.5.

The board operates within a typical power envelope of 5W to 10W, making it suitable for energy-constrained embedded and edge-computing scenarios.
The platform supports the NVIDIA JetPack SDK, which includes optimized libraries such as CUDA, cuDNN, and TensorRT for hardware-accelerated deep learning inference [7].
The presence of a dedicated GPU enables on-device execution of neural networks for tasks such as object detection and image segmentation, reducing latency compared to cloud-based solutions and enabling near real-time inference in edge environments.

Despite its limited memory capacity (2GB LPDDR4) and constrained power budget, the Maxwell GPU architecture allows the Jetson Nano to achieve competitive inference performance for several state-of-the-art deep learning models. To quantitatively assess its computational capabilities, representative inference benchmarks are reported in Table 3.1 [40]. The values are expressed in frames per second (FPS) under optimized inference configurations.

Model	**Performance (FPS)**
Inception-v4	10.51
VGG-19	9.95
Super-resolution (BSD500)	15.12
U-Net (segmentation)	16.57
Pose estimation	14.47
YOLOv3-Tiny (416×416)	47.11
ResNet-50 (224×224)	36.42
SSD MobileNet-v1	42.00

Table 3.1: Representative inference performance of NVIDIA Jetson Nano 2GB.

As shown in Table 3.1 [40], lightweight architectures such as YOLOv3-Tiny and SSD MobileNet-v1 achieve real-time performance, while more computationally demanding networks (e.g., VGG-19 and Inception-v4) operate at lower frame rates. These results highlight the importance of selecting architectures that balance accuracy and computational complexity when deploying deep learning models on resource-constrained embedded platforms.
It should be noted that the reported performance values may vary depending on the software configuration, precision mode (e.g., FP16 or INT8), and system workload.

Considering the trade-off between computational performance, power consumption, cost, and ecosystem support, the Jetson Nano 2GB represents a suitable platform for prototyping and deploying computer vision algorithms in embedded systems, as in the present application.

3.1.2 Azure Microsoft Kinect DK

For the acquisition of RGB-D data at Site 1, the **Microsoft Azure Kinect DK** [8] was employed. This device is a spatial sensing platform developed by Microsoft, designed for advanced computer vision, depth sensing, and AI-based perception applications.

Figure 3.6: Microsoft Azure Kinect DK

The Azure Kinect DK integrates a high-resolution RGB camera and a depth sensor based on Time-of-Flight (ToF) technology. The RGB camera provides up to 12 MP resolution (4096×3072), while the depth sensor delivers depth maps with a maximum resolution of 1024×1024 pixels.

The depth sensing module operates using modulated infrared light to estimate per-pixel distance information, enabling accurate 3D scene reconstruction [41].

The device supports multiple depth operating modes, including Narrow Field of View (NFOV) and Wide Field of View (WFOV), each configurable with binned and unbinned resolutions, as illustrated in Figure 3.7 and in Table 3.3. This flexibility allows adaptation to different working distances and spatial accuracy requirements. The depth measurement range typically varies between 0.25 m and 5.5 m, depending on the selected configuration and environmental conditions [41].

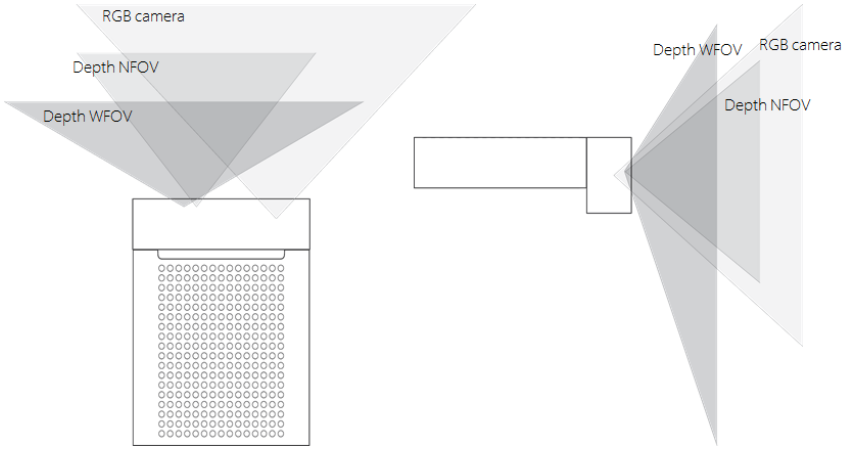


Figure 3.7: Available depth sensor fields of view (NFOV and WFOV).

Depending on the selected RGB resolution and depth operating mode, the camera parameters, field of view, frame rate, and operational range vary significantly.
The main RGB camera configurations are reported in Table 3.2, while the available depth modes are summarized in Table 3.3.

RGB Resolution (HxV)	Aspect Ratio	Format	FPS	Nominal FoV (HxV)
3840×2160	16:9	MJPEG	0, 5, 15, 30	90°×59°
2560×1440	16:9	MJPEG	0, 5, 15, 30	90°×59°
1920×1080	16:9	MJPEG	0, 5, 15, 30	90°×59°
1280×720	16:9	MJPEG / YUY2 / NV12	0, 5, 15, 30	90°×59°
4096×3072	4:3	MJPEG	0, 5, 15	90°×74.3°
2048×1536	4:3	MJPEG	0, 5, 15, 30	90°×74.3°

Table 3.2: RGB camera operating modes and specifications.

Mode	Resolution	FoV	FPS	Operating Range	Exposure Time
NFOV unbinned	640×576	75×65°	0, 5, 15, 30	0.5 – 3.86 m	12.8 ms
NFOV 2×2 binned	320×288	75×65°	0, 5, 15, 30	0.5 – 5.46 m	12.8 ms
WFOV 2×2 binned	512×512	120×120°	0, 5, 15, 30	0.25 – 2.88 m	12.8 ms
WFOV unbinned	1024×1024	120×120°	0, 5, 15	0.25 – 2.21 m	20.3 ms
Passive IR	1024×1024	N/A	0, 5, 15, 30	N/A	1.6 ms

Table 3.3: Depth camera operating modes and specifications.

From a hardware standpoint, the Azure Kinect DK includes:

- one 12 MP RGB camera,
- one Time-of-Flight depth sensor,
- one 6-axis inertial measurement unit (IMU),
- one USB-C interface for power and data transmission,
- one external synchronization port for multi-device configurations.

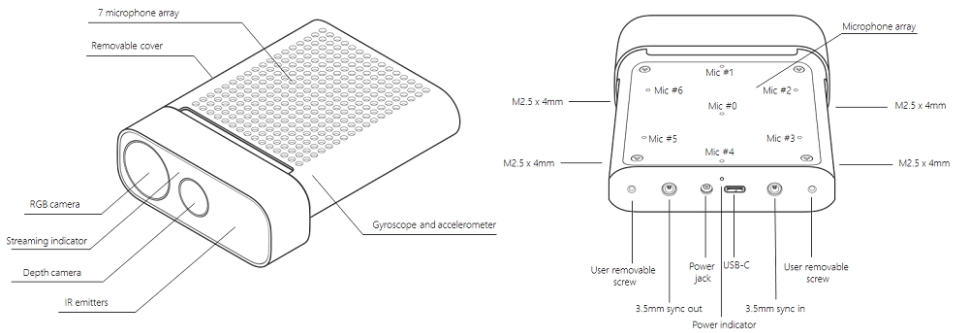


Figure 3.8: Microsoft Azure Kinect DK hardware and connectivity.

The availability of synchronized RGB and depth streams enables the generation of geometrically aligned RGB-D frames, which are particularly suitable for 3D reconstruction, volumetric

estimation, and point cloud generation. In the present application, depth information is exploited to estimate the volume of dough samples by reconstructing their three-dimensional point-cloud.

Time-of-Flight Depth Sensing Principle

The depth sensing module of the Microsoft Azure Kinect DK is based on the Time-of-Flight (ToF) principle, a technique used to estimate the distance between the sensor and objects in the scene by measuring the time required for emitted light to travel to the target and back to the sensor.

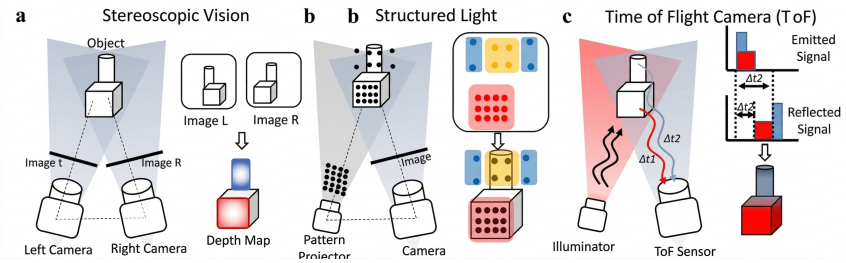


Figure 3.9: Different depth estimation approaches

In general, the depth Z of a point can be computed as:

$$Z = \frac{c \cdot \Delta t}{2} \tag{3.1}$$

where:

- c is the speed of light in air,
- Δt is the round-trip travel time of the emitted optical signal.
- The division by two accounts for the two-way propagation path (sensor-to-object and object-to-sensor).

Direct measurement of Δt requires extremely high temporal resolution, as light travels approximately 30 cm in 1 ns. For this reason, most modern ToF cameras, including the Azure Kinect, employ a *continuous-wave modulation* approach rather than direct pulsed time measurement.

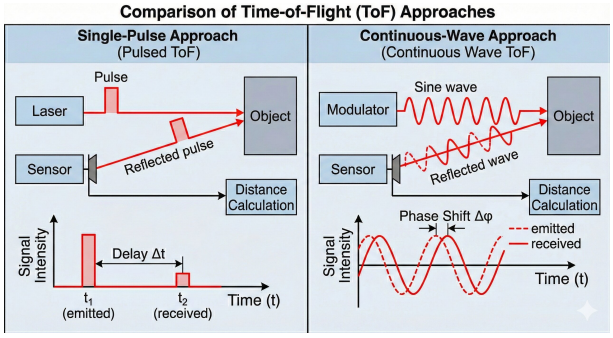


Figure 3.10: Different ToF approaches

In continuous-wave ToF systems, the emitted infrared light is amplitude-modulated at a known frequency. The reflected signal received by the sensor exhibits a phase shift ϕ proportional to the propagation delay. The depth is then computed as:

$$Z = \frac{c}{4\pi f}\phi \tag{3.2}$$

where:

- f is the modulation frequency,
- ϕ is the measured phase difference between emitted and received signals.

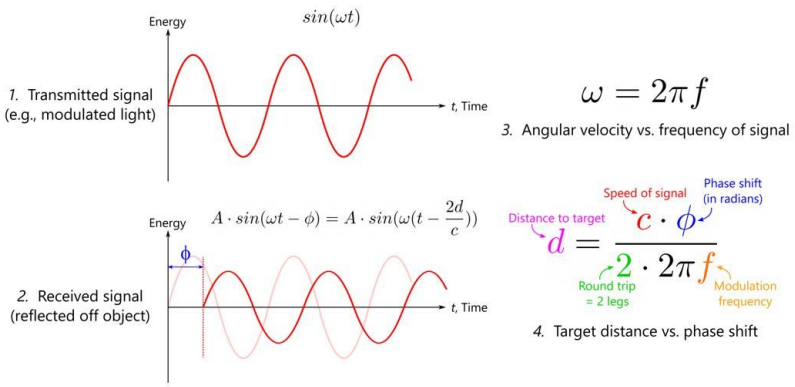


Figure 3.11: Time of flight equations derivation

By estimating the phase shift at each pixel, the sensor generates a dense depth map, enabling per-pixel 3D reconstruction of the observed scene.
The advantages of ToF-based depth sensing include:

- direct acquisition of metric depth measurements,
- dense depth maps without requiring stereo matching,
- robustness to moderate textureless surfaces.

However, ToF systems are subject to several sources of error, including:

- multipath interference (MPI), caused by indirect reflections,
- ambient infrared light interference,
- phase ambiguity at larger distances due to periodic modulation,
- reduced accuracy on highly reflective or absorptive materials.

Despite these limitations, Time-of-Flight technology provides accurate and dense depth measurements in indoor environments, making it particularly suitable for volumetric estimation and 3D reconstruction tasks, as required in the present application.

Azure Kinect Sensor SDK

The Azure Kinect DK is supported by the Azure Kinect Sensor SDK [42], which provides low-level APIs for device control, stream acquisition, intrinsic and extrinsic calibration handling, depth-to-color transformation, and point cloud extraction [41].

The SDK exposes calibrated camera parameters, including focal lengths, principal points, and distortion coefficients, enabling accurate geometric transformations between RGB and depth coordinate systems. Moreover, built-in functions allow real-time depth image rectification, coordinate mapping, and conversion of depth frames into 3D point clouds expressed in camera reference coordinates.

Such capabilities are particularly relevant in volumetric estimation tasks, where precise spatial alignment between depth measurements and RGB data is required to ensure geometrical consistency.

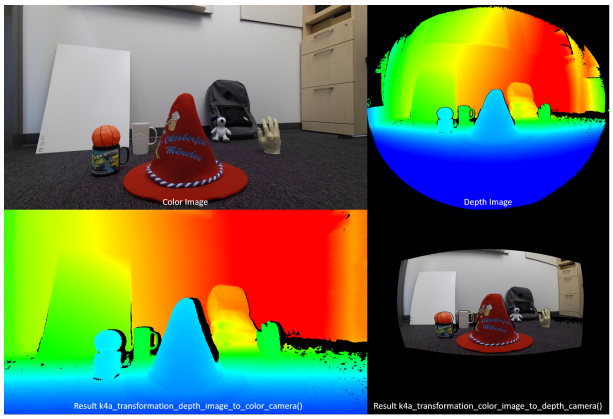


Figure 3.12: Example of SDK transformation from depth image to color camera FOV and viceversa

Python Wrapper: pyKinectAzure

For integration within the data processing pipeline, a Python interface was adopted through the *pyKinectAzure* wrapper [43]. This open-source library provides Python bindings to the native Azure Kinect Sensor SDK, allowing device initialization, stream acquisition, frame synchronization, and calibration access directly from Python environments.

The wrapper enables rapid prototyping and seamless integration with scientific computing libraries such as NumPy and OpenCV. Through pyKinectAzure, depth frames can be converted into NumPy arrays, facilitating further processing steps such as filtering, segmentation, and 3D point cloud generation within the same software ecosystem used for machine learning and computer vision tasks.

The use of a Python wrapper significantly simplifies the development workflow while maintaining access to the underlying calibrated sensor data and geometric transformation utilities provided by the official SDK.

Considering its depth accuracy, configurable field of view, synchronized RGB-D acquisition capabilities, and extensive software support, the Microsoft Azure Kinect DK represents a suit-

able sensing solution for industrial computer vision applications requiring reliable volumetric measurements.

3.2 Site 2: Baked Bread Dataset Acquisition and Anomaly Detection

Site 2 corresponds to the acquisition and inspection station dedicated to baked bread loaves. In contrast to Site 1, where RGB-D data are exploited to estimate volumetric properties of raw dough balls, Site 2 focuses on the visual analysis of the final baked product. The primary objective at this stage is the creation of a dedicated image dataset of baked loaves, followed by the training and deployment of the DEIMv2 object detection network for automated detection and anomaly-based quality assessment.

3.2.1 Industrial Context and Monitoring Objectives

At Site 2, the product has completed the baking process and is transported along the production line as finished bread loaves. At this stage, geometric variability is less critical than the overall visual appearance of the final product, including crust condition, browning patterns, and other surface irregularities. These characteristics may reflect deviations in baking quality and in the underlying process parameters.
The system deployed at Site 2 aims to:

- Acquire a representative dataset of baked bread images,
- Train a dedicated DEIMv2 object detection model,
- Detect bread loaves in real time,
- Support anomaly-based quality monitoring of the final product.

Figure 3.13: Point of view of the Raspberry Pi Camera Module 3 Wide during the manual extraction phase, showing operators removing baked bread loaves from the trays under real production conditions

3.2.2 Dataset Creation

During the initial phase, the system operates in data acquisition mode. Following the same motion-triggered acquisition principle adopted at Site 1, images of baked bread loaves are captured by using background subtraction as an event trigger. At this site, the assessment is not based on depth cues but on visual characteristics such as color, illumination, and surface texture. For this reason, images are acquired under controlled lighting conditions to reduce visual variability due to the environment (see *Practical challenges and system robustness* in Chapter 1) and to improve the consistency of both the collected dataset and the resulting analysis.

