



Finanziato
dall'Unione europea
NextGenerationEU



Ministero
dell'Università
e della Ricerca



Italiadomani
PACOLLAZZIPAL
DI SPERANZA E RESILIENZA



UNIVERSITÀ
DEGLI STUDI
DI UDINE
HEC SUAT FUTURA

Corso di dottorato di ricerca in:
Informatica e Intelligenza Artificiale
in convenzione con Fondazione Bruno Kessler

Ciclo 38°

Titolo della tesi

**Visual Anomaly Detection for Industrial Environments with
Three-Dimensional Structural Complexity**

Dottorando
Enrico Marchi

Supervisore
Prof. Gian Luca Foresti

Anno 2026

INSTITUTE CONTACTS

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Università degli Studi di Udine

Via delle Scienze, 206

33100 Udine — Italia

+39 0432 558400

<https://www.dmif.uniud.it/>

Acknowledgements

Special thanks go to the AIMV business unit of beanTech and to the Vicos laboratory, directed by Professor Danijel Skočaj of the University of Ljubljana, for their fruitful collaboration. Finally, sincere thanks are extended to Łukasz Leon Grzywacz (<https://www.seymouria.pl/>) for providing the 3D LEGO-based models used in this work.

Abstract

The aim of this thesis is to explore visual anomaly detection solutions for industrial environments, with particular emphasis on quality control scenarios characterized by a strong intrinsic 3D nature. Realistic systems are affected by challenges arising from the unconstrained degrees of freedom in space, which demand robust and reliable methods capable of handling complex 3D variability in real-world industrial settings.

The thesis is structured as follows: it first reviews key concepts in artificial intelligence and computer vision, which provide the background for the proposed visual quality control systems. Then, two complementary scenarios based on realistic industrial settings are presented, describing the proposed architectures, providing qualitative and quantitative results, and comparing them with other state-of-the-art methods.

Contents

1	Introduction	1
1.1	General Context and Motivation	1
2	Artificial Intelligence Background	5
2.1	Symbolic AI	6
2.2	Machine Learning	6
2.3	Statistical Learning Theory	6
2.4	Learning Paradigms	8
2.5	Anomaly Detection	9
2.6	Bias and Variance	9
2.7	Maximum a Posteriori and Maximum Likelihood	11
2.8	Model Selection	11
2.9	Evaluation Metrics	12
2.10	Deep Learning	15
2.10.1	Perceptron	15
2.10.2	Feedforward Networks	18
2.10.3	Modern Networks	22
2.10.4	Transfer Learning	26
2.10.5	Augmentation	26
3	Computer Vision	29
3.1	Recognition Tasks	29
3.2	Metrics	30
3.3	Camera model	32
3.4	3D reconstruction	35
4	Problem Definition	43
5	Proposed Systems	47
5.1	Overlapping Screws Detection in 3D and Color Images	47
5.1.1	The Plant	50
5.1.2	Datasets	53
5.1.3	Experimental Settings	54
5.1.4	Neural Network Models	56
5.2	Unsupervised pose-agnostic visual anomaly detection in realistic industrial scenes	57

5.2.1	Gaussian Splatting	59
5.2.2	Pose Estimation and Anomaly Detection	61
5.2.3	Methodology	62
5.2.4	Implementation	64
5.2.5	Dataset	66
6	Results	69
6.1	Overlapping Screws Detection in 3D and Color Images	69
6.1.1	Performance on Real Dataset	70
6.1.2	Performance on Synthetic Datasets	70
6.1.3	Performance on Test Data with Optimized Hyperparameters	71
6.2	Unsupervised pose-agnostic visual anomaly detection in realistic industrial scenes	74
6.2.1	Metrics	74
6.2.2	Quantitative Evaluation	74
6.2.3	Qualitative Evaluation	76
6.2.4	Ablation Study	76
7	Conclusions	85

1

Introduction

1.1 General Context and Motivation

In recent years, the industrial world has been shaped by the concepts of Industry 4.0 and Industry 5.0.

Industry 4.0 builds on previous paradigms based on traditional automation and focuses on creating automatic systems that integrate electro-mechanical automation with digital technologies. The goal is to develop “smart” factories in which systems are interconnected and the entire production process can be coordinated, monitored, and optimized. This is achieved by collecting data through sensors and analyzing it in real time using efficient digital solutions and advanced analytics.

While Industry 4.0 primarily aims at enhancing efficiency and productivity, Industry 5.0 emphasizes sustainability and the central role of human operators within industrial ecosystems. Industry is called to invest in the human dimension by continuously developing workers’ skills and protecting their well-being; in this way, the synergy between empowered operators and cutting-edge technologies can have a positive impact on production performance and quality. Furthermore, technological progress should be aligned with the responsible use of resources and the reduction of environmental impact, with the broader goal of contributing to climate change mitigation and supporting long-term social stability.

Both paradigms promote the adoption of advanced technologies, such as artificial intelligence, which is intended to support rather than replace human decision-making, robotics solutions such as collaborative robots (cobots) designed to share tasks and workspaces with operators, and digital tools that contribute to monitoring and managing workspaces, with the goal of increasing safety, productivity, and sustainability.

In quality control AI models are employed to analyze sensor and image data to detect defects and reduce waste in materials, energy, and time, while cobots can take over repetitive or hazardous inspection tasks in environments where machines in motion and complex equipment expose operators to risk, and digital platforms can provide real-time feedback on process conditions, enabling safer and more reliable inspection procedures that remain centered on human supervision and expertise.

In this context, artificial intelligence applied to computer vision plays a fundamental role. In the last decade, a large amount of research has focused on designing intelligent systems for visual tasks. These systems typically address core objectives such as classification, object detection, segmentation, and key-point estimation.

Classification is the task of assigning a global label to an image or to a region of interest, for example deciding whether a part is normal or defective; this label is then used to trigger an appropriate response, such as keeping the part on the production line or removing it. Object detection and segmentation extend this idea: in addition to assigning a label, they also locate where objects or defects are in the image. Detection does this by predicting bounding boxes around the instances of interest, while segmentation assigns a category to each pixel, trying to precisely match the shape of each object or defect. Finally, key-point estimation focuses on finding specific characteristic points on an object, which can then be used, for example, to estimate its pose and support later inspection or manipulation steps.

The mentioned tasks are nowadays a mature technology and are already successfully deployed in many real industrial scenarios. However, despite this considerable progress, several challenges remain when these methods are applied to production lines. Industrial environments are often complex and variable. Backgrounds, even when carefully optimized during the initial setup, tend to degrade over time due to changing lighting conditions, dust accumulation, or modifications in the production area. This variability directly affects the robustness of vision-based systems, which may start to generate more false alarms or miss subtle defects when the visual appearance of the scene drifts away from the initial conditions.

There are also structural difficulties, since parts and materials are not always easy to position or keep in a stable configuration. Components can move, tilt, overlap, or appear with different orientations on conveyors, introducing an additional source of variability that complicates automatic inspection which should remain effective even when the pose of the objects is not perfectly controlled.

The problem of data availability adds further complexity. Defects are rare by nature, and collecting extensive datasets of anomalous cases is difficult and often expensive. In many applications, only a few examples are available for each type of defect, and in some cases certain failure modes are not observed at all during development. This scarcity limits the training of supervised methods and reduces the coverage of potential failure modes.

In addition, building high-quality datasets requires manual labeling by experts, who must carefully identify and classify defects on a large number of images. This process is time-consuming and can introduce inconsistencies, especially when the distinction between acceptable variation and real defect is subtle.

For these reasons, attention has been directed towards models that can be deployed with minimal retraining or adapted quickly with limited supervision. Instead of collecting large new datasets every time something changes in the production line, recent approaches focus on techniques such as transfer learning and domain adaptation, where a model trained in one setting is reused and adjusted to new conditions with only a small amount of additional data.

In the same direction, few-shot and even zero-shot methods aim to recognize new classes or defect types starting from very few examples or without any labeled images

at all.

Another critical challenge is the speed of inspection. Anomaly detection must be performed in line with production rates so that the system can provide immediate feedback and trigger either automatic correction by robotic mechanisms or timely manual intervention by operators. Models must then be lightweight enough, yet still robust to the variability of real environments, so that they can achieve both accuracy and efficiency under such time constraints.

Finally, the geometric complexity of industrial components poses one of the most demanding challenges. Many parts have complex three-dimensional shapes, with holes, edges, curved areas, and very small details that can easily hide or cover defects. As a result, these defects can be very subtle and hard to detect with standard imaging techniques. In these cases, problems such as self-occlusion, shadows, reflections, and changes in appearance when looking from different viewing angle become particularly relevant and can compromise the visibility of critical areas.

Complex geometries therefore limit the effectiveness of conventional 2D vision systems and often require more advanced solutions, such as 3D sensing or multi-view setups which must be carefully designed to ensure sufficient coverage of all relevant surfaces, while keeping acquisition time and system complexity within acceptable limits.

For all the reasons stated above, quality control in modern industry remains an open research problem.

This thesis, in particular, focuses on industrial visual anomaly detection, with a particular emphasis on realistic scenarios where three-dimensionality plays a pivotal role in the intrinsic complexity of the task.

2

Artificial Intelligence Background

In the mid-20th century, scientists introduced the concept of artificial intelligence (AI), defined as the simulation of human intelligence processes by computer systems. These processes include learning, reasoning, problem-solving, and decision-making, as well as the ability to interact with the environment and exchange information with it. AI research has been mainly driven by two paradigms: symbolic AI and machine learning. This chapter is based on the works [1, 2, 3, 4, 5, 6].

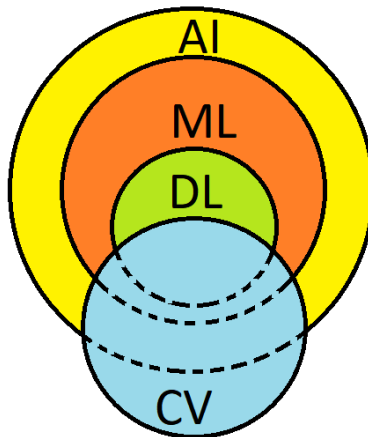


Figure 2.1: Schematic view of the AI field, highlighting Machine Learning (ML), Deep Learning (DL), and Computer Vision (CV).

2.1 Symbolic AI

The first one aims to develop algorithms that model human intelligence through explicit representations of knowledge. It relies on high-level symbolic descriptions of problems, using strict logical rules and formal patterns that are understandable by humans. In practice, it tries to mimic the logical structure of human reasoning. The main advantage of this approach is its transparency and explainability, since the correspondence with human reasoning is direct. However, its main limitation lies in handling uncertainty, variability, and large amounts of unstructured data.

2.2 Machine Learning

On the other hand, machine learning (ML) avoids explicit logical representations and relies on mathematical models that learn from data and generalize to unseen cases, performing tasks without being explicitly programmed. The advantages are evident: these models can process complex inputs with high variability and are able to deal with intrinsic uncertainty. A typical example is given by digital images, which can be represented as an $H \times W$ matrix of pixels, repeated for three channels in the case of RGB. The same scene can be captured by many different images; for instance, a slight change in camera position produces significant differences at the pixel level, especially in terms of pixel arrangement. Even with a fixed position, changes in environmental light or the presence of dust can alter many pixels without being perceptible to the human eye. Nevertheless, the global idea, concept, and information conveyed by these images remain the same. However, these systems build their own internal representations of knowledge with limited human control, which makes them difficult to interpret. For this reason, machine learning models are often considered black boxes: the input and the output are known, but the process that links them remains opaque. Another drawback is that they usually require large amounts of data, which is often costly and difficult to collect and manage. Finally, while these models can learn a dataset very effectively, they often struggle to generalize beyond its scope.

2.3 Statistical Learning Theory

According to Vapnik, machine learning is rooted in statistical learning theory, which provides a formal definition of the learning process. According to this framework, learning from examples can be described by three main elements. The first is a generator that produces input vectors $x \in \mathbb{R}^n$, sampled independently from an unknown probability distribution $P(x)$. The second is a supervisor who associates each input vector with an output value y , following a conditional distribution $P(y|x)$, also fixed but unknown. The third element is a learning machine that can realize a family of functions $f(x, \theta)$, where $\theta \in \Lambda$ represents the parameters of the model. The objective of learning is to identify, within this family, the function that provides the best approximation of the supervisor's response. In practice, this selection is carried out using a training set of ℓ independent and identically distributed observations

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\} \sim P(x, y), \quad (2.1)$$

drawn from the joint distribution of inputs and outputs. To approximate the supervisor's response, the learning process requires a measure of the discrepancy between the predicted value and the true outcome. This discrepancy is expressed through a loss function $L(y, f(x, \theta))$, which compares the output y provided by the supervisor with the output $f(x, \theta)$ generated by the learning machine for the same input x . The overall quality of a function can then be assessed by considering the expected value of this loss with respect to the joint distribution of the data. This leads to the definition of the risk

$$R(\theta) = \int L(y, f(x, \theta)) dP(x, y). \quad (2.2)$$

The learning task is therefore equivalent to finding, among the family of candidate functions $\{f(x, \theta), \theta \in \Lambda\}$, the one that minimizes the risk $R(\theta)$. The difficulty of this problem is that the distribution $P(x, y)$ is unknown, so optimization cannot be performed directly on it. The only source of information available to the learner is the finite set of independent samples introduced above. Therefore, a common strategy is to replace the unknown distribution with the empirical distribution induced by the training data. In this case, the integral that defines the risk is approximated by the average of the losses computed on the available samples. This quantity is called the empirical risk and is defined as

$$\hat{R}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} L(y_i, f(x_i, \theta)). \quad (2.3)$$

The principle of empirical risk minimization (ERM) states that the learning machine should select the function $f(x, \theta)$ that minimizes $\hat{R}(\theta)$. In other words, the parameters are adjusted in order to achieve the best performance on the training set. This provides a practical way to approximate the optimal solution, since it relies only on observed data rather than on the unknown probability distribution. The function f is also called a hypothesis or a model, and these terms are used interchangeably. Within statistical learning, two fundamental problems are usually considered: classification and regression. For simplicity, the discussion here is limited to the binary classification case, where the output variable, also called label, can take only two possible values, $y \in \{-1, +1\}$. The objective is to teach a classifier to assign each input vector x to its proper class. During train the loss function should penalizes incorrect predictions. A common choice is

$$L(y, f(x, \theta)) = H(-y f(x, \theta)), \quad (2.4)$$

where $H(z)$ is equal to 1 if $z \geq 0$ and 0 otherwise. This formulation enforces that the loss is zero when the prediction $f(x, \theta)$ assigns the input to the correct class, and one when the prediction is wrong. Learning in this setting therefore amounts to minimizing the average number of classification errors over the available data. In regression, the output variable is real-valued, $y \in \mathbb{R}$, and the goal is to learn a function $f(x, \theta)$ that predicts y from x with small error. A standard choice for measuring the discrepancy is the squared loss

$$L(y, f(x, \theta)) = (y - f(x, \theta))^2. \quad (2.5)$$

With this loss, the function that minimizes the expected risk is the regression function,

defined as the conditional expectation

$$f^*(x) = \mathbb{E}[y \mid x] = \int y dP(y \mid x). \quad (2.6)$$

The learning problem is therefore to approximate $f^*(x)$ by minimizing the empirical risk,

$$\hat{R}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} (y_i - f(x_i, \theta))^2. \quad (2.7)$$

2.4 Learning Paradigms

In machine learning, there are several paradigms of learning, which differ according to the type of training dataset available to the learner. In all these paradigms, the dataset is split between the training set and the test set, which are used to train and evaluate a model respectively. The two sets are disjoint in their inputs to ensure fairness in the evaluation.

Supervised learning. In this setting, the dataset is composed of pairs of input and output and the objective is to learn a function that maps each input to its corresponding output.

Unsupervised learning. In this case, the learner observes only the inputs, without any explicit output or label. The goal is to discover useful patterns or structures in the data, such as grouping similar inputs into clusters.

Reinforcement learning. Here the learner interacts with an environment and receives feedback in the form of rewards or penalties. The task is to discover a strategy for choosing actions that maximizes the cumulative reward over time.

Semi-supervised learning. In this paradigm the learner is provided with a small set of labeled examples together with a much larger collection of unlabeled inputs. The objective is to exploit the information contained in the unlabeled data, guided by the few available labels, in order to improve the quality of the learned model. This approach has become increasingly common, since in many real scenarios labeling is expensive or time-consuming, while unlabeled data are abundant.

Self-supervised learning. In this setting, the supervision signal is automatically derived from the data itself. The learner solves a pretext task, such as predicting missing parts of the input, and in doing so acquires useful internal representations that can later be used for downstream tasks.

2.5 Anomaly Detection

Anomaly detection is the problem of finding patterns in the data that do not follow the expected behavior. A dataset for anomaly detection is usually partitioned into normal and anomalous data, with the latter class also referred to as outliers. Different paradigms of anomaly detection resemble those commonly found in machine learning.

Supervised anomaly detection. This paradigm is characterized by the presence of both anomalous and normal instances in the dataset, and it can be reduced to a binary classification problem, where the labels correspond to the anomaly and normal classes. In practice, however, this approach faces several challenges. As the word "anomalous" suggests, outliers are rare, otherwise they would be considered normal. This makes data collection difficult, since the time required to gather a sufficient number of anomalies can be large. Moreover, datasets are typically unbalanced toward the normal class, making it difficult to learn the anomalous class, as the learning process rarely encounters anomalies. Finally, anomalies themselves can be substantially different from each other, which makes it infeasible to extract common patterns among them. One way to overcome this hindrance is to augment the dataset with synthesized anomalies.

Semi-supervised anomaly detection. This is a common scenario in which the training set is assumed to contain only normal instances, while the test set may include both normal and anomalous values. The model is trained to capture the patterns of the normal class, with the goal of constructing a boundary around its region in the feature space. Any instance that falls outside this boundary is then considered anomalous.

Unsupervised anomaly detection. In this case, there is no distinction between training and test data. The entire dataset is directly used during inference to identify anomalous elements, under the assumption that anomalies occur much less frequently than normal instances.

2.6 Bias and Variance

Learnable functions belong to a space, called the hypothesis space, which can be restricted or enlarged with different consequences for the learning process. An hypothesis with space too small can lead to the phenomenon called **underfitting**, which means that the process has failed to capture the relevant patterns of the data. Opposite to underfitting there is **overfitting**, which means that the process has learned a tight function around the training data, but fails to generalize when seeing new inputs from the test set. Usually, underfitting occurs with simple and weak models, while overfitting is more common with highly complex ones. Visual examples are shown in Figure 2.2. To discuss these aspects more clearly, it is useful to introduce the concepts of bias and variance.

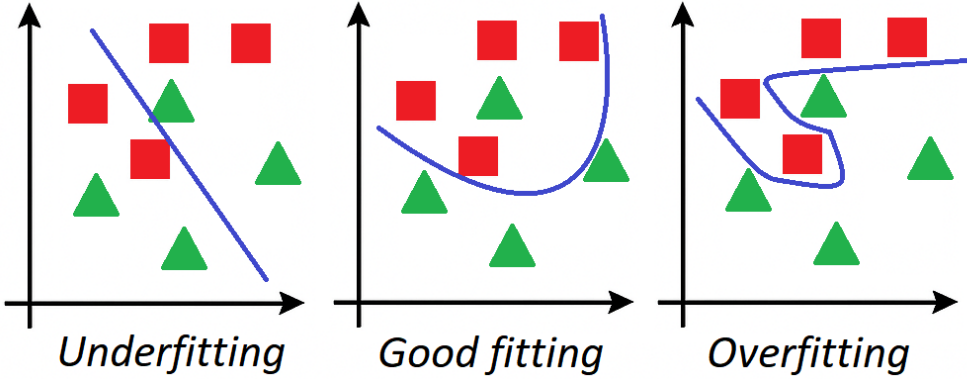


Figure 2.2: Visual examples of underfitting, good fitting and overfitting.

Bias Given a statistical model parametrized by θ , it gives rise to a distribution $P(x | \theta)$, which generates a dataset $\{x_1, \dots, x_n\}$. An estimator is a function $\hat{\theta}(x_1, \dots, x_n)$, meaning that it provides an estimate of θ based on the observations $\{x_1, \dots, x_n\}$. The bias is defined as

$$\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta} - \theta] = \mathbb{E}[\hat{\theta}] - \theta,$$

where the expectation is taken with respect to the distribution $P(x | \theta)$. An estimator is said to be unbiased if the bias is equal to zero, which means that averaging the estimator over multiple datasets drawn from $P(x | \theta)$ returns the true parameter θ . On the contrary, if the bias is not zero, then the estimator is biased, and the interest lies in quantifying how much it deviates from the true parameter θ . If the hypothesis space is too restricted, this may introduce bias, since the true parameter θ could fall outside the admissible space. Conversely, a sufficiently large space should allow the model to capture the true parameter more accurately. Bias tries to capture the error that is introduced when choosing the learning method.

Variance The variance is defined as

$$\sigma_{\hat{\theta}_n} = \sqrt{\mathbb{E}[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^2]}.$$

In this case, the interest is in measuring how much the estimator deviates from its mean under the distribution $P(x | \theta)$. It reflects how much the estimation is influenced by the particular dataset, since it quantifies the variability of the estimator when computed on different train sets drawn from the same distribution. A larger hypothesis space generally increases the variance, because with many degrees of freedom the model may become too expressive, leading to multiple ways of fitting datasets generated by the same underlying distribution. This results in the so called bias-variance tradeoff, which means that usually, there is the necessity to choose between simple hypothesis spaces with strong bias and complex spaces with high variance. Practically, to measure if an hypothesis generalizes well enough, the model should be evaluated using a test set,

independent from the training set.

2.7 Maximum a Posteriori and Maximum Likelihood

A common way to look at optimization problems in machine learning is through the principles of maximum a posteriori and maximum likelihood estimation. Given a model parametrized with θ and a dataset X , the goal of maximum a posteriori principle is to find

$$\arg \max_{\theta} P(\theta | X).$$

$P(\theta | X)$ is called posterior probability because it represent the probability of the estimated model after observing the dataset. The posterior distribution is usually unknown then, by Bayes' rule, it is easier to rewrite the optimization problem as

$$\arg \max_{\theta} \frac{P(X | \theta) P(\theta)}{P(X)} = \arg \max_{\theta} P(X | \theta) P(\theta),$$

where the denominator has been removed because it is independent from θ . $P(\theta)$ is the prior and can be assumed to be independent from θ . In such a case, the problem can be rewrite as

$$\arg \max_{\theta} P(X | \theta),$$

leading to the formulation called maximum likelihood estimation. If it is assumed that the $x_i \in X$ are *i.i.d* , which means independent and identical distributed, then the likelihood can be expressed as

$$\arg \max_{\theta} P(X | \theta) = \arg \max_{\theta} \prod_{i=1}^n P(x_i | \theta).$$

By applying the logarithmic function, this expression can be reduced to

$$\arg \min_{\theta} \left(- \sum_{i=1}^n \log P(x_i | \theta) \right),$$

which is easier to compute, since it is written as sums and not as products.

2.8 Model Selection

Usually, many different models can be formulated to solve the same learning problem. These models can be intrinsically different because the solutions differ methodologically, or they may belong to the same methodological class and differ only in a few structural parameters. In the latter case, these parameters are called hyper-parameters; they differ substantially from the other parameters because they do not belong to the learning process, but are selected directly by the designer of the learning model. As already mentioned, models are validated by testing them against the test dataset. In the case of multiple models that differ only in their hyper-parameters, it would be unfair to select

the one that performs best on the test set. Instead, a separate dataset is required, called the **validation set**, whose purpose is to compare models with different hyper-parameters and select the one that performs best on it. After selecting the best model, it is then evaluated on the test set.

It may happen that the entire dataset is not large enough to be split into training, validation, and test sets. In such cases, the common procedure is to perform **k -fold cross-validation**. This consists of dividing the entire dataset into k folds and running an iterative process for k rounds. In each round, one fold is taken as the test set and the remaining folds as the training set. The model is trained and evaluated in every round, and finally the scores are averaged across the k results.

This procedure, however, does not take into account hyperparameter tuning. For this reason, it is often preferable to perform **nested cross-validation**. This is a nested procedure made of two loops, called the outer loop and the inner loop. First, the dataset is split into folds as in standard cross-validation: in each outer round, one fold is retained for testing and the remaining folds form the outer training set. The outer training set is then further split into new inner folds. At each round of the inner loop, one inner fold is used as the validation set, while the others are used for training. These inner rounds are used to train and evaluate models with different hyper-parameters. After completing the inner loop, the scores of the models with the same hyper-parameters are averaged, and the hyper-parameters that achieve the best average performance are selected. The model is then retrained on the entire outer training set with the selected hyper-parameters and finally evaluated on the outer test fold. After completing all outer rounds, the results are averaged to obtain the final performance estimate.

This procedure evaluates the overall performance of the model while properly accounting for hyperparameter tuning. However, it does not directly provide the model with the selected hyper-parameters for deployment in a real scenario. Indeed, to finally choose the best hyper-parameters, a new simple k -fold cross-validation can be run on the entire dataset. The models with different hyper-parameters are tested on the validation folds, their results are averaged, and the model with the best average performance is selected. Importantly, the reported performance should always be the one derived from nested cross-validation, since it provides the most reliable estimate of generalization ability.

The illustrated procedure, though exhaustive, lacks practicality, as it may take too long to evaluate all the different hyperparameter configurations. If the number of hyper-parameters is small, this can be done through grid search, which consists of testing all possible combinations. Otherwise, it is common to perform manual tuning, which relies on intuition and prior knowledge to guess the most promising hyperparameter values.

2.9 Evaluation Metrics

When dealing with classification, and consequentially with anomaly detection, the quality of the model is assessed considering its ability of correctly classifying the instances within their class respect an external judgment taken as truthful which is called **ground truth**. In binary classifications classes are called positive and negative class and every decision of the model can be classified in four categories, based on the truthfulness of the prediction. If the model predicts correctly for the positive class then the result is a

true positive TP ; similarly if it predicts correctly the negative class then the results is a **true negative** TN . If an element belonging to the negative class is classified as positive then the results is a **false positive** FP ; finally, if a positive element is classified as negative then the result is a **false negative** FN . These four categories allow the definition of **precision**, **recall** and **accuracy**, which are metrics defined as

$$\text{Precision} = \frac{TP}{TP + FP},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

Precision measures the ability of a classifier to avoid the wrong classification of negative elements as positive. High precision is therefore obtained when the number of FP is low. Recall, on the other hand, measures the ability to identify all positive elements without leaving any out; high recall is achieved when the number of FN is low. These metrics evaluate the performance of the system with respect to the positive class, which is usually the one of primary interest. However, it is also possible to calculate precision and recall with respect to the negative class by applying the same formulas and swapping the roles of positive and negative. In this case, the resulting metrics are called **negative predictive value** and **true negative rate**, respectively. Accuracy, on the other hand, considers both classes simultaneously and measures the proportion of correct predictions over the total number of predictions.

