Master Thesis at the Department of Information Technology and Electrical Engineering

Spring Semester 2024

Region-level, Similarity-based Interactive Image Retrieval for Visual Quality Inspection

Yutong Xiang

Jul 23, 2023

Advisor:	Dr. Yawei Li, yawei.li@vision.ee.ethz.ch
Supervisor:	Prof. Dr. Luc Van Gool, vangool@vision.ee.ethz.ch
Industry Supervisor:	Lucas Vandroux, lucas.vandroux@viun.tech

Acknowledgements

I would like to express my deepest gratitude to my industry advisor, Lucas Vandroux, for giving me the opportunity to work on this project at VIUN and for his continuous support and guidance throughout the project. I would also like to thank my academic advisors, Dr. Yawei Li and Prof. Dr. Luc Van Gool, for their valuable feedback and suggestions for the project.

This thesis tackles the challenge of distinguishing fine-grained visual similarities among nearly identical images, a critical foundation for advanced search applications in visual quality inspection. We identified gaps in current solutions regarding the distinguishability, representativeness, and efficiency of image embeddings used as search indices. To address these, we proposed a novel deep learning-based approach, starting with curating a new dataset comprising defect images from open-source domain datasets annotated with human preference data to represent fine-grained visual similarity. The dataset is used to train the model and evaluate the performance of fine-grained visual similarity searches. Then, we improved the distinguishability of search indices at the fine-grained level by fine-tuning an embedding model with a contrastive learning framework in both supervised and self-supervised learning manners. We also established a comprehensive evaluation protocol to assess the performance of the proposed solutions on the dataset. In summary, the proposed method demonstrated significant performance gain and generalizability across different products and defect types and scalability to large-scale and unannotated datasets. Finally, we present a prototypical search application that showcases how the proposed solution works in real-world scenarios. The proposed solution provides a solid foundation for developing advanced image search applications for visual quality control in manufacturing and other industries.

Contents

1. Introduction

1.1. Background

In today's fast-paced industrial environment, ensuring the highest product quality is paramount. Visual quality inspection systems play a crucial role because they help identify and address defects that could compromise product integrity and customer satisfaction. However, traditional inspection methods often fail to provide insights to tackle recurring problems effectively. Consider a scenario where an industry professional encounters a new defect on the production line that can halt operations and incur significant costs. An advanced, interactive application capable of analyzing an extensive database of past defects would offer valuable insights and comparisons to previously encountered similar issues. This tool streamlines troubleshooting processes and enhances the ability to predict and prevent future defects. By leveraging sophisticated search capabilities based on visual defect similarities, industry experts can quickly pinpoint potential causes and solutions, fostering a more proactive quality control approach.

Considering these crucial industry requirements, researchers have long sought to develop practical search applications. The most recent solutions leverage computer vision technologies, particularly deep learning-based techniques. Among the available techniques, a popular approach is vector-based semantic search. This method uses embeddings generated by deep learning models to represent images. These embeddings capture the semantics of the corresponding images. They can be used as search indices for similarity-based retrieval, enabling the identification of similar images based on the similarity of their embeddings.

However, despite previous achievements, we have found that current applications do not meet the growing needs of industry professionals. More specifically, in an industrial setting, the defect database often contains various defect images that appear very similar on the entire product image, with tiny yet distinctive differences at the defect level. It is referred to as the **fine-grained visual similarity problem**, whose key challenge lies in understanding fine-grained visual differences that sufficiently discriminate between highly similar objects in overall appearance but differ in fine-grained features.

Considering these developments, the underlying vector-based semantic search systems should be improved to capture the fine-grained visual similarity between defect areas. Typically, in such systems, the quality of the embeddings is a critical factor for successful image retrieval. We determined that the insufficient performance of the current systems can be attributed to the following three main characteristics of these embeddings that are not well addressed:

- **Distinguishability:** The embeddings of similar regions should be similar, while those of different regions should be dissimilar; however, the current model training recipe **does not explicitly enforce similarity** over embeddings for similar objects.
- **Representativeness:** the embeddings should capture the visual characteristics to facilitate performative retrieval; however, the current pre-trained backbones' **performance on domain-specific datasets like visual inspection is still limited**, especially when it comes to fine-grained features.
- **Efficiency:** the embeddings should be computationally and memory-efficient to minimize costs; however, previous methods have primarily focused on feature extraction **from the whole image**, which involves processing a massive amount of data, where only specified small regions are of interest.

Therefore, we laid the foundation of this thesis on enhancing the performance of image search applications by improving the embedding characteristics. Through this work, we empower professionals with the tools they need to maintain excellent quality standards and drive continuous improvement in their operations.

1.2. Main Contributions

In this thesis, we have developed a simple yet effective deep-learning solution to distinguish fine-grained visual similarities among industrial defect images. Our primary contributions are as follows:

- **New Annotated Dataset**: We curated the Fine-Grained Defect Similarity (FGDS) dataset comprising defect images from open-source domain datasets and annotated human preference data to represent fine-grained visual similarity using triplets. The proposed dataset can be used to train the embedding model and evaluate the performance of fine-grained visual similarity searches.
- **Innovative Search Index**: We propose an innovative method to generate search indices from a Vision Transformer (ViT) embedding model fine-tuned with a SimCLR-based contrastive learning framework. This process was performed in supervised and self-supervised modes using the training subset of the FGDS dataset. The indices were used for similarity-based retrieval from the database subset of the FGDS dataset using the evaluation subset.
- **Comprehensive Evaluation Protocol**: We established a comprehensive set of metrics to accurately assess similarity-based retrieval performance at both the defect category (label) and fine-grained levels. This setup uses triplet-based evaluation metrics to directly leverage the triplet annotations collected from humans to evaluate the retrieval performance at the fine-grained level.