The collected dataset is subsequently annotated through an automatic labelling strategy based on Lang-SAM [11] and used to train the DEIMv2 [1] object detection network in an offline environment with higher computational resources. The details of the automatic labelling strategy and training procedure are described in Chapter 4.

This dataset creation phase is fundamental to ensure robustness of the detection model against variations in loaf shape, surface texture, illumination changes, and partial occlusions.

3.2.3 Hardware Configuration

Unlike Site 1, which relies on an RGB-D sensor and embedded GPU platform, Site 2 adopts a lightweight and cost-effective vision setup composed of:

- A Raspberry Pi 5 [44] for edge computation,
- A Raspberry Pi Camera Module 3 Wide [45] for RGB image acquisition.

The Raspberry Pi Camera Module 3 Wide provides a large field of view, enabling coverage of the table where the loaves are baked (as shown in Figure 3.13). Since volumetric estimation is not required at this stage, depth sensing is unnecessary, and a monocular RGB configuration is sufficient.

3.2.4 Online Detection and Downstream Analysis

Once training is completed, the DEIMv2 model is deployed on the Raspberry Pi 5 [44] for inference. For each acquired RGB frame, the bread loaves are detected and the corresponding image regions are extracted. These regions are then passed to a downstream anomaly-detection stage for product-quality assessment. The detailed pipeline and anomaly-detection procedure are presented in Chapter 4.

3.2.5 Role of Site 2 in the Overall Architecture

Site 2 complements the functionality of Site 1 by extending the inspection framework from geometric analysis of raw dough to anomaly-based visual assessment of the final baked product. While Site 1 leverages RGB-D sensing for volumetric estimation, Site 2 employs a monocular RGB setup for anomaly detection on baked bread loaves.

Together, the two sites demonstrate the flexibility of the proposed DEIM-based architecture, which can be adapted to different sensing modalities and quality metrics while maintaining a consistent edge-computing paradigm.

3.2.6 Raspberry Pi 5

For the computations at Site 2, namely RGB image acquisition, local control of the sensing pipeline, and execution of lightweight edge-processing tasks, the **Raspberry Pi 5** was adopted.[44] In contrast to Site 1, where an RGB-D sensor and an embedded GPU platform were required for volumetric estimation, Site 2 relies on a monocular RGB setup and therefore benefits from a more compact and cost-effective embedded computer.

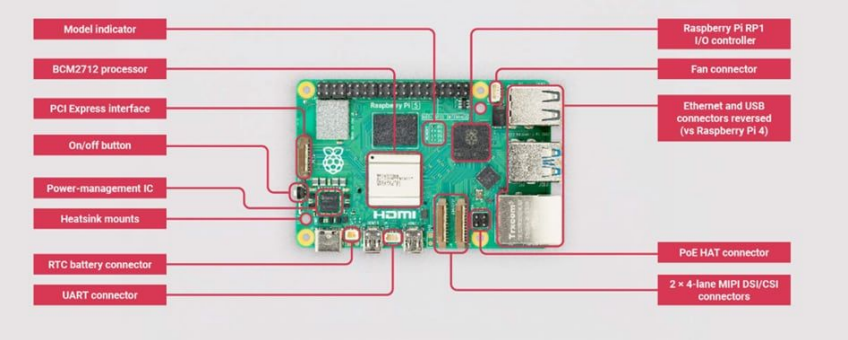


Figure 3.14: Raspberry Pi 5 board.

The Raspberry Pi 5 is a single-board computer developed for embedded and edge-computing applications.[44] It is based on the Broadcom BCM2712 system-on-chip, which integrates a **quad-core Arm Cortex-A76 CPU running at 2.4 GHz**, a **VideoCore VII GPU**, and **8 GB of LPDDR4X memory** in the configuration adopted for this project.[44]

From a hardware standpoint, the Raspberry Pi 5 provides a wide set of interfaces that are particularly relevant for computer-vision deployments, including:

- two USB 3.0 ports and two USB 2.0 ports,
- Gigabit Ethernet,
- dual-band Wi-Fi and Bluetooth connectivity,
- two four-lane MIPI transceivers for camera and display interfaces,
- a PCIe 2.0 interface for high-speed peripheral expansion,
- a USB-C power input.[44]

These characteristics make the platform flexible enough to support local image acquisition, communication with external devices, and integration with dedicated sensing peripherals such as the Raspberry Pi Camera Module 3 Wide discussed in the following subsection.[46]

An important practical aspect of the Raspberry Pi 5 is that its increased computing performance, compared with earlier Raspberry Pi boards, is accompanied by a higher thermal load

under sustained workloads. Since the device in the present application is expected to operate continuously for long periods, potentially day and night and without constant supervision, it is important to equip it with an active cooling system in order to reduce the risk of thermal throttling and to maintain stable performance under prolonged load.[44] For this reason, the board was installed inside a protective case equipped with a fan, as shown in Figure 3.15.

Figure 3.15: Raspberry Pi 5 installed on Site 2 inside a protective case with active cooling.

From the perspective of the present application, the Raspberry Pi 5 offers several advantages: compact size, low power requirements, straightforward integration with official Raspberry Pi peripherals, and sufficient computational capability for a lightweight vision pipeline at the edge. At the same time, compared with more specialized embedded AI platforms such as the Jetson Nano, it provides a less dedicated acceleration stack for deep-learning inference. In this sense, its adoption at Site 2 is justified by the lower sensing complexity of the setup, which is based only on RGB imagery and does not require real-time RGB-D reconstruction or point-cloud processing. Considering the trade-off between computational capability, ease of integration, cost, and deployment simplicity, the Raspberry Pi 5 represents a suitable hardware platform for Site 2.[44, 46]

3.2.7 Raspberry Pi Camera Module 3 Wide

For RGB acquisition at Site 2, the **Raspberry Pi Camera Module 3 Wide** was employed.[47] This device is an official Raspberry Pi imaging sensor designed for embedded vision applications and directly interfaced to the Raspberry Pi 5 through the CSI camera interface.

The Camera Module 3 family is built around the **Sony IMX708** image sensor, which provides a resolution of approximately **11.9 MP** (4608×2592 pixels).[47] In addition to its rela-

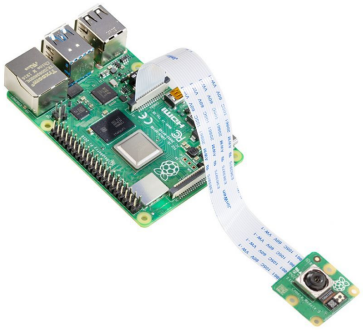


Figure 3.16: Raspberry Pi Camera Module 3 Wide mounted on a Raspberry Pi 5 board.

tively high spatial resolution, the module supports **autofocus** through phase-detection autofocus and includes **HDR** capabilities, making it suitable for embedded vision tasks in which scene conditions may vary over time.[47, 45]

The **Wide** variant was selected because it provides a **120° diagonal field of view**, which is particularly useful in industrial environments where the camera must cover a relatively broad working area from a short mounting distance.[47] In the present setup, this characteristic is important because the system must observe the table area in which baked loaves are manually removed from the trays, which is relatively large and located at a distance of approximately 1.5 m from the camera.

Figure 3.17: 3D model of the Raspberry Pi Camera Module 3 Wide.

To improve the practical integration of the camera into the Site 2 workstation, a dedicated housing solution was developed by starting from an existing online case model and modifying it for the specific constraints of the installation. The resulting enclosure was then fabricated through 3D printing. This solution was engineered to fit the geometry of the available station and to support a stable and repeatable mounting configuration for data acquisition. Besides providing mechanical protection, it improved cable management and helped maintain a more consistent camera placement under the specific operating conditions of the industrial environment, as shown in Figure 3.18.

Figure 3.18: Custom 3D-printed case for the Raspberry Pi Camera Module 3 Wide, placed in the Site 2 workstation.

From an operational point of view, the camera module supports still-image acquisition at full sensor resolution and multiple video modes suitable for embedded computer-vision pipelines, including Full HD video and higher-frame-rate lower-resolution modes.[47, 45] The native integration with the Raspberry Pi ecosystem also simplifies software development through the official camera stack based on *libcamera* and *Picamera2*.[45]

For the purposes of this thesis, the Camera Module 3 Wide offers several practical advantages, including a field of view well suited to the inspection area, autofocus capability, and tight hardware and software integration with the Raspberry Pi 5. At the same time, the use of a monocular RGB camera also introduces some limitations. Unlike the Azure Kinect employed at Site 1, the Raspberry Pi Camera Module 3 Wide does not provide depth information and therefore cannot support direct 3D reconstruction. Moreover, because the sensing modality is purely RGB-based, the quality of the acquired data remains sensitive to lighting conditions, shadows, and viewpoint-dependent appearance changes. For this reason, illumination control and camera placement remain important aspects of the Site 2 setup, as illustrated in Figure 1.2. Overall, the Raspberry Pi Camera Module 3 Wide represents a suitable sensing solution for Site 2 thanks to its wide field of view, autofocus capability, compact form factor, and seamless compatibility with the Raspberry Pi platform.[47, 45]

Picamera2 Library

For the integration of the Raspberry Pi Camera Module 3 Wide within the acquisition pipeline, the *Picamera2* library was adopted.[48, 49] Picamera2 is the Python interface built on top of the Raspberry Pi *libcamera* stack and represents the modern replacement for the legacy Picamera interface.[49]

The library provides high-level access to camera initialization, stream configuration, frame acquisition, metadata handling, and camera controls such as exposure, gain, and autofocus.[48]

These features are particularly useful in embedded computer-vision applications, where the camera must be configured programmatically and integrated into a broader software pipeline.

In the present project, Picamera2 enabled direct access to RGB frames from Python, thereby simplifying the interaction between the camera and the processing software based on NumPy and OpenCV. This made it possible to implement image acquisition, parameter tuning, and frame extraction within the same software environment used for the rest of the pipeline.

Compared with lower-level camera handling, Picamera2 provides a practical abstraction that accelerates development while still exposing the controls needed for embedded deployment. This was particularly useful not only for the implementation of the acquisition pipeline itself, but also for the two-week automated data-collection phase, in which a stable and well-supported software interface was essential to manage continuous image acquisition and reliable data storage over extended periods. For this reason, Picamera2 represented a suitable software interface for Site 2, where ease of integration, reliability, and compatibility with the Raspberry Pi ecosystem were important design requirements.[48, 49]

Chapter 4

Multimodal quality inspection framework

4.1 Pipeline

The implemented framework follows a shared offline–online logic across the two industrial sites, while adapting the final processing stage to the different monitoring objectives of raw dough and baked bread. In both cases, the first phase consists of autonomous image acquisition on an embedded edge device. The collected data are then transferred to an offline workstation, automatically labelled, and used to train a DEIMv2 detector. Once training is completed, the detector is redeployed on the edge hardware and becomes the entry point of the online inspection pipeline.[50, 51] The overall architecture was designed in a **modular** way, so that different detection, reconstruction, and downstream-analysis modules could be tested, replaced, or extended without rewriting the entire workflow.[50]

At **Site 1**, the acquisition stage relies on synchronized RGB-D frames from the Azure Kinect.[8] After online detection, each dough ball is associated with aligned depth information and converted into a local 3D representation for volumetric analysis. The adopted workflow includes an initial color-clustering stage, subsequent foreground segmentation and refinement with respect to the reference table model, 3D completion of the observed geometry, and volume estimation through multiple geometric methods.[50]

At **Site 2**, the acquisition stage is based on RGB images from the Raspberry Pi Camera Module 3 Wide.[47] After online detection, the localized bread loaves are cropped and forwarded to an anomaly-detection stage implemented with EfficientAD through anomalib.[5, 12] In this case, the detector acts as a region proposal mechanism, while the downstream anomaly model estimates how strongly each loaf deviates from the normal appearance learned from production data.[50]

4.2 Motion Detection

Background Subtraction for Motion Triggering

During the data-acquisition phase, images are not recorded continuously. Instead, the acquisition framework monitors the scene through a configurable background-subtraction stage and stores a frame only when motion is detected inside the selected region of interest (ROI).[22, 51]
In this way, the saved dataset is concentrated around the actual passage of dough balls or bread loaves, while long intervals of static background are discarded.

Figure 4.1: Example of background subtraction for motion triggering at Site 2. The operator is placing a tray of bread loaves on the output table. The tray appears as foreground, whereas the table is correctly modelled as background.

Implementation details

The implementation supports the two OpenCV subtractors MOG2 and KNN, both wrapped inside a motion-detection interface that computes a foreground mask and converts it into a normalized motion score.[22, 51] This score is then compared with configurable thresholds in order to decide whether the current frame is relevant enough to be stored. The main parameters of the acquisition logic, such as the selected sensor branch (Kinect[8] or Raspberry Pi Camera[47]), subtractor type, history length, variance threshold, motion threshold, ROI geometry, image resolution, and initial warm-up offset, are collected in a YAML configuration file (shown in Figure 4.2) so that the system can be retuned without modifying the source code.[51]

The same logic is reused across the two sites, but the sensing modality depends on the available hardware. In the Kinect branch, ROI-based triggering can be performed on the transformed depth image, the infrared image, or the RGB image. The transformed depth image is particularly convenient because it is substantially invariant to illumination changes. In RGB-only branches, such as the Raspberry Pi acquisition setup used at Site 2, the foreground mask is instead computed directly from the RGB stream.[51] The framework also skips an initial number of frames in order to stabilize the background model before the actual recording starts, thereby reducing false triggers caused by transient initialization artefacts.

```yaml
kinect: false
video_cam: ''
raspberry_cam: true
kinect_folder: /home/flexsight/Desktop/flexsight-image-acquisition/Acquisition/kinect
video_cam_folder: /home/flexsight/Desktop/flexsight-image-acquisition/Acquisition/video_cam
raspberry_cam_folder: /home/flexsight/Desktop/flexsight-image-acquisition/Acquisition/raspberry_cam
aruco: false
algo: KNN
offset: 20
motion_threshold: 0.02
var_threshold: 30
history: 50
rectangle: true
rectangle_width: 200
rectangle_height: 200
rectangle_x: 640
rectangle_y: 360
motion_threshold_rectangle: 0.02
show_mask: false
resolution: 720
fps: 15
format: jpg
enable_door_checking: false
door_threshold: 0.1
manual_mode: false
gain: -1
exposure: -1
split: true
obj_points:
```

Figure 4.2: Example of the YAML configuration file used to tune the motion-triggering module without modifying the source code.

When the motion score exceeds the configured threshold within the selected ROI, the current frame is written to disk together with the modality-specific data associated with the active sensor. In the Kinect setup, the acquisition stage stores RGB, depth, infrared, and mask data. The RGB frames are used for automatic labelling and detector training, whereas the aligned depth data are used to develop and validate the volume-estimation workflow. In the Raspberry Pi setup, the stored outputs are instead the RGB frame and the associated foreground mask, and these data are later used to train the detector and support the downstream bread-analysis pipeline.[51, 50] In order to better subdivide the events, the codebase also supports the automatic creation of timestamped folders when motion starts, so that separate acquisition sessions can be organized without manual intervention.[51]

Data Management and Robustness

Since the local storage capacity of the embedded devices is limited, the acquisition framework also includes a flushing mechanism that reacts to low-space conditions. When enabled, this mechanism pauses the acquisition logic and allows the oldest files to be transferred to an external SSD, thereby freeing local space for new data.[51] This feature is particularly useful during long autonomous campaigns, where the amount of generated data can quickly exceed the internal storage capacity of the device.

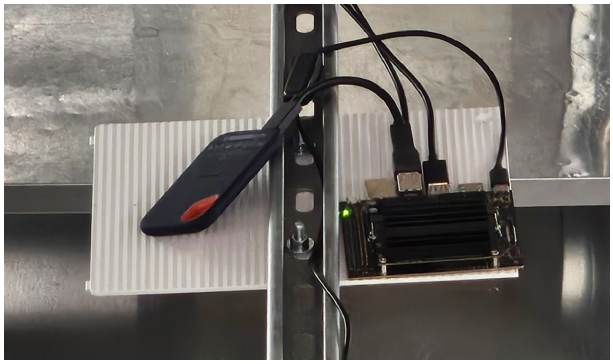


Figure 4.3: Jetson Nano with the external SSD used for data flushing during long acquisition campaigns.