The F_1 score is the harmonic mean of precision and recall, and it is used as a summary metric that takes both into account simultaneously.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The metrics introduced so far require the model to make a *hard prediction*, meaning that the output must be a specific class label. In this case, the model assigns each instance directly to one class, for example positive or negative. However, many models naturally produce *soft predictions*, that is, continuous scores or probabilities indicating the degree of belonging to a class. Relying only on hard predictions prevents the evaluation of models in this more general setting, since the richer information contained in soft outputs is ignored. For this reason, the **AUROC** was introduced. It measures the performance of a model without requiring hard decisions, as it operates directly on the soft scores and does not depend on fixing a specific threshold to convert them into class labels. Before introducing AUROC it is necessary to define the **false positive rate**

$$\text{FPR} = \frac{FP}{FP + TN} = \frac{FP}{N}.$$

The false positive rate (FPR) measures the proportion of false positives with respect to the total number of elements belonging to the negative class. The **true positive rate** (TPR), instead, coincides with the already defined recall, as it measures the proportion

of true positives with respect to the total number of elements belonging to the positive class. Figure 2.3 shows all the introduced concepts for binary classification.

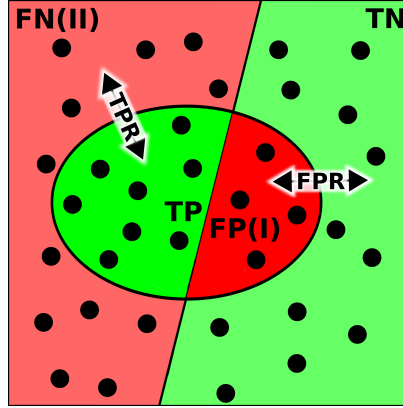


Figure 2.3: Binary classification schema.

When dealing with soft scores, both TPR and FPR are functions of the threshold chosen as the decision boundary for classification. The **ROC curve** is defined in the x - y plane, where the FPR is plotted on the x -axis and the TPR on the y -axis. Choosing a threshold value t , it is possible to compute the pair $(\text{FPR}(t), \text{TPR}(t))$, which corresponds to a point on the ROC curve. By varying t , the entire ROC curve can be drawn. An example is shown in Figure 2.4.

Finally, the **AUROC** is defined as the area under the ROC curve. Since TPR and FPR vary between zero and one, the maximum value of AUROC is one, which corresponds to the area of the unit square. The ROC curve is monotone: by increasing the threshold for accepting an element as positive, fewer elements are classified as positive, which reduces both FP and TP , leading to lower values of FPR and TPR. Conversely, lowering the threshold makes it easier for elements to be classified as positive, which increases both FP and TP , and decreases TN and FN , thereby increasing both FPR and TPR.

So far, the discussion has focused on binary classification, but the introduced concepts can be extended to multiclass classification as well. The idea is to reduce the multiclass problem to multiple binary classification tasks. This can be done in two ways, known as **one-vs-rest** and **one-vs-one**. In the first case, each class is considered in turn as the positive class, while all the other classes are grouped together and treated as the negative class. In the second strategy, every possible pair of classes is considered, and one is treated as positive while the other is treated as negative.

In these settings, the metrics already introduced can be computed, but they need to be summarized into a single value through averaging. Two common approaches are **macro averaging** and **micro averaging**. In the case of macro averaging, the metric is the arithmetic mean of the values computed for each class, thereby assigning equal weight to all classes. In the case of micro averaging, the counts of TP , FP , TN , and FN are aggregated across all classes before computing the metric, which gives equal weight to the individual instances rather than to the classes.

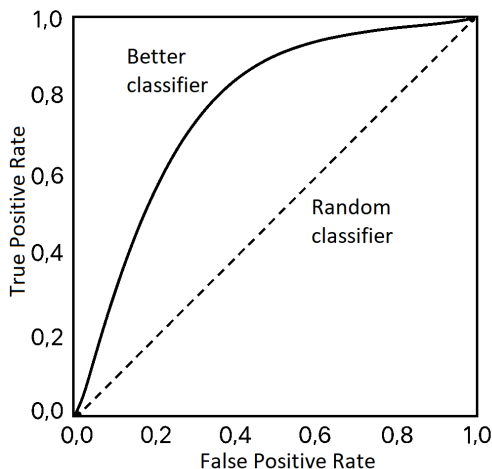


Figure 2.4: AUROC.

2.10 Deep Learning

Deep learning is a subset of machine learning that focuses on models based on specific architectures called deep neural networks, and all the principles described earlier still apply to deep learning. In this framework, the models are learnable functions defined by inputs, outputs, and parameters that are adjusted during training which are also called weights. This reflects the general principle of machine learning, where the goal is to approximate an unknown function through learning from data. Compared to other classical approaches, the power of neural networks lies in the fact that the number of parameters is on the order of millions, and in many cases can be far greater. At first sight, this aspect might seem to limit their practical applicability, since a larger number of parameters implies longer training and deploying times; moreover, as discussed earlier, many degrees of freedom usually lead to poor generalization and models prone to overfitting. The success of deep learning, however, resides in its ability to address these issues, resulting in models that are both large and general, while still allowing their application in realistic time frames. This is made possible by two key ingredients: GPUs and big data. GPUs are specialized hardware capable of executing a large number of computations in parallel; algorithms must be designed to exploit parallelization, and deep learning architectures are particularly well suited for this. On the other hand, neural networks require large amounts of data for training in order to achieve sufficient generalization and while big datasets were rare in the past, they are now widely available.

2.10.1 Perceptron

Before introducing neural networks, it is necessary to discuss the fundamental building block, the perceptron. It is a mathematical model of a biological neuron and can be

described as a parametrized function. It is defined as

$$f(x) = h \left(\sum_{i=1}^n w_i x_i + b \right) \quad (1)$$

where x_i are the input features, w_i are the learnable weights, b is the bias term, and all of them are real numbers; h is the activation function. An example is shown in Figure 2.5. When it was first proposed, h was the sign function, which returns 1 if the input is greater than or equal to zero and -1 otherwise. However, many variations of h have since been introduced. Modern activation functions are typically differentiable, which allows the use of gradient-based optimization methods for training neural networks. The most common choices include:

- **Sigmoid:**

$$h(z) = \frac{1}{1 + e^{-z}},$$

which maps the input into the range $(0, 1)$.

- **Hyperbolic tangent (tanh):**

$$h(z) = \tanh(z),$$

which maps the input into the range $(-1, 1)$ and is symmetric around the origin.

- **Rectified Linear Unit (ReLU):**

$$h(z) = \max(0, z),$$

which is widely used due to its simplicity and effectiveness in deep networks.

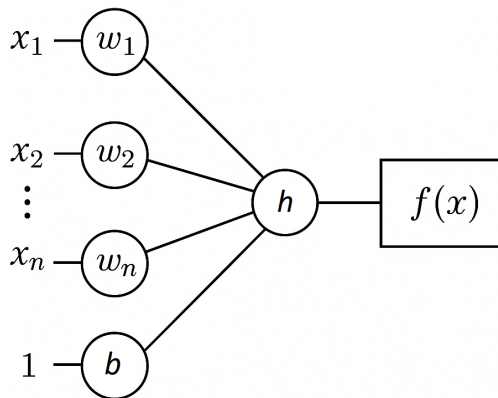


Figure 2.5: Perceptron.

The perceptron was originally introduced as a stand-alone binary classification model. Its training is performed through **gradient descent**, an optimization method that relies on the computation of the gradient of a cost function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. This procedure

makes it possible to iteratively approach a minimum of the function. Given a point $x_0 \in \mathbb{R}^n$, the gradient

$$\nabla f(x_0) = \left. \frac{\partial f}{\partial x} \right|_{x_0}$$

indicates the direction of the steepest local increase of f in a neighborhood of x_0 . Taking a step of length $\epsilon > 0$ in the direction of the gradient produces the maximum increase of the function achievable with a step of that length. On the other hand, moving in the opposite direction it produces the maximum decrease of the function. Since this is a local property, it does not guarantee reaching an optimum point (a maximum or a minimum) within a single step. For this reason, the procedure is repeated iteratively: starting from an initial point, the gradient is computed, a step toward the minimum is taken, and a new point is reached. From this new point the procedure is repeated, and so on, until convergence. The goal is to reach a global minimum, which is the point of lowest possible cost; however, in practice the procedure often converges only to a local minimum. Informally, a local minimum is a point that is optimal with respect to its neighborhood, but from which better minima may exist elsewhere. The algorithm terminates once convergence has been achieved, meaning that no further significant improvements in the cost function are observed, which is hopefully a sign that a minimum has been reached. Training a perceptron in the classical supervised setting requires the definition of a cost function, generally referred to as the *Loss*. Given an input vector x and a target value y , the perceptron is expected to learn a mapping from x to y . In practice, this means that

$$\hat{y} = h\left(\sum_{i=1}^n w_i x_i + b\right)$$

should approximate y as closely as possible. The optimization is carried out with respect to the perceptron weights w . Let the training dataset be defined as

$$\mathcal{D} = \{(x^{(j)}, y^{(j)}) \mid j = 1, \dots, m\},$$

where each $x^{(j)} \in \mathbb{R}^n$ is an input vector and $y^{(j)}$ is the corresponding target value. For a given example $(x^{(j)}, y^{(j)})$, the output of the perceptron is

$$\hat{y}^{(j)} = h\left(\sum_{i=1}^n w_i x_i^{(j)} + b^{(j)}\right).$$

Accordingly, the total error over the entire dataset is expressed as

$$\mathcal{L}(w) = \sum_{j=1}^m \text{Loss}\left(h\left(\sum_{i=1}^n w_i x_i^{(j)} + b^{(j)}\right), y^{(j)}\right).$$

Once a step size $\epsilon > 0$ (the learning rate) has been chosen for gradient descent, the update rule for the weights is

$$w^{(t+1)} = w^{(t)} - \epsilon \nabla_w \mathcal{L}(w^{(t)}),$$

where the gradient is computed with respect to the total loss function, and t denotes the iteration index.

As already stated, gradient descent is an iterative algorithm and requires an initial state; the easiest way is to initialize randomly the weights. An improvement of gradient descent is **stochastic gradient descent** which differs from the previous technique because for every iteration it does not take the entire batch of examples to compute the total loss, but at every step it selects just one element j , such that the loss is only

$$\mathcal{L}(w) = \text{Loss} \left(h \left(\sum_{i=1}^n w_i x_i^{(j)} + b^{(j)} \right), y^{(j)} \right).$$

The main advantages are that at each iteration the computation of the loss is fast, since it only requires evaluating a single training example, and that the inherent stochasticity can sometimes help to escape local minima, as the optimization is guided by different gradients across the training examples.

2.10.2 Feedforward Networks

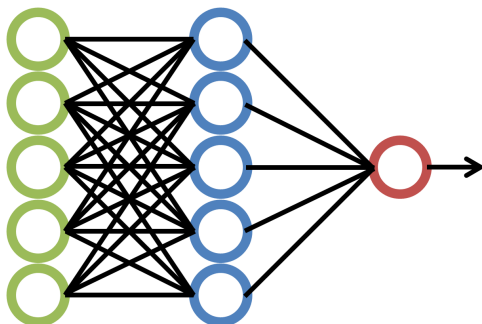


Figure 2.6: Feedforward neural network.

The original perceptron model is not powerful enough to solve complex tasks, as it can only represent linear functions. A classical example is the computation of the XOR operation between two boolean variables, which cannot be achieved with linear models. This limitation led to the idea of stacking multiple perceptrons together. The simplest architecture based on this idea is the feedforward network, also called multilayer perceptron. As the name suggests, it is composed of several layers of neurons, where the outputs of one layer are forwarded and used as inputs for the next. Each neuron can be regarded as a perceptron and follows the mathematical model described in Equation (1), with the activation function h selected among those introduced earlier. Every neuron has its own learnable weights, which act as coefficients in the linear combination of the outputs of the neurons from the previous layer, before the result is passed through the activation function. All the intermediate layers between the input and the output are referred to as **hidden layers**. In standard feedforward networks, the output of any neuron in a given layer cannot be used as input for neurons in the same layer. This

property makes the computation parallelizable across neurons within a layer, since each neuron depends only on the outputs of the previous layer. The first layer takes as input the data from the dataset, while the last layer, called the output layer, produces the predictions. An example is shown in Figure 2.6. Its structure may vary depending on the task for which the network is designed. In the case of binary classification, the output consists of a single neuron with a sigmoid activation function. Since its values range between 0 and 1, a threshold at 0.5 makes it straightforward to separate two classes. To guide the learning process in binary classification, where class labels are represented by the values $\{0, 1\}$, a common choice for the loss function is the **binary cross entropy**, defined as

$$\text{Loss}(y, f(x)) = -(y \log f(x) + (1 - y) \log(1 - f(x))).$$

The structure is fully differentiable, and it is easy to see that when the correct class is 0, the model is encouraged to predict $f(x)$ close to 0, since this value minimizes the term $-\log(1 - f(x))$. Conversely, when the correct class is 1, the prediction is pushed toward $f(x)$ close to 1, as this minimizes the term $-\log f(x)$. In the case of multiclass classification, the output layer, also called the **logits layer**, contains a number of neurons equal to the number c of classes. Each logit $o_i(x)$ is then transformed by applying the **softmax** function:

$$f_i(x) = \frac{\exp o_i(x)}{\sum_{j=1}^c \exp o_j(x)}.$$

The softmax function maps all logits to values greater than zero and normalizes them so that their sum equals one. In practice, it represents the logits as a probability distribution over the c classes. The prediction is then obtained by selecting the class with the highest probability:

$$y^* = \arg \max_{i \in [1, c]} f_i(x).$$

Training for multiclass classification is carried out using the **cross-entropy loss**. Given the correct class $y \in [1, c]$, the loss is defined as

$$\text{Loss}(y, f(x)) = -\log f_y(x),$$

where $f_y(x)$ denotes the probability assigned by the model to the correct class. In this way, the training process encourages the model to maximize the score of the true class, so that it is predicted with the highest probability. In the case of regression, the output neuron directly produces the predicted real value. The error function commonly used to train the model is the **mean squared error** (MSE):

$$\text{Loss}(y, f(x)) = (y - f(x))^2,$$

where $f(x) = o(x)$ denotes the output of the last neuron.

The **universal approximation theorem** states that a feedforward neural network with a single hidden layer containing a finite but sufficiently large number of neurons, and with a sigmoidal activation function (or, more generally, any non-constant, bounded, and continuous activation function), is capable of approximating arbitrarily well any continuous function defined on a compact subset of $\mathbb{R}^n$. This result demonstrates the expressive power of feedforward neural networks, but it does not provide a practical

method for constructing the required model, since it is only an existence theorem.

Since neural networks are parametrized with millions of parameters, it would be impractical to repeat the entire optimization procedure for each weight without efficiently sharing computations across neurons. The procedure that enables this parallelization is called **backpropagation**, whose purpose is to propagate the learning signal backward from the output layer to the input layer, in the opposite direction with respect to the forward pass used for computing the loss. The idea behind backpropagation is that the computations in each layer depend on inputs that are functions of the weights in the previous layers. When differentiating with respect to the weights of earlier layers, the **general chain rule** can be applied.

In the classical one-dimensional form it is written as

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x),$$

where $(f \circ g)(x) = f(g(x))$ with $f, g : \mathbb{R} \rightarrow \mathbb{R}$.

In the general multivariate setting, let

$$f : \mathbb{R}^m \rightarrow \mathbb{R}, \quad g : \mathbb{R}^n \rightarrow \mathbb{R}^m,$$

and consider the composition $h(x) = f(g(x))$. The chain rule then states that

$$\nabla h(x) = J_g(x)^\top \nabla f(g(x)),$$

where $J_g(x)$ denotes the Jacobian of g at x , and ∇ the gradient. Equivalently, it can be written as

$$\nabla f(g_1(x), \dots, g_m(x)) = \sum_{i=1}^m \frac{\partial f}{\partial y_i}(g(x)) \nabla g_i(x), \quad (2)$$

where $\frac{\partial f}{\partial y_i}$ denotes the partial derivative of f with respect to its i -th argument, evaluated at $g(x)$.

The following derivation of backpropagation, shown in Figure 2.7, considers the simplest setting, namely a feedforward neural network composed of N hidden layers plus the output scalar layer. This choice does not reduce generality, since with appropriate modifications the procedure can be adapted to any specific architecture. The procedure starts with the differentiation at the output layer and proceeds backward toward the input layer. To facilitate the formulation, it is useful to introduce the following definitions:

$$a_j = \sum_{i=1}^n w_i^{(j)} x_i^{(j)} + b^{(j)}, \quad \phi_j = h(a_j),$$

where j is the index of a neuron, a_j is its pre-activation value, and ϕ_j is its output after applying the activation function h .

Backpropagation requires computing for every weight i belonging to the neuron j

$$\frac{\partial \text{Loss}(W)}{\partial w_i^{(j)}},$$

where W denotes the set of all weights.

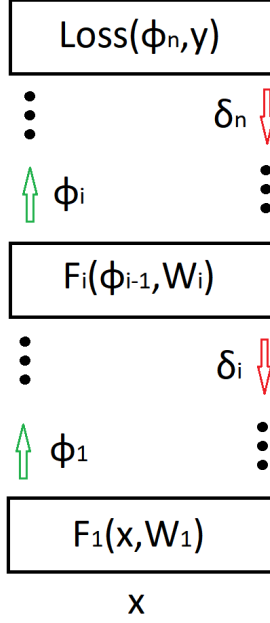


Figure 2.7: Backpropagation.

The computation of the local derivative is straightforward:

$$\frac{\partial a_j}{\partial w_i^{(j)}} = x_i^{(j)},$$

since a_j is a linear combination of the inputs.

The procedure starts from the output layer, where

$$\delta_o = \frac{\partial \text{Loss}(W)}{\partial a_o},$$

with o denoting the output neuron. Then,

$$\frac{\partial \text{Loss}(W)}{\partial w_i^{(o)}} = \delta_o \frac{\partial a_o}{\partial w_i^{(o)}}.$$

This computation depends only on the chosen loss function. Similarly to the principle of mathematical induction, the next step consists in showing how to compute the gradients with respect to the weights of neurons in layer m , under the assumption that the corresponding computations are already available for layer $m + 1$.

Let p denote a neuron belonging to layer m , and let $ch[p]$ be the set of neurons in layer $m + 1$ that receive a_p as input (the children of p). The loss with respect to a_p can be regarded as a function

$$\text{Loss}(a_p) : \mathbb{R} \rightarrow \mathbb{R},$$

which can be expressed as the composition of two functions:

$$f : \mathbb{R}^{|ch[p]|} \rightarrow \mathbb{R}, \quad g : \mathbb{R} \rightarrow \mathbb{R}^{|ch[p]|},$$

where $|ch[p]|$ denotes the cardinality of the set $ch[p]$.

Specifically,

$$f(a_{q_1}, \dots, a_{q_{|ch[p]|}}) = Loss(a_{q_1}, \dots, a_{q_{|ch[p]|}}), \quad q_i \in ch[p],$$

and

$$g(a_p) = (a_{q_1}(a_p), \dots, a_{q_{|ch[p]|}}(a_p)).$$

By applying the chain rule in the form given in (2), it follows that

$$\begin{aligned} \delta_p &= \frac{\partial Loss(W)}{\partial a_p} \\ &= \sum_{q \in ch[p]} \frac{\partial Loss(W)}{\partial a_q} \frac{\partial a_q}{\partial a_p} \\ &= \sum_{q \in ch[p]} \delta_q \frac{\partial a_q}{\partial \phi_p} \frac{\partial \phi_p}{\partial a_p} \\ &= \sum_{q \in ch[p]} \delta_q w_{qp} \frac{\partial \phi_p}{\partial a_p}, \end{aligned}$$

where w_{qp} denotes the weight associated with the connection from neuron p to neuron q , that is, the weight multiplying the activation a_p when computing the input of neuron q . The obtained derivation shows that δ_p depends only on the δ_q values belonging to the next layer, as well as on the computation of ϕ_p . The latter is straightforward to evaluate, since it is a continuous function determined by the chosen activation. The terms δ_q , instead, are obtained during the backward pass, once the gradients for layer $m + 1$ have been computed.

2.10.3 Modern Networks

Convolutional Neural Networks In computer vision, inputs are typically images, which can be represented as three-dimensional arrays of size $C \times H \times W$, where H and W denote the height and width of the image, and C denotes the number of channels. For RGB images, $C = 3$. Unlike punctual data, images are structured data: pixels that are far apart usually carry independent information, while pixels in the neighborhood of a given location jointly convey additional information beyond their individual values. Convolutional networks are specifically designed to preserve the spatial structure of the data. The discrete convolution between two two-dimensional inputs f and g is defined as

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b),$$

where (i, j) denotes the spatial position in the output. In this formulation, the function f acts as the kernel or filter, while g denotes the input signal like images. The indices

a and b span the spatial support of the kernel, which is typically restricted to a small neighborhood around the position (i, j) . This locality enables the convolution to extract features by combining information from pixels that are spatially close, while disregarding distant ones. The previous two-dimensional definition can be naturally extended to the general case of multi-channel inputs and outputs, as in the case of RGB images. For the input, the convolution is applied channel-wise, and the resulting contributions are summed across channels. Moreover, the input signal g can be convolved with multiple filters, each producing a distinct feature map. By concatenating these feature maps, the output acquires as many channels as the number of filters employed in the convolutional layer. A neural network that includes convolution operations is called a **convolutional neural network** (CNN). In this architecture, the learnable parameters correspond to the weights of the convolutional filters. The convolution operation is used to manipulate and extract local information, and it usually preserves the spatial dimensions of the input. However, at certain stages it becomes necessary to aggregate local information and produce a more abstract representation. This is achieved through **pooling** operations, which apply a sliding window over the input and compute, for each position, either the maximum or the average value within the window. Since pooling introduces no learnable parameters, it is a fixed operation that reduces the spatial resolution of the input. In the case of multi-channel inputs, pooling is applied independently to each channel, so that the number of channels remains unchanged, while the height and width are reduced according to the size and stride of the pooling window. Another important component of modern deep neural networks is **batch normalization**. This operation can be applied after any of the common layers in a network and requires training with mini-batches of data. For each randomly selected mini-batch B , the forward pass is computed, the total error is accumulated, and the backward pass is then performed. The normalization is carried out by applying the following transformation to the activations:

$$\text{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \hat{\mu}_B}{\hat{\sigma}_B} + \beta,$$

where $\hat{\mu}_B$ and $\hat{\sigma}_B$ denote the mean and standard deviation of the batch B . Each activation is thus centered and scaled so that the batch has approximately zero mean and unit variance. The parameters γ and β are learnable, and allow the network to restore an appropriate scale and shift if needed, ensuring that the normalization does not reduce the representational power of the model. Batch normalization improves training stability by keeping activations within comparable ranges, thereby reducing the risk that different layers operate on values with very different magnitudes. This standardization often has also a regularizing effect that can reduce overfitting. When dealing with deep neural networks, the learned weights can sometimes behave unpredictably and lead to problematic situations. For instance, if the weights of a layer converge towards zero, the output of that layer will also be close to zero, causing the propagated signal to vanish. To mitigate this issue, residual connections are introduced, allowing the information from previous layers to be directly incorporated into the current computation.

Given an activation function h and weight matrices W and V , a residual block can be expressed as

$$y = h(x + V \cdot h(W \cdot x)),$$

where the term $V \cdot h(W \cdot x)$ represents the *residual*. In this formulation, the input x from the previous layer is directly forwarded to the output and combined with the transformed signal produced by the block. The matrix V modulates the contribution of the residual: if $V = 0$, the block reduces to the identity mapping. Since V is a learnable parameter, the network can adaptively control how much the residual modifies the propagated input.

Another technique used to reduce overfitting is **dropout**. It is applied during training and consists in deactivating, with a random probability p , some of the units of the network. This is done independently at each forward-backward pass, with the set of deactivated units changing every time. During backpropagation, no information flows through the deactivated units. Dropout reduces overfitting by preventing the network from relying too heavily on specific connections, effectively reducing the number of active parameters at each pass and encouraging better generalization.

Transformers Although convolutional networks are particularly effective in computer vision, where data is highly structured, other architectures have been shown to outperform them in various contexts. These architectures are called **transformers** and are applied to many varieties of tasks. The building block of transformers is the so-called **self-attention**.

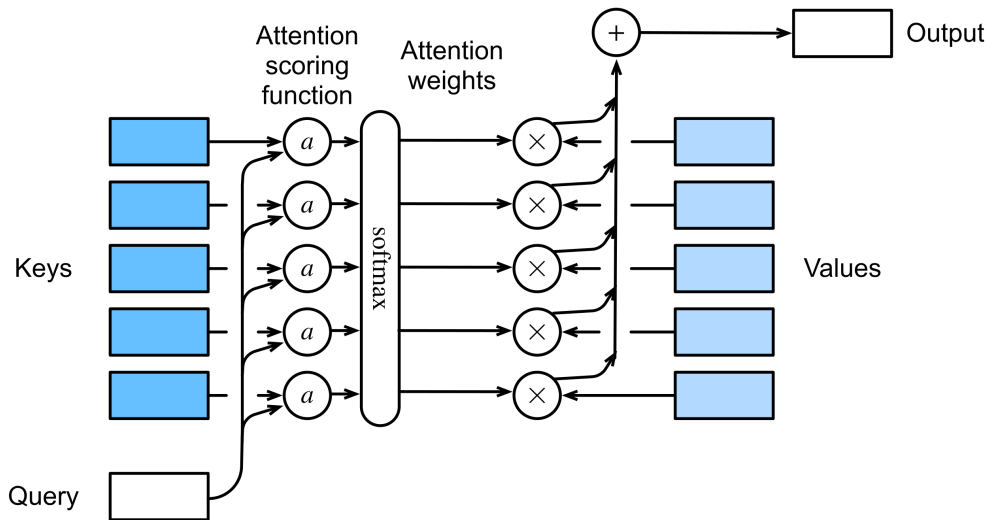


Figure 2.8: Attention mechanism output. By Zhang, Aston; Lipton, Zachary C.; Li, Mu; and Smola, Alexander J. (2023). Source: <https://github.com/d2l-ai/d2l-en>. Licensed under CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>).