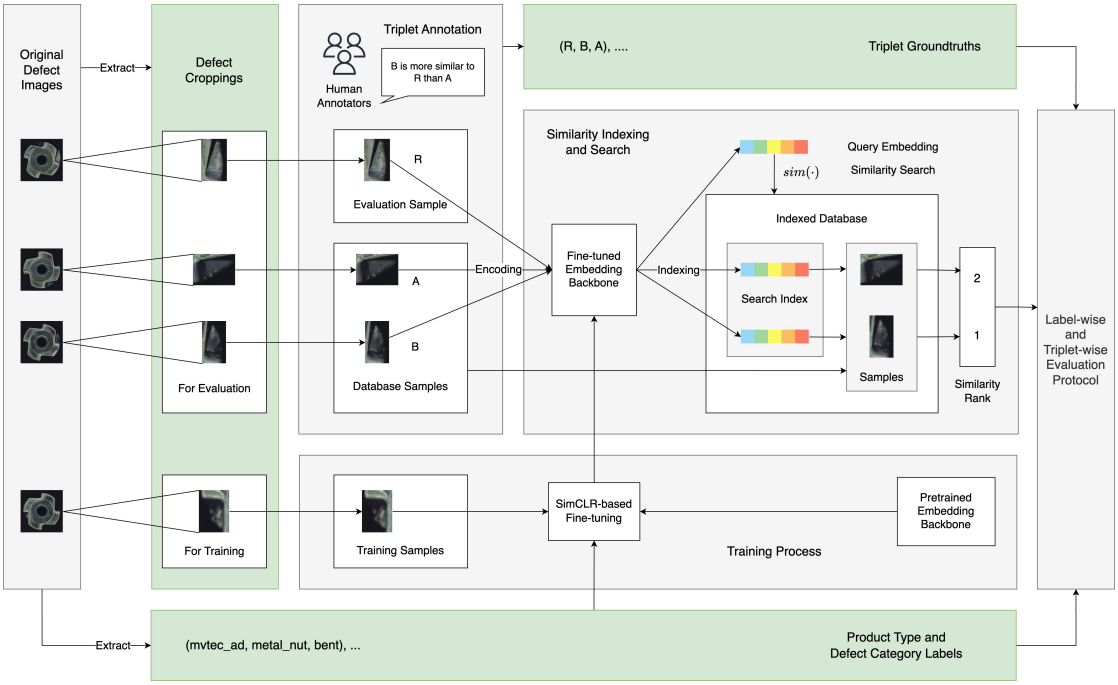


Figure 1: The overall pipeline of the thesis work.

Figure 1 illustrates the overall workflow of our work. The process begins by extracting defect croppings from the original defect images in the open-sourced domain datasets, which are then used for training and evaluation. More specifically, the croppings for evaluation were further divided into two subsets: evaluation and database. Then, human annotators provide triplet annotations to identify which defect image from the database subset is more similar to a reference image from the evaluation subset than another image. These annotations are used as the ground truths to evaluate the search results, together with product and defect category labels extracted from the original dataset. The pre-trained embedding backbone is fine-tuned with the training samples via a SimCLR-based training framework (with label information for the supervised setting). Then, the fine-tuned embedding backbone encodes the defect images into embeddings and indexes them with these embeddings. For similarity indexing and search, the query embedding is compared to the indices using some similarity metric function $sim(\cdot)$. The returned results are rank lists based on similarity, and the performance is evaluated in both label-wise and triplet-wise manners using the rank lists and ground truths collected earlier. Note that the green boxes represent the components of the FGDS dataset.

Through evaluation of the FGDS dataset with the established metrics, we realized the following capabilities with the proposed solution:

- **Effectiveness**: Notable performance gain after fine-tuning in terms of label-level precision (5-8%) and triplet-level precision (>10%), compared to zero-shot perfor-

mance.

- **Generalizability**: Notable performance gains were observed for unseen products and defect types after fine-tuning with a self-supervision signal (5-7%), compared to zero-shot performance, indicating the generalizability of the self-supervised solution.
- **Scalability**: Comparable and better performance of self-supervised methods to that of supervised methods in fine-grained visual similarity search tasks, indicating the scalability of self-supervised solutions to large, unannotated datasets.

In addition, we present an interactive demonstration application that showcases the use of the proposed solution in real-world scenarios.

1.3. Thesis Structure

The remainder of this thesis is structured as follows:

- **Section 2: Related Works** provides an overview of the existing literature on fine-grained image retrieval and contrastive representation learning.
- **Section 3: The FGDS Dataset** describes the curating process of the FGDS dataset, including defect cropping extraction, data splitting, and data annotation.
- **Section 4: Training Methods** details the proposed method for fine-tuning the embedding model, including the self-supervised learning framework and other training details.
- **Section 5: Evaluation Metrics and Tools** presents the evaluation metrics and protocols used to assess the performance of the proposed solution on the FGDS dataset.
- **Section 6: Experiment Design and Result Analysis** outlines the experimental setup for both vertical and horizontal comparisons and analyzes the obtained results.
- **Section 7: Prototype** introduces the interactive image search application prototype, showcasing the capabilities of the proposed solution.

2. Related Works

2.1. Fine-grained Image Retrieval

Fine-grained image analysis(FGIA) is the primary focus of this thesis.

According to [1], FGIA involves understanding subtle visual differences that distinguish similar objects with overall appearance but differ in fine-grained features. FGIA tasks can be categorized as recognition and retrieval. In fine-grained retrieval tasks, the goal is to retrieve images related to a given query based on relevant fine-grained features. It can be formalized as follows: Given an input query $\boldsymbol{x}^q$, the goal of a fine-grained retrieval system is to rank all instances in a retrieval set $\Omega = \left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{M}$ of the same category based on their fine-grained relevance to the query. Let $\mathcal{S}_\Omega = \left\{s^{(i)}\right\}_{i=1}^{M}$ represent the similarity between $\boldsymbol{x}^q$ and each $\boldsymbol{x}^{(i)}$ measured via a predefined metric applied to the corresponding fine-grained representations, i.e., $h\left(\boldsymbol{x}^q;\delta\right)$ and $h\left(\boldsymbol{x}^{(i)};\delta\right)$. Here, δ denotes the parameters of a retrieval model h. For instances whose labels are consistent with the fine-grained category of $\boldsymbol{x}^q$, we form them into a positive set $\mathcal{P}_q$ and obtain the corresponding $\mathcal{S}_P$ The retrieval model $h(\cdot;\delta)$ can be trained by maximizing the ranking-based score

$$\max_{\delta} \frac{\mathcal{R}\left(i,\mathcal{S}_P\right)}{\mathcal{R}\left(i,\mathcal{S}_\Omega\right)}$$

w.r.t. all query images, where $\mathcal{R}\left(i,\mathcal{S}_P\right)$ and $\mathcal{R}\left(i,\mathcal{S}_\Omega\right)$ refer to the rankings of instance i in $\mathcal{P}_q$ and Ω, respectively.

Fine-grained image retrieval tasks can be further categorized into content-based and sketch-based approaches. The former focuses on analyzing the content of the query image, while the latter utilizes hand-drawn sketches as queries. In this thesis, we focus on content-based fine-grained image retrieval tasks.

While [1] appraises the success of supervised deep metric learning-based approaches that map image data to an embedding space, where similar fine-grained images are close together, and dissimilar images are distant, it failed to catch up to the more recent trends of contrastive representation learning techniques to achieve this goal. In the next section, we introduce contrastive representation learning.

2.2. Contrastive Representation Learning

According to [2], the goal of (deep-metric) contrastive representation learning is to learn an embedding space in which similar sample pairs remain close while dissimilar samples are distant from each other. This goal aligns with the deep metrics learning approach for fine-grained image retrieval tasks, as described in Section 2.1 and addresses the distinguishability characteristics discussed in Section 1.1.