To further increase robustness, the acquisition scripts can be launched automatically at startup through system services or desktop autostart entries, so that the data-collection process can resume after power interruptions or device reboots without requiring manual intervention.[51, 13] As discussed in Chapter 1, this aspect was essential for ensuring continuity during two-week acquisition campaigns in an industrial environment.

Overall, this motion-triggered acquisition strategy proved particularly suitable for long autonomous campaigns, because it reduces redundant storage while preserving the samples that are most informative for the subsequent labelling and training stages. At the same time, the framework remains modular enough to support different sensors and acquisition modes without requiring a complete redesign of the system.[51]

4.3 Automatic Labelling

To reduce the manual effort required to annotate the collected datasets and to speed up the work, an automatic labelling strategy based on Language Segment-Anything (Lang-SAM) was adopted on an offline workstation.[11] This choice is consistent with the overall workflow adopted in this thesis, since annotation generation is computationally more suitable for a higher-resource environment than the embedded devices used for data acquisition and online inference. Lang-SAM is a zero-shot framework that combines text-guided localization and segmentation, making it possible to obtain object masks or bounding boxes directly from natural-language prompts describing the target product.

In the implemented workflow, prompts referring to dough balls or bread loaves are used to detect the products in the acquired frames. The generated bounding boxes are then manually

checked for correctness and exported in a detector-compatible annotation format, in this case the COCO dataset format. This procedure provides a common labelling pipeline for both sites, significantly reduces manual annotation time, and keeps the detector-training stage consistent across the two datasets.[50]

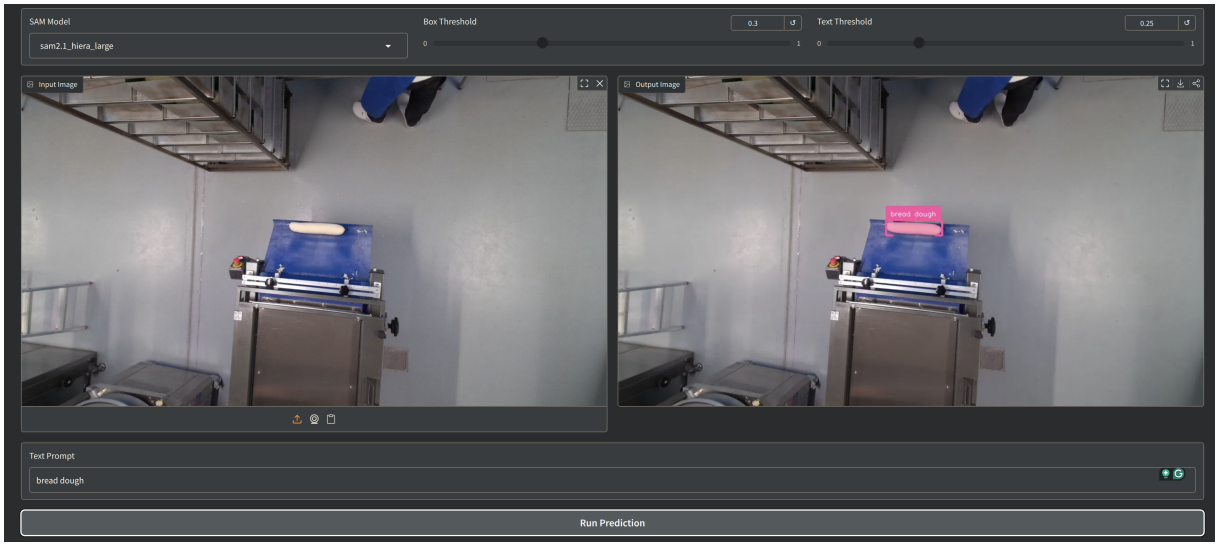


Figure 4.4: Example of automatic labelling using Lang-SAM. The left image shows the original frame with a natural-language prompt describing the target product (e.g., "dough ball" or "bread loaf"). The right image displays the resulting segmentation mask and bounding box generated by Lang-SAM. This approach significantly reduces manual annotation effort while maintaining consistency across datasets.

4.4 DEIMv2 Training

After acquisition and automatic labelling, the DEIMv2 detector is trained offline on a workstation with higher computational resources than the embedded devices used on site.[1] This choice is necessary because the edge platforms are intended primarily for acquisition and online inference, whereas detector training requires longer optimization runs and larger memory budgets. The automatically labelled images and the corresponding COCO-format annotations described in the previous section are used to build the detector datasets, which are then partitioned into training, validation, and test subsets with an 80/10/10 split.

Although the two industrial sites lead to different downstream tasks, the role of DEIMv2 is the same in both cases: the network must localize the product instances and provide reliable regions of interest for the subsequent processing stages. The model is therefore trained on two site-specific datasets, while preserving the same detector architecture and the same general training logic.[50] At Site 1, the detected regions are associated with aligned depth data and used for volumetric reconstruction. At Site 2, they are converted into loaf crops that feed the anomaly-detection branch. In this sense, DEIMv2 constitutes, together with the automatic labelling stage, the common detection layer of the overall framework.

During training, the optimization dynamics are monitored through TensorBoard, and the final checkpoint is selected according to the lowest validation loss observed during the training schedule. The resulting curves and the quantitative detection metrics obtained on the two

datasets are discussed in Chapter 5.

Once the training stage is completed, the selected checkpoint is redeployed on the embedded target platform and becomes the detector used during online acquisition and inference. This offline-to-online transition allows the computationally demanding optimization stage to remain separated from the lightweight deployment stage, while preserving a consistent detection pipeline across the two monitoring sites.[50]

4.5 Feature Extraction and Quality Assessment

4.5.1 Site 1: Volume Estimation

The volume-estimation branch at Site 1 was developed through several increasingly refined workflows. A first baseline estimated volume by computing a convex hull directly from the points contained within each detected bounding box. However, this bounding-box-only approximation was not sufficiently accurate, because the selected region also included non-dough elements that biased the reconstructed geometry. Subsequent variants therefore introduced an explicit segmentation stage before volumetric reconstruction. SAM [37] was first explored as a foreground-segmentation tool and produced very accurate masks, but its computational cost proved too high for the intended workflow, whose long-term goal is deployment on edge devices. The final implementation therefore relies on a more efficient pipeline based on color clustering, refined through depth-based background subtraction and followed by axis-based 3D completion.[50]

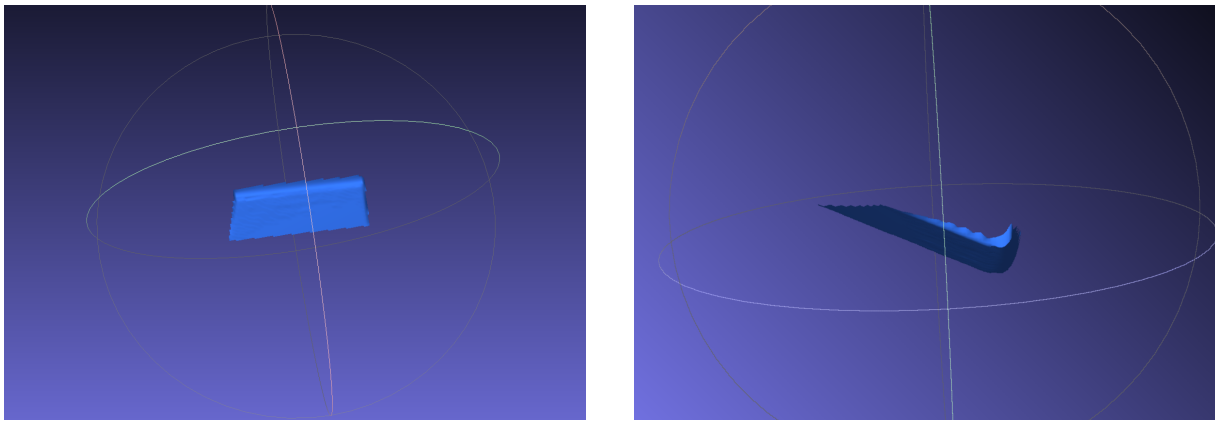
Detection Filtering

Before volumetric processing, the detections are filtered so as to retain only the most reliable samples. In particular, a dedicated preprocessing stage builds a filtered detection cache by keeping only images containing a single valid dough-ball detection inside a predefined ROI polygon. This step reduces ambiguities caused by multiple simultaneous objects, spurious detections near the borders of the acquisition area, or colliding dough balls, all of which can be problematic for the subsequent volume-estimation stage.

This filtering stage was introduced because the main aim of this part of the thesis is to analyze the consistency of the volume-estimation algorithms on a well-defined set of samples, rather than to evaluate the robustness of the detector under crowded or ambiguous conditions. According to the nominal operating conditions of the line, multiple dough balls should not be present simultaneously on the output table; frames exhibiting such cases were therefore treated as outliers and excluded from the analysis.

Reference Table Modelling

The supporting table is modelled separately from a set of empty-scene acquisitions. These depth images are aggregated through a robust median operation in order to produce a reference depth background for the table. The resulting model is stored as a numerical depth map for subsequent computations and can also be exported as a point cloud or mesh for debugging and visualization purposes, as shown in Figure 4.5.[50] This reference surface is later used both to remove the support contribution from the dough point cloud and to define the geometric relation between the visible dough region and the table plane.



(a) Front view **(b)** Side view

Figure 4.5: Reference table model reconstructed from empty-scene depth acquisitions.

Foreground Extraction and Point-Cloud Generation

Once a dough-ball detection has been selected, the corresponding RGB bounding box is associated with the aligned depth image through the Kinect calibration and reprojection pipeline. Because the RGB and depth streams of the Azure Kinect have different intrinsic parameters, extrinsic relationships, and fields of view, a dedicated reprojection and alignment stage was implemented using the calibration data provided by the Azure Kinect SDK and accessed in Python through the *pyKinectAzure* interface.[42, 43] This stage makes it possible to map the RGB detections onto the corresponding depth coordinates and therefore to associate each RGB bounding box with the correct 3D measurements.

```
# ---- Kinect params for volume.py (reprojection) ----
fx_depth = 252.43328857421875
fy_depth = 252.50279235839844
cx_depth = 162.43429565429688
cy_depth = 169.75067138671875
depth_scale = 0.001

fx_rgb = 608.0234375
fy_rgb = 607.9978637695312
cx_rgb = 642.4961547851562
cy_rgb = 364.57635498046875

R_rgb_to_depth = np.array([
    [0.9999880790710449, -0.004373531322926283, 0.0021795618813484907],
    [0.004568530712276697, 0.9950369596481323, -0.09940114617347717],
    [-0.0017340105259791017, 0.09940991550683975, 0.9950450658798218],
])
t_rgb_to_depth = np.array([0.03207849711179733, 0.002367959590628743, -0.003765013534575700 8])
# ----------------------------------------------------
```

Figure 4.6: Kinect reprojection parameters. The figure shows the RGB and depth intrinsic parameters, as well as the extrinsic transformation between the two sensors, which are used to align the RGB detections with the depth data.

The local 3D point cloud is then obtained from the aligned depth pixels by means of the standard pinhole camera model:

$$X = \frac{(u - c_x) \cdot Z}{f_x}, \quad Y = \frac{(v - c_y) \cdot Z}{f_y}, \quad Z = D(u, v) \tag{4.1}$$

where (u, v) are the pixel coordinates, $D(u, v)$ is the depth value, (c_x, c_y) are the principal-point coordinates, and f_x and f_y are the focal lengths expressed in pixel units.
The implementation supports different segmentation modes, namely `sam`, `bbox`, and `color`.[50] These modes are configurable through the experiment configuration file, whereas the workflow adopted for the main experiments initializes the foreground in `color` mode and then refines it in depth (`color_then_depth`). More specifically, a central rectangular seed (shown in yellow in Figure 4.7) is first derived from each detection, and the clustering stage can also be extended by a configurable percentage beyond the predicted bounding box so as to remain robust even when the detector localization is not perfectly tight. The color-based stage suppresses the blue background, constrains the admissible lightness range, optionally performs adaptive clustering in the LAB color space, and can recover dark dough regions that would otherwise be missed by a purely brightness-based filter.[50]

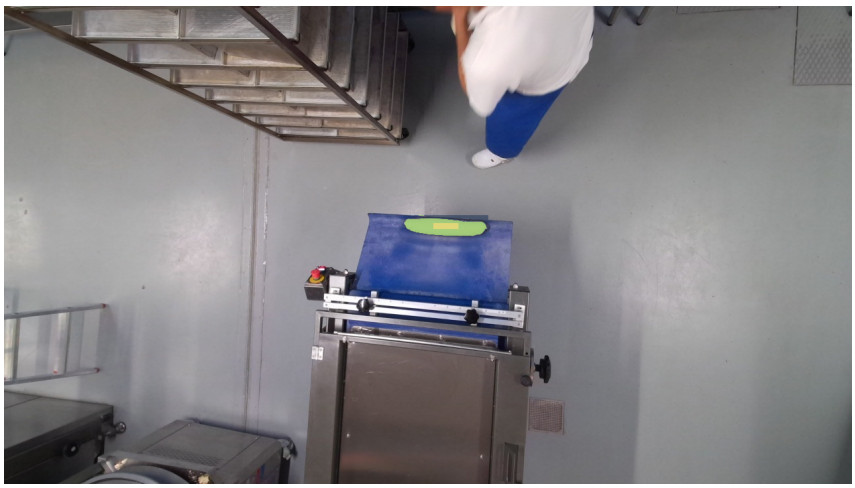
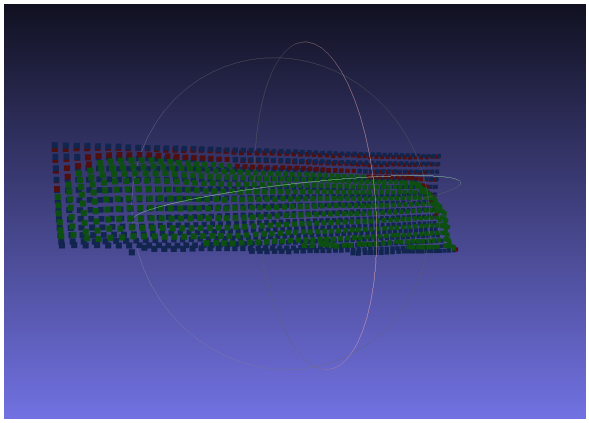


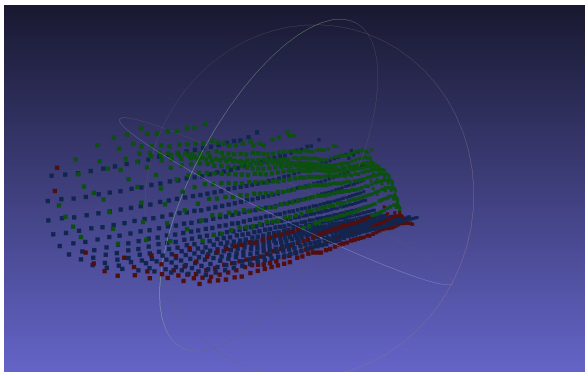
Figure 4.7: Example of the color-based segmentation stage. The dough point cloud is shown in green, filtered points are highlighted in red, the seed rectangle is shown in yellow, and the background region excluded by clustering is shown in blue.

In an earlier version of the workflow, this clustering step was not yet present; it was introduced later because it produced a cleaner foreground estimate before depth refinement. The resulting mask is then projected onto the depth data and further refined through background subtraction against the precomputed table model, with additional morphological filtering to improve spatial coherence.[50] A practical difficulty in this stage is that the depth measurements near the border of the table, especially along its final curved portion, may exhibit singular or unstable values that make the support surface harder to filter correctly. In this respect, the two refinement stages complement each other: color clustering helps suppress unstable depth regions near the table border before they enter the volumetric estimation, whereas background subtraction removes regions that may be chromatically similar to the dough but are not geometrically consistent with the support model. This issue is also mitigated by combining an adaptive background-

subtraction threshold with the distance from the reference table surface. These morphological operations remove points in the point cloud that intersect, penetrate the table model, or are closer than a certain threshold to the background, thereby enforcing a geometrically consistent foreground prior to volumetric reconstruction.