This mechanism works extremely well with complex and structured data, such as text and images, where every part of the input can be semantically linked with any other part regardless of its position. These parts are called **tokens**, which represent the semantic entities of the input. Since every token can have strong semantic relations

with some tokens and weak relations with others, it is important to capture the context of each token by emphasizing its strongest links. This is achieved through self-attention. In abstract, the attention mechanism operates on three elements: queries Q , keys K , and values V . The **values** represent the information associated with each token. Each value is a vector of length l , and if the input sequence contains m tokens, then there are m values in total, so that $V \in \mathbb{R}^{m \times l}$. The **keys** represent the information used to retrieve the corresponding values. Each key is a vector of dimension d , and there is one key for each value; therefore $K \in \mathbb{R}^{m \times d}$. Finally, the **queries** are the elements used to interrogate the set of keys in order to retrieve the values. Each query is also a vector of dimension d , and there can be n queries in total, so that $Q \in \mathbb{R}^{n \times d}$. Given a query $q \in Q$, the attention mechanism computes a weight for each key $k_i \in K$, which is then used to combine the corresponding values $v_i \in V$. The resulting output is a weighted sum of all the values:

$$\text{Attention}(q) = \sum_{i=1}^m \alpha(q, k_i) v_i,$$

where α is a function that computes a scalar weight from q and k_i . The weights $\alpha(q, k_i)$ are required to form a probability distribution over the elements in V , meaning that they must be non-negative and sum to one. This ensures that the values in V are combined consistently, and that the overall scale of the output does not depend on the particular query q . To enforce this property, the softmax operation is applied. Hence, for every query q the attention weights are computed by applying the softmax function:

$$\alpha(q, k_i) = \frac{\exp(q^\top k_i / \sqrt{d})}{\sum_{j=1}^m \exp(q^\top k_j / \sqrt{d})},$$

where m is the number of keys and the factor $\sqrt{d}$ is introduced to keep the variance of the dot product under control. This operation is carried out independently for each query $q \in Q$. In matrix form, the attention mechanism can be expressed compactly as

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V \in \mathbb{R}^{n \times l},$$

where the softmax is applied row-wise, that is, separately for each query. In self-attention, the goal for each token is to capture its relationships with all the other tokens. For this reason, the number of queries n and the number of values m equal the number of tokens.

The general structure of a transformer consists of a trainable encoder and decoder. The **encoder** is responsible for extracting a contextual representation of the input sequence through the self-attention mechanism. The resulting context is then passed to the decoder in the form of keys and values. The **decoder** generates the output sequence token by token. To model the dependencies among the tokens that have already been produced, it also employs self-attention. However, in this case a **masked self-attention** mechanism is applied: each token is allowed to attend only to itself and to previous tokens, but not to future ones, ensuring causality during generation. In addition, the decoder makes use of **encoder-decoder cross-attention**. Here, the queries are derived from the decoder states (corresponding to the partially generated output), while

the keys and values come from the, already mentioned, encoder representations of the input. This allows each generated token to depend not only on the previously generated tokens, but also on the global context provided by the encoder. This architecture was originally designed for text, as it naturally handles sequential data. However, it has also been extended to the vision domain with the **Vision Transformer (ViT)**. In this case, images are divided into patches, and each patch is treated as a token. In the original ViT, only the encoder is used: it captures the relationships between patches through self-attention, and the resulting contextual representation is employed for classification. Other vision transformers, instead, also make use of a decoder. This is typically the case in tasks such as the reconstruction of partially masked images. Here, the image is divided into patches and a subset of them is masked. The encoder processes the visible patches to build a contextual representation, while the decoder takes this representation and attempts to reconstruct the full image, including the masked patches.

2.10.4 Transfer Learning

Beyond architectural design, the strength of neural networks lies in their ability to generalize when trained on large amounts of data. However, powerful models require training on massive datasets, which comes with long training times, high energy consumption, and significant human effort, especially in the case of annotated data. As a result, retraining every model from scratch is often too costly, particularly when the same model must be deployed across different scenarios. To address this issue, the concept of **transfer learning** has been introduced. The idea is to exploit models that have already been trained on large datasets and adapt them to the task at hand. A common approach is **fine-tuning**, which consists of taking a pretrained model, freezing its initial layers, and retraining only the final layers to match the requirements of the new task. In this way, the pretrained model retains the general knowledge it has already acquired across many domains, while fine-tuning serves as a specialization step that adapts this knowledge to the specific task. This approach is commonly employed with models for classification as well as for tasks such as localization and segmentation, which will be described in the following chapters. Pretrained models are downloaded and fine-tuned on smaller, task-specific datasets in order to adapt them to the number of classes required by the task. Another approach to transfer learning consists of retaining only the **backbone** of a pretrained model, regardless of whether it was trained in a supervised or self-supervised fashion. In this case, the final layers dedicated to the original training task are discarded, and the backbone is used purely as a feature extractor. Typically, the backbone is frozen and additional task-specific layers are trained on top of it, so that the pretrained representations can be reused effectively in downstream tasks.

2.10.5 Augmentation

In real-world scenarios, unlike competition settings where data is often abundant, the problem of data scarcity is common. In particular, the issue may not lie in the numerical quantity of data, but rather in the fact that it is often collected under very similar conditions. This is problematic because neural networks tend to generalize only within the range of variations they have already seen, and are not able to invent new ones from scratch. In practice, environments can be highly dynamic, with variations

in illumination, perspective, and background. To address this issue without the need to imagine and collect all possible scenarios, a common strategy is to apply **data augmentation**. This consists of artificially perturbing the available data through various transformations, with the goal of filling gaps in the distribution that are not covered by the dataset itself. In the case of images, the most common augmentations include:

- **Flipping**: mirroring the image horizontally or vertically;
- **Cropping**: extracting a subregion of the image, simulating zoom or partial visibility;
- **Shearing**: applying a geometric transformation that slants the image along one axis;
- **Color jittering**: randomly altering brightness, contrast, or color balance to simulate different lighting conditions.

3

Computer Vision

Computer vision is the field that deals with computer systems that extract and process information from images and videos. Its two main tasks are **reconstruction** and **recognition**. The goal of reconstruction is to model the physical world from a set of images by creating a geometric representation that resembles it. The goal of recognition, instead, is to extract semantic meaning from images, where the notion of semantics depends on the specific task being performed. This chapter is based on works [7, 8].

3.1 Recognition Tasks

Image Classification. The goal of classification is to assign a general semantic label to the entire image. Traditional convolutional networks are well suited for this task: they process the whole image and output a discrete probability distribution over the available classes, typically using a softmax function.

Semantic Segmentation. This task goes beyond image classification, as it aims to assign a semantic label to each pixel of the image. In this way, the extracted information is more detailed, since it provides localized semantics that cover the entire image. The output is generally a mask of the same size as the input image, where each pixel contains an integer label indicating its class. The background is treated as a special class, usually indexed as zero. Models are not explicitly designed to detect it, since there is generally no real interest in identifying the background itself. Rather, its identification is a side effect of recognizing the other classes, as the background is simply defined as everything that does not belong to the target classes.

Object Detection. The goal of object detection is to locate in an image all the instances belonging to the selected classes, with a **bounding box** that tightly encloses the object. In most cases, bounding boxes are rectangles with sides parallel to the image axes. A more precise variant of the task is to predict rotated bounding boxes, which also requires estimating the orientation angle of the box in addition to the classical

parameters: the coordinates of the center and the lengths of the sides. Each bounding box must also be assigned to the correct semantic category.

Instance Segmentation. A limitation of semantic segmentation is that multiple objects of the same class within a single image are not distinguished from one another: they are all assigned to the same semantic category. In many scenarios, however, it is crucial to separate different instances of the same class. This is the objective of instance segmentation, which combines semantic segmentation with object detection. The output includes both semantic masks and instance masks. The latter are similar in structure to semantic masks, but each pixel is assigned an integer label that identifies the specific instance it belongs to. Instance segmentation can also be seen as a way to perform object detection, since bounding boxes can be easily derived from the instance masks by surrounding each mask with the corresponding box.

3.2 Metrics

Semantic segmentation can be regarded as pixel-level classification and, similarly to image classification, it follows the standard metrics for classification introduced in the previous chapters. Each pixel is assigned a predicted label, which can result in a TP , TN , FP , or FN . By considering all the classified pixels across the entire dataset, it is possible to compute metrics such as accuracy, precision, and recall. In these computations, however, the background class is usually excluded, since it does not represent a semantic category of interest. In segmentation, an important evaluation metric is the **Intersection over Union (IoU)**, which measures the accuracy of the localization. It is defined as the ratio between the intersection and the union of the predicted region A and the ground truth region B :

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

In semantic segmentation tasks, the IoU is expressed in terms of pixel classification counts, and it is written as

$$IoU = \frac{TP}{TP + FP + FN},$$

where TP denotes the number of true positive pixels, FP the false positives, and FN the false negatives. The IoU is computed separately for each class across all images, with the background class typically excluded. In the multiclass case, the final score is obtained by averaging the IoU values over all the considered classes, following the same procedure described previously. In the case of detection, the IoU between a detected box A and a ground truth box B is defined as the ratio between the area of their intersection and the area of their union. When dealing with object detection and instance segmentation, the Intersection over Union (IoU) is not used as a standalone metric; rather, it serves as a criterion for defining evaluation measures that quantify the quality of instance localization. One of the most widely used metrics is the **Mean Average Precision (mAP)**. The term mean indicates that, in multiclass settings, the average precision

(AP) is first computed separately for each class and then macro-averaged across all classes. For a given class, the AP is defined as the area under the precision–recall curve (AUC-PR), which plots how precision varies as a function of recall. Since both precision and recall take values in $[0, 1]$, the maximum possible area is the unit square 1×1 , and therefore $AP \in [0, 1]$. To compute the Average Precision (AP), a fixed IoU threshold τ is used (commonly $\tau = 0.5$ or $\tau = 0.75$, in which case the metrics are referred to as AP_{50} and AP_{75} , respectively). This threshold determines whether a detection can be considered correct: a detection is valid if its IoU with a ground-truth box exceeds τ . The procedure for a given class is as follows:

1. Collect all ground-truth instances and predictions of the class.
2. Sort the predictions by confidence score in descending order.
3. For each prediction (in order):
 - (a) Find the ground-truth box with the highest IoU.
 - If the IoU is $> \tau$ and the gt-box has not yet been assigned, the prediction is counted as TP .
 - Otherwise, the prediction is counted as FP .
 - (b) Compute the **cumulative precision** and **cumulative recall** up to this step:

$$\text{Precision}[i] = \frac{TP_c[i]}{TP_c[i] + FP_c[i]}, \quad \text{Recall}[i] = \frac{TP_c[i]}{N_{GT}},$$

where $TP_c[i]$ and $FP_c[i]$ denote the cumulative number of true positives and false positives up to step i , and N_{GT} is the total number of ground-truth instances of that class.

At each step, the cumulative recall is non-decreasing, while the precision may decrease as false positives accumulate. The pairs $(\text{Recall}[i], \text{Precision}[i])$ for all steps define the **precision–recall curve**. To obtain the Average Precision, this curve is summarized by an interpolation method called **11-point interpolation**. In this method, the precision is sampled at 11 fixed recall levels:

$$r \in \{0.0, 0.1, 0.2, \dots, 1.0\}.$$

For each recall threshold r , the interpolated precision is defined as:

$$p_{\text{interp}}(r) = \max_{\tilde{r} \geq r} \text{Precision}(\tilde{r}),$$

that is, the maximum precision achieved for any recall greater than or equal to r . Finally, the Average Precision is computed as the arithmetic mean:

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \dots, 1.0\}} p_{\text{interp}}(r).$$

This produces a single number between 0 and 1 that summarizes the trade-off between precision and recall for the class under the chosen IoU threshold. The methodology

just explained, however, considers only a fixed IoU threshold τ . A more precise metric, mAP@[.5:.95] , computes the AP for each threshold in the range from 0.50 to 0.95 with a step size of 0.05. The values of AP obtained for the same class at the different thresholds are then averaged to produce the final AP for that class. Furthermore, this metric uses a finer interpolation scheme based on 101 recall points instead of 11. The procedure described above does not specifically refer to bounding boxes or instance masks; this means that it can be applied both to object detection and to instance segmentation. The only difference lies in how the IoU is computed, namely between bounding boxes in the former case and between masks in the latter.

Usually, there can be multiple detections for the same instance. In such cases, it is useful to reduce the number of redundant detections by applying **Non-Maximum Suppression (NMS)**. This technique works as follows: all predicted bounding boxes are first sorted by their confidence score in descending order. Then, starting from the highest-scoring detection, all other detections that have an IoU greater than a given threshold with this box are suppressed, since they are considered duplicates of the same object. The algorithm then proceeds iteratively with the next highest-scoring box among the remaining ones, repeating the suppression step until no boxes remain to be processed.

Pixel-level AUROC for segmentation is computed by treating all pixels as individual binary classification instances and assigning each a continuous soft score; the AUROC is then obtained as the area under the ROC curve by varying the decision threshold over these scores.

Similarly, at the pixel level, the **AUPRO** metric is defined. Like AUROC, it is the area under a curve, specifically the per-region overlap (PRO) curve. For a decision threshold t , let P_t be the set of pixels predicted as anomalous and let $\{C_i\}_{i=1}^N$ be the connected components of the ground-truth anomaly mask. The PRO at threshold t is

$$\text{PRO}(t) = \frac{1}{N} \sum_{i=1}^N \frac{|P_t \cap C_i|}{|C_i|}.$$

Plotting $\text{PRO}(t)$ against the false positive rate $\text{FPR}(t)$ as t varies yields the PRO curve, with FPR on the x-axis and PRO on the y-axis. The **AUPRO** is the area under this curve.

3.3 Camera model

The purpose of this section is to introduce the fundamentals of image formation and 3D reconstruction. Image formation is described using the **pinhole** camera model, schematized in Figure 3.1. In this model, the actors are the 3D world, the camera and the image. The world is a set of 3D points that should be projected in the 2D image, which can be considered as a rectangular portion of plane characterized by a center whose coordinates in the 3D world are known and by the parameters of height and width. The camera act as the device that maps the 3D points in the 2D points and relies on the usage of projective geometry. The idea behind the model is that light rays emitted or reflected by points in the 3D scene pass through the pinhole, which in the limiting case is represented as a single point. Each of these rays intersects the image plane, and the

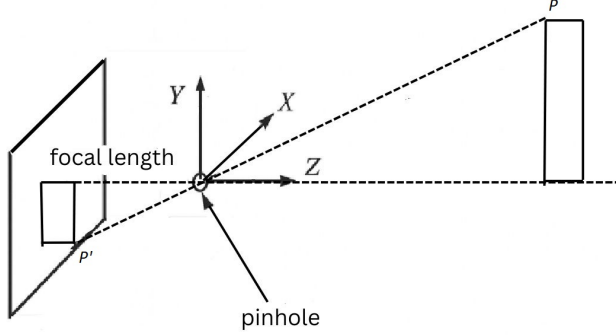


Figure 3.1: Pinhole model

intersection defines the projection of the corresponding 3D point, thereby forming the image. In this setting, the object is located in front of the camera and the pinhole, while the image plane is placed behind them. Points in the 3D world are denoted by capital letters, in general $P = (P_x, P_y, P_z)$, whereas their projections on the image plane are represented by lowercase letters $p = (p_x, p_y, p_z)$. Although images are inherently two-dimensional, the third coordinate p_z is introduced to represent the position of the image plane along the Z axis, and this representation is called homogeneous coordinates. The projection is described in the camera coordinate system, whose origin is located at the pinhole. The X and Y axes are aligned with the dimensions of the image plane, while the axis passing through the pinhole and orthogonal to the image plane is called the principal axis. The point where the principal axis intersects the image plane is referred to as the principal point. The distance between the pinhole and the principal point is called **focal length** indicated as f . The principal axis is aligned with the Z axis of the camera coordinate system. The choice of coordinate system may vary depending on the adopted convention. In the case illustrated in Figure 3.1, the Z axis is oriented forward toward the 3D scene, the X axis points to the right, and the Y axis points upward. However, an alternative but equally valid dextro-rotatory convention is often adopted, in which the Z axis points inward toward the image plane, the Y axis points upward, and the X axis points to the left. The principle of projection is simply to preserve proportions. A 3D point $P = (P_X, P_Y, P_Z)$ is mapped to an image point $p = (p_x, p_y, p_z)$, where the depth coordinate p_z corresponds to the focal length f . What remains is to determine p_x and p_y using the proportionality rules. Considering, for instance, the point of the rectangle in Figure 3.1, its projection is obtained by following the corresponding dotted ray until it intersects the image plane. Along the vertical axis Y , the proportion between world and image coordinates is expressed as

$$\frac{P_Y}{P_Z} = \frac{p_y}{f},$$

and similarly, along the horizontal axis X , the relation is

$$\frac{P_X}{P_Z} = \frac{p_x}{f}.$$

From these relations, the projection equations can be written as

$$p_x = f \frac{P_X}{P_Z}, \quad p_y = f \frac{P_Y}{P_Z}, \quad p_z = f.$$

The relevant coordinates on the image plane are the first two, (p_x, p_y) . These values, however, are expressed in the reference frame of the camera. In practice, an image has its own 2D coordinate system, typically defined with the origin at one corner of the rectangular image. Let (p_w, p_h) denote the coordinates of the principal point in the image coordinate system. The final pixel coordinates are then obtained by applying a translation:

$$u = f \frac{P_X}{P_Z} + p_w, \quad v = f \frac{P_Y}{P_Z} + p_h.$$

This translation shifts the coordinates from the camera reference frame into the image coordinate system, aligning them with the convention used to represent pixels in the image. All this computation can be summarized using the **intrinsic** calibration matrix K , defined as

$$K = \begin{bmatrix} f & 0 & p_w \\ 0 & f & p_h \\ 0 & 0 & 1 \end{bmatrix};$$

then, given $P = (P_X, P_Y, P_Z)^\top$, the homogeneous image point is obtained as

$$\hat{p} = KP = \begin{bmatrix} \hat{p}_x \\ \hat{p}_y \\ p_z \end{bmatrix},$$

and the Euclidean pixel coordinates follow by normalization:

$$u = \frac{\hat{p}_x}{p_z}, \quad v = \frac{\hat{p}_y}{p_z}.$$

Thus, the mapping from a 3D point in camera coordinates to a 2D point in the image plane is expressed as $p = (u, v)^\top$. The intrinsic matrix K can be generalized to account for different focal length parameters along the X and Y directions in the case of anisotropy, leading to the use of f_x and f_y instead of a single focal length f . Furthermore, a skew parameter s may appear in the second column of the first row, although this effect is not considered in this work. As already mentioned, the projection is defined for 3D points expressed in the camera coordinate system, whose origin is located at the pinhole. In practice, however, objects and their points are usually represented in the world coordinate system, in which cameras are also defined. Each camera is then associated with a pose, which specifies the position of the camera origin in the world system as well as its orientation with respect to the world axes. This means that, before

projection, points must be transformed from world coordinates to the camera coordinate system through a rigid body transformation, composed of a rotation followed by a translation. Let $C \in \mathbb{R}^3$ be a column vector representing the camera center in world coordinates, and let $R \in SO(3)$ be the rotation matrix describing the orientation of the world axes with respect to the camera coordinate system; in particular $R = R_c^\top$ where $R_c^\top$ is the rotation matrix that describes the orientation of the camera coordinate system respect the world coordinate system. The extrinsic transformation can then be written as

$$P_c = R_c^\top (P_w - C),$$

which is equivalent to

$$P_c = R_c^\top P_w + t, \quad \text{with } t = -R_c^\top C.$$

In homogeneous coordinates this becomes

$$\tilde{P}_c = \begin{bmatrix} R_c^\top & -R_c^\top C \\ 0 & 1 \end{bmatrix} \tilde{P}_w,$$

where the matrix is the **extrinsic** matrix and the homogeneous vector $\tilde{P}_w$ is obtained from the 3D world point $P_w = (X_w, Y_w, Z_w)^\top$ by appending a 1 at the end,

$$\tilde{P}_w = \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}.$$

Analogously, the homogeneous camera point $\tilde{P}_c$ has the form

$$\tilde{P}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix},$$

and the Euclidean camera coordinates are recovered by discarding the last component. The extrinsic matrix is also called world-to-camera matrix, and its inverse is called camera-to-world matrix. The combination of the intrinsic and extrinsic parameters defines the projection matrix, which maps 3D points into the image. Its standard form is given by $P = K[R \mid t]$.

3.4 3D reconstruction

In this section, **Structure from Motion (SfM)** is introduced. SfM is a technique for recovering three-dimensional structures (3D points) from a set of two-dimensional images. The presented workflow provides a general overview of SfM, while more sophisticated approaches and variations are also available. The idea behind SfM is that each image contains only a partial representation of the scene, while multiple images often share overlapping regions. By combining the common information between two images,

it becomes possible to recover 3D structure and reconstruct parts of the scene. The first step is to establish feature correspondences between pairs of images. This is typically achieved by extracting local features from both images and then matching them by comparing the similarity of their feature descriptors. These correspondences link points in one image to points in another, and each pair can be interpreted as the projection of the same 3D point. They form the basis for reconstructing the underlying structure, by performing **triangulation**. Triangulation is the process of estimating the position of a 3D point by intersecting the viewing rays that pass through its projections in two images. In other words, when the same feature is observed from different viewpoints, the intersection of the corresponding lines of sight yields its location in 3D space. The triangulation technique belongs to the field of epipolar geometry.

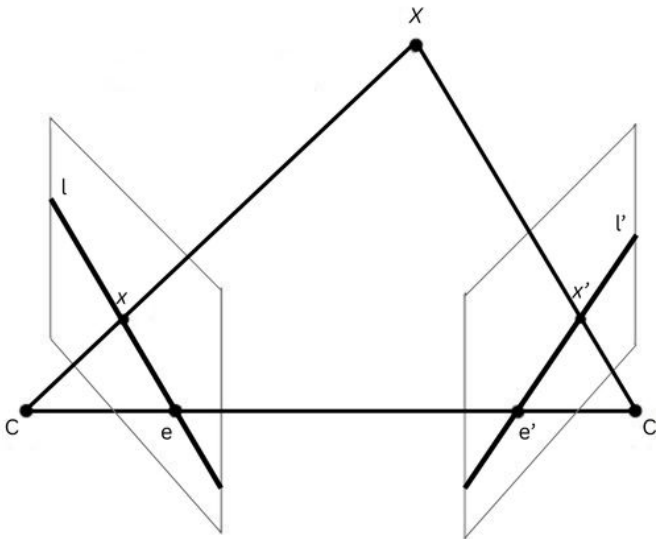


Figure 3.2: Epipolar geometry

Formally, given two images characterized by their projection matrices P and P' , two image points x and x' define a correspondence if there exists a 3D point X such that $x = PX$ and $x' = P'X$. Figure 3.2 illustrates the fundamental elements of epipolar geometry: the image points x and x' are the projections of the same 3D point X ; the epipoles e and e' are the projections of the opposite camera centers; and the lines l and l' are the corresponding epipolar lines. The line connecting the two camera centers is referred to as the baseline.

Fundamental Matrix When the cameras are uncalibrated, all geometric constraints between correspondences are encoded in the **fundamental matrix** F . This is a 3×3 matrix of rank two with 7 degrees of freedom that enforces the epipolar constraint for any correspondence through the relation $x'^T F x = 0$. If the two projection matrices P

and P' are given, the fundamental matrix F can be obtained as

$$F = [e']_{\times} P' P^+,$$

where P^+ denotes the pseudo-inverse of P , e' is the epipole in the second image. The notation $[e']_{\times}$ indicates the 3×3 matrix such that allows computing the cross product between e' and any vector $v \in \mathbb{R}^3$, i.e. $[e']_{\times} v = e' \times v$. Explicitly,

$$[e']_{\times} = \begin{bmatrix} 0 & -e'_z & e'_y \\ e'_z & 0 & -e'_x \\ -e'_y & e'_x & 0 \end{bmatrix},$$

where $e' = (e'_x, e'_y, e'_z)^T$. Conversely, estimating the fundamental matrix in an uncalibrated scenario is a crucial step for performing triangulation and for recovering the projection matrices P and P' . In particular, the canonical form of the camera matrices corresponding to a given fundamental matrix F can be chosen as

$$P = [I|0], \quad P' = [[e']_{\times} F | e'],$$

where I is the 3×3 identity matrix, e' is the epipole in the second image, and $[e']_{\times}$ is the skew-symmetric matrix associated with e' . However, there is no unique derivation of P and P' from F ; in fact, if (P, P') is a valid pair of projection matrices associated with F , then for any projective transformation $H : \mathbb{P}^3 \rightarrow \mathbb{P}^3$ the pair

$$\tilde{P} = PH, \quad \tilde{P}' = P'H$$

defines the same fundamental matrix. Hence, F determines the projection matrices only up to a 3D projective transformation. To recover the full metric reconstruction more information about the scene is needed.

Essential Matrix Sometimes, more knowledge about the cameras is available. In the case of calibrated cameras, instead of considering the fundamental matrix, it is convenient to work with the **essential matrix** E . Assuming that the camera matrix can be decomposed as $P = K[R \mid t]$ and $x = PX$ is a point in the image, then the effect of calibration can be removed by applying the inverse of K , so that the point in normalized coordinates is

$$\hat{x} = K^{-1}x.$$

Similarly to F , the essential matrix can be expressed algebraically assuming two canonical normalized camera configuration given by

$$P = [I \mid 0], \quad P' = [R \mid t].$$

In particular, the corresponding essential matrix has the form

$$E = [t]_{\times} R.$$

The defining constraint of the essential matrix is expressed in normalized coordinates as

$$\hat{x}'^T E \hat{x} = 0,$$

for corresponding points $\hat{x} \leftrightarrow \hat{x}'$. Given an essential matrix E , it can be factorized through SVD as

$$E = U \operatorname{diag}(1, 1, 0) V^T,$$

where U and V are orthogonal matrices. Then, let

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Taking the canonical first camera matrix as

$$P = [I \quad 0],$$

the second camera matrix P' admits four possible solutions:

$$P' \in \{ [UW^T \mid +u_3], [UW^T \mid -u_3], [UW^T V^T \mid +u_3], [UW^T V^T \mid -u_3] \},$$

where u_3 is the third column of U . Among these four candidates, only one corresponds to the physically valid reconstruction, since the reconstructed 3D points must lie in front of both cameras. This cheirality condition is used to disambiguate the correct metric reconstruction from the essential matrix. Finally, the relation between the essential matrix and the fundamental matrix is obtained by substituting $\hat{x} = K^{-1}x$ and $\hat{x}' = K'^{-1}x'$ in the epipolar constraint. This leads to

$$E = K'^T F K,$$

which shows that E and F are equivalent representations of epipolar geometry in the calibrated and uncalibrated cases, respectively.