More specifically, contrastive representation learning techniques can be applied in both supervised and unsupervised settings and can be scaled gradually from a completely supervised setting to data without extra supervision signals [3]. Although supervised methods have shown promising results previously, as stated in 2.1, we are interested in exploring the potential of self-supervised contrastive learning techniques, where data annotations are not required, and we compare their performance to that of supervised methods in fine-grained image retrieval tasks.

There are various approaches to self-supervised contrastive learning. According to [4], we can categorize these into four categories:

- **Deep-metric Learning Family**: As previously mentioned, these methods promote similarity between semantically transformed versions of an input. Typically, contrastive loss is used to convert this principle to a learning objective. A well-known method in this category is SimCLR [5], which learns visual representations by fostering the similarity between two augmented views of an image. After encoding each view, SimCLR employs a projector to map the initial embeddings into another space where contrastive loss is applied to encourage similarity between the views. Section 4.2 will provide more details on SimCLR.
- **Self-Distillation Family**: Unlike the deep metric learning family, these methods process two views of the same image using two encoders, mapping one view to the other via a predictor. Self-distillation prevents the encoders from collapsing by predicting a constant for any input. Several notable contrastive learning methods, such as BYOL [6], MoCo [7], and DINO [8], fall into this category. BYOL, which stands for Bootstrap Your Own Latent, was the first to introduce self-distillation. It uses a target network to generate the target representation of the input through image augmentations and a predictor to map the online/student network's output to the target/teacher representation. The student network is updated throughout training using gradient descent, while the teacher network is updated with exponential moving average (EMA) updates of the weights of the online network. MoCo, which stands for Momentum Contrast, maintains a queue of negative samples and updates the momentum encoder with the current encoder's parameters. DINO, which stands for DIstillation of knowledge with NO labels, applies self-distillation in the context of Vision Transformers (ViT) [9]. Like BYOL, DINO employs a teacher and a student network but centers the output of the student network using a running mean (to avoid sensitivity to mini-batch size) and discretizes the representations smoothly using softmax and the temperature parameter τ.
- **Canonical Correlation Analysis (CCA) Family**: These methods leverage the principles of CCA to maximize the correlation between two augmented views of the same data. These ideas were extended to deep learning in Deep Canonically Correlated Autoencoders (DCCAE), which uses an autoencoder regularized via CCA. Among these, two notable methods are SwAV [10] and VICReg [11]. SwAV computes the corresponding codes, and the loss quantifies the fit by swapping two

codes using the loss to measure the fit between a feature and a code. The swapped fit prediction depends on the cross-entropy between the predicted code and a set of trainable prototype vectors. The model is optimized by maximizing the similarity between the features and the prototypes [2]. VICReg, which stands for Variance-Invariance-Covariance Regularization, addresses the collapse problem by introducing three regularization terms: variance, invariance, and covarianceThe variance term ensures that the embeddings have nonzero variance along each dimension, thereby preventing collapse. The invariance term maximizes the similarity between the embeddings of different views of the same image, whereas the covariance term reduces redundancy by de-correlating different dimensions of the embeddings.

- **Masked Image Modeling**: These methods apply degradations to training images, such as decolorization, noise, or shuffling image patches, and teach models to undo these degradations. Although these early self-supervised methods have failed to prosper, some self-distillation methods like iBOT [12] and DINOv2 [13] also employ masked image modeling techniques. iBOT (Image BERT Pre-Training with Online Tokenizer) is an update of DINO, where students predict tokens for masked image patches from the representations of the entire image generated by the teacher network. DINOv2 [13] further builds on iBOT and improves its performance by improving several engineering aspects, such as training recipes and architecture choices. Now, DINOv2 is generally considered the state-of-the-art self-supervised contrastive learning method.

It is noteworthy that while some research papers like [8], [11], [12], and [13] do evaluate the performance of image retrieval tasks on the open-sourced benchmark datasets, it is mainly performed using the k-nearest neighbor search of labels on natural images like ImageNet [14]. To the best of our knowledge, no study has evaluated their performance at the fine-grained level in an industry scenario. Thus, it leaves a gap in the existing literature that we aim to fill with this thesis.

3. The FGDS Dataset

3.1. Introduction

As discussed in Section 1.1, the current image search system in the visual quality inspection domain faces a significant challenge in addressing fine-grained visual similarity. However, no existing datasets are available for training the model and evaluating fine-grained similarity search results. To overcome this, we must construct a dataset with fine-grained similarity annotations to evaluate the performance improvements of image search systems with our proposed methods.

Constructing this dataset involves answering two key questions:

- What type of data should be used to construct the dataset?
- How should the fine-grained similarity be represented when annotating the dataset?

To address the first question, we observed that the search application requires only defect areas, rather than entire images, for comparison with other defect areas. Based on this fact, we constructed our dataset by extracting only the defect areas from existing datasets. We refer to these as defect croppings, which will be used for training and evaluation.

To address the second question, we must first define a data structure that effectively represents fine-grained visual similarity. Inspired by Wang et al. [15], we chose to use a set of triplets in the form of (p, p^+, p^-), where p is the reference image and p^+ is more similar to the reference image (positive) than p^- (negative), to characterize the relative similarity ordering.

Following these observations and decisions, this section describes the process of constructing the dataset. The dataset is named the Fine-Grained Defect Similarity (FGDS) dataset. It comprises defect croppings extracted from large-scale, open-source domain datasets and fine-grained similarity annotations in triplets collected from human preferences.

It is important to note that, in this scenario, we evaluate visual similarity based solely on the visual appearances of these defect croppings without considering domain knowledge (e.g., defect nature). Although this approach can introduce some subjectivity, we mitigate this effect by collecting multiple annotations from different annotators for each triplet and using majority voting to determine the ground truth.