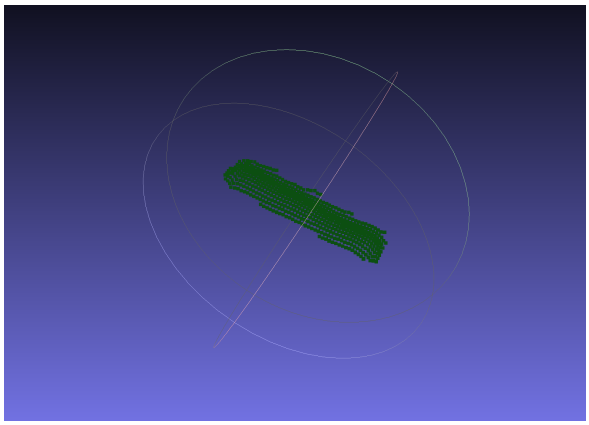


(a) Front view of table-collision filtering

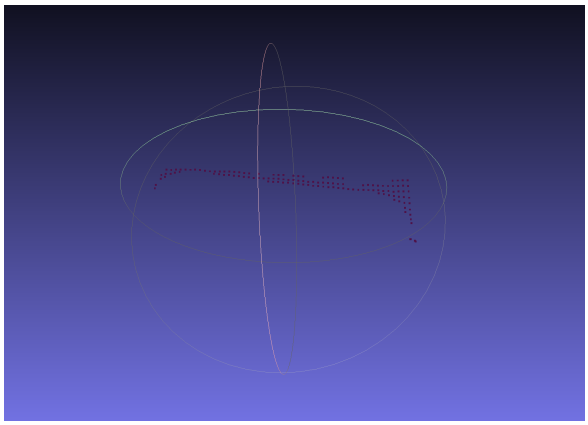


(b) Side view of table-collision filtering

Figure 4.8: Examples of depth-based refinement against the reference table model during foreground extraction. Red points are removed because they are too near or intersect the table model (blue points), while the remaining green points are retained as the refined dough foreground.



(a) Final foreground after refinement



(b) Points removed during refinement

Figure 4.9: Result of the depth-based refinement stage. The left image shows the final foreground after filtering, while the right image highlights the points that were removed because they were inconsistent with the reference table model. This refinement step helps to ensure that the subsequent volume estimation is based on a more accurate representation of the dough geometry.

Example of Combined Clustering and Background Subtraction

A representative example of this combined strategy is shown in Figures 4.10 and 4.11. In some cases, the color-clustering stage can include a region that is chromatically similar to the dough but does not belong to the actual product surface. When the corresponding mask is compared with the reference table model, the depth-based background subtraction removes the geometrically inconsistent portion, leaving only the region that is compatible with the dough foreground. This example highlights the practical advantage of combining appearance-based clustering with geometric refinement, since the first stage provides a broad candidate region while the second suppresses portions that are incompatible with the known support geometry.

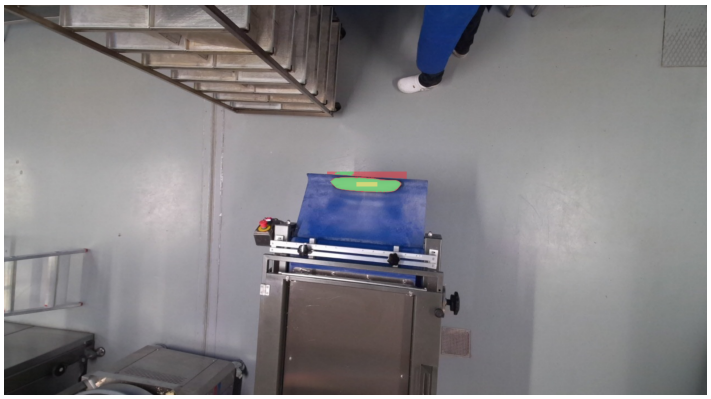
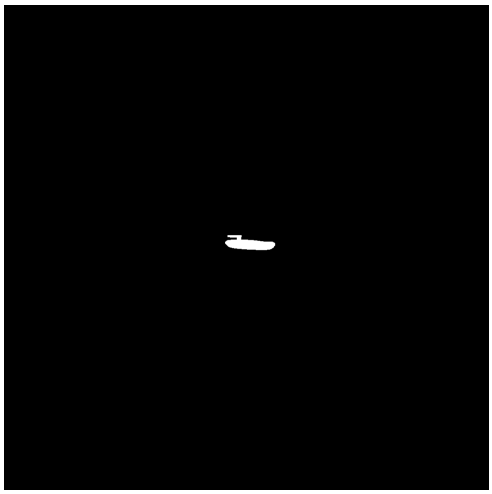
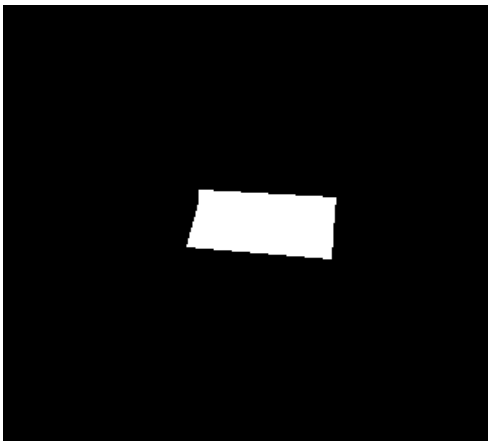


Figure 4.10: Example of bad clustering input. The color-based clustering stage includes an additional region in the upper-left area that is chromatically similar to the dough but does not belong to the actual product surface.

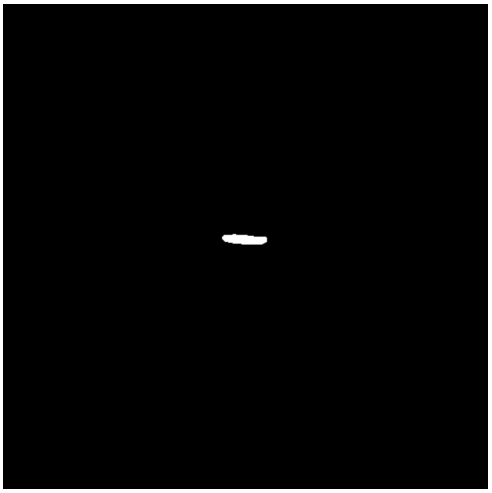


(a) Mask after color clustering. Note the additional spurious region in the upper-left area



(b) Reference table mask

(c) Cluster compared with table model

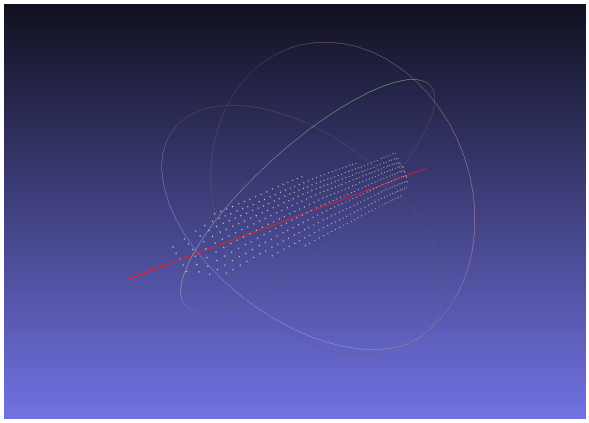


(d) Final mask after background subtraction

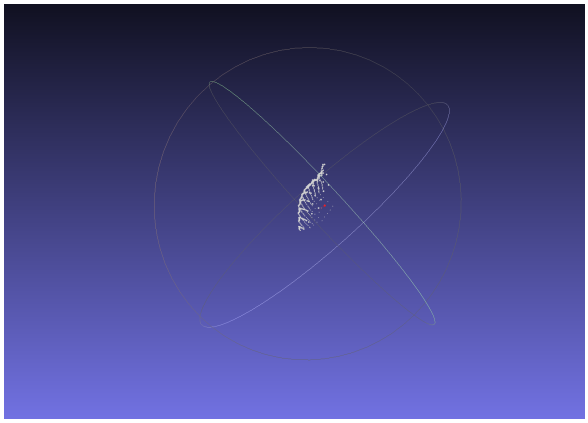
Figure 4.11: Example showing the complementarity between color clustering and depth-based background subtraction. The clustering stage initially includes an additional region outside the dough, which is then removed by enforcing consistency with the reference table model.

3D Completion and Volume Estimation

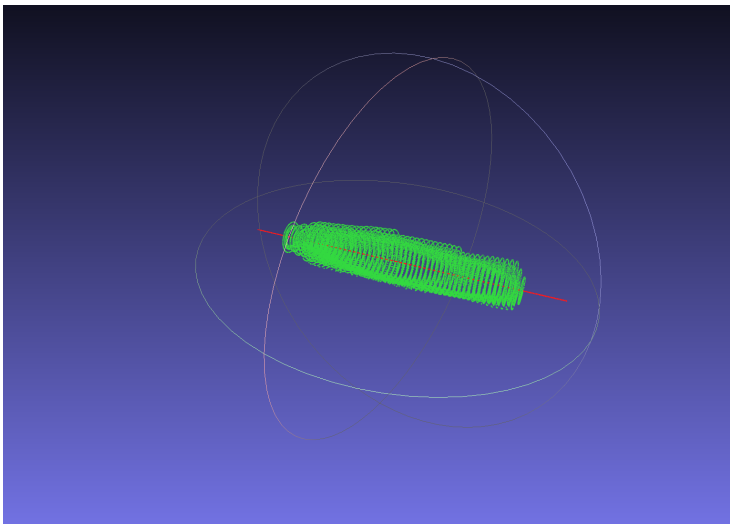
After foreground extraction, the visible dough points provide only a 2.5D observation of the object. To estimate the hidden lower portion, the implementation supports several completion strategies, including axis-construction modes based on PCA and variants explicitly tied to the support surface.[50] In the workflow adopted for the main experiments, the rotated estimators use the `mid_support_pca` strategy: the direction of the rotation axis is derived from PCA on the visible and already filtered dough cloud, while the axis itself is translated so as to lie halfway between the foreground and the support plane. This placement provides a more plausible geometric center for the subsequent rotational completion.



(a) Visible point cloud with rotation axis



(b) Side view of the same configuration



(c) Point cloud after rotational completion

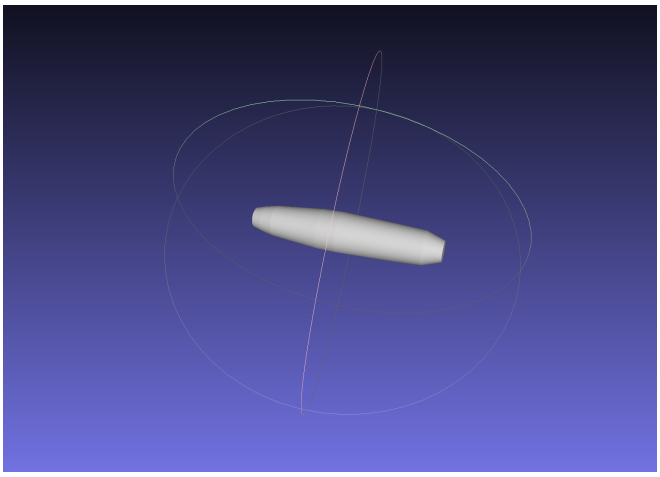
Figure 4.12: Example of the geometric completion stage. The first two views show the visible dough point cloud together with the adopted rotation axis (in red), while the third view shows the completed point cloud obtained after rotation.

The observed geometry is then rotated around this axis, using Rodrigues-based rigid rotations, in order to synthesize an approximation of the non-visible part of the dough. This rotational completion is used only for the rotated convex-hull and rotated alpha-shape estimators. After rotation, points that are geometrically inconsistent with the support model are discarded, and an additional statistical outlier removal stage is applied so that isolated or unstable points do not disproportionately affect the final volume estimate. This cleaning stage is especially important

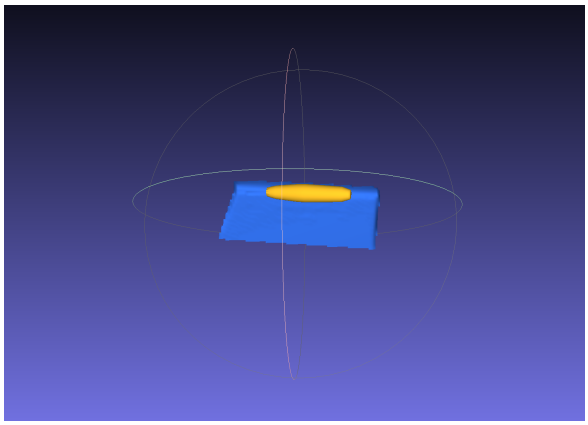
for the rotated estimators, since residual singular depth points near the border of the support can otherwise have a disproportionately large impact when the completion rotates them, leading to a much larger volume estimate.

The completed point cloud is then passed to different geometric volume estimators. A raw convex hull computed directly from the visible cloud is used only as an initial baseline and is not retained for the final comparative analysis.

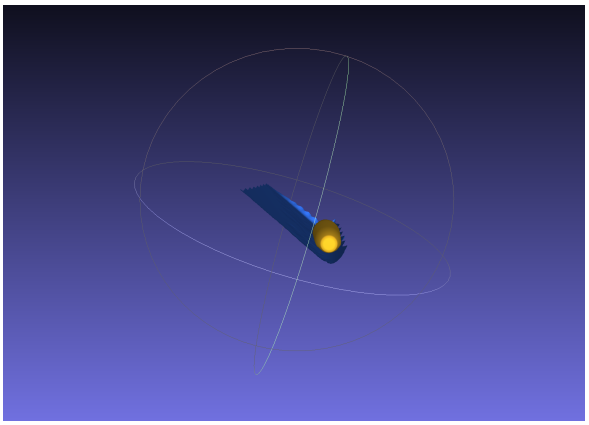
Rotated convex hull. A convex hull is computed after rotational completion of the dough geometry.



(a) Convex-hull mesh



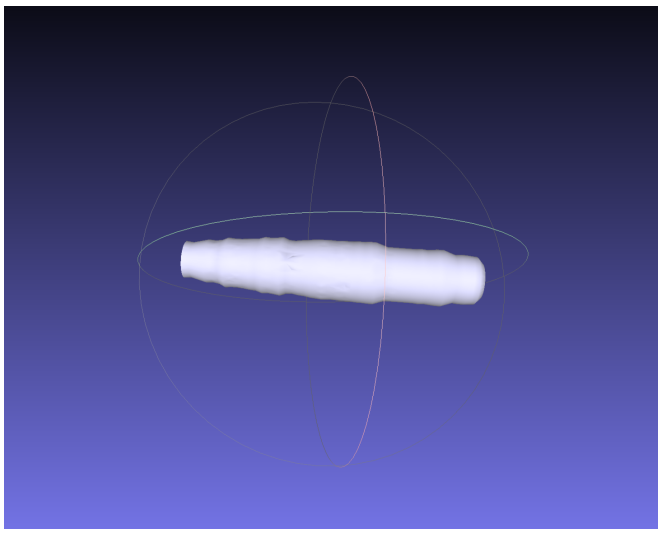
(b) Front view on the reference table



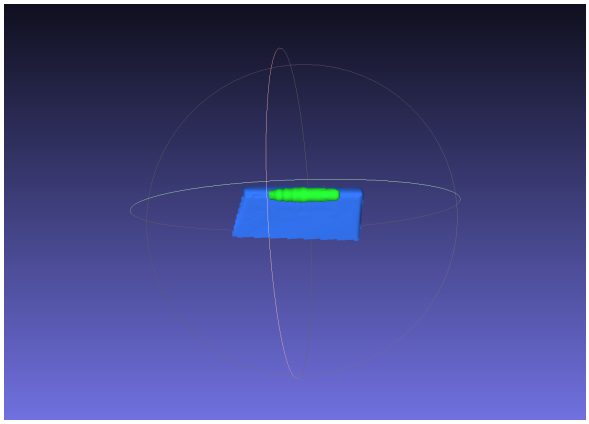
(c) Side view on the reference table

Figure 4.13: Representative output of the rotated convex-hull estimator.