Estimation of the Fundamental Matrix In principle, at least 7 point correspondences are required to estimate it, because F has 7 degrees of freedom and it can be done through **direct linear transformation (DLT)**. The starting point is the epipolar constraint $x'^T F x = 0$; indeed writing $x = (x, y, 1)^T$ and $x' = (x', y', 1)^T$, the equation expands as

$$x'x f_{00} + x'y f_{01} + x'f_{02} + y'x f_{10} + y'y f_{00} + y'f_{12} + x f_{20} + y f_{21} + f_{22} = 0,$$

which is linear in the nine unknown entries of F . Collecting these coefficients for n correspondences leads to a homogeneous linear system of the form

$$A f = 0,$$

where f is the 9-vector containing the entries of F in row-major order, and A is an $n \times 9$ matrix constructed from the image coordinates. In practice, image points are noisy and

$n > 8$ correspondences are typically available. In this case, the system is overdetermined and the fundamental matrix can be estimated through least-squares estimation. The solution is obtained by singular value decomposition: if $A = UDV^T$, then f is equal to the smallest singular value of A , which corresponds to the last column of V and the resulting vector is reshaped into the 3×3 matrix F . The fundamental matrix must have rank 2, but the linear estimate obtained from noisy correspondences generally does not satisfy this constraint. The standard fix is to replace F with a matrix $\tilde{F}$ of rank 2 that is closest to F in the Frobenius norm, i.e.,

$$\tilde{F} = \arg \min_{G: \text{rank}(G)=2} \|F - G\|_F,$$

by imposing that $\det(\tilde{F}) = 0$. Then $\tilde{F}$ is obtained by considering $F = UDV^T$ as the singular value decomposition, with

$$D = \text{diag}(r, s, t), \quad r \geq s \geq t \geq 0.$$

Then the optimal approximated solution is obtained by zeroing the smallest singular value:

$$\tilde{F} = U \text{diag}(r, s, 0) V^T.$$

This procedure is known as the **8-point algorithm**, which requires at least eight correspondences to find a unique solution.

Perspective n Point The Perspective- n -Point (PnP) problem is a general framework in which, starting from a set of correspondences between 3D points X_i in the scene and their 2D projections x_i in the image, it is possible to recover the pose of the camera that acquired the image. The formulation is valid also in the uncalibrated case, although a calibrated camera model is typically assumed for accurate pose estimation. A basic approach to solve the PnP problem is to employ the DLT method with at least 6 correspondences of non coplanar points. The projection of a 3D point $X_i = (X_i, Y_i, Z_i, 1)^T$ onto $x_i = (x_i, y_i, 1)^T$ through the 3×4 camera matrix $P = \{p_{jk}\}$ can be written as:

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \quad y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}.$$

Multiplying by the denominators and rearranging gives two linear equations in the 12 unknown entries of P :

$$\begin{aligned} p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03} - x_i(p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}) &= 0, \\ p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13} - y_i(p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}) &= 0. \end{aligned}$$

Stacking the 12 parameters into

$$\mathbf{p} = [p_{00}, p_{01}, p_{02}, p_{03}, p_{10}, p_{11}, p_{12}, p_{13}, p_{20}, p_{21}, p_{22}, p_{23}]^T,$$

each correspondence contributes the two rows

$$\underbrace{\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & -x_i X_i & -x_i Y_i & -x_i Z_i & -x_i \end{bmatrix}}_{\text{row } (2i-1)},$$

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_i X_i & -y_i Y_i & -y_i Z_i & -y_i \end{bmatrix}}_{\text{row } (2i)}.$$

Collecting $n \geq 6$ (non-coplanar) correspondences yields the homogeneous system

$$A \mathbf{p} = 0, \quad A \in \mathbb{R}^{2n \times 12}.$$

The estimate is obtained by finding $\mathbf{p}$ as described in the 8-point algorithm.

Triangulation The goal of triangulation is to recover the 3D point corresponding to a pair of matching image points assuming that the camera matrices P and P' and the image correspondences $x = (x, y, 1)^\top$ and $x' = (x', y', 1)^\top$ are known. Then the 3D point to recover is denoted as $T = (X, Y, Z, 1)^\top$. Since the projection is independent of the scale, it is possible to write

$$x \times (PT) = 0,$$

which can be written by developing the cross product and taking the first two equations as

$$\begin{aligned} x (p_2^\top T) - (p_0^\top T) &= 0, \\ y (p_2^\top T) - (p_1^\top T) &= 0, \end{aligned}$$

where P_i are the rows of P . Taking two independent equations per image, the system can be written in homogeneous form $AT = 0$ with

$$A = \begin{bmatrix} x p_2^\top - p_0^\top \\ y p_2^\top - p_1^\top \\ x' p_2'^\top - p_0'^\top \\ y' p_2'^\top - p_1'^\top \end{bmatrix} \in \mathbb{R}^{4 \times 4}, \quad T \in \mathbb{P}^3.$$

Finally, T is recovered as usual through SVD.

Bundle Adjustment After initializing cameras and 3D points, the reconstruction is refined by bundle adjustment, which jointly optimizes camera and structure parameters by minimizing the total re-projection error:

$$E = \sum_{i=1}^n \sum_{j=1}^m d(\pi(P_i, T_j), x_{ij})^2.$$

Here, n is the number of cameras and m the number of 3D points; P_i collects the parameters of camera i , T_j is the 3D point j ; x_{ij} is the measured image location of point j in camera i ; π is the camera projection model and $d(\cdot, \cdot)$ denotes the metric distance. The objective is a nonlinear least-squares problem, solved iteratively with optimizers such as **Levenberg-Marquardt**.

Structure for Motion With the elements introduced above, a standard SfM pipeline from uncalibrated images proceeds as follows. Two images with overlapping content are selected; local features are extracted and matched. From the resulting correspondences, the fundamental matrix F is estimated. Given F , a canonical pair of projection matrices (P, P') is recovered, and the inlier correspondences are triangulated to obtain an initial set of 3D points. The procedure then continues incrementally by adding images one at a time: for each new image, its pose is estimated via PnP with respect to the current 3D points; newly observed points are triangulated and tracks referring to the same 3D point are merged. After each registration, bundle adjustment jointly refines camera parameters and 3D structure. When the pipeline is initialized from the fundamental matrix, the reconstruction is *projective* and is determined only up to a projective transformation. If instead the process relies on the essential matrix E , the reconstruction is *metric*, determined up to an overall scale.

The preceding overview of machine learning, deep learning, computer vision, and 3D reconstruction is not intended to be comprehensive; it only introduces the theoretical tools and ideas that will be used to set up the problems addressed in the following chapters.

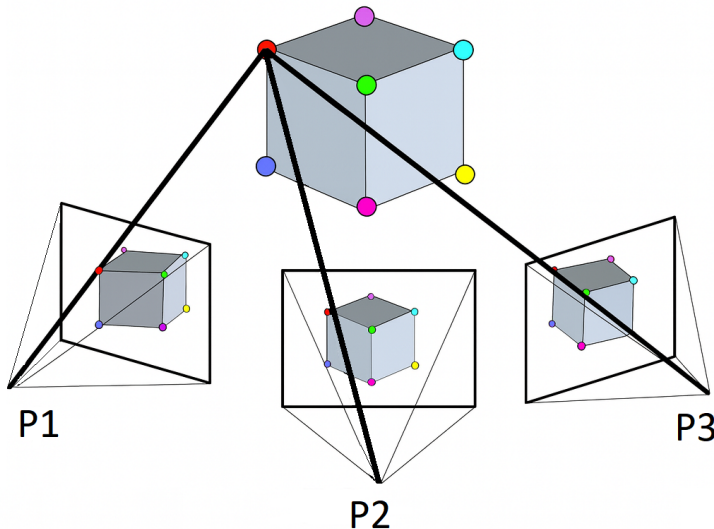


Figure 3.3: Structure from Motion illustration. Adapted from Wikipedia (2016). Author: Wikipedia contributors. Licensed under CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>).

4

Problem Definition

This work focuses on visual anomaly detection for industrial quality control, addressing both structural anomalies, meaning defects directly observable on the object's morphology or surface such as scratches, cracks, deformations, missing material, or contamination, and logical anomalies, meaning violations of expected functional or assembly constraints such as misplaced parts, or inconsistent configurations, which, although not limited to surface imperfections, can still be identified visually because they appear as perceptible deviations from the expected spatial or relational structure.

In real deployments, inspection takes place in an unconstrained three-dimensional workspace where multiple agents interact and must be coordinated. The agents include the items to be inspected, the background actuators and carriers that move or hold the items such as robotic arms, grippers, and conveyors, and the sensors that capture visual data such as RGB cameras and depth sensors. Since these agents share the same physical space and operate at the same time, their actions and data streams must be synchronized, which means the inspection algorithms need to be fast and responsive. The scene itself adds further difficulties, including noise and visual distractors, so the background is not negligible. Consequently, robust background removal and reliable foreground segmentation are often required before anomaly detection, ensuring that the inspection focuses on the target object. Beyond these scene-level effects, geometric complexity is a primary challenge. Objects are volumetric and must be examined over their entire surface to detect localized anomalies; indeed, they can be bulky, introducing clutter and self-occlusions. In addition, since objects exist in 3D space, they can move and take many different and unpredictable poses. This requires adapting the viewpoints during inspection, which makes both pose estimation and subsequent handling more difficult. For these reasons, effective inspection needs representations and algorithms that can handle occlusions, noise, and clutter while maintaining sufficient surface coverage.

Furthermore, anomalies are rare events by definition; this poses a challenge because it is difficult to obtain in advance a complete taxonomy, as anomalies may occur in new forms that have never been seen before. Collecting a sufficient sample of anomalous cases is often unfeasible, since it may take a long time before they appear or, as mentioned, they are simply unpredictable. As a direct consequence, conventional supervised

training suffers from severe class imbalance, and models often fail to generalize beyond the patterns seen during training, making them unable to handle previously unseen anomalies. As a result, methods that learn primarily from non-anomalous data become essential, since normal samples are easy to acquire at scale, and practical systems therefore favor one-class, semi-supervised, or self-supervised approaches.

The main objective of this thesis is to address inspection scenarios with a strong intrinsic 3D nature characterized by viewpoint changes, occlusions and clutter, that make industrial anomaly detection challenging. This objective is pursued through two complementary studies, which are described in detail in the next chapter.

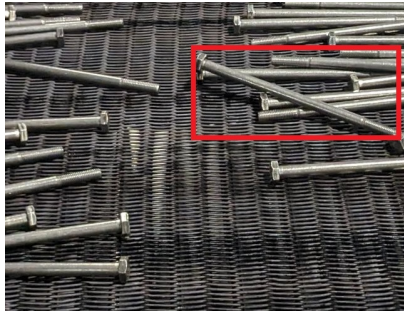


Figure 4.1: Overlapping screws highlighted by the red bounding box.

The first study tackles an anomaly detection problem on a real industrial production line. The target is to identify overlapping screws while they travel on a conveyor. This defect is a logical anomaly: the screws may look intact at the surface level, but their spatial arrangement is wrong from a functional and handling perspective, because two or more pieces partially cover each other. Figure 4.1 highlights a representative anomalous case where the screw heads overlap in the image, revealing an invalid configuration even though no single screw is physically damaged.

The study begins by identifying sensing options for conveyor inspection and selects depth imaging as the primary data source. On the algorithmic side, the pipeline adopts supervised machine vision with modern neural networks for object detection. Training is designed to reflect real industrial conditions, with many normal samples and few overlaps, and the deployed system runs in real time.

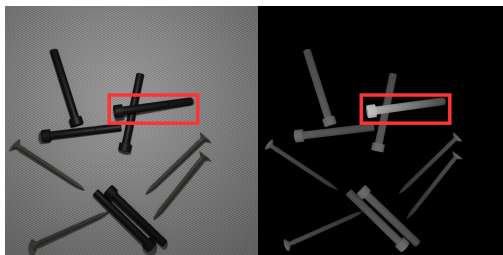


Figure 4.2: RGB (left) and depth (right) views of the same scene.

Once trained, the detector is integrated into a robotic subsystem that manages and resolves the detected anomalies.

In addition, the study examines an alternative sensing option based on RGB images, implementing an RGB-only pipeline to compare against the deployed setup that is based on depth. To enable a fair comparison, a controlled synthetic evaluation environment is built that reproduces the same screw arrangements and poses, while the scene is rendered through different sensor models so each modality yields its own representation of the identical configuration. An example is shown in Figure 4.2.

Finally, comparative experiments are conducted under these matched conditions to evaluate each solution in terms of accuracy, latency, and operational cost, providing a balanced view of overall performance and deployability.

The second work addresses a more cutting-edge setting: pose-agnostic anomaly detection. The underlying motivation is that objects on a production line are free to move in 3D space, so their pose is not known a priori and must be estimated during detection. Anomalies may occur anywhere on the surface and remain hard to localize due to self-occlusions. In this setting, the detector cannot rely on fixed viewpoints or pre-aligned inputs; instead, it must remain robust across rotations and translations and operate under partial visibility.

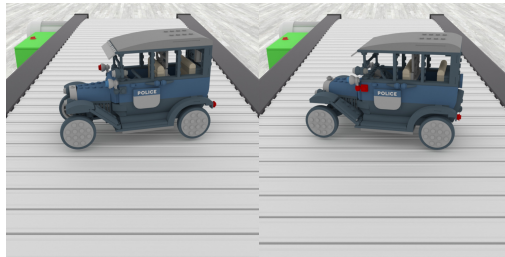


Figure 4.3: Left: non-anomalous item. Right: anomalous item (small red brick).

The problem is addressed in an unsupervised setting that reflects the scarcity of anomalous samples, assuming only normal data are available beforehand. The strategy learns a representation of normality and identifies deviations at test time by measuring discrepancies between feature-similarity maps; to cope with unknown pose, the pipeline includes an explicit pose estimation step.

Although pose-agnostic anomaly detection has been explored in the literature, most available datasets contain objects captured against clean and uniform backgrounds, limiting understanding of how to mitigate false detections caused by realistic scene clutter. To investigate this aspect under more representative conditions, a simulation environment is developed in which target objects appear within complex backgrounds, preserve strong three-dimensionality, and are observed under variable viewpoints reflective of industrial inspection scenarios. An example of the proposed scenario is shown in Figure 4.3, which depicts two items of the same category in different, unpredictable poses, with and without an anomaly.

Finally, a dedicated pipeline is introduced for this setting. It builds on pose-agnostic, unsupervised pipelines and adds components to handle realistic background clutter.

Comparisons are carried out on two identical test scenarios that differ only in the

background. An example is shown in Figure 4.4. Prior methods are evaluated on the clean version, consistent with their original setup, while the proposed pipeline is tested on the cluttered version with the same poses and viewpoints.

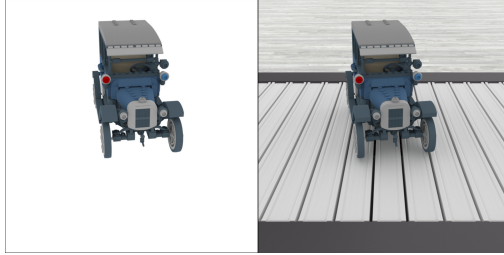


Figure 4.4: Left: item shown without background. Right: item shown with background.

5

Proposed Systems

5.1 Overlapping Screws Detection in 3D and Color Images

This section introduces the first major contribution [9] of this thesis, regarding bin picking for quality-control inspection, applied to a real case study targeting the detection of overlapping screws in a screw manufacturing line. As items move along a conveyor belt during production, instances of “crossing” or “overlapping” screws occasionally occur. The aim of the real system is to detect and locate these overlaps and remove them with a robotic system. A basic solution to this problem involves the usage of a 3D laser profiler to capture a point cloud that describes the 3D geometry of the objects. From this point cloud, the screws’ shapes can be reconstructed, retaining depth information that indicates the distance of points from the profiler. By applying a depth threshold, it is possible to identify blobs that are closer to the profiler, typically corresponding to parts of screws overlapping with others. While this method is simple and effective, it is prone to errors, especially in cluttered or noisy industrial environments, which can interfere with the profiler’s accuracy. Additionally, this approach requires the development of a machine vision algorithm capable of precisely determining the pose of the target screw to enable effective robotic picking. The aim of this work is twofold: first, to investigate software approaches based on neural networks; and second, to explore the usage of different sensors in conjunction with these. To achieve the first goal, the capability of a neural model to accurately recognize and locate overlapping screws in depth images obtained from a laser profiler is verified. The second objective is pursued by training a model on RGB images and comparing its performance with a model trained on depth images, both captured from the same perspective. The RGB approach inherently provides less information since it lacks the depth data, which is useful for detecting overlaps. Indeed, reduced depth indicates an overlap. However, this method is pursued because RGB cameras are generally more affordable than 3D profilers. To conduct experiments for the second step, two synthetic datasets were created using BlenderProc [10]. The first dataset consists of depth and RGB images, where the objects

have identical shapes and sizes. This setup allows a direct comparison between the two modalities under highly controlled conditions. The second dataset, also containing both depth and RGB images, was designed to be more general. It includes images with two different types of screws mixed within the same scene and introduces random variations in lighting positions while maintaining a consistent perspective. This more diverse dataset evaluates the robustness of the approaches when faced with greater variability in object types and illumination. In all synthetic datasets, depth and RGB images are captured from identical perspectives to ensure consistency. The results emphasize the effectiveness of using RGB images despite their lack of depth information. While depth images provide a clear advantage in detecting overlaps due to their ability to capture 3D structure, the RGB-based approach demonstrates competitive performance. This highlights the viability of RGB cameras as a cost-effective alternative to 3D profilers.

Based on available information, no existing studies treat the concept of overlapping or occluding object as a distinct class instance that can be identified by a neural network, as proposed in this work. Instead, most research in this area focuses on related challenges such as handling overlap or occlusion through indirect means. Two key approaches that could potentially address the problem explored in this paper are amodal instance segmentation and relationship reasoning between objects. Amodal instance segmentation, introduced in [11, 12], performs segmentation by identifying visible instances of objects and simultaneously attempting to reconstruct the amodal mask for occluded objects. This mask represents a prediction of the object’s complete shape, including the portions hidden behind occluders. As suggested in [13, 14, 15], this technique could be applied to the task considered in this article; indeed, by detecting the amodal mask for each instance, it is possible to infer areas of overlap and, once the overlap is identified, the relative order of the objects can be reconstructed, determining which object is in front and which is behind. Various datasets have been developed for amodal segmentation, along with solutions for identifying amodal masks. However, these datasets generally consist of collections of common objects with high interclass variation [11], [16, 17, 18, 19, 20]. In contrast, the scenario proposed in this paper involves multiple occurrences of identical objects within the same scene, and it is closer to the work presented in [21]. Many frameworks, including variational autoencoders (VAEs), transformers, Bayesian models, and graph neural networks, have been applied to amodal segmentation [12], [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36]. Some of these methods specifically focus on learning a priori the shape of the objects, which can then be used to reconstruct their full shape, even when parts are occluded. Other works, such as [37, 38, 39], focus on recovering the relative order of objects before performing segmentation, a task extensively discussed in [40, 41, 42, 43, 44, 45, 46, 47, 48]. Specifically, this task involves observing a scene and constructing a graph where each node represents an object, and the edges represent relationships between the objects. For instance, a “parent-child” relationship is established when one object occludes another (‘occluder’ and ‘occludee’), while no connection is drawn between objects that do not overlap. A pioneering work in this area is [42], which introduced an RGB-D dataset along with a solution to address the problem. However, like previous datasets, this one also features high inter-class variation, rather than multiple instances of the same object. Several other works propose different approaches to identify relationships between objects. For example, [40] uses both RGB and depth information to order objects by

analyzing their 3D positional data. On the other hand, methods like [41, 42] rely solely on RGB images. In these approaches, objects are first identified, and their features are compared through a classification module to establish relationships. Works such as [43, 44, 45, 46] employ a similar approach but utilize graph neural networks (GNNs) to model the relationships between objects. In contrast, [47] uses a conditional random field (CRF) to capture the relationships. Finally, [48] leverages both RGB and depth information to segment objects and, based on the area of the detected masks, attempts to infer whether an object is occluded.

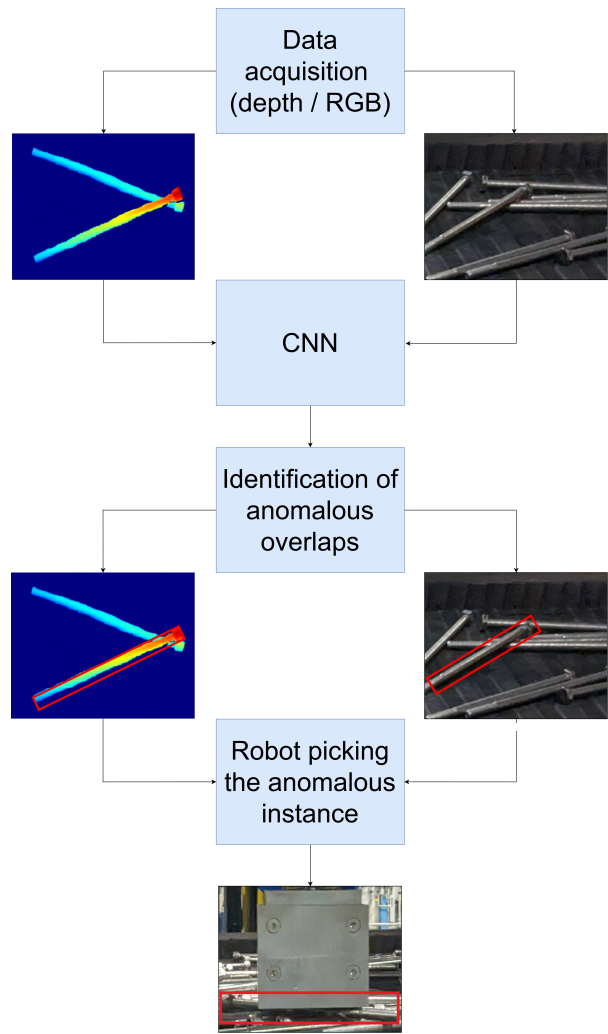


Figure 5.1: Diagram of the proposed solution, illustrated with a running example using both RGB and depth inputs. A colormap has been applied to the depth images for better visualization.

5.1.1 The Plant

The manufacturing plant is equipped with a 3D profiler, a Ruler3000 by Sick, which can sequentially scan surfaces to reconstruct their 3D shapes. The device emits laser beams in response to an external trigger from an encoder mounted on the conveyor, ensuring consistent resolution throughout the process. Each beam is reflected to the profiler by the portion of the surface it strikes. A laser beam spreads along one dimension which means that for every beam the profiler reconstructs a portion of surface arranged along one dimension, and this portion is known as “profile”. To reconstruct the entire surface, it is necessary to move the profiler in the orthogonal direction with respect to the direction of the profiles. Then, by aligning the acquired profiles side by side, the whole surface emerges. Every profile contains a list of values. The number of values is determined by setting the resolution of the profiler, which is the number of points (or pixels) into which the laser beam is divided. In the remainder of the article, this resolution is referred to as the resolution along the X axis. Every value represents the distance between the profiler and the target point of the surface identified by the corresponding pixel. The sequence of profiles is used to reconstruct a bidimensional image of the surface; in particular, every profile is employed to produce a single row of pixel; then a 2D image is built by placing the rows corresponding to the sequence of the profiles one after the other. Throughout the rest of the article, the sequence of profiles is arranged along the Y axis, in particular, the resolution of the 2D image along Y is determined by the time-rate the profiler uses to schedule the emission of the laser beams. The resulting 2D image is a single-channel image, where the value of the pixel, called range, encodes the distance from the profiler to the points in the surface, and it is arranged along the Z axis. The profiler can be configured to provide range values at different resolutions, depending on the required precision. Given that the values of the range can be many orders of magnitude higher than the admitted interval of the value of pixels in standard images (0-255), it is necessary to extrapolate and then rescale a portion of the range to fit it into 0-255. The convention chosen for this work is that points that are closer to the profiler are whiter (higher value of the corresponding pixel) than the points that are more distant.

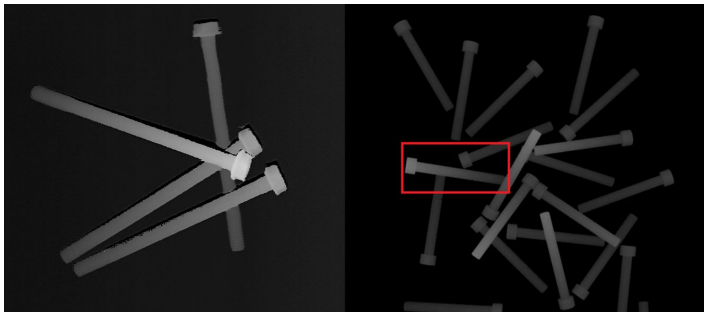


Figure 5.2: (a) Reconstructed depth image of a scene generated using the profiles obtained from the profilometer. (b) The red bounding box highlights a screw in the depth image, which is slightly overlapped by another screw.

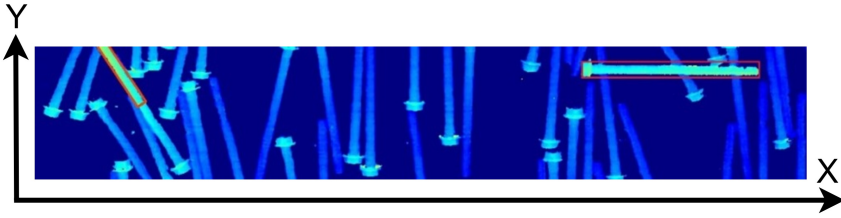


Figure 5.3: Identification of overlaps by thresholding: brighter areas suggest the presence of overlaps. A colormap has been applied to the image for better visualization. The Y-axis represents the direction in which the screws move, corresponding to the conveyor belt's motion, while the X-axis indicates the perpendicular direction across the belt.