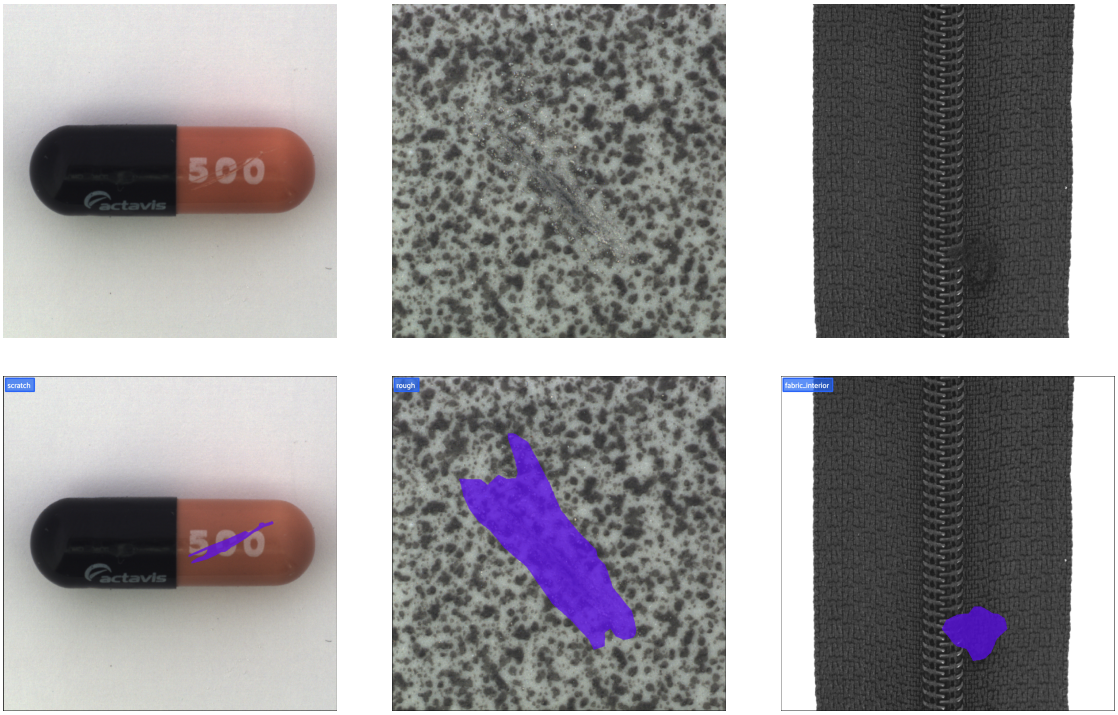
3.2. Data Source

There are several open-sourced datasets of defect images available for industrial inspection. Among them, we consider using the MVTec anomaly detection dataset (MVTec AD) [16] and VISION dataset [17] as the source to construct our dataset due to their popularity and sizes.

3.2.1. MVTec AD

MVTec AD is a dataset for benchmarking anomaly detection methods in industrial inspection. It contains 5354 high-resolution images from 15 products and 73 defects [16]. Each category comprises a set of defect-free training images and a test set of images with defects. For each defect image, MVTec provides pixel-accurate ground truth regions.

Figure 2 shows some example images from the MVTec AD dataset.



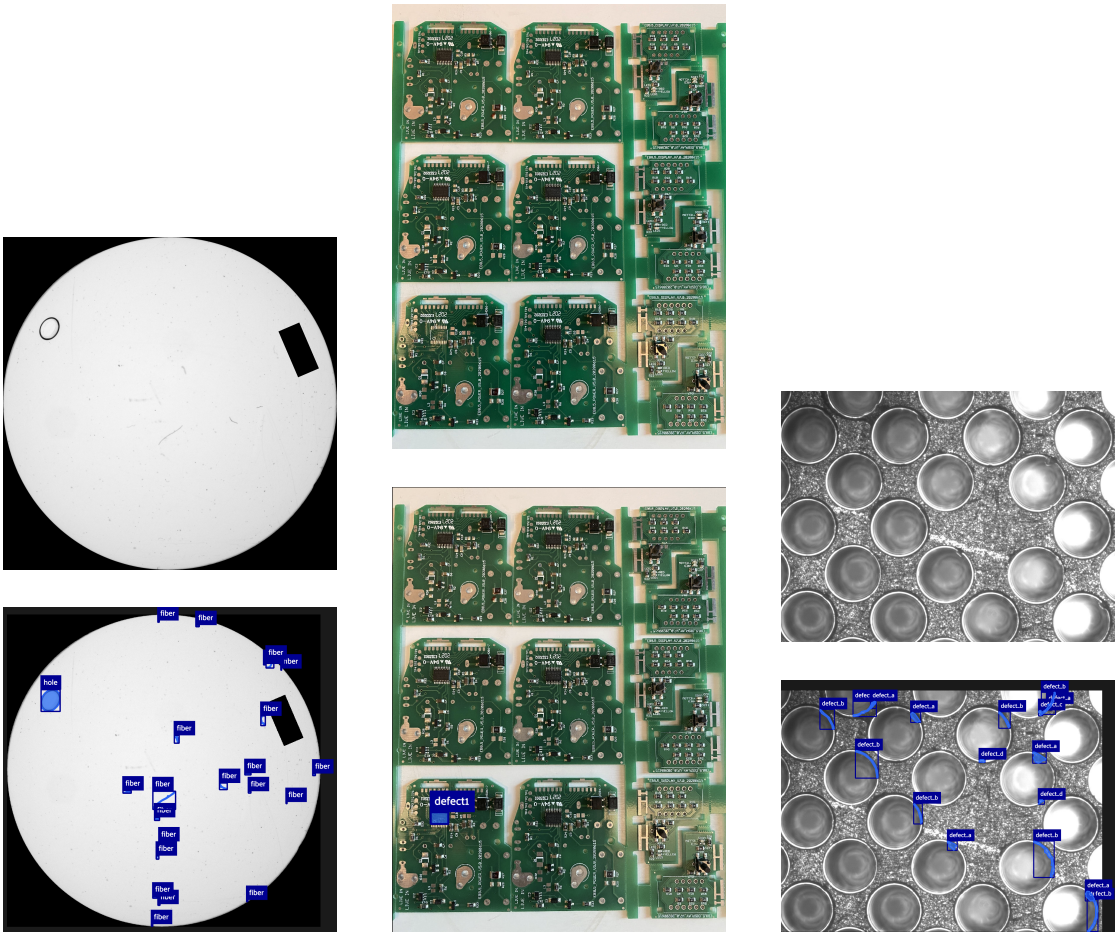
(a) `capsule` (b) `tile` (c) `zipper`

Figure 2: Example images from the MVTec AD dataset. The top row shows the original images, while the bottom row shows the images with the defect areas masked and the defect categories annotated.

3.2.2. VISION

The VISION dataset is a collection of 14 industrial inspection products, with 18422 images encompassing 44 defect types [17]. Each product has object detection and instance segmentation annotations in training and validation splits, while the test splits are not annotated.

Figure 3 shows some example images from the VISION dataset.



(a) `lens` (b) `pcb_2` (c) `hemisphere`

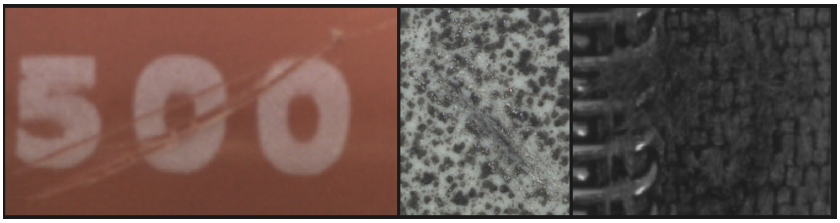
Figure 3: Example images from the VISION dataset. The top row shows the original images, while the bottom row shows the images with the defect areas masked.