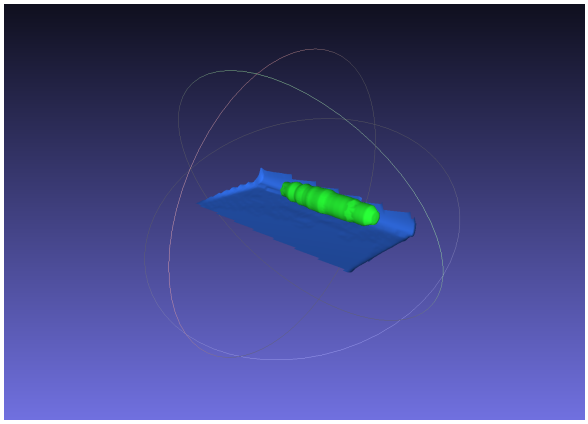
Rotated alpha shape. An alpha-shape reconstruction is computed after rotational completion of the dough geometry.



(a) Alpha-shape mesh



(b) Front view on the reference table



(c) Side view on the reference table

Figure 4.14: Representative output of the rotated alpha-shape estimator.

Depth-gap estimator. The *depth-gap* method follows a different logic and does not rely on rotational completion. Instead, it directly fills the space between the segmented foreground and the reference support model.

Figure 4.15: Representative output of the *depth-gap* estimator.

Representative outputs of the three main estimators considered in this thesis are shown in Figures 4.13, 4.14, and 4.15. The full implementation therefore makes it possible to compare simple geometric baselines with progressively more structured reconstructions. In the comparative analysis reported in this thesis, the three main estimators considered over the entire test set are the rotated convex hull, the rotated alpha shape, and the depth-gap method. Their resulting volume distributions are then analyzed and compared in order to assess the consistency of the reconstructed dough volumes, as discussed in Chapter 5.

4.5.2 Site 2: Anomaly Detection

At Site 2, the downstream task is not geometric reconstruction but visual anomaly assessment of the final baked product. For this reason, the anomaly-detection branch operates on localized loaf crops rather than on full production-line frames. In the implemented framework, these crops are processed by EfficientAD through anomalib, so that each loaf can be assigned an image-level anomaly score together with spatial diagnostic outputs such as anomaly maps and predicted masks.[5, 12]

Dataset Preparation from Loaf Crops

The first step of the anomaly-detection branch consists of exporting bread-loaf crops from the detector annotations. A dedicated utility reads COCO-style bounding-box annotations, optionally enlarges the crop through configurable padding, discards excessively small detections, and organizes the resulting images into a structure compatible with anomaly-detection training.[50]

Two complementary dataset-construction strategies are supported. The first is a simple automatic split, which partitions the crops into train, validation, and test sets according to a 0.8/0.1/0.1 ratio. The second is an interactive review tool through which exported crops can be inspected and manually assigned to good or defective/anomalous subsets before being distributed across the dataset folders.[50] This strategy is particularly useful when anomalous samples are few, heterogeneous, externally sourced, or when a small but carefully controlled exploratory dataset is needed. In the present work, this tool was also used in a preliminary test on about 300 good samples, while crops partially occluded by a hand or by the tray were operationally assigned to the defect group. The aim of this test was not to define a representative taxonomy of industrial defects, but to verify whether the network could rapidly separate clearly non-good samples from the good dataset in the event that defective products became available during operation. Since the results of this preliminary test were promising, the model was then trained on the full collected dataset, which was subsequently partitioned into training, validation, and test subsets according to the adopted split.

In the experiments described in this thesis, the normal class is composed of *good* bread images collected during the two-week acquisition campaign. Since no representative defective loaves were provided either by the production site or by the SMACT [10] operators, **the collected on-site dataset was treated as entirely normal**. To obtain an initial qualitative indication of the model behavior on clearly non-conforming products, a small auxiliary set of **18 defective bread images** collected **from online sources** was used as an external reference, with 9 images assigned to the validation set and 9 distinct images assigned to the test set. These images were not used to define the normal training distribution, but only as auxiliary anomalous examples for preliminary qualitative evaluation. They depict clearly non-conforming products, such as burnt or mouldy loaves, but they are not fully representative of either the bread type or the visual environment observed in the industrial setup.

Figure 4.16: Examples of externally sourced defective bread images used only for qualitative validation and testing of the anomaly-detection branch.

For this reason, they are not expected to provide a rigorous estimate of industrial anomaly-detection performance; rather, they offer a **practical sanity check** for the anomaly scores and maps produced by the trained model.[50] This choice remains consistent with the one-class training paradigm, which focuses on learning the normal class and then relies on anomaly scores to identify deviations from that normality, even when the specific nature of the anomalies is not known in advance.[5]

EfficientAD Training

Once the loaf crops have been organized, EfficientAD is trained through anomalib.[12] The method remains one-class: it learns the appearance of normal bread from the training set of good samples and later measures deviations from this learned normality at inference time.[5] Accordingly, the actual training phase is based on the normal samples only, whereas any external anomalous examples are reserved for exploratory validation or qualitative inspection.

The training scripts are designed to be practical for real experimentation. They expose parameters such as image size, batch size, mixed-precision usage, and validation limits, thereby making it possible to adapt the training run to the available GPU memory.[50] The implementation also supports scenarios in which anomalous validation or test samples are absent, while still allowing external anomalous images to be included in the validation or test folders whenever a preliminary qualitative check is needed. In this way, the same codebase can be used both for strict one-class training and for exploratory validation of anomaly scores on externally sourced defect examples.

Inference and Reporting

After training, the best checkpoint is loaded for inference on batches of loaf crops. Because large inference sets may exceed memory or I/O limits if processed at once, the repository includes a chunked inference routine that iterates over the dataset in manageable groups and stores the predictions incrementally.[50]

For each analyzed crop, the implemented pipeline produces an image-level anomaly score, a predicted class label, an anomaly map, and a binary mask. The outputs are then saved both in numerical form, through CSV and text summaries, and in visual form, through heatmaps, overlays, and dedicated debug folders containing the most anomalous examples.[50] In the

context of this thesis, the most relevant quantity is the anomaly score itself, since it provides the scalar measure used to compare samples and to analyze the distribution of potentially problematic loaves over the inspected set. These results are then shown and analyzed in Chapter 5.

Chapter 5

Results

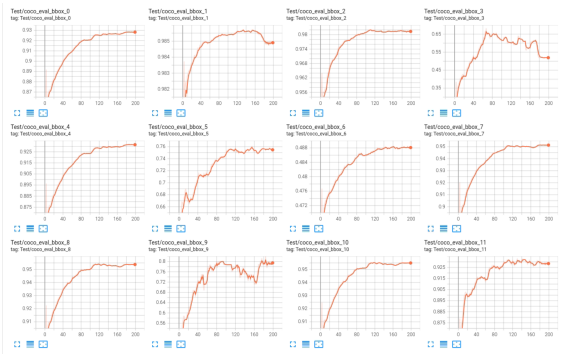
5.1 Training results

The DEIMv2 training process was monitored for both site-specific datasets through **TensorBoard**, so as to track the evolution of validation accuracy and of the main loss terms across the **200 training epochs**. These curves provide a compact view of the optimization dynamics and support the selection of the final checkpoint used for deployment.

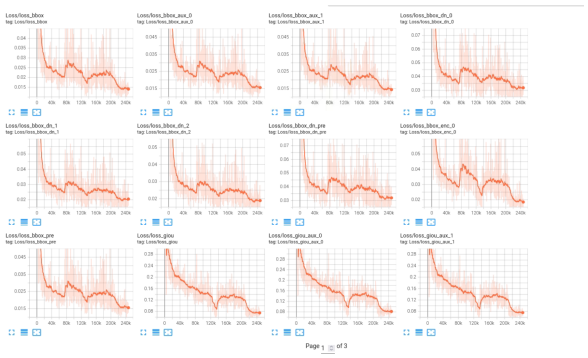
In both experiments, the detector was trained for 200 epochs and the final model was selected according to the **lowest validation loss** observed during training. This criterion provided a consistent model-selection rule across the two datasets and made it possible to retain the checkpoint corresponding to the most favorable validation regime within the fixed training schedule.

5.1.1 Site 1: Training on the Dough-Ball Dataset

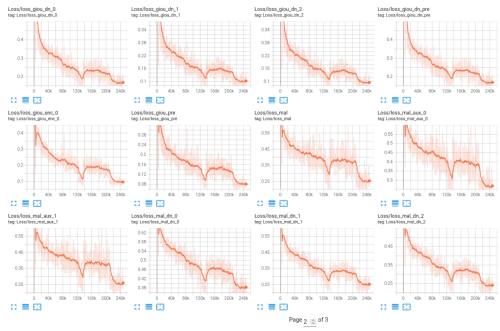
The first training experiment was carried out on the dough-ball dataset collected at Site 1, which consisted of approximately 50,000 images partitioned into training, validation, and test subsets with an 80% - 10% - 10% split. Figure 5.1 shows the TensorBoard plots recorded during training. Overall, the curves indicate a stable optimization process, with progressive convergence of the loss terms and a validation trend that supports the selected final checkpoint.



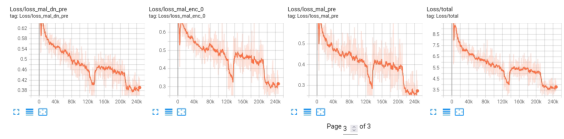
(a) Validation accuracy



(b) Loss term 1



(c) Loss term 2



(d) Loss term 3

Figure 5.1: TensorBoard monitoring for the DEIMv2 training on the Site 1 dough-ball dataset. The plots report the evolution of validation accuracy and loss terms over the 200-epoch training schedule. As can be observed, the best validation checkpoint was reached before the end of the training schedule, confirming that the selected model corresponds to a favorable validation regime.

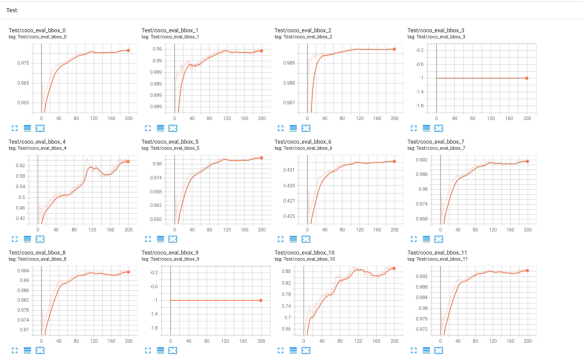
The final detector achieved an Average Precision (AP) of 0.929 and an Average Recall (AR) of 0.953 on the evaluation split, indicating that the model was able to localize dough balls with high reliability despite the challenging visual configuration of the site.

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.929
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.985
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.981
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.621
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.932
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.751
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.489
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.951
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.953
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.954
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.923
 Average Recall     (AR) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.995
```

Figure 5.2: Quantitative DEIMv2 detection results obtained on the Site 1 dough-ball dataset.

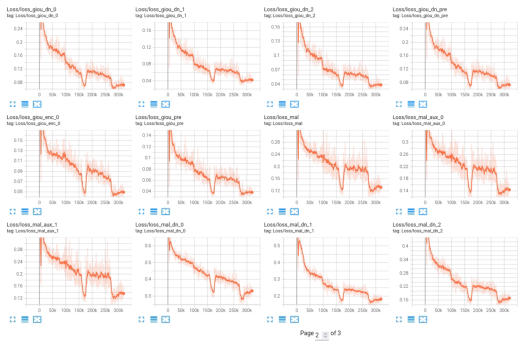
5.1.2 Site 2: Training on the Bread-Loaf Dataset

The second training experiment was carried out on the bread-loaf dataset collected at Site 2, which consisted of approximately 70,000 images partitioned into training, validation, and test subsets with an 80% - 10% - 10% split. Also in this case, Figure 5.3 shows a stable optimization process and supports the checkpoint selected through the validation-loss criterion.

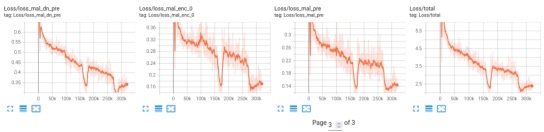


(a) Validation accuracy

(b) Loss term 1



(c) Loss term 2



(d) Loss term 3

Figure 5.3: TensorBoard monitoring for the DEIMv2 training on the Site 2 bread-loaf dataset. The plots report the evolution of validation accuracy and loss terms over the 200-epoch training schedule. As can be observed, the best validation checkpoint was reached before the end of the training schedule, confirming that the selected model corresponds to a favorable validation regime.

The final detector achieved an outstanding Average Precision (AP) of 0.982 and an Average Recall (AR) of 0.994 on the evaluation split, confirming that the model was able to localize bread loaves very accurately in the Site 2 setting.

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.982
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.644
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.983
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.432
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.992
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.994
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.876
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.994
 Average Recall     (AR) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
```

Figure 5.4: Quantitative DEIMv2 detection results obtained on the Site 2 bread-loaf dataset.

Result comparison and discussion

Comparing the two experiments, Site 2 achieved slightly higher detection performance than Site 1. A plausible explanation is the different visual complexity of the two datasets: at Site 1, dough balls may touch or partially overlap during production, making instance separation more difficult, whereas at Site 2 the bread loaves are generally more isolated and easier to localize. Since the same detector architecture and the same general training logic were used in both cases, this difference is likely associated more with dataset difficulty than with changes in the training procedure itself. Overall, both training experiments produced highly accurate detectors that are suitable for deployment in the respective acquisition sites, confirming the effectiveness of the DEIMv2 training approach for real-time detection in challenging industrial environments.

5.2 Site 1: dough-ball volume estimation

5.2.1 Quantitative results

The quantitative analysis of dough-ball volume was carried out on the three estimators introduced in Chapter 4: the depth-gap method, the rotated alpha-shape reconstruction, and the rotated convex-hull reconstruction. Figures 5.5, 5.6, and 5.7 provide visual representations of the corresponding volume distributions across the valid test samples, which are approximately 15,000 for each estimator after filtering, while Tables 5.1, 5.2, and 5.3 summarize the main descriptive statistics exported by the evaluation scripts.

For ground-truth comparison, the expected volume interval was derived from the nominal weight of the dough balls, reported by the SMACT [10] operators as 1.54 kg, together with the typical dough density, approximately between *0.9 kg/dm*^{3} and *1.1 kg/dm*^{3}. This yields an expected volume range of **[1.38, 1.72] liters**. Before computing the descriptive statistics, non-plausible volume estimates, such as extremely small or extremely large values caused by reconstruction failures or borderline cases, were excluded from the analysis. For this reason, the tables report the number of valid samples retained for each estimator together with the mean, median, standard deviation, minimum and maximum valid volumes, and the percentage of samples falling inside this interval, so as to provide a complete picture of the reconstructed volume distributions and their consistency with the nominal product characteristics.

Depth-gap estimator

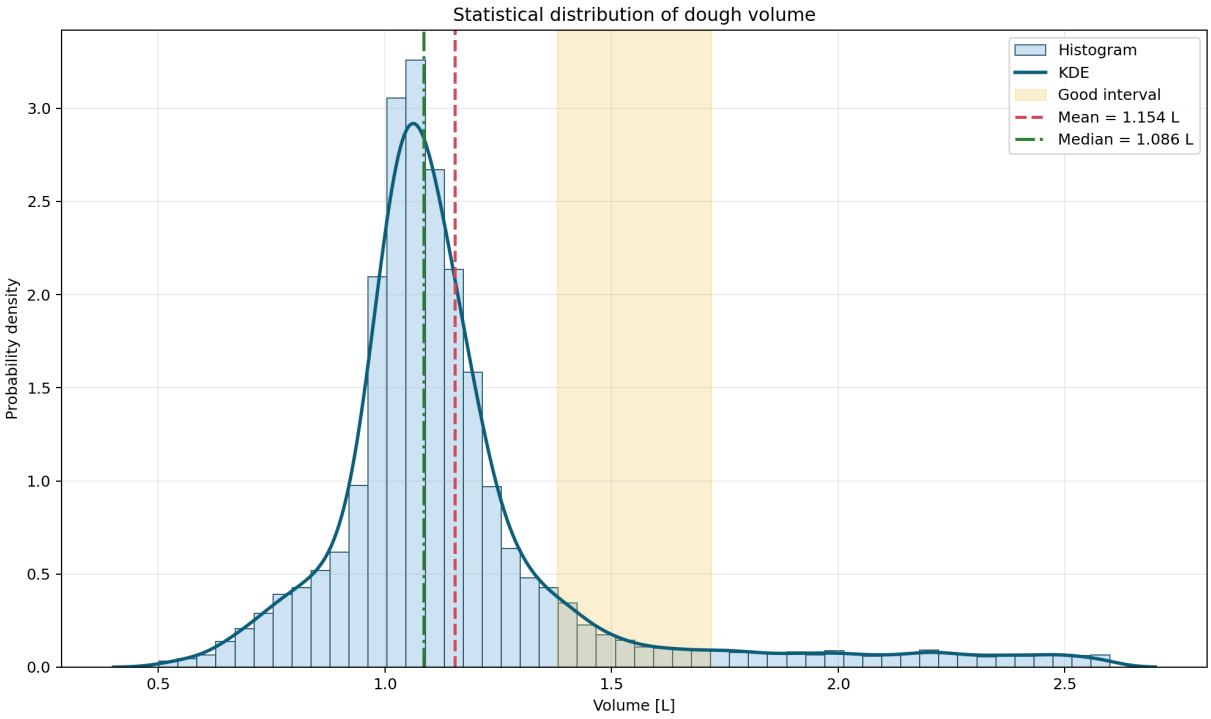


Figure 5.5: Volume distribution obtained with the depth-gap estimator.