Summing up, the resolution of the image corresponds to the resolution along the X and the Y axis; instead, the value contained in every pixel depends on the resolution of the range along Z and on the operations of extrapolation and rescaling executed on it. The process just described is a crucial element of the system architecture, as it enables the encoding of 3D spatial information into a two-dimensional image. The image in Figure 5.2(a) not only captures the surface shape but also retains depth information, as conveyed by pixel intensity values. The manufacturing plant can produce screws of varying sizes and calibers, but for each production batch, these parameters are fixed. This ensures that screws of different dimensions do not appear on the production line simultaneously. During the manufacturing process, screws are transported via a conveyor to an oven. For this process, the screws must be arranged without overlapping. Currently, a machine vision quality control system is employed to meet this requirement. It uses simple thresholding of depth information: if a white blob is detected in the depth image, it is identified as a potential overlap because lighter pixels indicate that those parts are closer to the profiler as in Figure 5.3, where a colormap has been applied for better visualization. An overlap, indeed, suggests that a part of a screw is elevated compared to others. The position of this blob is then communicated to a robotic system, which is responsible for removing the improperly positioned screw. This system works effectively when all screws are of the same size. However, if screws of different sizes are present on the conveyor, setting an appropriate threshold becomes challenging. The overlap of two smaller screws could have the same height as a single larger screw, complicating the identification process. Additionally, this approach can be compromised by spurious values that may appear in images captured in real-world environments, where varying lighting conditions can mislead the sensors during the acquisition process, potentially creating false overlapping blobs. Finally, this approach has another significant drawback: identifying blobs that indicate the parts of screws closest to the profiler does not necessarily provide information about the orientation of the overlapping screw. This makes it difficult to grasp the upper screw without disturbing the properly positioned screws around it. [49] offers a robust solution for identifying optimal grasping points in general environments. However, given that the target screws are positioned higher relative to other items and have a simple, almost convex shape, it is sufficient to identify the complete pose and shape of the overlapping screw. To acquire this information, the approach in this study involves finding the instance segmentation

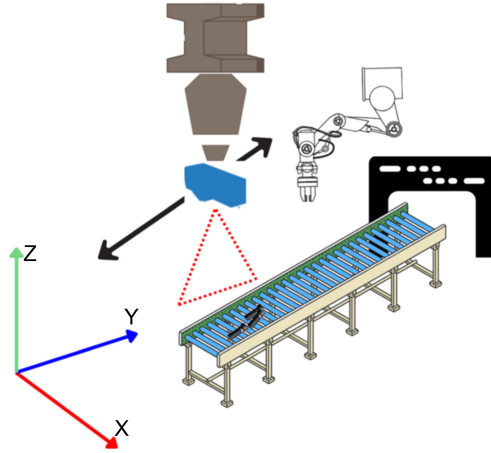


Figure 5.4: Plant diagram.

mask of the overlapping screw. In fact, segmentation has been widely used in related works [50, 51, 52, 53, 54], demonstrating its effectiveness in addressing tasks involving identification in complex scenarios. Furthermore, the segmentation mask allows for easy calculation of the centroid and orientation. The positional information is then passed to the robotic system for precise picking. The robotic system comprises three motorized axes and a rotating electromagnetic end effector. When an electric current flows through it, the end effector becomes a magnet capable of attracting the target screws. For a more precise pick-and-place process, the end effector should be aligned with the orientation of the selected screw. The conveyor has a maximum speed of 6 mm per second and a width of 1200 mm. For clarity, a Cartesian coordinate system is established as follows: the Y axis aligns with the length of the conveyor, extending towards the oven. The X-axis aligns with the width of the conveyor, extending to the right when viewed from the positive Y direction. The Z-axis is orthogonal to the X-Y plane and points upwards. The laser beam creates an isosceles triangle with the vertex at the laser's emission point. The triangle's height is 1445 mm, and the base extends 1680 mm along the X-axis. The profiler is positioned so that the laser vertex is 1200mm above the conveyor, covering a maximum length of approximately 1395 mm when hitting the conveyor. Positioned equidistantly at 600 mm from each side of the conveyor, the laser covers the entire belt width. Along the Z-axis, the pixel values indicate proximity to the laser's emission point. The system removes only one overlapping screw at a time, even if multiple overlaps are detected in a single image. After the robot removes a selected overlap, the system verifies whether the removal has been completed correctly. To do this, since the conveyor moves during the picking process, the profiler is repositioned forward respect the original overlap location along the Y-axis using an electro-mechanical actuator. It then scans the area backward to capture an image which is analyzed by the system, continuing this process until it returns to the original position to confirm whether the overlap has been successfully removed as shown in Figure 5.4.

5.1.2 Datasets

The real dataset was acquired directly from the plant and consists of depth images with a fixed resolution of 4096 x 800 pixels. This equates to an X-axis resolution of 4096 pixels per 1200mm. To ensure that the resolution is consistent along both the X and Y axes, the trigger rate is calculated by accounting for the speed of the conveyor, which is monitored using an encoder. This setup ensures precise synchronization and uniform resolution in the depth images. Each depth image is composed of 800 profiles. The choice of this relatively small number is intentional: it allows the system to quickly scan each image for anomalies and limits the number of anomalies per image. By minimizing the area covered, the system can efficiently handle and process any irregularities as soon as they are detected. For the depth images, the Z resolution is set to 0.018mm per unit. Due to this high precision, the depth values exceed the standard 0-255 range used for 8-bit image representation. To manage this, a normalization process is applied. Initially, depth values corresponding to the conveyor belt are adjusted to zero. Any outliers—such as excessively high values which indicate proximity to the profiler or negative values—are also set to zero. The remaining depth values are then rescaled to fit within the 0-255 range. Since the neural network input requires 1024 x 1024 dimensions, the images were scaled to 1024 x 200 pixels. The initial dataset consisted of 250 unique real images; due to the limited number of samples, data augmentation was applied to increase the dataset to a total of 400 images. Specifically, the 1024 x 200 images were cropped, and the obtained sections were then recomposed while applying a 50% random vertical flip. The composition process ensured no overflow beyond 1024 pixels in width, and if the width was less than 1024, zero-padding was added. Finally, zero-padding was also applied to the top and bottom of the figure to achieve the required 1024 x 1024 size. After obtaining the cropped sections, they were divided into three groups to ensure distinct and non-overlapping data instances for training, validation, and testing phases. The screw instances were distributed as 75% for training, 15% for validation, and 10% for testing. Subsequently, recomposed images for each phase were constructed exclusively using the crops belonging to their respective phases. This approach avoids having identical elements shared across different phases, ensuring a clear separation of data for robust evaluation. The real dataset consists exclusively of depth images. Two synthetic datasets, named Single-Screw Synthetic Dataset and Mixed-Screws Synthetic Dataset, were generated; both comprise 600 RGB images and 600 depth images. Specifically, for every RGB image, there is a corresponding depth image representing the same scene, captured from the same perspective and with identical dimensions and resolution. To construct the experimental configurations, 3D models of screws were used. A single 3D screw model was employed for the Single-Screw Synthetic Dataset, while two different 3D screw models were utilized for the Mixed-Screws Synthetic Dataset to introduce variability. The setup began with the preparation of a plane to serve as the base. This plane was textured to resemble a metallic grid, simulating the appearance of a conveyor belt. A camera was positioned perpendicularly above the centre of this plane to ensure a uniform viewpoint. Lighting was implemented using a point light source that radiates in all directions. For the Single-Screw Synthetic Dataset, the light source was fixed at the centre above the plane. In contrast, for the Mixed-Screws Synthetic Dataset, the point light source was placed randomly within a defined region around the centre of the plane, allowing variations

in position along the x, y, and z axes to increase the generality of the dataset. One kind of screw was given a black opaque texture, while the other received a grey opaque texture to mimic the appearance of real screws. The dataset generation process utilized BlenderProc and its physics simulation capabilities. Screws were randomly positioned above the plane, and the physics simulation allowed them to fall naturally, potentially stacking on top of each other. A boundary or enclosure was introduced to ensure that screws remained within the defined area during the simulation. This enclosure was removed before capturing snapshots, ensuring a clean scene without visual artifacts. For each simulation, up to 30 screws were placed randomly above the plane, and the physics simulation ensured realistic positioning without any intersection between screws. After every simulation, a snapshot was captured from the camera’s viewpoint, generating a pair of RGB and depth images. Then the process was repeated by resetting the setup, running the simulation, and capturing a new snapshot. BlenderProc does not directly return a depth image; instead, it provides depth information that is used to reconstruct the depth map. Specifically, for each pixel in the RGB image, BlenderProc provides the distance from the profiler to the plane that passes through that pixel and is parallel to the base plane. This means that pixels representing points on the same plane parallel to the base plane will have the same distance value. This setup ensures that the distance provided is not the straight-line distance connecting the profiler directly to the pixel. Instead, it accurately represents the perpendicular distance from the profiler to the plane passing through each pixel, ensuring that pixels on the same plane, despite their spatial position, receive the same depth value. After rendering, depth images undergo post-processing. Pixels corresponding to the background (base plane) are uniformly set to a depth value of 0. Conversely, pixels corresponding to objects like screws are rescaled uniformly, up to 255, prioritizing a lighter shade for pixels closer to the camera, as for the real dataset. This adjustment enhances the distinction between foreground objects and the background in the final depth representation. The RGB and depth images in the synthetic datasets are standardized to a resolution of 1024 x 1024 pixels; an example is shown in Figure 5.5.

5.1.3 Experimental Settings

The approach in this work relies on fully neural methods. Figure 5.1 shows the system architecture with a running example. In the real plant, only a laser profiler is available, not an RGB camera; therefore, the deployed system corresponds to the left side of Figure 5.1. The right side illustrates a possible implementation using an RGB camera. Specifically, YOLOv8n-seg by Ultralytics [55] and Mask R-CNN by Torchvision [56], which are models that perform both object detection and instance segmentation simultaneously, were utilized. Both models are described in section 3.4. The decision to compare different neural architectures stems from the need to demonstrate the system’s independence from any single model. These models were used as-is, with adjustments limited to hyperparameter tuning and setting the appropriate number of classes. A challenge present in the real dataset, but absent in the synthetic datasets, is the occurrence of partially visible screws. In some cases, indeed, photos may only capture a portion of a screw in the upper or lower part, continuing into the previous or subsequent image. To address this, a practical solution is to ensure that consecutive images overlap in a common area rather than being strictly sequential. This approach ensures that

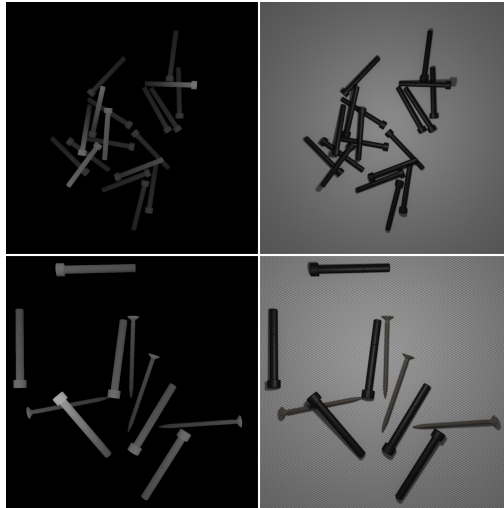


Figure 5.5: Top row:(a) Single-screw synthetic depth image. (b) Single-screw synthetic RGB image. Bottom row: (c) Mixed-screws synthetic depth image. (d) Mixed-screws synthetic RGB image.

a screw appearing truncated in one image should appear complete in the overlapping area of the next image. However, due to the absence of strictly consecutive common areas, achieving this was not possible. Therefore, truncated screws cannot be entirely discarded. Consequently, in the real case, the models were trained to detect incomplete overlapping parts as well. Another challenge involves scenarios where screws that overlap others are themselves overlapped, creating a complex hierarchy of interactions. To address this, screws are categorized as 'topmost screws', which overlap others without being overlapped, and 'intermediate screws', which both overlap and are overlapped. Screws that are either fully overlapped without overlapping others or completely free (neither overlapping nor overlapped) are treated as background and are not explicitly detected. The difficulty with intermediate screws lies in picking them without interfering with the screw above. A potential solution, as suggested in [49], involves identifying optimal grasping points. However, considering the practical objective of this work, which involves using a magnet aligned with the screw's pose, the focus shifts to detecting and distinguishing topmost screws from intermediate screws, prioritizing the removal of the former before addressing the latter.

For simplicity, this distinction was not implemented, like the real plant, which removes both without distinction. Regarding the synthetic datasets, both neural models were applied to the depth and RGB synthetic images, resulting in a total of eight experimental combinations. Unlike the experiments on the real dataset, this setting distinguishes between topmost and intermediate screws by introducing two distinct classes. The first class is the most critical; high precision and recall are essential for this class because the real system removes one screw at a time. By accurately identifying the position of these screws, an intermediate screw may no longer be overlapped after the screw above it is removed. For the second class, precision is the most important metric,

as it is crucial to ensure that these screws are not mistakenly identified as topmost. This prevents the system from attempting to remove them, allowing it to prioritize the removal of topmost screws. Following this solution, another important issue emerged: it was sometimes difficult to determine intermediate screws. This confusion arose particularly when a screw overlapped another screw by a very small margin Figure 5.2(b). In such cases, although the bottom screw is technically overlapped, it is almost not occluded. From a practical standpoint, it is not significant whether these screws are categorized within the first or second category. If classified in the first category, they are easy to pick because they are almost not overlapped. Conversely, if classified in the second category, they should not be considered for picking. Eventually, after the overlapping screw is removed, the bottom screw will be identified as a truly non-overlapped screw and can then be correctly removed. However, for the sake of fairness, these screws were assigned to one of the two classes and included in the statistical calculations, even though practically it does not matter if they are assigned to the wrong class. One motivation for using a segmentation network instead of a detection network is particularly evident in cluttered scenarios where conveying accurate information about the target can be challenging. In such cases, it is common for multiple targets to appear nearly completely within a single bounding box.

5.1.4 Neural Network Models

The chosen model for YOLO is the YOLOv8n-seg (Nano), using the implementation provided by Ultralytics for Python, accessible via their CLI. A general overview of the network can be found in the paper [57]. Training was conducted for 500 epochs with a patience of 100 epochs. During both training and test, to enhance mask quality, the `retina_masks` parameter was set to `True`, the image size was maintained at 1024 (`imgsz = 1024`), corresponding to the original image resolution and `mask_ratio` was set to 1. The model was not pretrained (`pretrained = False`), and a batch size of 32 was used during training. All other parameters were kept at their default values as defined in the standard Ultralytics implementation [55]. The Mask R-CNN architecture utilized in this study is based on the implementation provided by Torchvision and described in [56]. It features a backbone network (ResNet-50) with a Feature Pyramid Network (FPN)[58] constructed from four intermediate layers of the backbone (C2 to C5). The FPN produces five levels of output (P2 to P6) that feed into five Region Proposal Network (RPN) heads. These RPN heads extract features and predict both the presence or absence of objects and the adjustments to the anchor box coordinates (bounding box refinement). The region proposals generated by the RPN are subsequently processed through the MultiScale RoIAlign layer, which extracts fixed-size feature maps from the appropriate level of the FPN. These feature maps are then passed to three distinct heads within the ROI heads module: a classification head that predicts the class label for each proposal, a bounding box regression head that refines the coordinates of the bounding boxes and a mask prediction head which outputs binary masks for each object class. The model was trained for 500 epochs without using pretrained weights. Dropout (0.5) was added to the fully connected layers of the classification and bounding box regression heads within the ROI heads module to improve generalization. The `image_mean` and `image_std` values, specific to each dataset, were computed and used to normalize the input images. The optimizer used was AdamW, with a learning rate of 0.001 applied to

the RPN and ROI heads and 0.0001 applied to the backbone. The batch size was set to 4. Additionally, a degree of randomness was introduced during training via the application of a RandomHorizontalFlip augmentation, which was applied with a probability of 0.5. The best epoch for the model was selected based on validation set performance.

5.2 Unsupervised pose-agnostic visual anomaly detection in realistic industrial scenes

The work exposed in this section refers to the work [59] and is situated within a broader body of research [60, 61, 62, 63, 64, 65, 66, 67, 68] that integrates modern machine learning and advanced computational methods into real-world engineering systems, enabling effective and scalable solutions for practical applications. Specifically, recent research efforts increasingly focus on developing innovative approaches for visual anomaly detection, aiming to identify structural defects through deep learning and machine vision techniques. Within this context, the second major contribution of this thesis is introduced, focusing on pose-agnostic anomaly detection.

Recent years have witnessed a growing interest in unsupervised 3D anomaly detection, driving the creation of increasingly challenging benchmarks. The first widely adopted dataset, MVTech 3D-AD [69], consists of single-view RGB-D images, where the depth maps can be uniquely converted into point clouds. This was followed by Eyecandies [70], a synthetic dataset providing both single-view RGB-D images and complementary 3D information in the form of object normals, and PD-Real [71], designed to better approximate the complexity of real-world scenarios. Early benchmarks generally offered only partial object views. Real3D-AD [72] marked a major step forward by providing point cloud data capturing the complete object geometry without blind spots. However, it contains only structural information, while 3D-ADAM [73] augments point clouds with RGB data, enabling more realistic industrial scenarios. Similarly, Anomaly-ShapeNet [74] includes complete point clouds reconstructed from original ShapeNet models. Other dataset like MAD [75], Real-IAD [76], and RAD [77] achieve full object coverage through multi-view RGB images, with the latter two addressing more challenging real-world conditions. Notably, MAD was designed for pose-agnostic anomaly detection (PAD), where the object pose is unknown at test time and must be estimated before anomaly detection. PIAD [78] further extends this setup by introducing illumination changes alongside unknown poses, significantly increasing task difficulty. Furthermore, the PCAD dataset [79] bridges the gap between synthetic and real-world benchmarks by combining real multi-view RGB images with corresponding CAD models, enabling more realistic scenarios.

In parallel, numerous unsupervised approaches have been proposed. Since anomalies are absent from the training set, these methods learn a model of normality solely from normal samples, classifying any deviation at inference time as anomalous. A prevalent strategy is reconstruction-based detection, where models are trained to reconstruct normal inputs but fail to accurately reproduce anomalies. Among these, student-teacher architectures are widely used: students learn to match the teacher’s outputs for normal data, and deviations between the two emerge naturally when processing anomalies, and this scheme is adopted in works such as [80, 81, 82]. Other methods, including

[74, 83, 84], also employ reconstruction but differ in their design: [74] uses a transformer to reconstruct patches extracted from point clouds; [83] applies an encoder-decoder network tailored for point clouds to the same task; [84] reconstructs the entire point cloud directly, but here anomalies are inter-class rather than intra-class. Like [81], works such as [85, 86, 87, 88] also attempt to leverage RGB information by fusing it with depth data. In [89, 90, 91, 92, 93], the architecture generates simulated anomalies on normal images for training purposes; it then learns to reconstruct the normal appearance from the anomalous image, and at inference time employs the extracted features to drive the anomaly detection process using the reconstructed normal object as a guide. Similarly, [94] generates anomalies during training by locally deforming the point cloud and learning to detect deviations from the object’s normal shape. Works such as [74, 95, 96, 97, 98, 99, 100] adopt a different approach based on the paradigm introduced by PatchCore, in which features from normal images are extracted during training and stored in a feature dictionary, referred to as a memory bank. At test time, features extracted from the query item are compared against the memory bank, and an anomaly score is assigned based on their distance from the stored features.

With the paper [101], the authors introduced a powerful technique for 3D scene modeling called Gaussian splatting. This method leverages RGB images with known camera poses and, through optimization, learns a point cloud-based representation composed of primitives called splats. Using the learned model, it becomes possible to render images from arbitrary viewpoints by exploiting the information provided by the most relevant splats for that pose. Each splat represents the local 3D geometry of the scene, and the combination of neighboring splats enables precise reconstruction of the overall scene. Since its introduction, Gaussian splatting has been applied to the PAD problem. The first approaches, SplatPose [102] and IGSPAD [103], leverage the differentiability of Gaussian splatting to design a pose estimation module that solves an optimization problem with respect to the pose. In contrast, SplatPose+ [104] departs from this idea and estimates object poses at test time via PnP-based optimization.

This section outlines the second major contribution of this thesis, building on the aforementioned literature. In the proposed work, the experimental setting was synthetically constructed to simulate an industrial inline inspection scenario, with objects moving along a conveyor and exhibiting varied, unpredictable poses, resulting in images with realistic backgrounds that closely resemble an actual production environment. Regarding illumination, a uniform lighting condition was assumed, as illumination-invariant detection lies outside the scope of this work. Nevertheless, the proposed synthetic dataset remains sufficiently challenging, as it includes substantial background clutter. The aim is to identify surface anomalies that can occur at arbitrary locations on the objects. The training set is assumed to contain only non-defective samples, while anomalies are artificially introduced as small and subtle defects, to evaluate the robustness of the proposed methodology under realistic and complex conditions. Furthermore, defect detection is performed over the whole potentially visible surface of each object, excluding the bottom area that remains in contact with the conveyor.

The proposed anomaly-detection pipeline builds upon the SplatPose+ framework and leverages the Gaussian splatting technique [101], which is formalized in the next paragraph. The subsequent paragraphs briefly summarize SplatPose+ and then describe in detail the proposed system and the introduced dataset.

5.2.1 Gaussian Splatting

Gaussian splatting is a 3D scene representation technique that reconstructs a model from 2D images with known camera poses, enabling the synthesis of new views from arbitrary viewpoints. Its main advantages are fast rendering and full differentiability, which make it suitable for efficient optimization and novel-view generation. The following description is based on the work [105].

Splats are the basic primitives, stored as a sparse point cloud. Each splat is defined by a 3D spatial extent, color, and opacity. The spatial extent is modeled as a 3D Gaussian density distribution in space. Its mean μ corresponds to the 3D position of the splat, stored as 3D coordinates in the point cloud, while its shape and orientation are encoded in the covariance matrix. The covariance matrix is parametrized as

$$\Sigma = RSS^T R^T,$$

where R is a 3×3 real rotation matrix and S is a real 3×3 scaling matrix. This parametrization derives from the singular value decomposition (SVD) of a real square matrix, which decomposes a matrix into the product of a rotation, a scaling (SS^T), and another rotation. In this setting, $S = \text{diag}(s)$, where s^2 is a vector that contains the three singular values. These singular values correspond to the square roots of the eigenvalues of $\Sigma \times \Sigma^T$.

Geometrically, this can be interpreted as a 3D ellipsoid. The matrix R defines the orientation of the ellipsoid in 3D space, while the singular values represent the magnitudes of its semi-axes. Therefore, the singular values determine the anisotropic spread of each splat along its principal directions.

Since R is a rotation matrix, it can be represented through a quaternion parameterization. Given a normalized quaternion $q = (u, x, y, z)$, the corresponding rotation matrix R can be expressed as

$$R = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - uz) & 2(xz + uy) \\ 2(xy + uz) & 1 - 2(x^2 + z^2) & 2(yz - ux) \\ 2(xz - uy) & 2(yz + ux) & 1 - 2(x^2 + y^2) \end{bmatrix}.$$

From this description, the parameters of the model that represent the position and extent of each splat are: the splat location μ , the scaling vector s and the rotation quaternion q .

The remaining parameters of each Gaussian splat are the opacity α , which ranges from 0 (completely opaque) to 1 (fully transparent), and the color.

To represent color, spherical harmonics (SH) are employed which are a differentiable basis for approximating functions of the form $f : \mathbb{S}^2 \rightarrow \mathbb{R}$ where $\mathbb{S}^2$ is the unit sphere. This choice enables the modeling of view-dependent appearance rather than assigning a single constant color to each splat. Indeed, since the appearance of a point on a surface may vary with the viewing direction, representing color as a function on $\mathbb{S}^2$ allows a splat to change its color depending on the viewpoint. Thus, each RGB channel is modeled as a function $c : \mathbb{S}^2 \rightarrow \mathbb{R}$.

In particular, given a viewing direction $d \in \mathbb{S}^2$, the corresponding color value is obtained by evaluating the spherical harmonic expansion at d :

$$c(d; \mathbf{k}) = S \left(\sum_{\ell=0}^{\deg_{\max}} \sum_{m=-\ell}^{\ell} \mathbf{k}_{\ell}^m Y_{\ell}^m(d) \right),$$

where $\deg_{\max}$ is the maximum SH degree employed, $Y_{\ell}^m(\cdot)$ are the real spherical harmonic basis functions and $\mathbf{k} = (\mathbf{k}_{\ell}^m)_{0 \leq \ell \leq \deg_{\max}, -\ell \leq m \leq \ell}$ are the learnable SH coefficients. The function $S : \mathbb{R} \rightarrow [0, 1]$ denotes the sigmoid, which is applied to constrain each RGB channel to the valid range $[0, 1]$.

The optimal values of $\boldsymbol{\mu}$, $\mathbf{s}$, $\mathbf{q}$, $\mathbf{o}$ and $\mathbf{k}$ are estimated during the optimization process used to construct the 3D Gaussian splatting model.

In Gaussian splatting, each pixel p_i is obtained by front-to-back alpha compositing of the splats' colors $\{c_j^i\}_j$ weighted by their opacity parameters $\{\alpha_j^i\}_j \in [0, 1]$ and the residual transmittance $\{Tr_j^i\}_j \in [0, 1]$:

$$p_i = \sum_{j=1}^{N^i} Tr_j^i \alpha_j^i c_j^i, \quad (5.1)$$

where N^i is the number of splats involved in pixel i and

$$Tr_j^i = \prod_{k < j} (1 - \alpha_k^i). \quad (5.2)$$

Intuitively, Tr_j^i measures the residual mass of the pixel i available to be filled after the accumulation of the first $j - 1$ contributions.

In the previous formulas, the opacity parameters $\{\alpha_j^i\}_j$ are obtained multiplying the learnable opacities $\{o_j^i\}_j$ by the Gaussian weights $\{w_j^i\}_j$ associated with the i^{th} pixel so as to account for the spatial weighting of each splat. To compute these weights, for each splat the associated 3D gaussian distribution is projected onto the image plane as a 2D gaussian distribution. During the 2D projection stage, splats are filtered in clip space to discard those that are too far or too close to the camera plane. The overall process is described by the following equations.

T_{cw} is the extrinsic matrix that operates from world space to camera space, and P is the projection matrix that operates from camera space to normalized clip space. The parameters f_x, f_y, c_x, c_y are the focal lengths and the coordinates of the principal points respectively; w, h , are the output image width and height and n, f are the near and far clipping planes depths. These parameters are used to build:

$$T_{cw} = \begin{bmatrix} R_{cw} & t_{cw} \\ 0 & 1 \end{bmatrix} \in SE(3), \quad P = \begin{bmatrix} \frac{2f_x}{w} & 0 & 0 & 0 \\ 0 & \frac{2f_y}{h} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Then, these transformations are applied to yield μ' , which is the mean of the 2D

projected Gaussian distribution:

$$t = T_{cw} \begin{bmatrix} \mu \\ 1 \end{bmatrix}, \quad t' = Pt, \quad \mu' = \begin{bmatrix} \left(w \cdot \frac{t'_x}{t'_w} + 1 \right) / 2 + c_x \\ \left(h \cdot \frac{t'_y}{t'_w} + 1 \right) / 2 + c_y \end{bmatrix}.$$

Instead, the covariance matrix of the 2D distribution Σ' is obtained as:

$$\Sigma' = JR_{cw} \Sigma R_{cw}^\top J^\top,$$

where,

$$J = \begin{bmatrix} \frac{f_x}{t_z} & 0 & -\frac{f_x t_x}{t_z^2} \\ 0 & \frac{f_y}{t_z} & -\frac{f_y t_y}{t_z^2} \end{bmatrix}.$$

Then, every weight w is obtained as

$$w = \exp(-\sigma), \quad \sigma = \frac{1}{2} \Delta^\top \Sigma'^{-1} \Delta,$$

where Δ is the offset between the pixel and μ' .