3.3. Create Defect Croppings

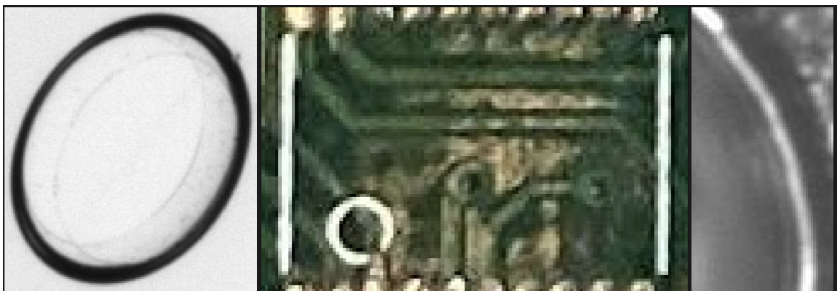
As discussed in Section 3.1, we constructed our dataset by starting from cropping out the defect areas from the annotated subsets of the MVTec AD and VISION datasets, using the provided segmentation masks with the tightest bounding boxes. We also left out the 4 "combined" defects in MVTec AD, as they contain more than one defect per image without specifications of defect categories. The defect croppings were saved as individual images, along with their source dataset, product type, and defect category information.

The resulting dataset comprises 5731 croppings, covering 29 products and 113 unique defects. Of these, 1258 croppings are from the MVTec AD dataset, while the remaining 4473 croppings are from the VISION dataset.

Figure 4 shows some example defect croppings from the dataset, echoing the examples from Figure 2 and 3.



(a) MVTec AD



(b) VISION

Figure 4: Example defect croppings from the dataset, aligning with the respective images shown in Figures 2 and 3.

In practice, the resulting dataset is created and managed using FiftyOne [18], an open-source Python package for building high-performance computer vision datasets.

3.4. Data Splits

3.4.1. Split Definition

As discussed in Section 1.1, the defect croppings will be used for training and evaluation. Consequently, it is necessary to divide the defect croppings into specific subsets tailored for different tasks. More specifically, to evaluate the performance of the image search system, two distinct subsets need to be established:

- **Evaluation Subset:** This subset contains samples with known product and defect information, which are used as queries to search for similar samples.
- **Database Subset:** This subset comprises samples with known product and defect information, which will be inferred and queried by the samples from the evaluation subset.

In addition to these subsets, a training subset is required for fine-tuning. Therefore, the defect croppings should be divided into training, evaluation, and database subsets.

3.4.2. Split Criteria

For constructing the evaluation and database subset, we randomly chose three samples from every defect category in the defect croppings as the evaluation subset and at least five samples, or ten if the number of samples is more than 15, as the database subset.

We determined the training subset based on product-level and defect-level difficulty using zero-shot performance of a pre-trained ViT-L-14 backbone from DINOv2 [13]. We also left some products out of the training set to test the generalization ability of the training methods. More specifically, we define the label-level performance thresholds as follows:

- **High Precision**: mean precision@10 > 0.9
- **High Average Precision**: mean AP@10 > 0.9
- **Low Precision**: mean precision@10 < 0.3
- **Low Average Precision**: mean AP@10 < 0.3

where Precision@K is defined as the number of relevant documents retrieved by the search engine divided by the total number of documents retrieved, and AP@K is defined as the precision of the top k documents averaged by the number of relevant documents in the top k search results [19]. More detailed definitions are in Section 5.1.1. We define the defect-level difficulty as follows:

- **Easy Defects**: defects that have both high precision AND high AP
- **Hard Defects**: defects that have either low precision OR low AP

- **Medium Defects**: the rest of the defects

These defects are then grouped by their respective product types. Among them, the two products containing only easy and medium defects and seven containing only medium defects are left out from training to test the generalizability of the propose method. The remaining five products, which have a standard deviation of product-wise precision@10 OR AP@10 > 0.2, are still used for training due to their large variance in performance. This way, the left-out products contain mostly medium defects with low product-wise standard deviations. We call them non-train products in the rest of the report.

3.4.3. Split Statistics

As a result, the evaluation subset contains 339 croppings, and the database subset contains 1021 croppings respectively. The training subset consists of 3079 croppings in total. Among them, 20 out of 113 defect categories are considered "hard" defects, and 9 out of 29 products (containing 35 defects) are left out from the training set to test the generalization ability of the training methods. Appendix A shows a detailed list of these defects and products.

3.5. Create Triplets

Following the definition of triplets in Section 3.1 and the subsets obtained in Section 3.4, we use each image R in the evaluation subset as the reference image. Every image pair (A, B) within the same defect category in the database subset is then used to construct the triplet (R, A, B). Annotators are subsequently asked to choose one of three options:

- Image A is more similar to the reference image.
- Image B is more similar to the reference image.
- Cannot determine which image is more similar.

This method allows us to obtain positive and negative samples as perceived by humans within the triplet framework.

As a result, we obtained a total of 13,008 triplets. Of these, 4,305 (33.1%) are triplets in the same defect category as the non-train products. Among the remaining 8,703 triplets, 1,944 (22.3%) belong to the same defect category as the hard defects.

3.6. Annotation Pipeline

We use the Amazon Sagemaker GroundTruth service to get the human preference annotation for the triplets obtained in 3.5. It is a managed data labeling service that makes it easy to collect data annotations for machine learning using other AWS services. In particular, Sagemaker GroundTruth gives access to the Amazon Mechanical Turk workforce, an on-demand, scalable human workforce that can help process large datasets quickly and accurately. This section describes the design of the annotation pipeline using Amazon Sagemaker GroundTruth.

3.6.1. Design Overview

Figure 5 shows the diagram of the annotation pipeline.