Table 5.1: Volume statistics obtained with the depth-gap estimator. Volumes are expressed in liters.

Metric	Value
Original samples	17766
Valid samples after filtering	15902
Discarded outliers	1864
Mean volume	1.153842
Median volume	1.085705
Standard deviation	0.313228
Minimum valid volume	0.500858
Maximum valid volume	2.599818
Expected interval	[1.380000, 1.720000]
Samples within expected range	874
Samples within expected range (%)	5.50

Alpha-shape estimator

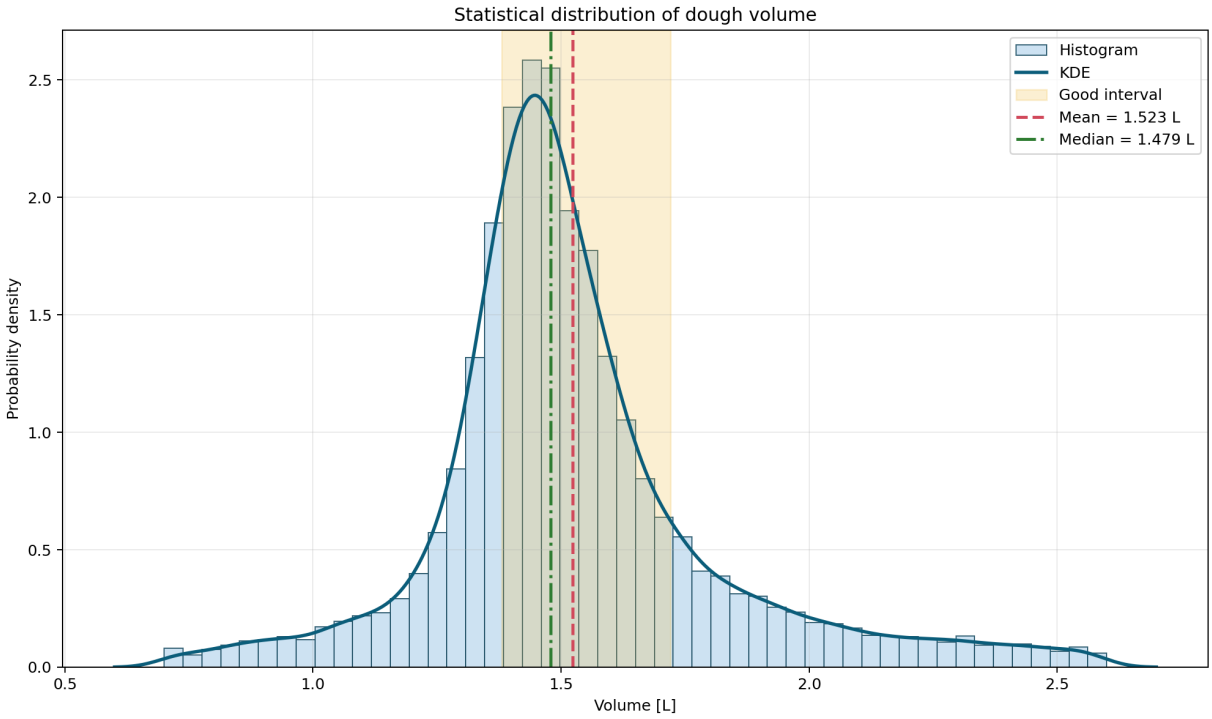


Figure 5.6: Volume distribution obtained with the rotated alpha-shape estimator.

Table 5.2: Volume statistics obtained with the rotated alpha-shape estimator. Volumes are expressed in liters.

Metric	Value
Original samples	17739
Valid samples after filtering	15185
Discarded outliers	2554
Mean volume	1.523359
Median volume	1.478649
Standard deviation	0.286593
Minimum valid volume	0.700107
Maximum valid volume	2.599819
Expected interval	[1.380000, 1.720000]
Samples within expected range	8746
Samples within expected range (%)	57.60

Convex-hull estimator

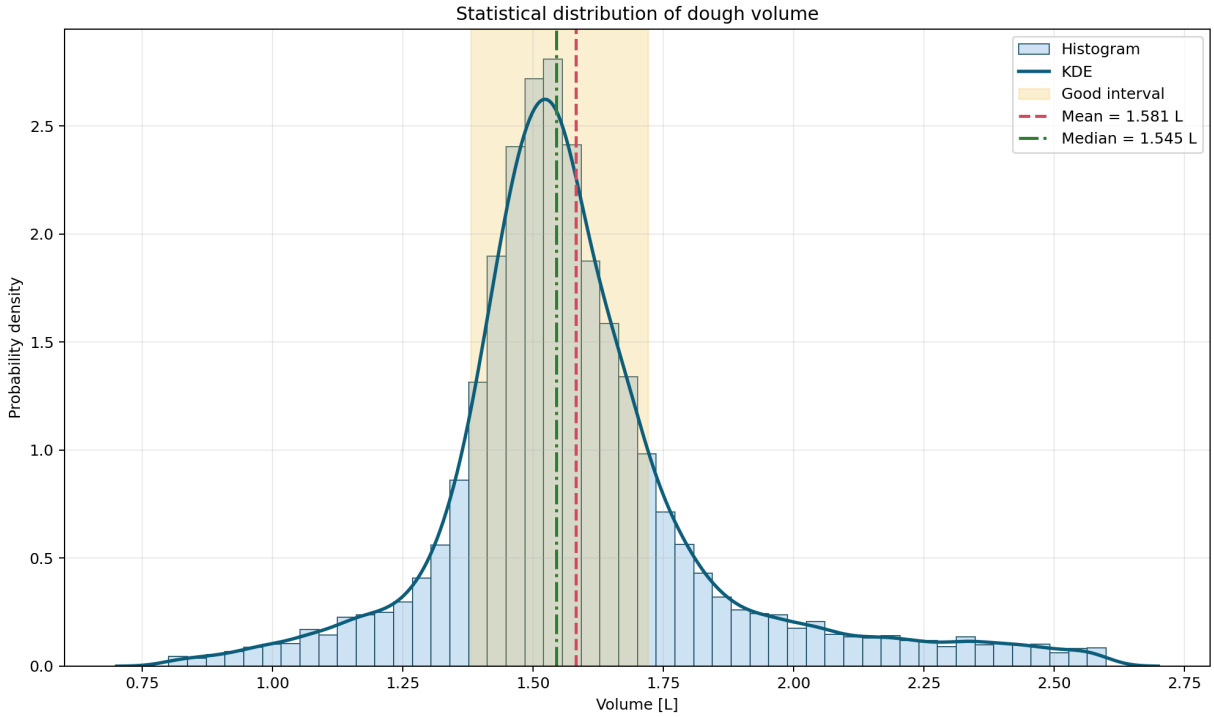


Figure 5.7: Volume distribution obtained with the rotated convex-hull estimator.

Table 5.3: Volume statistics obtained with the rotated convex-hull estimator. Volumes are expressed in liters.

Metric	Value
Original samples	17740
Valid samples after filtering	15261
Discarded outliers	2479
Mean volume	1.581481
Median volume	1.544832
Standard deviation	0.260279
Minimum valid volume	0.800663
Maximum valid volume	2.599514
Expected interval	[1.380000, 1.720000]
Samples within expected range	10338
Samples within expected range (%)	67.74

Discussion

The results show a clear difference among the three methods. The depth-gap estimator yields a significantly lower percentage of samples inside the expected interval, indicating a systematic tendency to underestimate the final volume. By contrast, both rotational estimators achieve a much higher agreement with the nominal target range, suggesting that the geometric completion stage provides a more plausible reconstruction of the dough shape. Among them, the **rotated convex-hull estimator** achieves the highest percentage of samples within the expected interval (67.74%), followed by the rotated alpha-shape estimator (57.60%). Although both methods

are computed from the same completed point cloud, the convex hull provides a slightly larger enclosing geometry, which in this case results in volume estimates that are more consistent with the expected range.

Figure 5.8 provides a visual comparison of the volume distributions obtained with the three estimators, highlighting the differences in their central tendencies and variabilities. The depth-gap method shows a distribution that is skewed towards lower volumes, while the rotational methods exhibit distributions that are more centered around the expected interval, with the convex-hull method showing a tighter concentration of values within the target range. Overall, these results suggest that geometric completion and reconstruction techniques play a crucial role in improving the accuracy of dough-ball volume estimation, and that the choice of reconstruction method can have a significant impact on the final estimates. The depth-gap method, while simpler and computationally less intensive, does not capture the full three-dimensional geometry of the dough ball and therefore tends to produce systematically lower volume estimates. In contrast, the rotational methods, by providing a more complete reconstruction of the dough shape, are able to produce volume estimates that are more aligned with the expected values based on the nominal weight and density of the product.

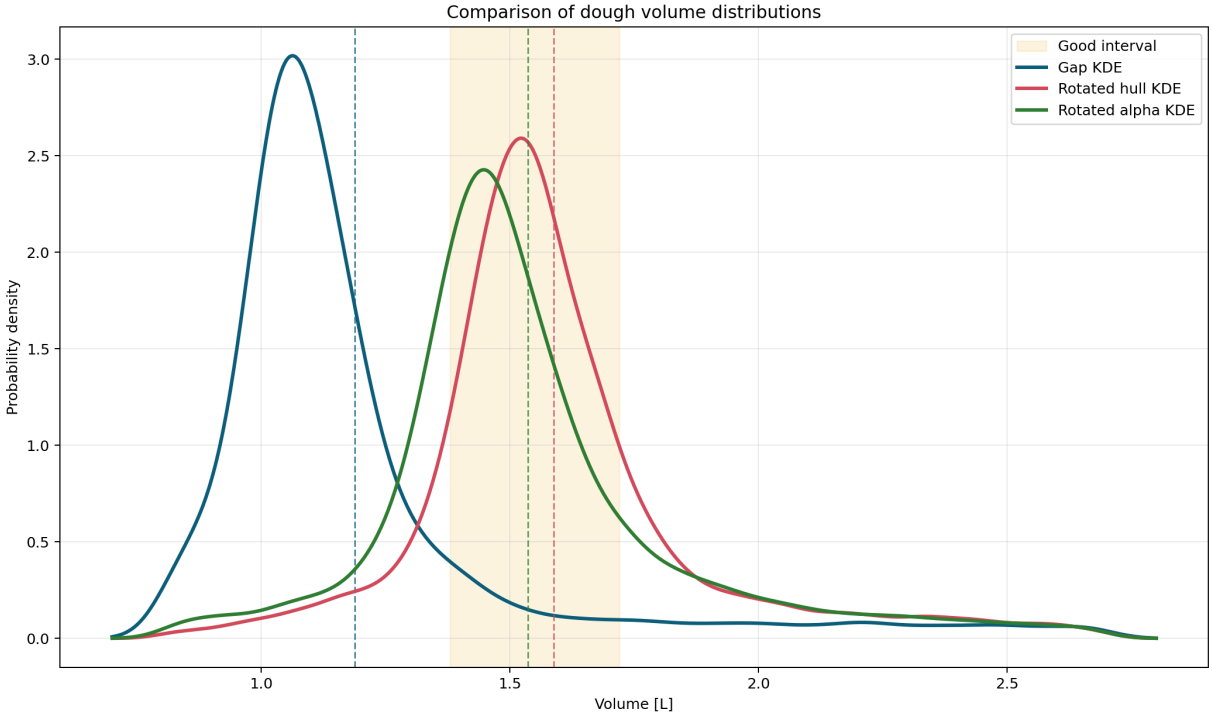


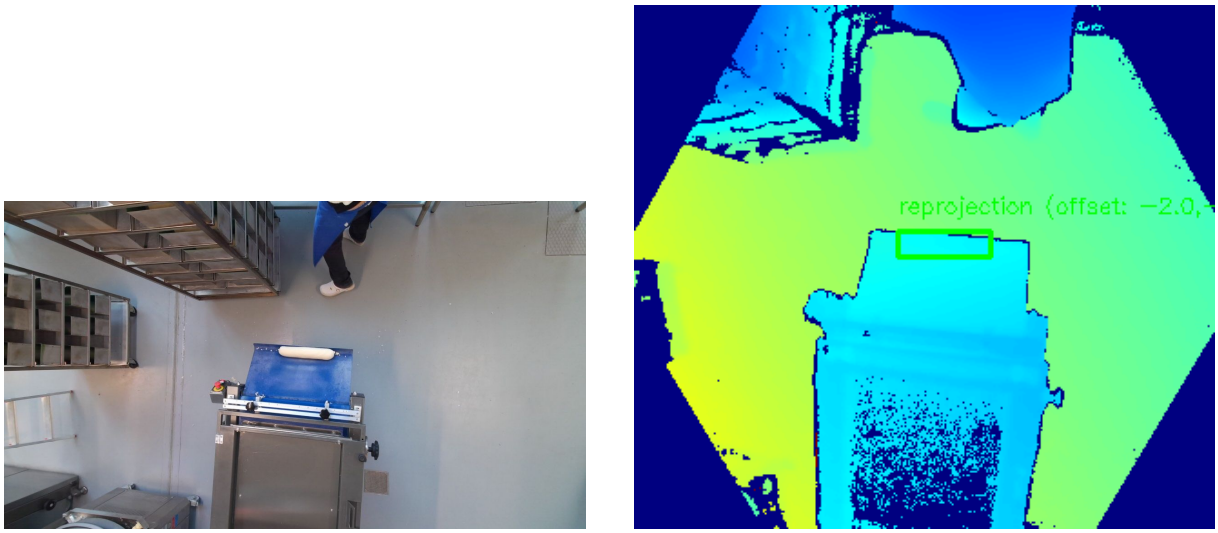
Figure 5.8: Comparison of the volume distributions obtained with the three estimators.

The reconstruction-based methods, particularly the convex-hull estimator, demonstrate a significant improvement in volume-estimation accuracy, making them more suitable for practical applications in the industrial context of dough-ball quality assessment. Despite the limited resources available for this study and the fact that the expected reference volume is itself derived indirectly from nominal weight and estimated density rather than from direct volumetric measurements, the obtained agreement is already very promising. It should also be considered that the rotational methods approximate the dough geometry through a surface generated around

the estimated axis, thus producing an idealized cylinder-like completion, whereas the real dough ball may deviate from that regular shape. For this reason, some discrepancy between estimated and expected volume is natural even if the reference were measured more directly. Nevertheless, the fact that a large portion of the samples reconstructed with the geometric methods falls within the expected interval suggests that these approaches are able to capture the overall shape of the dough ball effectively and to provide estimates that remain coherent with the nominal product characteristics. In this sense, the results represent a substantial improvement over the simpler depth-gap baseline and provide encouraging evidence for the practical applicability of the proposed geometric-reconstruction approach in a real industrial setting. A more precise calibration of the expected reference volume, for example through direct measurements on representative samples, could further improve the reliability of the final comparison; this point will be discussed again in Chapter 6.