The procedure above describes the computation for a single pixel. In practice, the entire rendering is parallelized using CUDA. The image is partitioned into 16×16 tiles, which are processed independently and in parallel.

Before rendering, each splat is checked for whether it can contribute to the tiles. The projected 2D Gaussian of the splat is used, taking its 3σ contour as an effective radius, covering about 99.7% of the mass). A bounding rectangle that encloses this 3σ area is then computed on the image plane, and the splat is assigned to a tile whenever this rectangle overlaps the tile. All splats assigned to a tile are instantiated for that tile and sorted in parallel by depth (front to back). The per-pixel accumulation of Gaussian contributions is then performed within each tile to produce the final image.

5.2.2 Pose Estimation and Anomaly Detection

In SplatPose+, the 3D representation of non-anomalous models is learned through Gaussian splatting.

To initialize the Gaussian splatting model, a preliminary 3D reconstruction is obtained using Structure-from-Motion (SfM). SfM produces a sparse point cloud representation of the object by exploiting feature correspondences across multiple RGB images captured from different viewpoints. Specifically, distinctive features are first extracted from the images, matched across views, and then triangulated to recover their 3D positions. Since the intrinsic parameters and camera poses are known, the reconstruction is metric and scale-consistent. The generated SfM point cloud thus provides an approximate yet reliable representation of the object's geometry. This point cloud, which contains only 3D positional information, is then used to initialize the Gaussian splatting model by assigning each 3D point as the initial position of a splat. Subsequent optimization refines these splats, enriching them with the additional attributes such as

color, opacity, and spatial extent.

In addition, SplatPose+ leverages the SfM reconstruction also for estimating the poses of the test images. Specifically, during inference, correspondences are established between 2D feature points extracted from the test image and their associated 3D points in the SfM model. Given these 2D–3D matches, the camera pose is estimated by solving a Perspective-n-Point (PnP) optimization problem.

Once the pose of the test image has been estimated with respect to the Gaussian splatting model, anomaly detection can be carried out directly in the image domain. A synthetic view of the object is rendered from the same viewpoint using the learned Gaussian splatting model, which represents the normal, defect-free appearance. This rendered image serves as a reference and is compared with the corresponding real test image. Both images are processed through a feature extraction backbone, and the resulting feature maps are analyzed to measure their discrepancy. Regions showing significant differences between the two sets of features indicate deviations from normality, allowing the detection and localization of surface anomalies.

5.2.3 Methodology

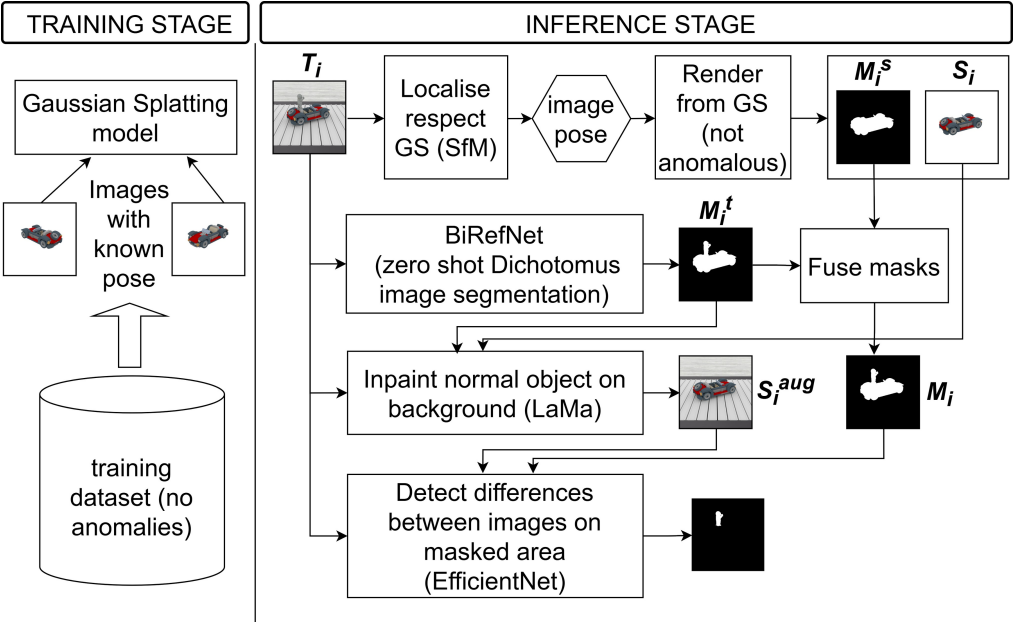


Figure 5.6: Proposed architecture.

In this work, particular attention is devoted to addressing the pose-agnostic anomaly detection problem under conditions that more closely resemble real industrial environments. The main challenge arises from background elements in the test images. Because objects have no fixed pose, neither their pose is known a-priori nor is their arrangement with respect to the background consistent across different objects. As a result, training

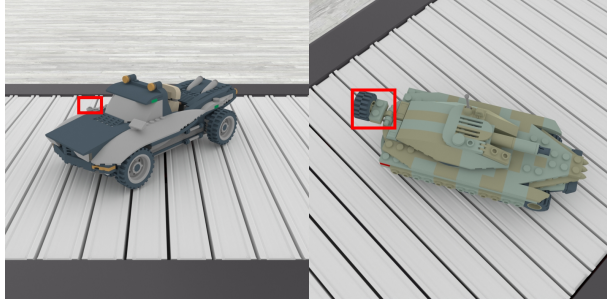


Figure 5.7: Left: (a) Removed part (mirror) visible against the background (red box). Right: (b) Added defect visible against the background (red box).

a Gaussian splatting normal model that includes background content and comparing it to a test image with background is unreliable: even when the object view matches, the background pose does not, and vice versa. To mitigate this, the processing pipeline aims to remove the influence of the background. Accordingly, the Gaussian splatting model is trained on images with a null background, represented by a uniform white surface, as in previous published works. To ensure a consistent comparison between rendered and test images, the test background must be suppressed. This requires an accurate object segmentation and restricting the comparison to the object mask, thereby eliminating background influence.

As will be explained in the dataset section, anomalies can involve missing, modified, or added parts. In the case of missing parts, detection must focus on the region where the part should be. The object mask from the test image is therefore insufficient, because it can exclude the removed part, as would happen in the example shown in Figure 5.7(a) where the removed part is above the background. Then, it is necessary to also use the object mask from the synthesized normal image, which delineates the expected extent of the part, so that the corresponding area can be analyzed.

On the contrary, for added parts, relying only on the synthesized normal mask would miss the anomalous region, since anomalies can lie outside the expected support, like in the example shown in Figure 5.7(b); the test-image object mask must therefore be used to include those extra areas. In practice, the union of the two masks is analyzed: regions present only in the normal mask indicate missing parts, regions present only in the test mask indicate added parts, and discrepancies within the overlap correspond to modified parts.

Formally, let M_i^s denote the object mask in the synthesized view S_i and M_i^t the object mask in the test image T_i . In the proposed example Figure 5.8(a) corresponds to T_i , while Figure 5.8(b) corresponds to S_i . The mask M_i^s is obtained by thresholding the final residual transmittance Tr_i of the synthesized image. In particular, for background pixels the residual transmittance map satisfies $Tr_i \approx 1$, as rendering yields negligible splat contribution; in contrast, for pixels depicting the object, $Tr_i \ll 1$, because these pixels are saturated by the accumulated contributions of the splats. Therefore, a clean foreground–background separation for the synthesized image is obtained by thresholding Tr_i .

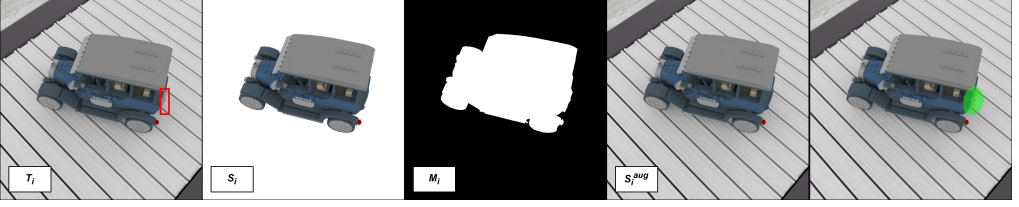


Figure 5.8: From left to right: (a) test image with the anomaly highlighted (box); (b) synthetic normal image; (c) foreground mask; (d) inpainted normal object on the reconstructed background; (e) anomaly detection image.

To obtain M_i^t , a BiRefNet[106] architecture is used. BiRefNet is a zero-shot model for dichotomous image segmentation, designed to separate foreground from background. Then M_i^s and M_i^t are fused by a pixel-wise logical OR to obtain the final mask $M_i = M_i^s \vee M_i^t$, which specifies the region over which anomaly detection is performed when comparing the test and synthesized images. In the proposed example, Figure 5.8(c) corresponds to M_i .

This is still not sufficient, as residual errors depend on mask accuracy. In particular, if the mask spills beyond the foreground object, the comparison will include background regions, which may be detected as false anomalies. To mitigate this issue, an inpainting strategy is applied to suppress background content within the comparison region. Specifically, the test image T_i is processed using the LaMa model [107], which removes the object and reconstructs the underlying background. The inpainting region is defined by the mask M_i^t . Once background reconstruction is complete, the anomaly-free object from the synthesized image S_i is extracted using M_i^s and composited onto the reconstructed background, yielding S_i^{aug} . In the proposed example Figure 5.8(d) corresponds to S_i^{aug} .

Finally, the anomaly detection is performed between T_i and S_i^{aug} restricted to the region defined by M_i . Figure 5.8(e) shows the detected anomaly in the proposed example. The whole pipeline is shown in Figure 5.6.

5.2.4 Implementation

Following the SplatPose+ pipeline, an SfM model is built both to initialize Gaussian splatting training and to localize test images. For each training image, image retrieval is performed using NetVLAD[108] to select the K closest images within the training set to form overlapping pairs. On these pairs, SuperPoint[109] extracts local features and LightGlue[110] performs feature matching. The matched training pairs are triangulated with hloc[111] to obtain a sparse SfM model of the scene. At test time, for each image, NetVLAD retrieves the top- K most similar reference images from the training set. Then, SuperPoint and LightGlue are applied between the query and the retrieved references to obtain 2D correspondences. The correspondences are then lifted to 2D–3D by associating keypoints in the test image with 3D points from the SfM model. Finally, hloc solves a PnP problem from these 2D–3D matches to recover the camera pose of the test images.

The Gaussian splatting model is trained for 15000 iterations with a densification interval of 1000 iterations and the spherical harmonics degree is set to 3.

Anomaly detection is performed following the same general paradigm as OmniposeAD [75], where features are extracted from both the synthesized reference and the test image using a pre-trained EfficientNet backbone [112]. Let $\mathbf{x}^{\text{ref}}$ and $\mathbf{x}^{\text{test}}$ denote the reference and test images, respectively. Both images are first resized to a spatial resolution of 224×224 and normalized, and this input level is denoted as

$$F_0^{\text{ref}}, F_0^{\text{test}} \in \mathbb{R}^{3 \times 224 \times 224},$$

which corresponds to the RGB image representation itself. Subsequently, the EfficientNet backbone produces four additional feature maps at progressively coarser resolutions:

$$\begin{aligned} F_1^{\text{ref}}, F_1^{\text{test}} &\in \mathbb{R}^{24 \times 112 \times 112}, \\ F_2^{\text{ref}}, F_2^{\text{test}} &\in \mathbb{R}^{32 \times 56 \times 56}, \\ F_3^{\text{ref}}, F_3^{\text{test}} &\in \mathbb{R}^{56 \times 28 \times 28}, \\ F_4^{\text{ref}}, F_4^{\text{test}} &\in \mathbb{R}^{160 \times 14 \times 14}. \end{aligned}$$

Here, the first dimension corresponds to the number of feature channels, while the last two dimensions represent the spatial resolution (height and width) of the feature maps.

For each level ℓ , a pixel-wise discrepancy map between the test and reference features is computed using an L_1 distance, which is less sensitive to noise than the L_2 metric adopted in OmniposeAD. Specifically, for each spatial location (i, j) , the discrepancy is defined as

$$A_\ell(i, j) = \sum_{c=1}^{C_\ell} |F_\ell^{\text{test}}(c, i, j) - F_\ell^{\text{ref}}(c, i, j)|,$$

where C_ℓ denotes the number of feature channels at level ℓ . This yields a single-channel anomaly map

$$A_\ell \in \mathbb{R}^{H_\ell \times W_\ell}.$$

Each map A_ℓ for $\ell = 1, \dots, 4$ is then upsampled to the input resolution 224×224 to match the resolution of A_0 . These upsampled maps are denoted by $\tilde{A}_\ell \in \mathbb{R}^{224 \times 224}$. The multi-scale information is aggregated by a pixel-wise sum across all levels:

$$A(i, j) = \sum_{\ell=0}^4 \tilde{A}_\ell(i, j),$$

yielding a single-channel anomaly map $A \in \mathbb{R}^{224 \times 224}$ in the image domain. This map is further processed by Gaussian smoothing and min-max normalization to obtain the final pixel-wise anomaly map. The image-level anomaly score is finally defined as the maximum value of this map:

$$s = \max_{i,j} A(i, j),$$

which quantifies the most anomalous region within the image.

For background reconstruction with LaMa, the inpainting region is obtained by dilating the mask M_i^t by 5 pixel. Indeed, dilating the mask avoids operating exactly on the object boundary, where recovering the underlying background is more difficult. After



Figure 5.9: Left: (a) Small added defect (box). Right: (b) Added defect visible against the background (box).

reconstruction, anomaly detection compares T_i with S_i^{aug} . In principle, this comparison could be performed over the entire image, since the anomaly-free object is composited onto the reconstructed background. However, because the inpainted region is dilated and the reconstruction, while generally accurate, may still contain local imperfections, evaluating everywhere could introduce false anomalies due to inpainting artifacts. To minimize such effects, the comparison is restricted to the minimal necessary region, namely the mask M_i .

5.2.5 Dataset

The dataset used in this work was developed in Blender [113], which enables the creation of consistent industrial scenarios.

Following the approach introduced in the MAD dataset, the proposed dataset was built using LEGO-based models specifically designed by expert modelers, who provided the original, defect-free base structures. Nine distinct vehicle models were employed which are police car, coupe car, amphibious car, off-road car, tank, desert car, truck, loader and racing car; these models were chosen to represent objects with increasing structural complexity. While simpler models such as the coupe exhibits compact and mostly closed shapes, the amphibious car, the tank, the truck, the loader, the off-road car, the racing car, the desert car introduce more intricate geometries, and the police car model features a high number of detailed parts and open regions. This progression from simple to complex shapes not only increases the number of components but also introduces concave and perforated structures, where background visibility through small openings may occur, potentially leading to false positives in anomaly detection. Although LEGO bricks offer a clear advantage in terms of ease of constructing a wide variety of models, their main drawback is that the bricks have fixed shapes and colors, which limits their ability to represent the true complexity of the real world. Nevertheless, complex models can still be built, helping to support good generalization.

For each class, corresponding to one vehicle model, 100 defect-free training images were rendered with known intrinsic and extrinsic camera parameters defining the pose P . The camera viewpoints were distributed over the visible upper hemisphere of the object which is the part not in contact with the conveyor. Additionally, 13 further objects per

class were generated for testing and each of them was placed in a random pose distinct from the pose of the training object. Seven of them were modified by adding small, subtle surface defects to simulate realistic flaws. The considered defects fall into the categories of additions, removals, and color alterations, which are implemented by respectively adding, deleting, or replacing LEGO components. Specifically, for each class, two of the modified objects exhibit color changes, two present removed components, and three include added components. The remaining six objects were left intact, representing non-defective items with unseen orientations; this setup was intended to simulate a realistic production scenario in which normal items are also inspected and should not be classified as anomalous. For each test object, 9 RGB images at 800×800 resolution were captured with unknown poses, and therefore a total of 117 test images were obtained for each class.

The modifications were intentionally kept minimal with respect to the overall object volume, so that the proposed method can be evaluated under challenging conditions. An example is shown in Figure 5.9(a). Moreover, some anomalies were made larger, especially in the case of added parts. This choice was made to account for perspective effects: depending on the viewpoint, anomalies may appear either over the object or over the background as in Figure 5.9(b). By introducing sufficiently large anomalies, it becomes more likely to include cases where the added parts appear on the background rather than always overlapping the object itself. Indeed, if anomalies overlap just the object and not the background, their detection is simplified, as background clutter is eliminated.

During rendering, the test images included a realistic background, whereas the final training images were rendered on a white background. This choice allowed the construction of reference normal models representing only the object of interest, without the background, which would be unnecessary and possibly misleading for the anomaly detection stage. However, to maintain consistent illumination between the training and test data, all renders were produced within the same scene and lighting setup; the training objects were subsequently cropped and placed on a white background, which is a reasonable assumption since training data can be pre-processed offline. The background scene consisted of a conveyor model and a textured floor, designed to avoid monochromatic regions while keeping the setup minimal. The conveyor was modeled as a series of metallic plates, providing a moderately complex yet regular texture. Moreover, for each test object, the conveyor position was kept fixed across its rendered views but slightly shifted between different objects, ensuring that the background pattern remained consistent within the same object while varying across samples. This promotes generalization and reduces background bias.

6

Results

6.1 Overlapping Screws Detection in 3D and Color Images

The experiments on overlapping screw detection were conducted using distinct datasets, each comprising 2500 instances of overlapping screws. Each dataset was divided into training, validation, and test sets with proportions of 75%, 15%, and 10%, respectively. The synthetic datasets exhibit a class distinction between intermediate and topmost screws, with 14% classified as intermediate and 86% as topmost. In contrast, the real dataset does not include such a distinction between screw classes. This configuration, with fewer intermediate screws, is common in scenarios where screws are primarily laid out without overlaps, as observed in the real-world setup. It is important to note that for each intermediate screw, there must be a topmost screw covering it and, given that scenes featuring more than three screws stacked on top of each other are rare, the dataset naturally tends to have fewer intermediate screws. All the experiments were run on a NVIDIA Tesla P40. The metric chosen to evaluate and compare the performance of the models is the mean average precision (mAP) with Intersection over Union (IoU) thresholds ranging from 0.5 to 0.95 in increments of 0.05, denoted as $\text{mAP}@[.5:.95]$. Average Precision (AP) is macro-averaged across the two classes (topmost and intermediate screws). In the case of the real experiments, there is no need for macro-averaging since the dataset contains only one class. This metric is automatically provided by MaskRCNN through Torchvision and by YOLO through Ultralytics. Aside from $\text{mAP}@[.5:.95]$, mAP50 is employed to assess model performance in the case of the real dataset. Finally, in the case of synthetic datasets, to provide a clearer understanding of the results, the following subsection "Performance on Test Data with Optimized Hyperparameters" presents the performance of the models on their test sets after having identified the optimal hyperparameters on the validation set, which maximized the F1 score. The hyperparameters considered include the model confidence threshold and the IoU threshold used for non-maximum suppression to select the final predictions. To calculate the F1 score, precision and recall were first computed. This required determining true pos-

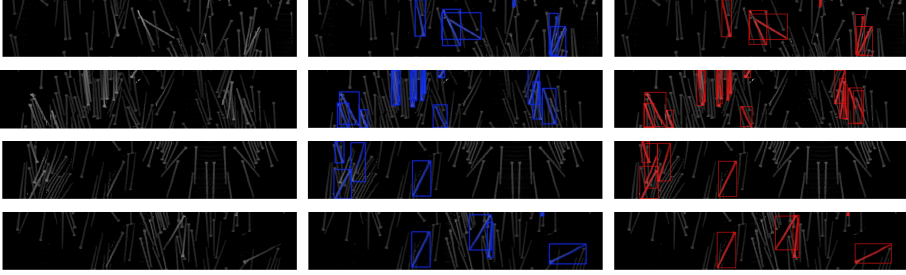


Figure 6.1: Four comparisons of YOLO (blue) and MaskRCNN (red) on the real dataset.

itives, false positives, and false negatives. Following the guidelines in [114], in cases where multiple bounding boxes are present, a match is established between a ground truth bounding box and the predicted bounding box with the highest IoU. The IoU threshold used to match the predicted bounding boxes with the ground truth is set to 0.5, meaning that a predicted bounding box is considered a match if its IoU with a ground truth bounding box is greater than 0.5.

6.1.1 Performance on Real Dataset

Table 6.1 and Table 6.2 report the models' performance in terms of $\text{mAP}@[.5:.95]$ and inference time on the real dataset. At $\text{IoU} = 0.5$, YOLO achieves a mAP_{50} of 0.97, while Mask R-CNN attains 0.95. These results indicate that the system effectively detects overlaps, even under a relatively loose yet sufficient IoU threshold of 0.5. The method is also computationally efficient, enabling timely detection within the case study. Figure 6.1 illustrates examples from the real dataset, showing both models with a confidence threshold of 0.5.

6.1.2 Performance on Synthetic Datasets

In the case of the synthetic datasets, the distinction between topmost and intermediate screws was taken in consideration. From Table 6.1 and Table 6.2, it is evident that both models perform reasonably well in both depth and RGB settings for the Single-Screw Synthetic Dataset and the Mixed-Screws Synthetic Dataset. Mask R-CNN demonstrates greater stability, showing higher $\text{mAP}@[.5:.95]$ values compared to YOLO. However, YOLO excels in terms of speed, operating near twice as fast as Mask R-CNN. Despite Mask R-CNN being slower, it still processes each image in less than one-tenth of a second, making it viable for real-world applications. The expectation was that the models would perform significantly better on the depth dataset, given that depth information should theoretically provide more clarity in identifying overlaps. Surprisingly, both models perform well in the RGB setting too, which is a notable finding. This implies that the models can effectively recognize overlaps even in RGB images. In depth images, the models might identify overlapping screws by detecting particularly bright areas. Nevertheless, the models have demonstrated their ability to infer overlaps in RGB images, even in challenging scenarios where the confluence of two screws makes

Table 6.1: mAP@[.5:.95] for YOLOv8n-seg and Mask R-CNN across datasets.

Setting	YOLOv8n-seg	Mask R-CNN
Depth – Real dataset	0.82	0.71
RGB – Real dataset	N/A	N/A
Depth – Single-Screw Synthetic	0.84	0.90
RGB – Single-Screw Synthetic	0.81	0.91
Depth – Mixed-Screws Synthetic	0.85	0.90
RGB – Mixed-Screws Synthetic	0.78	0.87

Table 6.2: Per-image inference time (milliseconds) for YOLOv8n-seg and Mask R-CNN.

Setting	YOLOv8n-seg	Mask R-CNN
Depth – Real dataset	31 ms	80 ms
RGB – Real dataset	N/A	N/A
Depth – Single-Screw Synthetic	31 ms	73 ms
RGB – Single-Screw Synthetic	44 ms	83 ms
Depth – Mixed-Screws Synthetic	31 ms	86 ms
RGB – Mixed-Screws Synthetic	46 ms	88 ms

it difficult to distinguish where one ends and the other begins, as exemplified in Figure 5.2(b). Furthermore, the models show good performance even in the Mixed-Screws Synthetic Dataset. This dataset is more complex because it includes different types of screws within the same images and incorporates random variations in lighting positions. Although industrial environments often feature production lines with identical objects and stable lighting conditions, this result is promising as it suggests that the models can adapt to more variable and challenging scenarios.

6.1.3 Performance on Test Data with Optimized Hyperparameters

Table 6.3, Table 6.4, and Table 6.5 present the F1-score, precision, and recall for both classes combined (macro-averaged), the topmost screws class, and the intermediate overlapping screws class, respectively. The most critical statistics are precision and recall for the topmost screws, and precision for the intermediate screws. From a technical standpoint, identifying and removing topmost screws is crucial as they can be easily extracted. Intermediate screws, although not readily removable, will eventually become topmost and thus easier to remove once the screws blocking them are removed. From the tables, it is evident that for the topmost class, both precision and recall are high for every model across both RGB and depth datasets. For the intermediate class, the worst-case scenario is observed when precision is notably low at 0.76. The remaining scenarios are relatively comparable and perform well, particularly in terms of precision. The intermediate class poses the most significant challenges; indeed, in the RGB setting it achieves the worst performance, with F1 scores of 0.69 and 0.64. Given the practical relevance of this class, improving its performance is important and warrants further investigation. Figure 6.2 shows the performance of the optimized models on RGB and

depth images of the synthetic datasets.

Table 6.3: Performance on all synthetic datasets (Macro F1, Precision, Recall).

	Single-Screw Synthetic		Mixed-Screws Synthetic	
	YOLOv8n-seg	Mask R-CNN	YOLOv8n-seg	Mask R-CNN
Depth	0.93 (F1)	0.87 (F1)	0.87 (F1)	0.80 (F1)
	0.96 (P)	0.92 (P)	0.87 (P)	0.90 (P)
	0.90 (R)	0.83 (R)	0.88 (R)	0.77 (R)
	0.83 (F1)	0.89 (F1)	0.77 (F1)	0.81 (F1)
RGB	0.87 (P)	0.91 (P)	0.83 (P)	0.87 (P)
	0.80 (R)	0.84 (R)	0.73 (R)	0.76 (R)

Table 6.4: Performance on the *topmost screws* class (F1, Precision, Recall).

	Single-Screw Synthetic		Mixed-Screws Synthetic	
	YOLOv8n-seg	Mask R-CNN	YOLOv8n-seg	Mask R-CNN
Depth	0.98 (F1)	0.98 (F1)	0.95 (F1)	0.92 (F1)
	0.97 (P)	0.97 (P)	0.94 (P)	0.91 (P)
	0.99 (R)	1.00 (R)	0.96 (R)	0.93 (R)
	0.97 (F1)	0.98 (F1)	0.89 (F1)	0.90 (F1)
RGB	0.96 (P)	0.97 (P)	0.85 (P)	0.89 (P)
	0.98 (R)	0.98 (R)	0.95 (R)	0.91 (R)

Table 6.5: Performance on the *intermediate screws* class (F1, Precision, Recall).