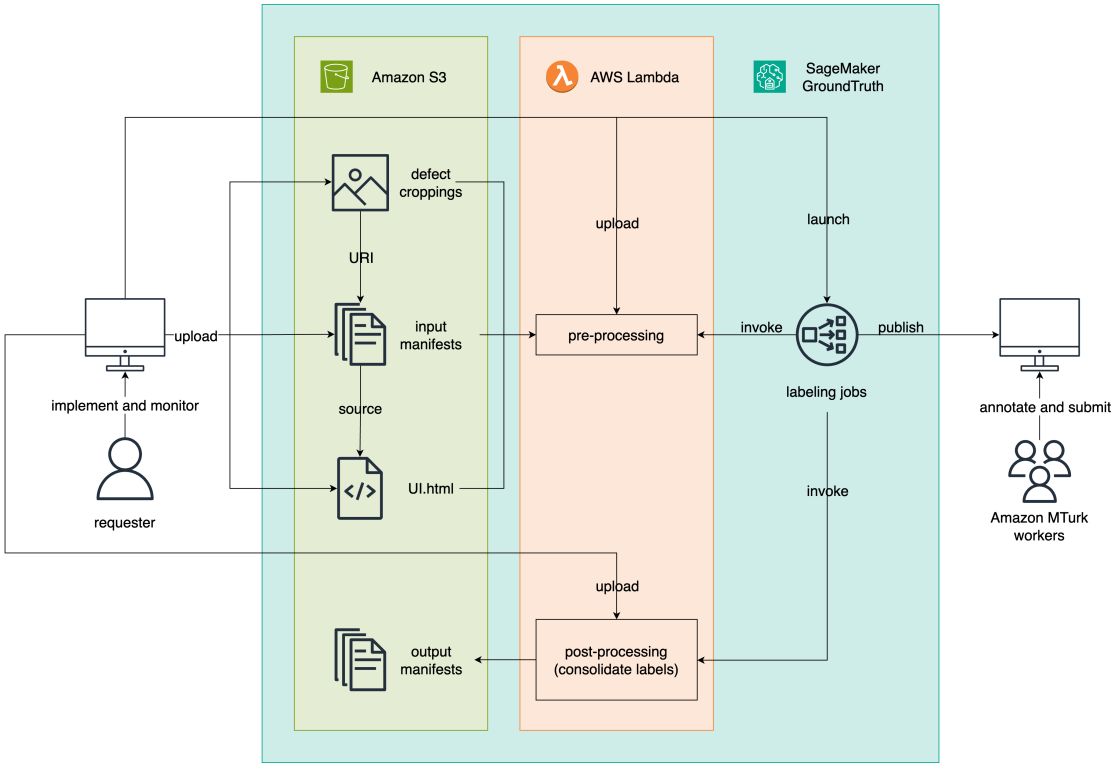


Figure 5: Annotation pipeline using Amazon Sagemaker GroundTruth.

On the SageMaker GroundTruth platform, the requester, who issues labeling tasks to the workforce, needs to prepare and upload to Amazon S3 storage service three essential components: images to be labeled (in our case, defect croppings), input manifests (JSON files as entry points for labeling), and an HTML file for the annotation UI. The requester must also prepare and upload a pre-processing function and a post-processing function

to consolidate the labeling results using the AWS Lambda service, which runs serverless functions triggered by various AWS services.

Once a labeling job is launched on the SageMaker GroundTruth platform, the pre-processing function is invoked to process the input manifest files for correct display in the annotation UI. If the data pre-processing is successful, the labeling jobs are published to the Amazon MTurk workers, who then annotate and submit their labeling tasks using the UI interface. Upon completion, the labeled data is re-uploaded to Amazon S3 storage. The post-processing function is invoked to consolidate the labels, ensuring all annotations are correctly combined. The final outputs are stored as JSON manifests, containing the fully labeled and processed data.

3.6.2. User Interface

As mentioned in 3.6.1, specifically for the triplet annotation task, we need a custom UI for the Amazon Mechanical Turk workers to annotate the triplets.

Figure 6 shows the example UI interface for annotating triplets.

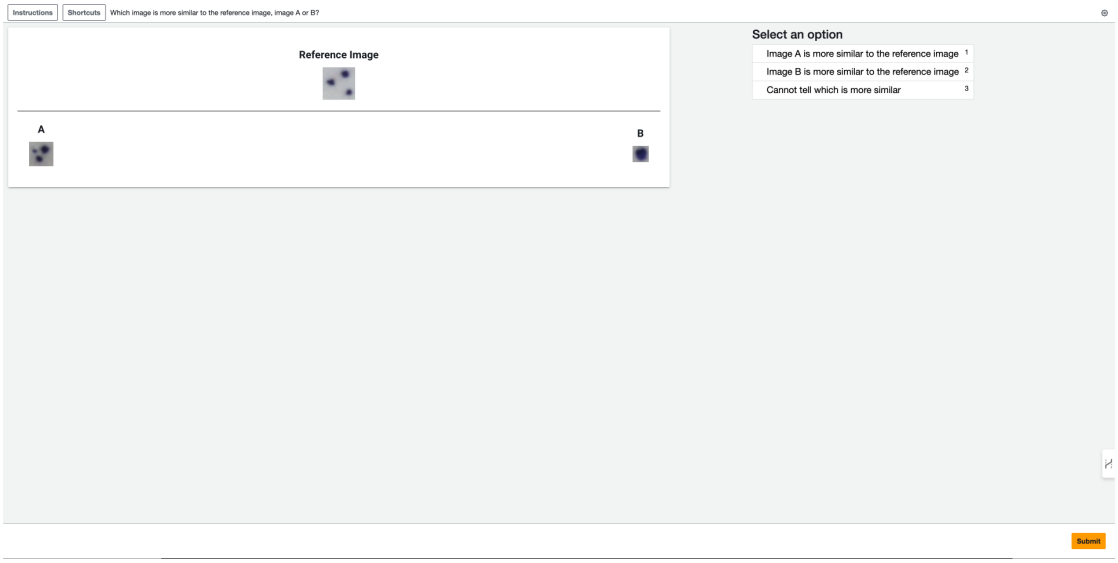


Figure 6: Example UI interface for annotating triplets.

The two images for comparison are placed at an equal distance from the reference image, and the annotators can choose one of the three options as defined in 3.5. The short and full instructions provided to the annotators are attached in the Appendix B.

3.6.3. Annotation Process

For each labeling job, we requested five workers from Amazon Mechanical Turk. The final decision was made using majority voting, with a threshold of three out of five workers: if no less than three workers marked the triplet as "Cannot tell which is more similar" or a consensus could not be reached, the triplet was denoted as "indistinguishable."

3.7. Annotation Results

3.7.1. Annotation Statistics

Consequently, among the 13008 triplets, 1075 were marked as "indistinguishable" by the workers, accounting for 8.26% of the total triplets. More than 90% of the samples are visually distinguishable by humans. These statistics indicate that the dataset is well-annotated and can be used to evaluate the similarity search performance.

3.7.2. The Ground Truth Table

To store the annotation results, we designed a relational database table named `TripletGTs`. The fields are as follows:

- `FileNameRef (Primary Key)`: The file name of the reference image.
- `FileNameFirst (Primary Key)`: The file name of the first sample in the triplet.
- `FileNameSecond (Primary Key)`: The file name of the second sample in the triplet.
- `GroundTruth`: The ground truth label.
  - 1 means the first sample is more similar to the reference image, and
  - 2 means the second sample is more similar to the reference image
  - 0 means the similarity of the two samples is not distinguishable by humans.

The table is populated from the output manifest files to the same SQLite database as the evaluation results mentioned in Section 5.2 for unified and centralized storage.