5.2.2 Qualitative results

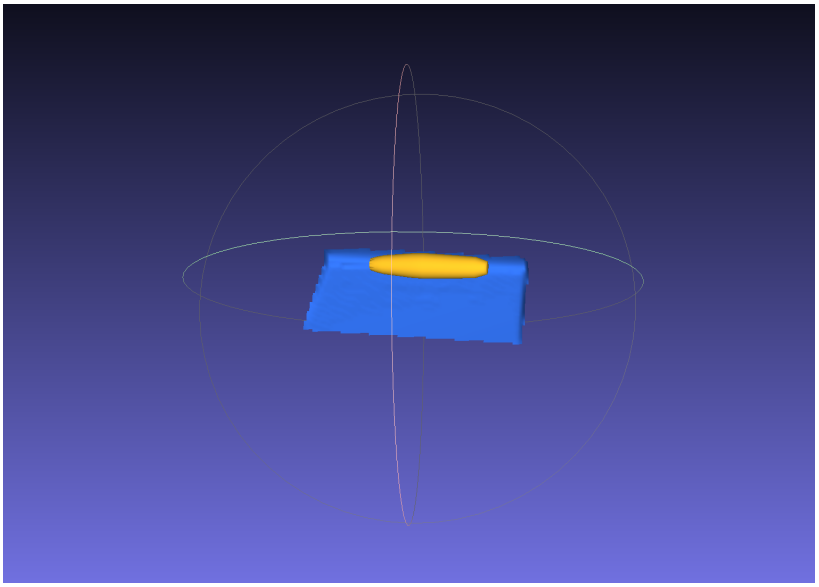
Figures 5.9 and 5.10 report a representative qualitative example of the three reconstruction strategies applied to the same dough-ball sample. The first figure shows the RGB input and the corresponding depth map, while the second reports the reconstructed geometries produced by the rotated convex-hull, rotated alpha-shape, and depth-gap estimators. These visual results complement the quantitative analysis by showing that the depth-gap method tends to provide a more compact reconstruction, whereas the rotational methods recover a more complete geometry of the dough ball. In particular, the convex-hull reconstruction appears visually more coherent with the expected global shape of the product, consistently with the quantitative results discussed above.



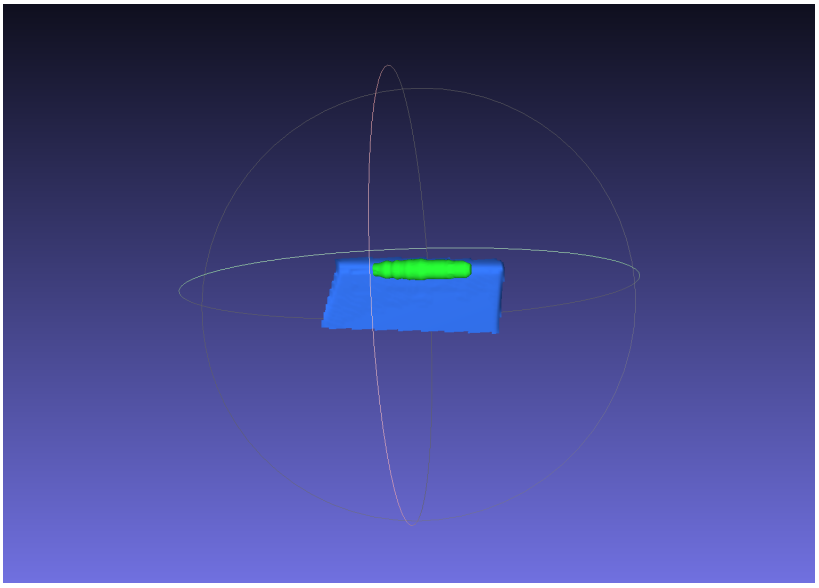
(a) RGB image of dough ball

(b) Depth map of dough ball

Figure 5.9: Representative RGB and depth inputs for the dough-ball sample used in the qualitative comparison.



(a) Rotated convex-hull reconstruction



(b) Rotated alpha-shape reconstruction

(c) Depth-gap reconstruction

Figure 5.10: Representative qualitative comparison of the three volume estimators on the same dough-ball sample.

5.3 Site 2: bread quality assessment

5.3.1 Quantitative results

The bread-quality branch produces image-level anomaly scores together with localization-oriented visual outputs. Under the adopted evaluation protocol, in which the normal test subset is contrasted with the nine external defective images reserved for testing, the anomaly-detection model achieved a near-perfect **image-level AUROC of 0.999958** and a **image F1-score of 0.7368**, where the latter is the harmonic mean of precision and recall, with formula (2.6).[24, 26] This second metric must be interpreted with care, since the anomalous reference set is small and externally sourced, and therefore does not constitute a consolidated industrial benchmark. Nevertheless, the obtained values provide a strong preliminary indication that the learned normality model is able to separate the collected *good* bread distribution from clearly non-conforming products. In practical terms, the model correctly predicts **7 out of the 9** anomalous samples as defective, while producing only **3 false positives** on the normal test set, which contains **40,253** samples.

Test metric	DataLoader 0
image_AUROC	0.9999585747718811
image_F1Score	0.7368420958518982

Figure 5.11: Results of the anomaly-detection branch on the test set.

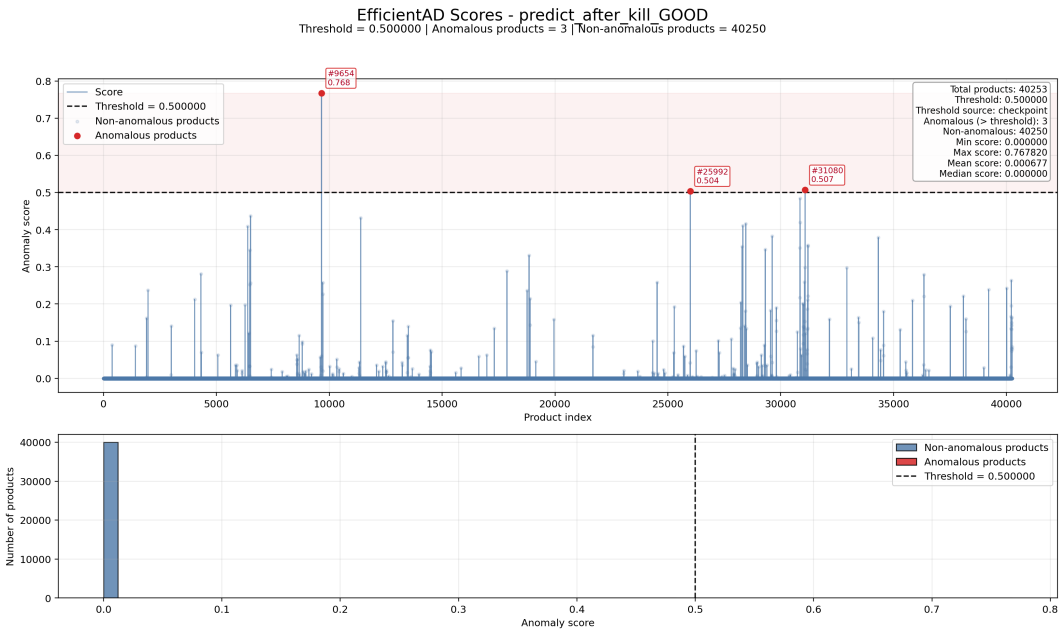


Figure 5.12: Anomaly-score trend obtained on representative *good* bread samples. The scores remain in the low-response regime expected for normal products.

The score values are also consistent with this interpretation. On the inspected *good* set, the anomaly scores remain extremely low, with an average value of approximately **0.00068**, and more than **99.3%** of the samples receiving an exact zero response. By contrast, the nine external defective samples produce substantially higher scores, ranging from approximately

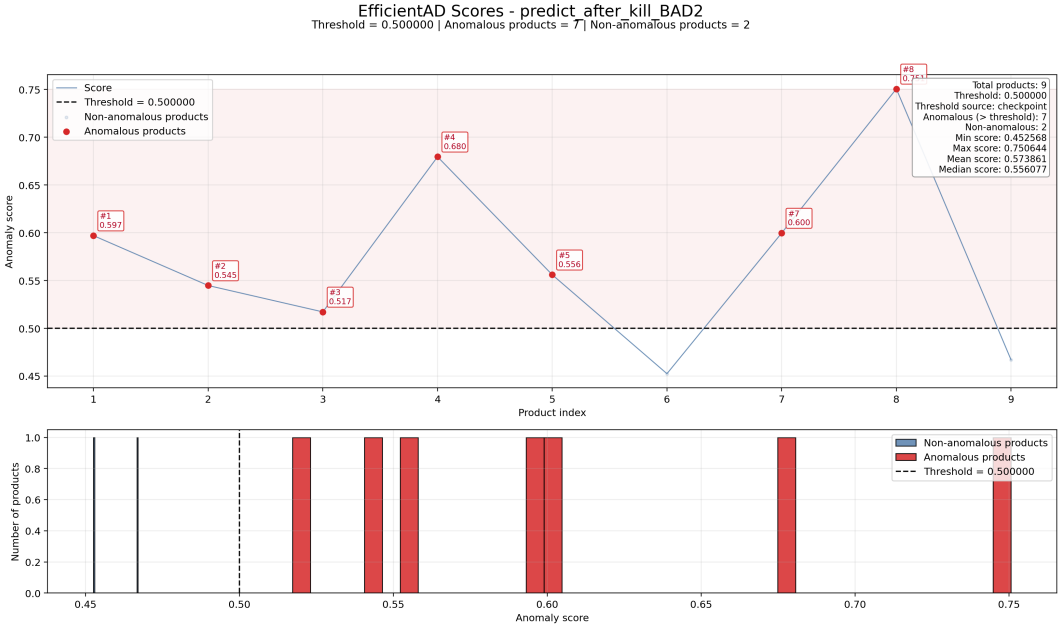


Figure 5.13: Anomaly-score trend obtained on external defective samples. Compared with the *good* set, these clearly non-conforming loaves produce substantially higher anomaly responses.

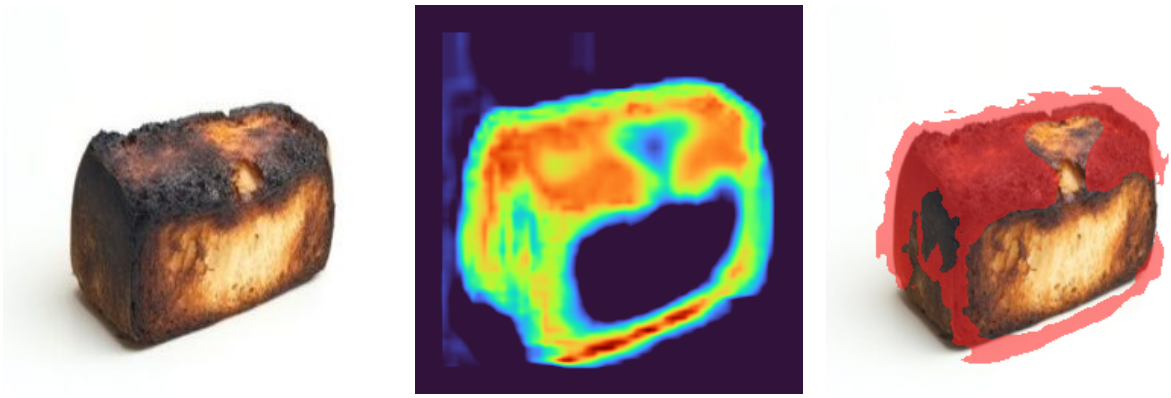
0.453 to 0.751, with an average of about 0.574. Figures 5.12 and 5.13 report representative score plots for normal bread samples and for external defective samples, respectively. Using the **default operating threshold of 0.5**, the average anomalous response remains above the decision boundary, while the normal samples stay far below it. This **large score separation** confirms that the model has learned a stable representation of the normal bread appearance within the monitored industrial scenario and reacts coherently when strong deviations are introduced.

From an operational perspective, this separation also leaves room for **threshold tuning** in order to balance sensitivity and specificity according to the actual monitoring needs of the production line. In particular, the two missed anomalous samples are associated with scores of approximately **0.453** and **0.467**, both close to the decision threshold, which suggests that a slight threshold reduction could improve recall. Similarly, two of the three false positives occur just above the threshold, with scores of approximately **0.504** and **0.507**, whereas the remaining one reaches about **0.768**. This indicates that part of the residual error is threshold-related, while a smaller fraction is more likely due to visual irregularities within otherwise normal bread samples. A broader normal dataset or additional augmentation strategies could therefore further improve robustness without changing the overall structure of the pipeline.

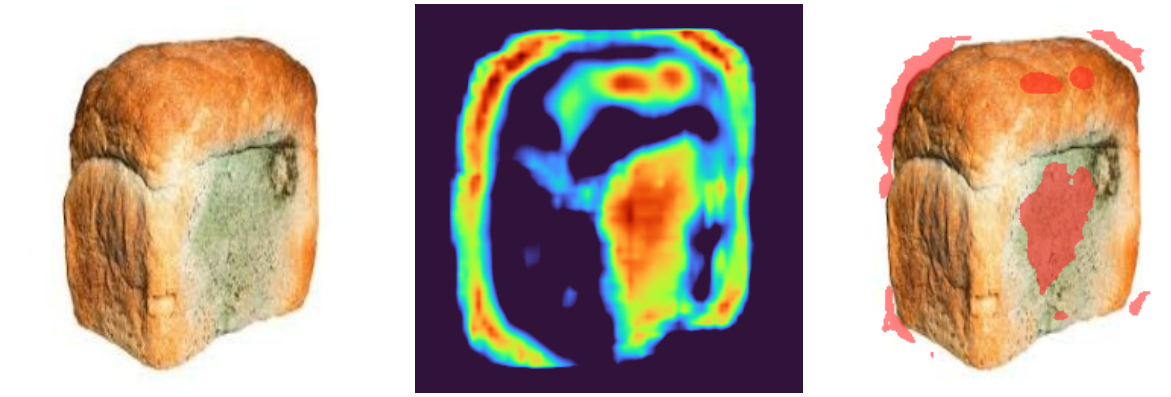
5.3.2 Qualitative results

Representative qualitative outputs are shown in Figures 5.14, 5.15, and 5.16. In the detailed visualization of Figure 5.14, the left image shows the input loaf crop, the middle image reports the predicted anomaly heatmap, and the right image shows the mask overlay used to highlight the regions that most strongly contributed to the final score. Figure 5.15 reports the same kind of evidence through the compact visualization format directly exported by the inference pipeline for anomalous bread samples, whereas Figure 5.16 shows representative *good* bread examples

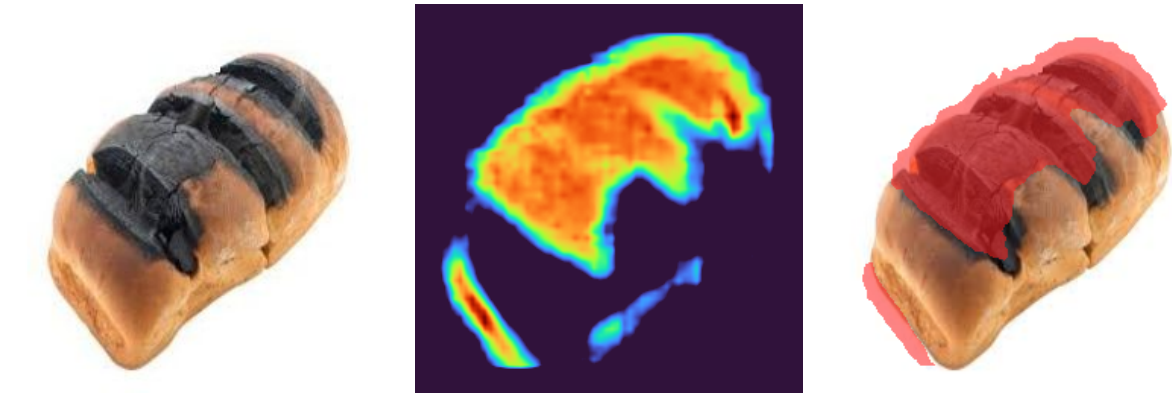
under the same visualization scheme.