	Single-Screw Synthetic		Mixed-Screws Synthetic	
	YOLOv8n-seg	Mask R-CNN	YOLOv8n-seg	Mask R-CNN
Depth	0.88 (F1)	0.75 (F1)	0.80 (F1)	0.68 (F1)
	0.96 (P)	0.85 (P)	0.80 (P)	0.76 (P)
	0.81 (R)	0.67 (R)	0.80 (R)	0.62 (R)
	0.69 (F1)	0.81 (F1)	0.64 (F1)	0.71 (F1)
RGB	0.77 (P)	0.95 (P)	0.76 (P)	0.85 (P)
	0.63 (R)	0.70 (R)	0.56 (R)	0.62 (R)

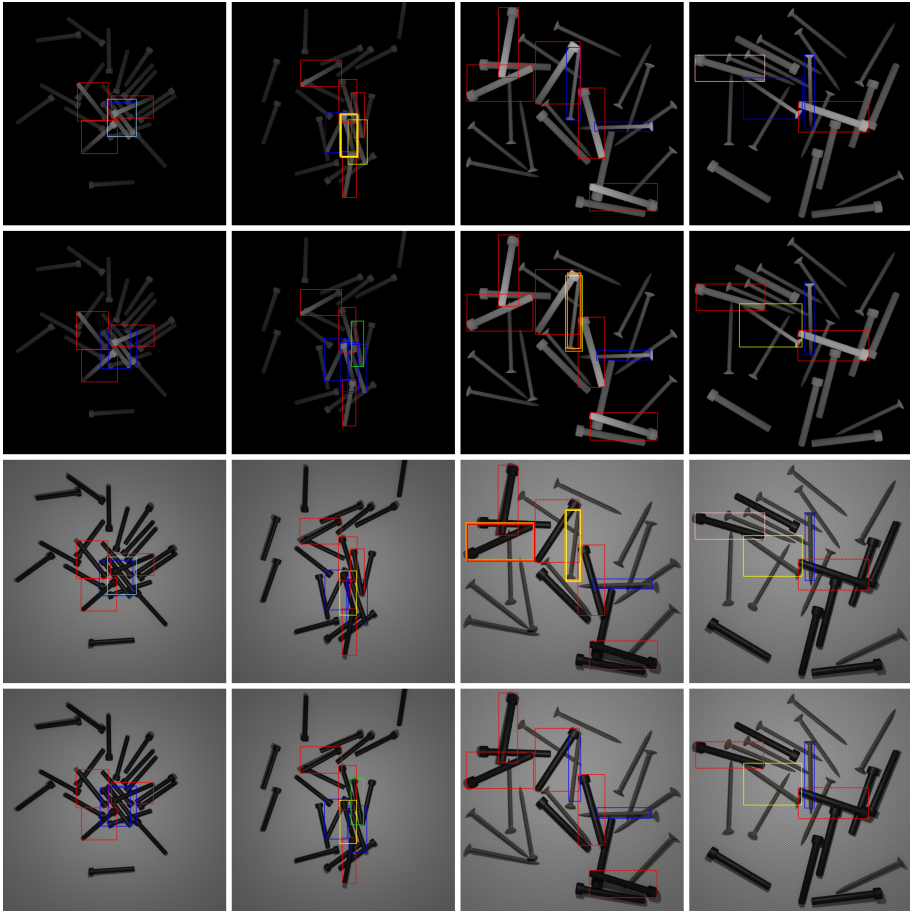


Figure 6.2: First row: YOLO model, depth images. Second row: MaskRCNN model, depth images. Third row: YOLO model, RGB images. Fourth row: MaskRCNN model, RGB images. First and second columns: Single-Screw-Synthetic Dataset. Third and fourth columns: Mixed-Screws-Synthetic Dataset. Red box: correctly identified topmost screw. Blue box: correctly identified intermediate screw. Yellow box: wrongly classified as topmost screw. Green box: wrongly classified as intermediate screw. Pink box: topmost screw not identified. Sky-blue box: intermediate screw not identified. Orange box: prediction of intermediate screw not associable with a ground truth.

6.2 Unsupervised pose-agnostic visual anomaly detection in realistic industrial scenes

The following sections report quantitative and qualitative evaluations of the proposed pose-agnostic anomaly-detection pipeline, in comparison with SplatPose+, the current state of the art, and with SplatPose.

6.2.1 Metrics

In accordance with prior works [75, 102, 104], the evaluation metrics includes pixel-level AUROC, AUPRO[69], and image-level AUROC, where image-level anomaly score is defined as the maximum of the pixel-wise anomaly map. Higher values of pixel-level AUROC and AUPRO indicate better localization of true anomalies, as explicitly assess the spatial accuracy of the pixel-wise map.

While the previous metrics are threshold-free, qualitative visualization and practical anomaly detection require setting a decision threshold, which in turn enables the computation of the F_1 -score to measure the quality of anomaly detection at localization level. The proposed pipeline produces a soft anomaly mask at the pixel level and aggregates it into a single image-level score by taking the maximum value across pixels. Then, a decision threshold is selected by maximizing the image-level F_1 -score. Therefore, an image is considered anomalous if its aggregated score exceeds this threshold; the same threshold is then applied at pixel level, classifying pixels with scores above it as anomalous. With this in mind it is possible to compute the F_1 -score at the anomaly-localization level as follows. After thresholding the pixel-wise anomaly map, connected components are extracted and the same procedure is applied to the ground-truth mask. A true positive area is counted when a predicted connected component overlaps with any ground-truth one. A false positive area is a predicted connected component that does not overlap with any ground-truth one. A false negative area is a ground-truth connected component that does not overlap with any predicted one. F_1 -score is then computed using the number of true positives, false positives and false negatives. The F_1 -score also measures how accurate the image-level classification is; specifically, whether an image labeled as anomalous reflects the detection of a genuine anomaly or merely spurious, noise-induced false positives.

6.2.2 Quantitative Evaluation

Table 6.6 reports the performance of the proposed pipeline and the per-image inference time for all nine classes, based on experiments run on an NVIDIA RTX-A4500 GPU. The results are reported up to the third decimal digit and this level of precision is required especially for pixel-level AUROC and AUPRO because the anomalous pixel class is unbalanced relative to the non-anomalous class. High numerical precision is therefore required to capture small variations in the detection of fine anomalies.

To the best of the authors’ knowledge, all existing methods addressing the PAD problem assume that backgrounds are removed from test images. A direct comparison with such approaches would therefore be misleading, as their performance is expected to degrade substantially under the proposed setting, which includes background clutter.

Then, to obtain a meaningful point of reference to the state-of-the-art SplatPose+, the comparison is conducted by evaluating SplatPose+ on a simplified dataset with same objects and poses, but with backgrounds removed, whereas the proposed method is evaluated on the original dataset that retains background clutter. The results show that the proposed pipeline attains comparable performance to SplatPose+ in this setting, indicating that any remaining gap is not due to the background-handling challenge addressed by the pipeline, but reflects difficulty intrinsic to the task even under simplified conditions. The resulting scores, are written in Table 6.7. Table 6.9 shows, instead, the per-class F_1 -scores for both the proposed method and SplatPose+. Results are reported to two decimal places, which is, in this case, sufficiently informative.

All evaluations are averaged over 50 runs of the same pipeline using the same data split, and the results are reported as mean and standard deviation. The variability is intrinsic to the Gaussian splatting technique. During model optimization, new splats may be introduced with a small degree of randomness, which can lead to slight differences across runs even when training on the same data.

Overall, the performances between methods are comparable even though the proposed method is evaluated on original images with background clutter, unlike SplatPose+, which is tested on background-removed images.

The easiest class, coupe car, achieves the best results, as expected, since it is relatively compact and lacks small holes or irregularities in its shape that could degrade the anomaly detection performance.

Conversely, complex objects, such as the police car present fine structural details like small open regions that make accurate detection more challenging, lowering the scores. More specifically, concave regions and small holes may be incorrectly detected by BiRefNet as part of the object rather than as background. In such cases, LaMa is employed to mitigate this issue; however, when holes are located within the object and are very small, the inpainting technique may still be insufficient to accurately reconstruct the background in these regions, as shown in Figure 6.5. These aspects represent the main limitations of the proposed pipeline. However, SplatPose+ also exhibits limitations when dealing with small holes or concavities, as illustrated in Figure 6.5. In particular, the Gaussian splatting technique may fail to accurately reconstruct small holes and cavities during training; as a consequence, during anomaly detection, these regions can be incorrectly identified as anomalous, leading to false positives.

The results may appear sometimes slightly inconsistent, since for some classes a higher image-level AUROC does not necessarily correspond to a higher F_1 -score. This is due to the fact that image-level AUROC disregards anomaly localization and can be affected by noise.

In addition, a comparison with SplatPose is reported, even though it is no longer state-of-the-art because SplatPose+ outperforms it in both prediction quality and inference time. This comparison is included for completeness. The tests are conducted under the same setting as for SplatPose+, with the background removed, since SplatPose is not designed to handle it. The resulting performance is substantially lower than that of both the proposed method and SplatPose+, in terms of both overall performance and inference time. The results are reported in 6.8 and 6.9.

6.2.3 Qualitative Evaluation

The masked regions in the third column of Figure 6.3 and Figure 6.4 indicate anomalous pixels obtained by applying the previously defined decision threshold, while the first and second columns show the test images and the synthesized normal images, respectively. Figure 6.5 highlights, within boxes, examples of false positives and false negatives for the proposed method in the first column and for SplatPose+ in the second column; the third column displays the ground-truth masks. The images show that both approaches can produce spurious or missing detections. Figure 6.6 and 6.7 show the results of SplatPose. The green anomalous area is clearly incorrect because the pose estimation is not accurate. This mismatch can be observed by comparing the first and second columns and using the red line as a visual reference to inspect the pose estimation error.

6.2.4 Ablation Study

Finally, two ablation studies are conducted. The first analyzes the impact of the LaMa inpainting and BiRefNet segmentation modules. The second reports the results obtained by running the proposed pipeline on background-free images, following the SplatPose+ setting.

Table 6.10 reports results obtained when comparing test and reference images only within the object mask of the synthesized normal image, which is the mask derived from residual transmittance. Performance remains strong because, in many views, anomalies are small and lie well inside the object rather than near its boundaries, making this mask sufficient in numerous cases. However, this cannot always be assumed: in worst-case scenarios where anomalies occur on the border, restricting the comparison to the synthesized mask misses relevant regions. In such cases, the full proposed pipeline is required to recover the complete comparison area. Consistently, Table 6.6 shows improved results, as it also addresses the border cases discussed above. Table 6.11 reports the F_1 -score performance and should be compared with the first column of Table 6.9. For classes such as amphibious car and coupe car, the results are essentially comparable across methods. For the remaining classes, however, the proposed method attains higher F_1 -scores, highlighting the improvements delivered by the proposed approach.

Table 6.12 reports the usual performance metrics of the proposed pipeline when applied to background-free images. The results are comparable to those obtained on images with background and to the performance achieved by SplatPose+ on background-free images.

Table 6.6: Proposed method evaluation results and inference times. Anomaly detection metrics: pixel-wise AUROC (p-AUROC), AUPRO, image-wise AUROC (i-AUROC) and image inference time (i-time) in seconds.

	p-AUROC	AUPRO	i-AUROC	i-time (s)
Amphibious car	0.999 ± 0.000	0.994 ± 0.000	0.985 ± 0.002	1.7 ± 0.0
Coupe car	0.999 ± 0.000	0.993 ± 0.000	0.986 ± 0.001	1.7 ± 0.0
Police car	0.992 ± 0.000	0.985 ± 0.000	0.921 ± 0.006	1.7 ± 0.1
Tank car	0.999 ± 0.000	0.990 ± 0.000	0.987 ± 0.001	1.7 ± 0.0
Off-road car	0.999 ± 0.000	0.994 ± 0.000	0.981 ± 0.002	1.7 ± 0.0
Desert car	0.999 ± 0.000	0.995 ± 0.000	0.940 ± 0.003	1.7 ± 0.0
Loader car	0.999 ± 0.000	0.997 ± 0.000	0.993 ± 0.003	1.7 ± 0.0
Racing car	0.996 ± 0.000	0.977 ± 0.000	0.947 ± 0.005	1.7 ± 0.0
Truck car	0.999 ± 0.000	0.993 ± 0.000	0.972 ± 0.003	1.7 ± 0.0

Table 6.7: SplatPose+ results. Anomaly detection metrics: pixel-wise AUROC (p-AUROC), AUPRO, and image-wise AUROC (i-AUROC).

	p-AUROC	AUPRO	i-AUROC	i-time (s)
Amphibious car	0.999 ± 0.000	0.994 ± 0.000	0.989 ± 0.002	0.9 ± 0.0
Coupe car	0.999 ± 0.000	0.991 ± 0.000	0.992 ± 0.001	0.9 ± 0.0
Police car	0.999 ± 0.000	0.993 ± 0.000	0.921 ± 0.011	0.9 ± 0.0
Tank car	0.998 ± 0.000	0.984 ± 0.000	0.977 ± 0.002	0.9 ± 0.0
Off-road car	0.999 ± 0.000	0.993 ± 0.000	0.986 ± 0.001	0.9 ± 0.0
Desert car	0.999 ± 0.000	0.994 ± 0.000	0.961 ± 0.001	0.9 ± 0.0
Loader car	0.999 ± 0.000	0.996 ± 0.000	0.993 ± 0.004	0.9 ± 0.0
Racing car	0.998 ± 0.000	0.989 ± 0.000	0.956 ± 0.003	0.9 ± 0.0
Truck car	0.999 ± 0.000	0.994 ± 0.000	0.981 ± 0.002	0.9 ± 0.0

Table 6.8: SplatPose results. Anomaly detection metrics: pixel-wise AUROC (p-AUROC), AUPRO, and image-wise AUROC (i-AUROC).

	p-AUROC	AUPRO	i-AUROC	i-time (s)
Amphibious car	0.983 ± 0.000	0.902 ± 0.001	0.727 ± 0.005	9.6 ± 0.0
Coupe car	0.990 ± 0.000	0.929 ± 0.000	0.803 ± 0.002	9.6 ± 0.0
Police car	0.960 ± 0.000	0.819 ± 0.001	0.613 ± 0.009	9.6 ± 0.0
Tank car	0.986 ± 0.000	0.882 ± 0.001	0.614 ± 0.001	9.6 ± 0.0
Off-road car	0.967 ± 0.000	0.862 ± 0.003	0.766 ± 0.003	9.6 ± 0.0
Desert car	0.954 ± 0.000	0.826 ± 0.000	0.655 ± 0.002	9.6 ± 0.0
Loader car	0.969 ± 0.000	0.847 ± 0.000	0.631 ± 0.002	9.6 ± 0.0
Racing car	0.947 ± 0.001	0.812 ± 0.005	0.648 ± 0.012	9.6 ± 0.0
Truck car	0.967 ± 0.000	0.864 ± 0.002	0.690 ± 0.014	9.6 ± 0.0

Table 6.9: Per-class F_1 -scores for the proposed method and SplatPose+.

	Proposed method	SplatPose+	SplatPose
Amphibious car	0.92 ± 0.01	0.95 ± 0.01	0.31 ± 0.00
Coupe car	0.94 ± 0.00	0.98 ± 0.00	0.42 ± 0.01
Police car	0.79 ± 0.02	0.78 ± 0.04	0.27 ± 0.03
Tank car	0.92 ± 0.01	0.93 ± 0.01	0.39 ± 0.04
Off-road car	0.92 ± 0.01	0.90 ± 0.02	0.34 ± 0.01
Desert car	0.83 ± 0.02	0.83 ± 0.02	0.34 ± 0.02
Loader car	0.95 ± 0.02	0.93 ± 0.02	0.25 ± 0.01
Racing car	0.88 ± 0.02	0.89 ± 0.01	0.26 ± 0.01
Truck car	0.88 ± 0.02	0.94 ± 0.01	0.26 ± 0.03

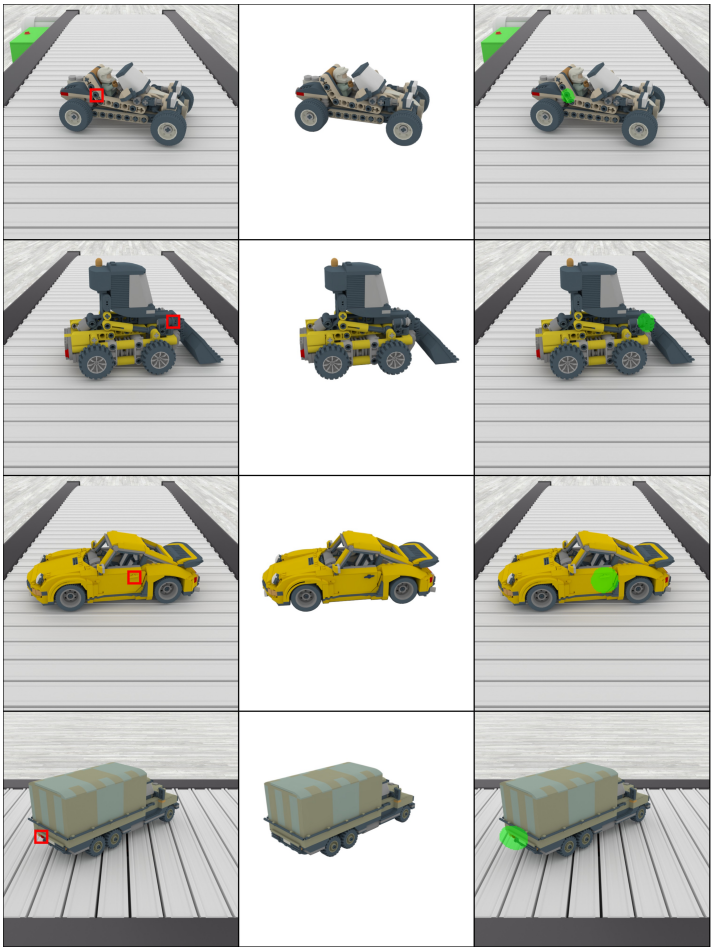


Figure 6.3: Examples for each class. First column: testing images. Second column: synthetic normal images. Third column: image with detected anomaly.

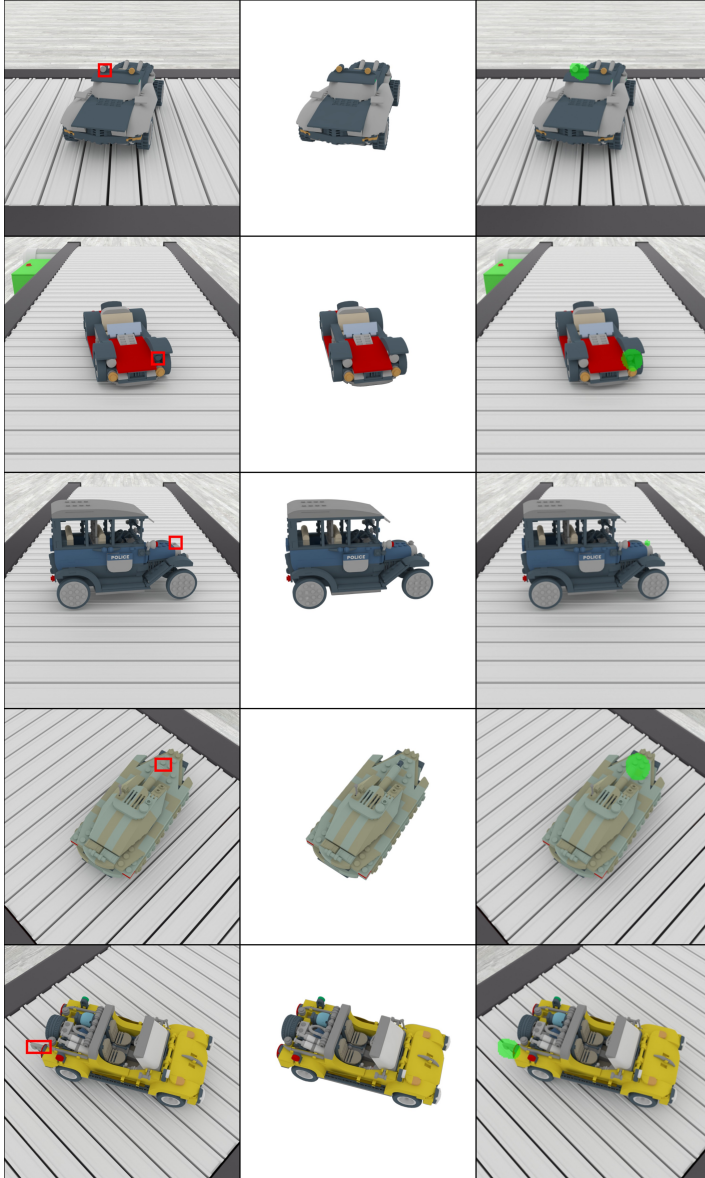


Figure 6.4: Examples for each class. First column: testing images. Second column: synthetic normal images. Third column: image with detected anomaly.



Figure 6.5: Qualitative comparison of detection errors. Examples of false positives and false negatives are highlighted with boxes. First column: proposed method; second column: SplatPose+; third column: ground-truth masks.

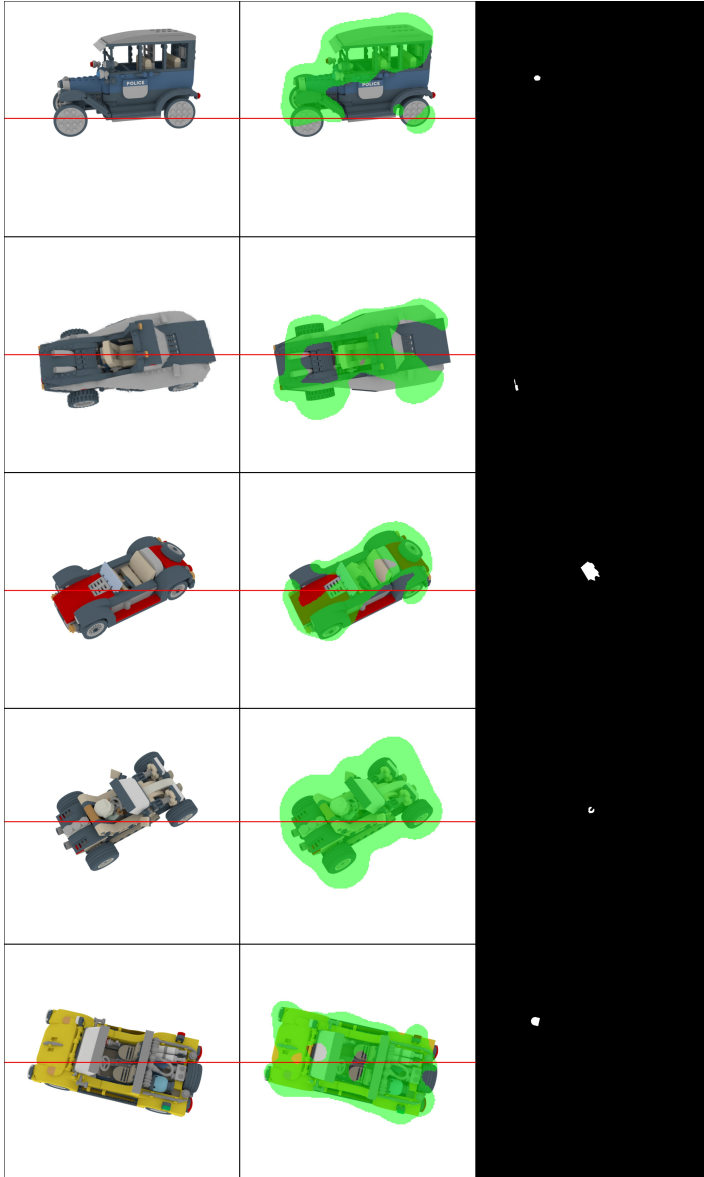


Figure 6.6: Examples of SplatPose results for each class. First column: test images. Second column: SplatPose outputs. Third column: ground-truth annotations. The red line provides a visual reference to inspect pose estimation errors.

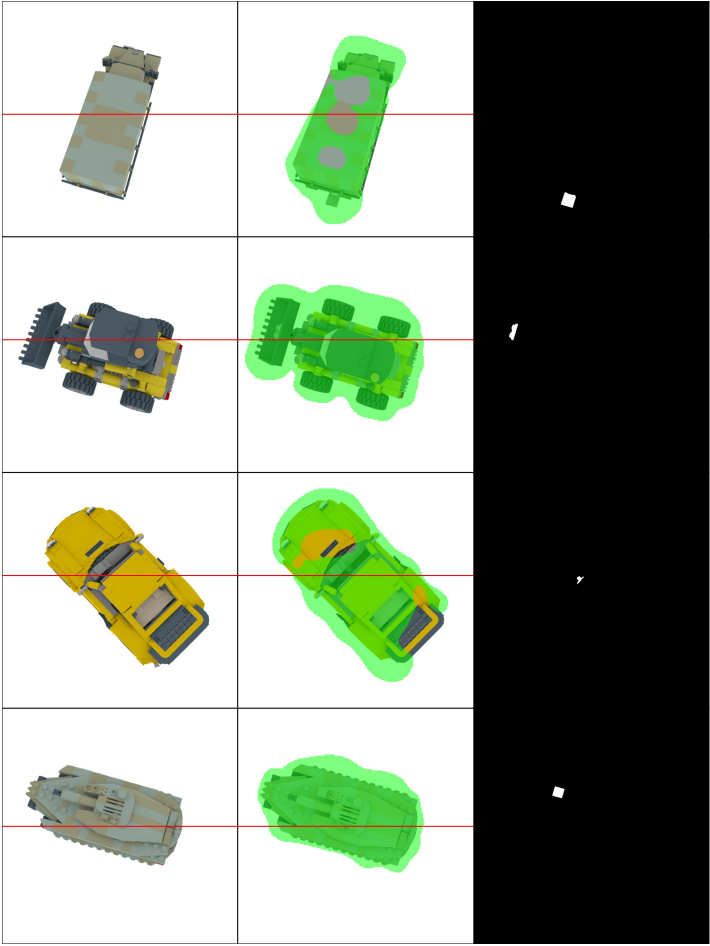


Figure 6.7: Examples of SplatPose results for each class. First column: test images. Second column: SplatPose outputs. Third column: ground-truth annotations. The red line provides a visual reference to inspect pose estimation errors..

Table 6.10: Results and inference times obtained without employing BiRefNet and LaMa modules. Anomaly detection metrics: pixel-wise AUROC (p-AUROC), AUPRO, image-wise AUROC (i-AUROC) and image inference time (i-time) in seconds.

	p-AUROC	AUPRO	i-AUROC	i-time (s)
Amphibious car	0.993 ± 0.000	0.983 ± 0.001	0.975 ± 0.004	0.9 ± 0.0
Coupe car	0.983 ± 0.000	0.980 ± 0.000	0.988 ± 0.001	0.9 ± 0.0
Police car	0.977 ± 0.000	0.940 ± 0.002	0.873 ± 0.009	0.9 ± 0.0
Tank car	0.956 ± 0.000	0.956 ± 0.000	0.974 ± 0.002	0.9 ± 0.0
Off-road car	0.971 ± 0.000	0.916 ± 0.001	0.911 ± 0.006	0.9 ± 0.0
Desert car	0.989 ± 0.001	0.961 ± 0.002	0.907 ± 0.005	0.9 ± 0.0
Loader car	0.979 ± 0.001	0.938 ± 0.002	0.939 ± 0.007	0.9 ± 0.0
Racing car	0.980 ± 0.000	0.927 ± 0.001	0.899 ± 0.008	0.9 ± 0.0
Truck car	0.977 ± 0.001	0.957 ± 0.002	0.930 ± 0.007	0.9 ± 0.0

Table 6.11: Per-class F_1 -scores obtained without employing BiRefNet and LaMa modules.