3.8. Conclusion

This section describes the process of constructing the Fine-Grained Defect Similarity (FGDS) dataset. It consists of defect croppings extracted from the MVTec AD and VISION datasets, along with fine-grained similarity annotations in triplets collected from human preferences. The dataset is divided into training, evaluation, and database subsets, and the triplets are annotated using the Amazon Sagemaker GroundTruth service. The resulting dataset can serve as a benchmark for evaluating the fine-grained visual similarity search performance of industrial defect search applications.

4. Training Methods

This section describes the training methods used to fine-tune the embedding model for the retrieval task. We first explain why we do both supervised and self-supervised contrastive learning. We then introduce the contrastive learning framework we used in the thesis, followed by training recipes like data augmentation, model architecture, and hyperparameter settings.

4.1. Supervised vs Self-supervised Contrastive Learning

As explained in Section 2.2, contrastive learning can be supervised and self-supervised. We find both worthy of exploration, as on the one hand, supervision labels are strong and efficient training signals if the defects are still not well classified, but using them for image-level supervision often drastically reduces the rich visual information in the image down to a single concept selected from a predefined set of object categories, which may destroy the fine-grained semantic features [8]; on the other hand, self-supervision methods, though works less efficiently, generates better features and retains the image details [20]; in addition, self-supervision does not require any labeling, so they have the potential to scale to large-scale and unlabeled datasets.

4.2. SimCLR-based Contrastive Learning Framework

As discussed in Section 2.2, many different contrastive learning frameworks are available. We use the SimCLR-based contrastive learning framework introduced in [5] due to its simplicity and effectiveness. In [5], the model learns representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space.

Figure 7 illustrates the SimCLR-based contrastive learning framework. Given an input batch of data, we first apply data augmentation twice to obtain two copies of the batch. Both copies are forward propagated through the encoder network to obtain a normalized embedding. This representation is further propagated during training through a projection network discarded at inference time. The supervised contrastive loss is computed on the outputs of the projection network [3].

To construct the training batch for SimCLR-based contrastive learning, we randomly sample a minibatch of N examples and define the contrastive prediction task on pairs of augmented examples derived from the minibatch, resulting in $2N$ data points. We call this a **multi-viewed batch**.

The supervised and self-supervised training is achieved by using different contrastive losses. To describe the losses, we first define some common denotations: within a multi-viewed batch, let $i \in I \equiv \{1 \dots 2N\}$ be the index of an arbitrary augmented sample,

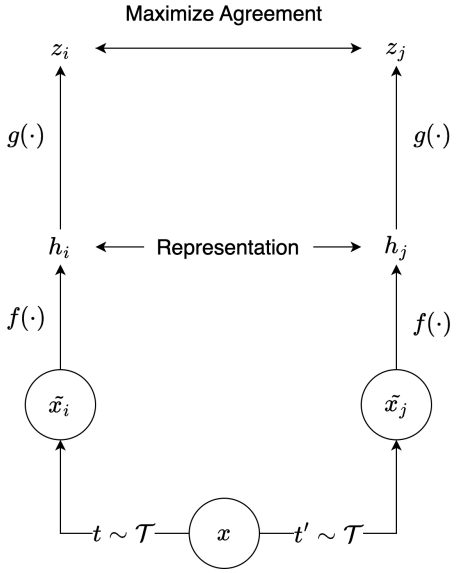


Figure 7: SimCLR-based contrastive learning framework. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation h for downstream tasks. Reproduced from [5]

and let $j(i)$ be the index of the other augmented sample originating from the same source sample, and $A(i) \equiv I \backslash \{i\}$ to denote the indices of all augmented samples in the multi-viewed batch distinct from i.

Following the denotations above and in Figure 7, we define the self-supervised contrastive loss in Section 4.2.1 and supervised contrastive loss in Section 4.2.2.

4.2.1. Self-Supervised Contrastive Loss

In self-supervised contrastive learning, a common choice for the contrastive loss is the InfoNCE loss introduced in Contrastive Predictive Coding [21] (or referred to as NT-Xent loss in the SimCLR paper [5]).It is defined as:

$$\mathcal{L}^{\text{infonce}} = \sum_{i \in I} \mathcal{L}_i^{\text{infonce}} = -\sum_{i \in I} \log \frac{\exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_{j(i)} / \tau\right)}{\sum_{a \in A(i)} \exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_a / \tau\right)}$$

where

- the index i is called the anchor, $i \in I \equiv \{1 \ldots 2N\}$

- index $j(i)$ is called the positive, and the other $2(N-1)$ indices ($k \in A(i) \backslash \{j(i)\}$) are called the negatives
- $\tau \in \mathcal{R}^+$ is a scalar temperature parameter

following the denotations in [3]. This temperature parameter is initially used as an adjustment to the softmax function, which controls the entropy of the distribution. In the context of contrastive learning, it is interpreted as an implicit control of the hard negative mining behavior, which we will investigate further in Section 6.1.

Note that each anchor i has 1 positive and $2N-2$ negative pairs. The denominator has a total of $2N-1$ terms (the positive and negatives). For convenience, we refer to this loss as "InfoNCE" in the rest of this report.

4.2.2. Supervised Contrastive Loss

The supervised contrastive loss introduced in [3] is a generalization of the InfoNCE loss to contain multiple positives (i.e., samples with the same label). It is defined as:

$$\mathcal{L}^{\text{supcon}} = \sum_{i \in I} \mathcal{L}^{\text{supcon}} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_p / \tau\right)}{\sum_{a \in A(i)} \exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_a / \tau\right)}$$

where

- the index i is called the anchor, $i \in I \equiv \{1 \dots 2N\}$
- $P(i) \equiv \left\{p \in A(i) : \tilde{\boldsymbol{y}}_p = \tilde{\boldsymbol{y}}_i\right\}$ is the set of indices of all positives in the multi-viewed batch distinct from i, and $|P(i)|$ is its cardinality.
- $\tau \in \mathcal{R}^+$ is a scalar temperature parameter

following the denotations in [3]. This temperature parameter is initially used as an adjustment to the softmax function, which controls the entropy of the distribution. In the context of contrastive learning, it is used to control the hard negative mining behavior, which we will investigate further in Section 6.1.

In this way, for any anchor, all positives in a multi-viewed batch (i.e., the augmentation-based sample as well as any of the remaining samples with the same label) contribute to the numerator instead of only using the augmented sample as the only positive as in InfoNCE. For convenience, we refer to this loss as "SupCon" in the rest of this report.

4.3. Data Augmentation

As mentioned in 4.2, SimCLR-based contrastive learning requires data augmentation to create noisy training sample versions. Proper data augmentation setup is critical for learning representative and generalizable embedding features, as the augmentation should significantly change its visual appearance but keep the semantic meaning unchanged, which encourages the model to learn the essential part of the representation [2].