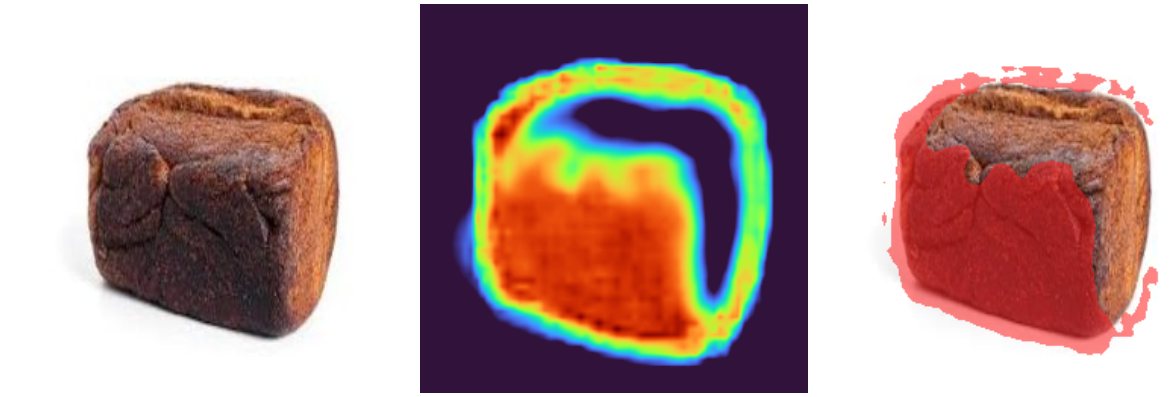
(a) Input example 1 **(b)** Heatmap example 1 **(c)** Mask overlay example 1



(d) Input example 2 **(e)** Heatmap example 2 **(f)** Mask overlay example 2



(g) Input example 3 **(h)** Heatmap example 3 **(i)** Mask overlay example 3



(j) Input example 4 **(k)** Heatmap example 4 **(l)** Mask overlay example 4

Figure 5.14: Representative qualitative outputs of the EfficientAD branch on four anomalous bread examples from the external reference set. For each case, the anomaly heatmap and mask overlay highlight the regions that most strongly deviate from the learned normal appearance.

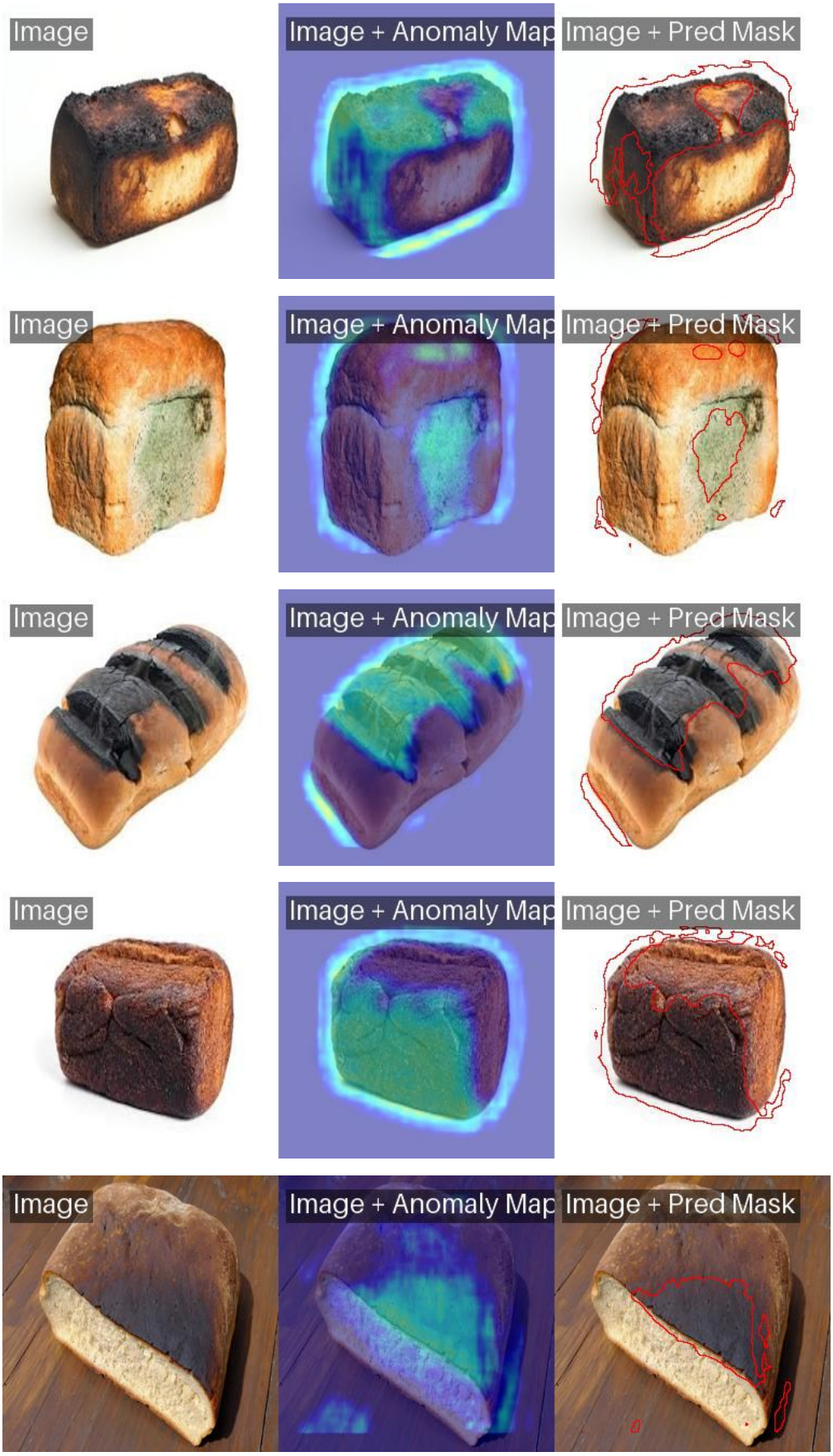


Figure 5.15: Compact qualitative visualizations exported by the inference pipeline for five anomalous bread examples. Each panel combines the input crop, anomaly heatmap, and predicted anomaly mask into a single view for faster qualitative inspection of the most anomalous regions.

Figure 5.16: Compact qualitative visualizations exported by the inference pipeline for five representative *good* bread samples. In these cases, the heatmaps remain very flat and the predicted anomaly masks are negligible, consistently with the low anomaly scores observed in the quantitative analysis.

Discussion

Taken together, the quantitative and qualitative results indicate that the anomaly-detection branch behaves consistently with the intended one-class formulation. Normal bread samples remain concentrated in a very low-score regime, whereas clearly non-conforming loaves generate substantially higher responses. In this sense, the obtained AUROC and the score distributions reported above confirm that the model has learned a stable representation of the normal baked-bread appearance within the monitored industrial scenario.

The qualitative outputs further support this interpretation. In the anomalous examples shown in Figures 5.14 and 5.15, the heatmaps and mask overlays highlight the visually irregular regions that motivate the final anomaly score, such as burnt or mouldy areas. By contrast, the representative *good* samples shown in Figure 5.16 exhibit very flat heatmaps and negligible anomaly masks, coherently reflecting the low-score regime observed on normal bread. This makes the results easier to inspect and potentially more useful for operator-oriented monitoring, since the system not only assigns a scalar anomaly response but also provides a spatial indication of the most suspicious image regions.

At the same time, the present evaluation should still be considered preliminary. The anomalous samples used in this thesis were collected from external online sources and therefore differ from the real industrial domain in terms of product appearance, acquisition conditions, and defect distribution. For this reason, the current results should not yet be interpreted as a definitive estimate of production-line anomaly-detection performance. Rather, they show that the proposed pipeline has learned the normal bread appearance effectively and reacts coherently when deviations are introduced even if they are not from the same data distribution. Even with these limitations, the current results are already very encouraging. A broader evaluation on representative defective samples collected directly from the production environment would be the natural next step to consolidate these findings.

Chapter 6

Conclusions and Future Work

This thesis has presented a multimodal quality-inspection framework for the bread-production process. The proposed system combines motion-triggered acquisition, automatic labelling, DEIMv2-based object detection, volumetric reconstruction of dough balls from RGB-D data, and anomaly detection for baked products. The implementation was developed in close interaction with the operators of the SMACT [10] Competence Center, so that the final solution could remain aligned with the practical constraints of the industrial environment as well as with the actual monitoring needs of the production line, while requiring no modifications to the existing production setup and operating without the need for additional infrastructure.

The results obtained in the two monitored sites highlight the effectiveness of the proposed framework. At Site 1, the volumetric analysis showed that the rotation-based methods clearly outperform the simpler depth-gap baseline, with the rotated convex-hull estimator providing the best agreement with the expected reference interval. The experiments also showed that, although SAM can provide very accurate segmentations, a lighter color-clustering plus background-subtraction strategy represents a more suitable and robust compromise for the intended workflow, since it preserves good reconstruction quality while remaining computationally more efficient. At Site 2, the anomaly-detection branch achieved a near-perfect image-level AUROC of 0.999958 under the adopted evaluation protocol, while also showing consistently low anomaly scores on normal bread samples and meaningful responses on external anomalous examples, with anomaly maps that correctly localized the most relevant irregular regions.

Overall, the quantitative and qualitative results demonstrate that the framework is capable of supporting quality monitoring in a real industrial environment, across both the dough-preparation and baked-product stages.

Future work may focus on improving both the accuracy and the industrial robustness of the proposed framework. For Site 1, a more reliable reference for volume evaluation could be obtained through direct volumetric measurements on representative dough samples, thus reducing the uncertainty associated with the current indirect estimate based on nominal weight and density. For Site 2, the collection of a broader and more representative set of defective bread samples would make it possible to validate the anomaly-detection branch more extensively and to better characterize its behavior under realistic fault conditions. Additional developments may

also include even tighter real-time integration with the production process, so as to provide immediate feedback to operators.

References

[1] Shihua Huang, Yongjie Hou, Longfei Liu, Xuanlong Yu, and Xi Shen. Real-Time Object Detection Meets DINOv3, Jan. 2026. arXiv:2509.20787 [cs].

[2] Yunjie Tian, Qixiang Ye, and David Doermann. YOLOv12: Attention-Centric Real-Time Object Detectors, Feb. 2025. arXiv:2502.12524 [cs].

[3] Tommie Kerssies, Niccolo Cavagnero, Alexander Hermans, Narges Norouzi, Giuseppe Averta, Bastian Leibe, Gijs Dubbelman, and Daan de Geus. Your ViT is Secretly an Image Segmentation Model, Mar. 2025. arXiv:2503.19108 [cs].

[4] Zhuo Li, Yuhao Yan, Xiangheng Wang, Yifei Ge, and Lin Meng. A survey of deep learning for industrial visual anomaly detection. *Artificial Intelligence Review*, 58:279, 2025.

[5] Kilian Batzner, Lars Heckler, and Rebecca Konig. EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies, Feb. 2024. arXiv:2303.14535 [cs].

[6] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. DETRs Beat YOLOs on Real-time Object Detection, Apr. 2024. arXiv:2304.08069 [cs].

[7] NVIDIA Corporation. Jetson Nano 2GB Developer Kit - Get Started. [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit.

[8] Microsoft. Azure Kinect DK hardware specifications. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/azure/kinect-dk/hardware-specification.

[9] FlexSight. FlexSight, 2026. [Online]. Available: https://flexsight.eu/.

[10] SMACT. SMACT Competence Center, 2026. [Online]. Available: https://www.smact.cc/?lang=en.

[11] Luca Medeiros. luca-medeiros/lang-segment-anything, Mar. 2026. [Online]. Available: https://github.com/luca-medeiros/lang-segment-anything.

[12] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A Deep Learning Library for Anomaly Detection, Feb. 2022.

[13] systemd. systemd.service, 2026. [Online]. Available: https://www.freedesktop.org/software/systemd/man/253/systemd.service.html.

[14] Dinh-Cuong Hoang, Phan Xuan Tan, Anh-Nhat Nguyen, Son-Anh Bui, Ta Huu Anh Duong, Tuan-Minh Huynh, Duc-Manh Nguyen, Viet-Anh Trinh, Quang-Huy Ha, Nguyen Dinh Bao Long, Duc-Thanh Tran, Xuan-Tung Dinh, Van-Hiep Duong, and Tran Thi Thuy Trang. Image-based anomaly detection in low-light industrial environments with feature enhancement. *Results in Engineering*, 25:104309, 2025.

[15] Ramona Kühlechner. Object detection survey for industrial applications with focus on quality control. *Production Engineering*, 19:1271–1291, 2025.

[16] FlexSight. About FlexSight, 2026. [Online]. Available: https://flexsight.eu/about/.

[17] Universita di Padova. SMACT Competence center, June 2017. [Online]. Available: https://www.unipd.it/smact-competence-center.

[18] IBM, Rina Diane Caballar, and Cole Stryker. What Is Computer Vision? | IBM. [Online]. Available: https://www.ibm.com/think/topics/computer-vision#691946467.

[19] IBM, Jacob Murel Ph.D., and Eda Kavlakoglu. What is Object Detection? | IBM, Jan. 2024. [Online]. Available: https://www.ibm.com/think/topics/object-detection.

[20] IBM, Dave Bergmann, and Cole Stryker. What Is Instance Segmentation? | IBM, Oct. 2023. [Online]. Available: https://www.ibm.com/think/topics/instance-segmentation.

[21] Himanshu Mittal, Avinash Chandra Pandey, Mukesh Saraswat, Sumit Kumar, Raju Pal, and Garv Modwel. A comprehensive survey of image segmentation: Clustering methods, performance parameters, and benchmark datasets. *Multimedia Tools and Applications*, 81(24):35001–35026, 2022.

[22] OpenCV. How to use background subtraction methods, 2026. [Online]. Available: https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html.

[23] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[24] scikit-learn developers. 3.4. metrics and scoring: quantifying the quality of predictions, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html.

[25] Detectron2 Contributors. detectron2.evaluation.coco_evaluation, 2020. [Online]. Available: https://detectron2.readthedocs.io/en/v0.6/_modules/detectron2/evaluation/coco_evaluation.html.

[26] Dominik Müller, Ivan Soto-Rey, and Frank Kramer. Towards a guideline for evaluation metrics in medical image segmentation. *Journal of Imaging*, 11(6):144, May 2022.

[27] TensorFlow. tf.keras.metrics.meaniou, 2026. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanIoU.

[28] scikit-learn developers. roc_auc_score, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.

[29] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9404–9413, June 2019.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[31] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NeurIPS 2012)*, pages 1097–1105, 2012.

[34] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929 [cs].

[36] Maxime Oquab, Timothee Darcet, Theo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning Robust Visual Features without Supervision, Feb. 2024. arXiv:2304.07193 [cs].

[37] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment Anything, Apr. 2023. arXiv:2304.02643 [cs].

[38] scikit-learn developers. train_test_split — scikit-learn documentation, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[39] Lutz Prechelt. Early stopping — but when? In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2012.

[40] Moe Long. NVIDIA Jetson Nano 2GB Developer Kit Review. [Online]. Available: https://www.electromaker.io/blog/article/nvidia-jetson-nano-2gb-developer-kit-review.

[41] Microsoft. About Azure Kinect DK. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/azure/kinect-dk/about-azure-kinect-dk.

[42] Microsoft. Azure-Kinect-Sensor-SDK github, Feb. 2026. [Online]. Available: https://github.com/microsoft/Azure-Kinect-Sensor-SDK.

[43] Ibai Gorordo Fernandez. pyKinectAzure, Feb. 2026. [Online]. Available: https://github.com/ibaiGorordo/pyKinectAzure.

[44] Raspberry Pi Ltd. Raspberry pi 5, 2026. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/.

[45] Raspberry Pi Ltd. Raspberry pi camera documentation, 2026. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[46] Raspberry Pi Ltd. Raspberry pi hardware documentation, 2026. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[47] Raspberry Pi Ltd. Camera module 3, 2026. [Online]. Available: https://www.raspberrypi.com/products/camera-module-3/.

[48] Raspberry Pi Ltd. The picamera2 library, 2026. [Online]. Available: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf.

[49] Raspberry Pi Ltd. Picamera2, 2026. [Online]. Available: https://github.com/raspberrypi/picamera2.

[50] Labby0811. Labby0811/thesis, 2026. [Online]. Available: https://github.com/Labby0811/thesis.

[51] Labby0811. Labby0811/kinect_acquisition, 2026. [Online]. Available: https://github.com/Labby0811/Kinect_acquisition.