	F_1
Amphibious car	0.91 ± 0.01
Coupe car	0.94 ± 0.00
Police car	0.70 ± 0.04
Tank car	0.89 ± 0.01
Off-road car	0.85 ± 0.01
Desert car	0.81 ± 0.01
Loader car	0.87 ± 0.02
Racing car	0.79 ± 0.05
Truck car	0.84 ± 0.02

Table 6.12: Evaluation results of the proposed method on background-free images. Anomaly detection metrics: pixel-wise AUROC (p-AUROC), AUPRO, image-wise AUROC (i-AUROC) and F_1 -score.

	p-AUROC	AUPRO	i-AUROC	F_1
Amphibious car	0.999 ± 0.000	0.993 ± 0.000	0.983 ± 0.002	0.91 ± 0.02
Coupe car	0.999 ± 0.000	0.991 ± 0.000	0.991 ± 0.001	0.92 ± 0.01
Police car	0.999 ± 0.000	0.992 ± 0.000	0.920 ± 0.008	0.79 ± 0.04
Tank car	0.999 ± 0.000	0.984 ± 0.000	0.980 ± 0.002	0.92 ± 0.02
Off-road car	0.999 ± 0.000	0.992 ± 0.000	0.987 ± 0.001	0.90 ± 0.02
Desert car	0.999 ± 0.000	0.993 ± 0.000	0.958 ± 0.002	0.86 ± 0.00
Loader car	0.999 ± 0.000	0.996 ± 0.000	0.993 ± 0.003	0.96 ± 0.01
Racing car	0.999 ± 0.000	0.990 ± 0.000	0.961 ± 0.005	0.88 ± 0.01
Truck car	0.999 ± 0.000	0.993 ± 0.000	0.979 ± 0.004	0.93 ± 0.02

7

Conclusions

In this thesis, two scenarios of visual anomaly detection for industrial applications with a strong intrinsic 3D nature were presented.

In the first case, object detection neural networks were employed. The results demonstrate their ability to "learn" in a supervised fashion the logical notion of overlapping screws, namely those screws that, due to their 3D freedom of movement, are positioned one on top of the other. Crucially, this detection can be achieved without relying on depth information, as shown by the results in the RGB setting compared with the depth configuration. This is particularly advantageous for the industrial context. Indeed, relying on depth sensing hardware, such as profilers, is significantly more expensive than utilizing RGB cameras. Furthermore, devices like profilers are often inherently linear and thus less flexible in terms of scene coverage compared to the fast and effective analysis achievable with matrix RGB cameras.

Some limitations were observed in managing borderline cases where screws are only slightly overlapped. These cases pose a challenge because such screws can be treated interchangeably as either topmost or intermediate. Moreover, while the masks generated are accurate enough to estimate the centroid and pose, they are not perfectly fitted to the screws, especially with the YOLO model.

Finally, the results obtained with mixed screws and varying lighting conditions are very close to those achieved with a single type of screw. This reinforces the idea that the network can understand the concept of overlap effectively, even in more complex and variable settings.

In the second proposed scenario, a dataset for pose-agnostic anomaly detection was introduced to reflect realistic industrial inspection conditions, together with a pipeline designed to handle the practical constraints present in the data.

Beyond the main task, some practical issues were encountered, primarily with the image-level decision rule. Indeed, small amounts of random noise can produce false positives, because nothing prevents the noise's features from exhibiting large differences in magnitude relative to normality. Then, since the aggregation uses the maximum across pixel scores, a single spurious peak can cause a normal image to be classified as anomalous. Similar failures arise in small surface holes. These regions are difficult

to separate from the object, often lie within the object silhouette, and are compared against non-anomalous exemplars that exhibit blank white backgrounds in those holes. This mismatch can introduce false positives. The proposed pipeline limits many of these cases, but performance is not yet perfect. Further investigation is required to refine these borderline scenarios and to improve robustness. Moreover, the computational time could be improved, and further research is needed to reduce it.

This work has some limitations, most notably the fact that the proposed approach is evaluated in a controlled synthetic environment. As a result, there is room for future work aimed at developing a pipeline that is closer to real industrial scenarios, where noise is more pronounced. Furthermore, illumination conditions, which were assumed to be uniform in this study, should be addressed in future work to reduce sensitivity to lighting changes, possibly building on PIAD [78], and to variability arising from the temporal degradation of equipment and environmental conditions.

Bibliography

- [1] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 2 edition, 2000.
- [2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006.
- [3] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023.
- [4] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4 edition, 2021.
- [5] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [6] Ethem Alpaydin. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition, 2010.
- [7] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.
- [8] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, Cham, 2 edition, 2022.
- [9] E. Marchi, D. Fornasier, A. Miorin, and G. L. Foresti. Segmentation networks for detecting overlapping screws in 3D and color images for industrial quality control. *Integrated Computer-Aided Engineering*, 32(3):244–257, 2025.
- [10] Maximilian Denninger, Dominik Winkelbauer, Martin Sundermeyer, Wout Boerdijk, Markus Knauer, Klaus H. Strobl, Matthias Humt, and Rudolph Triebel. Blenderproc2: A procedural pipeline for photorealistic rendering. *Journal of Open Source Software*, 8(82):4901, 2023.
- [11] Y. Zhu, Y. Tian, D. Metaxas, and P. Dollár. Semantic amodal segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3001–3009, Honolulu, HI, USA, 2017.
- [12] K. Li and J. Malik. Amodal instance segmentation. In *Computer Vision – ECCV 2016*, volume 9906 of *Lecture Notes in Computer Science*. Springer, 2016.

- [13] K. Wada, S. Kitagawa, K. Okada, and M. Inaba. Instance segmentation of visible and occluded regions for finding and picking target from a pile of objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2048–2055, Madrid, Spain, 2018.
- [14] K. Wada, K. Okada, and M. Inaba. Joint learning of instance and semantic segmentation for robotic pick-and-place with heavy occlusions in clutter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9558–9564, Montreal, QC, Canada, 2019.
- [15] Y. Inagaki, R. Araki, T. Yamashita, and H. Fujiyoshi. Detecting layered structures of partially occluded objects for bin picking. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5786–5791, Macau, China, 2019.
- [16] L. Qi, L. Jiang, S. Liu, X. Shen, and J. Jia. Amodal instance segmentation with kins dataset. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3009–3018, Long Beach, CA, USA, 2019.
- [17] P. Follmann, R. König, P. Härtinger, M. Klostermann, and T. Böttger. Learning to see the invisible: End-to-end trainable amodal instance segmentation. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1328–1336, Waikoloa, HI, USA, 2019.
- [18] X. Yan, F. Yu, F. Wang, W. Liu, S. He, and J. Pan. Visualizing the invisible: Occluded vehicle segmentation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7617–7626, Seoul, Korea, 2019.
- [19] G. Zhan, C. Zheng, W. Xie, and A. Zisserman. Amodal ground truth and completion in the wild. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 28003–28013, Seattle, WA, USA, 2024.
- [20] Y. Yan, Y. Qi, C. Xu, N. Su, and L. Yang. Building extraction at amodal-instance-segmentation level: Datasets and framework. *IEEE Transactions on Geoscience and Remote Sensing*, 62(1):1–18, 2024.
- [21] J. Ao, Q. Ke, and K. A. Ehinger. Amodal intra-class instance segmentation: Synthetic datasets and benchmark. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 280–289, Waikoloa, HI, USA, 2024.
- [22] Y. Xiao, Y. Xu, Z. Zhong, W. Luo, J. Li, and S. Gao. Amodal segmentation based on visible region segmentation and shape prior. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2995–3003, 2021.
- [23] Z. Li, W. Ye, T. Jiang, and T. Huang. 2d amodal instance segmentation guided by 3d shape prior. In S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, editors, *Computer Vision – ECCV 2022*, volume 13689 of *Lecture Notes in Computer Science*. Springer, 2022.

- [24] J. Chen, L. Niu, J. Zhang, J. Si, C. Qian, and L. Zhang. Amodal instance segmentation via prior-guided expansion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 313–321, 2023.
- [25] W.-D. Jang et al. Learning vector quantized shape code for amodal blastomere instance segmentation. In *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*, pages 1–5, Cartagena, Colombia, 2023.
- [26] Z. Li, W. Ye, T. Jiang, and T. Huang. Gin: Generative invariant shape prior for amodal instance segmentation. *IEEE Transactions on Multimedia*, 26:3924–3936, 2024.
- [27] M. Tran, W. Bounsavy, K. T. Vo, A. Nguyen, T. Nguyen, and N. Le. Shapeformer: Shape prior visible-to-amodal transformer-based amodal instance segmentation. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2024.
- [28] L. Ke, Y.-W. Tai, and C.-K. Tang. Deep occlusion-aware instance segmentation with overlapping bilayers. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4018–4027, Nashville, TN, USA, 2021.
- [29] Y. Sun, A. Kortylewski, and A. Yuille. Amodal segmentation through out-of-task and out-of-distribution generalization with a bayesian model. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1205–1214, New Orleans, LA, USA, 2022.
- [30] Minh-Triet Tran, Khoa T. Vo, Kashu Yamazaki, Arthur F. A. Fernandes, Michael T. Kidd, and Ngan T. H. Le. Aisformer: Amodal instance segmentation with transformer. *ArXiv*, abs/2210.06323, 2022.
- [31] J. Gao et al. Coarse-to-fine amodal segmentation with shape prior. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1262–1271, Paris, France, 2023.
- [32] Z. Li, R. Shi, T. Huang, and T. Jiang. Oaformer: Learning occlusion distinguishable feature for amodal instance segmentation. In *ICASSP 2023 - IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, Rhodes Island, Greece, 2023.
- [33] H. Ling, D. Acuna, K. Kreis, S. W. Kim, and S. Fidler. Variational amodal object completion. In *Advances in Neural Information Processing Systems*, volume 33, pages 16246–16257, 2020.
- [34] K. Nguyen and S. Todorovic. A weakly supervised amodal segmenter with boundary uncertainty estimation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7376–7385, Montreal, QC, Canada, 2021.
- [35] K. Ehsani, R. Mottaghi, and A. Farhadi. Segan: Segmenting and generating the invisible. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6144–6153, Salt Lake City, UT, USA, 2018.

- [36] S. Zhang, Z. Xue, Y. Jiang, and H. Wang. Opnet: Deep occlusion perception network with boundary awareness for amodal instance segmentation. In *ICASSP 2024 - IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2595–2599, Seoul, Republic of Korea, 2024.
- [37] C. Zheng, D. S. Dao, G. Song, et al. Visiting the invisible: Layer-by-layer completed scene decomposition. *International Journal of Computer Vision*, 129:3195–3215, 2021.
- [38] Z. Zhang, A. Chen, L. Xie, J. Yu, and S. Gao. Learning semantics-aware distance map with semantics layering network for amodal instance segmentation. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*, pages 2124–2132, New York, NY, USA, 2019.
- [39] X. Zhan, X. Pan, B. Dai, Z. Liu, D. Lin, and C. C. Loy. Self-supervised scene de-occlusion. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3783–3791, Seattle, WA, USA, 2020.
- [40] P. Wu, W. Chen, H. Liu, Y. Duan, N. Lin, and X. Chen. Predicting grasping order in clutter environment by using both color image and points cloud. In *2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pages 197–202, Beijing, China, 2019.
- [41] S. Yu, D.-H. Zhai, and Y. Xia. Egnet: Efficient robotic grasp detection network. *IEEE Transactions on Industrial Electronics*, 70(4):4058–4067, 2023.
- [42] H. Zhang, X. Lan, X. Zhou, Z. Tian, Y. Zhang, and N. Zheng. Visual manipulation relationship network for autonomous robotics. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 118–125, Beijing, China, 2018.
- [43] G. Zuo, J. Tong, H. Liu, W. Chen, and J. Li. Graph-based visual manipulation relationship reasoning network for robotic grasping. *Frontiers in Neurorobotics*, 15:719731, 2021.
- [44] M. Ding, Y. Liu, C. Yang, and X. Lan. Visual manipulation relationship detection based on gated graph neural network for robotic grasping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1404–1410, Kyoto, Japan, 2022.
- [45] G. Zuo, Z. Wang, D. Gong, and G. Huang. A graph reasoning method for multi-object unordered stacking scenarios. In *2023 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 888–892, Datong, China, 2023.
- [46] J. Luo, Y. Liu, H. Wang, M. Ding, and X. Lan. Grasp manipulation relationship detection based on graph sample and aggregation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4098–4104, Yokohama, Japan, 2024.

- [47] C. Yang, X. Lan, H. Zhang, X. Zhou, and N. Zheng. Visual manipulation relationship detection with fully connected crfs for autonomous robotic grasp. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 393–400, Kuala Lumpur, Malaysia, 2018.
- [48] C. Liu and C. Chen. The instance segmentation and grasping sequence method for stacked workpieces. In *2024 IEEE 13th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 2224–2229, Kaifeng, China, 2024.
- [49] C. Schwan and W. Schenck. A three-step model for the detection of stable grasp points with machine learning. *Integrated Computer-Aided Engineering*, 28(1), 2021.
- [50] I. García-Aguilar, J. García-González, R. M. Luque-Baena, E. López-Rubio, and E. Domínguez-Merino. Optimized instance segmentation by super-resolution and maximal clique generation. *Integrated Computer-Aided Engineering*, 30(3):243–256, 2023.
- [51] J. Á. Díaz-Francés, J. D. Fernández-Rodríguez, K. Thurnhofer, and E. López-Rubio. Semi-supervised semantic image segmentation by deep diffusion models and generative adversarial networks. *International Journal of Neural Systems*, 34(11):2450057, 2024.
- [52] S. Sun, P. Wang, H. Peng, and Z. Liu. Reference image segmentation with multimodal feature interaction and alignment based on convolutional nonlinear spiking neural membrane systems. *International Journal of Neural Systems*, 34(12):2450064, 2024.
- [53] J. Wang, L. Zhang, and Z. Yi. Mixture 2d convolutions for 3d medical image segmentation. *International Journal of Neural Systems*, 33(1):2250059, 2023.
- [54] G. Mirzaei and H. Adeli. Segmentation and clustering in brain mri imaging. *Reviews in the Neurosciences*, 30(1):31–44, 2019.
- [55] G. Jocher, A. Chaurasia, and J. Qiu. Ultralytics YOLOv8. <https://github.com/ultralytics/ultralytics>, 2023. Version 8.0.0.
- [56] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Venice, Italy, 2017.
- [57] Xu Zhao, Wen-Yan Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *ArXiv*, abs/2306.12156, 2023.
- [58] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, Honolulu, HI, USA, 2017.
- [59] Enrico Marchi, Matic Fučka, Danijel Skočaj, and Gian Luca Foresti. Unsupervised pose-agnostic visual anomaly detection in realistic industrial scenes. *Integrated Computer-Aided Engineering*, 2026. In press.

- [60] Timothy S. Newman and Anil K. Jain. A survey of automated visual inspection. *Computer Vision and Image Understanding*, 61(2):231–262, 1995.
- [61] C Garrido-Hidalgo, L Roda-Sanchez, A Fernández-Caballero, T Olivares, and F J Ramirez. Internet-of-things framework for scalable end-of-life condition monitoring in remanufacturing. *Integrated Computer-Aided Engineering*, 31:1–17, 2024.
- [62] P Michailidis, I T Michailidis, S Gkelios, G Karatzinis, and E B Kosmatopoulos. Neuro-distributed cognitive adaptive optimization for training neural networks in a parallel and asynchronous manner. *Integrated Computer-Aided Engineering*, 31:19–41, 2024.
- [63] F Neri, X Liu, P Yue, G Zhang, and R Haina. Deep deterministic policy gradient with constraints for gait optimisation of biped robots. *Integrated Computer-Aided Engineering*, 31(2):139–156, 2024.
- [64] C Zhou, L Fan, and F Neri. A spatio-temporal fusion deep learning network with application to lightning nowcasting. *Integrated Computer-Aided Engineering*, 31(3):233–247, 2024.
- [65] F S Marcondes, J J Almeida, and P Novais. An exploratory design science research on troll factories. *Integrated Computer-Aided Engineering*, 31:95–115, 2024.
- [66] P J Krassnig, M Haselmann, M Kremnitzer, and D P Gruber. Efficient surface defect detection in industrial screen printing with minimized labeling effort. *Integrated Computer-Aided Engineering*, 32:3–23, 2024.
- [67] Chris Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 246–252. IEEE, 1999.
- [68] J García-González, J M Ortiz-de Lazcano-Lobato, R M Luque-Baena, and E López-Rubio. Background subtraction by probabilistic modeling of patch features learned by deep autoencoders. *Integrated Computer-Aided Engineering*, 27:253–265, 2020.
- [69] P Bergmann, X Jin, D Sattlegger, and C Steger. The MVTec 3D-AD dataset for unsupervised 3D anomaly detection and localization. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 2022.
- [70] Luca Bonfiglioli, Marco Toschi, Davide Silvestri, Nicola Fioraio, and Daniele De Gregorio. The Eyecandies dataset for unsupervised multimodal anomaly detection and localization. In Lei Wang, Juergen Gall, Tat-Jun Chin, Imari Sato, and Rama Chellappa, editors, *Computer Vision – ACCV 2022*, pages 459–475, Cham, 2023. Springer Nature Switzerland.
- [71] J Qin, C Gu, J Yu, and C Zhang. Image-pointcloud fusion based anomaly detection using PD-REAL dataset. *arXiv preprint arXiv:2311.04095*, 2023.

- [72] Jiaqi Liu, Guoyang Xie, Ruitao Chen, Xinpeng Li, Jinbao Wang, Yong Liu, Chengjie Wang, and Feng Zheng. Real3D-ad: a dataset of point cloud anomaly detection. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023)*, NeurIPS '23, Red Hook, NY, USA, 2023. Curran Associates, Inc.
- [73] P McHard, F P Audonnet, O Summerell, S Andraos, P Henderson, and G Aragon-Camarasa. 3D-ADAM: a dataset for 3D anomaly detection in additive manufacturing. *arXiv preprint arXiv:2507.07838*, 2025.
- [74] Wenqiao Li, Xiaohao Xu, Yao Gu, Bozhong Zheng, Shenghua Gao, and Yingna Wu. Towards scalable 3D anomaly detection and localization: A benchmark via 3D anomaly synthesis and a self-supervised learning network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22207–22216. IEEE, June 2024.
- [75] Qiang Zhou, Weize Li, Lihan Jiang, Guoliang Wang, Guyue Zhou, Shanghang Zhang, and Hao Zhao. PAD: a dataset and benchmark for pose-agnostic anomaly detection. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023)*, NeurIPS '23, Red Hook, NY, USA, 2023. Curran Associates, Inc.
- [76] Chengjie Wang, Wenbing Zhu, Bin-Bin Gao, Zhenye Gan, Jiangning Zhang, Zhihao Gu, Shuguang Qian, Mingang Chen, and Lizhuang Ma. Real-IAD: A real-world multi-view dataset for benchmarking versatile industrial anomaly detection. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22883–22892. IEEE, 2024.
- [77] K Zhou, Y Cao, T Kim, H Zhao, H Dong, K M Ting, and Y Zhu. RAD: a dataset and benchmark for real-life anomaly detection with robotic observations. *arXiv preprint arXiv:2410.00713*, 2024.
- [78] K Yang, J Cao, Z Bai, Z Su, and A Tagliasacchi. PIAD: pose and illumination agnostic anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4734–4743, 2025.
- [79] R Maack, L Thun, T Liang, H Tercan, and T Meisen. PCAD: a real-world dataset for 6d pose industrial anomaly detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, pages 1132–1141, 2025.
- [80] Z Gencer, Y H Sahin, and G Unal. One-class classification of 3D point clouds using dynamic graph CNN. In *Proceedings of the 7th International Conference on Computer Science and Engineering (UBMK)*, pages 388–392, 2022.
- [81] M Rudolph, T Wehrbein, B Rosenhahn, and B Wandt. Asymmetric student-teacher networks for industrial anomaly detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2591–2601, 2023.

- [82] P Bergmann and D Sattlegger. Anomaly detection in 3D point clouds using deep geometric descriptors. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2612–2622, 2023.
- [83] R Lhoste, A Vacavant, and D Delhay. MAESTRO: a full point cloud approach for 3D anomaly detection based on reconstruction. In *Proceedings of VISIGRAPP 2025, Volume 2: VISAPP*, pages 717–724, 2025.
- [84] M Masuda, R Hachiuma, R Fujii, H Saito, and Y Sekikawa. Toward unsupervised 3D point cloud anomaly detection using variational autoencoder. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 3118–3122, 2021.
- [85] J Wang, Y Niu, and B Huang. Fusion-restoration model for industrial multimodal anomaly detection. *Neurocomputing*, 637:130073, 2025.
- [86] A Costanzino, P Z Ramirez, G Lisanti, and L Di Stefano. Multimodal industrial anomaly detection by crossmodal feature mapping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17234–17243, 2024.
- [87] Z Gu, J Zhang, L Liu, X Chen, J Peng, Z Gan, G Jiang, A Shu, Y Wang, and L Ma. Rethinking reverse distillation for multi-modal anomaly detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:8445–8453, 2024.
- [88] Zhongbin Sun, Xiaolong Li, Yiran Li, and Yue Ma. Memoryless multimodal anomaly detection via student-teacher network and signed distance learning. In *Pattern Recognition and Computer Vision: 7th Chinese Conference, PRCV 2024, Urumqi, China, October 18–20, 2024, Proceedings, Part XII*, pages 447–461, Berlin, Heidelberg, 2024. Springer-Verlag.
- [89] Zheyuan Zhou, Le Wang, Naiyu Fang, Zili Wang, Lemiao Qiu, and Shuyou Zhang. R3D-AD: Reconstruction via diffusion for 3D anomaly detection. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XXXVI*, volume 15094 of *Lecture Notes in Computer Science*, pages 91–107, Cham, 2024. Springer.
- [90] Ruitao Chen, Guoyang Xie, Jiaqi Liu, Jinbao Wang, Ziqi Luo, Jinfan Wang, and Feng Zheng. EasyNet: An easy network for 3D industrial anomaly detection. In *Proceedings of the 31st ACM International Conference on Multimedia (MM ’23)*, pages 7038–7046, New York, NY, USA, 2023. Association for Computing Machinery.
- [91] Vitian Zavrtanik, Matej Kristan, and Danijel Skocaj. Keep DRAEMing: discriminative 3D anomaly detection through anomaly simulation. *Pattern Recognition Letters*, 2024.
- [92] Chenyang Bi, Yueyang Li, and Haichi Luo. Dual-branch reconstruction network for industrial anomaly detection with RGB-D data. In Ram Bilas Pachori and Lei

- Chen, editors, *International Conference on Image, Signal Processing, and Pattern Recognition (ISPP 2024)*, volume 13180, page 1318033. International Society for Optics and Photonics, SPIE, 2024.
- [93] Vitian Zavrtanik, Matej Kristan, and Danijel Skocaj. Cheating depth: enhancing 3D surface anomaly detection via depth simulation. In *Proceedings of the IEEE/CVF Winter Conference on Computer Vision (WACV)*, pages 2153–2161, 2024.
 - [94] J Ye, W Zhao, X Yang, G Cheng, and K Huang. PO3AD: predicting point offsets toward better 3D point cloud anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1353–1362, 2025.
 - [95] Y-M Chu, C Liu, T-I Hsieh, H-T Chen, and T-L Liu. Shape-guided dual-memory learning for 3D anomaly detection. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pages 6185–6194, 2023.
 - [96] E Horwitz and Y Hoshen. Back to the feature: classical 3D features are (almost) all you need for 3D anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2968–2977, 2023.
 - [97] Y Cao, X Xu, and W Shen. Complementary pseudo multimodal feature for point cloud anomaly detection. *Pattern Recognition*, 156:110761, 2024.
 - [98] Y Wang, J Peng, J Zhang, R Yi, Y Wang, and C Wang. Multimodal industrial anomaly detection via hybrid fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8032–8041, 2023.
 - [99] Jiayu Liu, Shancong Mou, Nathan Gaw, and Yinan Wang. Uni-3DAD: Gan-inversion aided universal 3D anomaly detection on model-free products. *Expert Systems with Applications*, 272:126665, 2025.
 - [100] Yuanpeng Tu, Boshen Zhang, Liang Liu, Yuxi Li, Jiangning Zhang, Yabiao Wang, Chengjie Wang, and Cairong Zhao. Self-supervised feature adaptation for 3D industrial anomaly detection. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 75–91, Cham, 2025. Springer Nature Switzerland.
 - [101] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.
 - [102] Mathis Kruse, Marco Rudolph, Dominik Woiwode, and Bodo Rosenhahn. Splat-Pose & Detect: pose-agnostic 3D anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3950–3960. IEEE, June 2024.

- [103] B Jiang, Y Xie, J Li, N Li, B Chen, and S T Xia. IGSPAD: inverting 3D gaussian splatting for pose-agnostic anomaly detection. In *Proceedings of the ACM International Conference on Multimedia (ACM MM)*, 2024.
- [104] Yizhe Liu, Yan Song Hu, Yuhao Chen, and John Zelek. SplatPose+: real-time image-based pose-agnostic 3D anomaly detection. In *Computer Vision – ECCV 2024 Workshops: Milan, Italy, September 29–October 4, 2024, Proceedings, Part IV*, pages 378–391, Berlin, Heidelberg, 2025. Springer-Verlag.
- [105] Vickie Ye and Angjoo Kanazawa. Mathematical supplement for the **gsplat** library, 2023.
- [106] Peng Zheng, Dehong Gao, Deng-Ping Fan, Li Liu, Jorma Laaksonen, Wanli Ouyang, and Nicu Sebe. Bilateral reference for high-resolution dichotomous image segmentation. *CAAI Artificial Intelligence Research*, 3:9150038, 2024.
- [107] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with Fourier convolutions. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3172–3182. IEEE, 2022.
- [108] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1437–1451, 2018.
- [109] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperPoint: self-supervised interest point detection and description. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 337–349. IEEE, 2018.
- [110] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. LightGlue: local feature matching at light speed. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17581–17592. IEEE, 2023.
- [111] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: robust hierarchical localization at large scale. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12708–12717. IEEE, 2019.
- [112] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, June 2019.
- [113] Blender Online Community. *Blender: a 3D modelling and rendering package*, 2018.

- [114] R. Padilla, S. L. Netto, and E. A. B. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, Niteroi, Brazil, 2020.