However, in visual quality inspection, the semantic invariances of the defects are intrinsically heterogeneous. For example, a specific type of augmentation can create a meaningful representation of some defects while destroying others simultaneously: flipping an image can be a meaningful augmentation for a "scratch" defect, but it can be nonsensical for a "flip" defect where a part is not assembled in the right direction and creates a flip phenomenon as a consequence. This results in a dilemma: we want to use augmentations agnostic to the defect types for a large-scale dataset while also ensuring that the augmentations are meaningful for the invariances intrinsic to the defects. To address this issue, we manually chose a set of augmentations and experimented with different combinations to find the best-performing ones with all the data. The following subsections will discuss the basic augmentations and the resulting augmentation recipe.

4.3.1. Basic Augmentations

Over the years, the computer vision community has developed a set of basic augmentation techniques widely used in practice. These include spatial and geometric transformations (e.g., cropping, flipping, affine transformation), pixel-level transformations (e.g., noise, color jittering, brightness), and more advanced transformations (e.g., cutout, mixup) that encourage different semantic invariances. We briefly explain the basic augmentations we have experimented with in our experiments below (with names in the fashion of the Albumentations [22] library).

Spatial and Geometric Transformations

- `Affine`: it applies geometric changes to images, including scaling, rotation, translation, and shearing. This technique can simulate different viewpoints and object orientations.
- `Flip`: it flips the image horizontally (`HorizontalFlip`) or vertically (`VerticalFlip`).
- `Transpose`: transposes the image.
- `OpticalDistortion` simulates lens effects by warping the image, typically in a radial pattern. This augmentation can mimic the distortions found in real-world camera systems, such as barrel or pincushion distortions.

- `PixelDropout` randomly sets individual pixels in an image to zero or a specified value. This technique can help models become more robust to noise and partial occlusions.
- `RandomResizedCrop` combines random cropping and resizing operations. It crops a random portion of the input image and then resizes it to a specified size.
- `RandomRotate90`: rotates the image by 90 degrees.

Pixel-level Transformations

- `GaussNoise` adds random Gaussian noise to the image. This augmentation simulates camera sensor noise and can help models become more robust to low-quality or noisy inputs.
- `ISONoise`: it simulates noise patterns typically found in digital camera images with high ISO settings. This augmentation adds both color and luminance noise to the image. It benefits training models that need to work with low-light or high-sensitivity photography.
- `MultiplicativeNoise`: it applies element-wise random multiplication to the pixel values. It can create more realistic noise patterns than additive noise, as it preserves the relative brightness of different image regions. It is useful for simulating various lighting conditions and sensor artifacts.
- `UnsharpMask`: a sharpening technique that enhances image edge contrast. It works by subtracting a blurred version of the image from the original. This augmentation can help models focus on important features and edges.
- `RandomToneCurve`: it applies random adjustments to the image's tone curve, affecting brightness, contrast, and color balance. This augmentation simulates variations in image processing and color grading.

4.3.2. Augmentation Recipe

We experimented with all the augmentation methods and their parameters listed in 4.3.1. Among them, we figured out five basic augmentation methods that achieved the best performance when used in combination: `Flip`, `PixelDropout`, `OpticalDistortion`, `Affine`, and `GaussNoise`. The detailed configuration of these augmentations is in Appendix C. The detailed analysis of how these augmentations affect the model performance is discussed in Section 6.2.

4.4. Model Architecture

We use the Vision Transformer (ViT) [9] as the backbone for our contrastive learning framework due to its strong performance in various computer vision tasks. ViT is based on the transformer architecture, which leverages self-attention layers. It divides the input image into fixed-size patches and flattens them into a sequence of tokens, which are then fed into the transformer encoder. The transformer encoder processes the images as these sequences of tokens to learn the image representation. Then, FFN (Feed-Forward Network) layers are used to project the learned representation into the final embedding space. The transformer encoder is a stack of multiple layers containing a multi-head self-attention mechanism and a position-wise feed-forward network.

More specifically, in training, we use a ViT-B-14 backbone pre-trained by DINOv2 [13], which receives a patch size of 14 as the input and has 12 attention layers in the encoder with an output embedding size of 768, resulting in 86M parameters in total. We also used a single layer of MLP of size 512 as the projection head, following the design in [3] and [5].

4.5. Training Details

After experimenting with different training recipes, we found the following hyperparameters and implementation details to be the most effective for our task.

4.5.1. Hyperparameters

We used a layer-wise fine-tuning strategy to train the model. First, we used a constant learning rate of 4 to tune the MLP head for 50 epochs while keeping the backbone frozen, and then unfroze the last six attention layers of the backbone (more on this in Section 4.5.2) and fine-tuned them with an initial learning rate of 5e-2, together with the projection head. We used a cosine decay learning rate scheduler for the backbone and the projection head with a minimum learning rate 5e-3. We used the LARS [23] optimizer, a variation of SGD targeting large batch-size scenarios with a momentum of 0.9, trust coefficient of 1e-3, weight decay of 1e-4, and epsilon of 1e-8. The training was performed for 300 epochs with an early stopping patience of 20 and a minimum delta of 1e-4, which typically takes 1.5 hours. The batch size was set to 256. The training was performed on an AWS `g5.2xlarge` EC2 instance with an NVIDIA A10G 24GB GPU.

4.5.2. Implementation Details

According to [24], it is sufficient to fine-tune the attention layers of the vision transformers to achieve good performance while costing 10% less memory and speeding up the training

by 10%. Figure 8 illustrates the attention-only fine-tuning strategy compared to full fine-tuning.

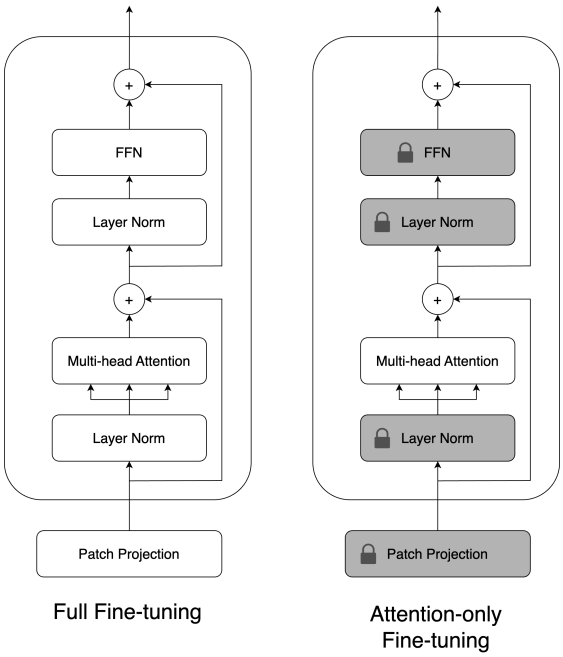


Figure 8: Attention only fine-tuning. Other parameters like FFN and normalization layers are frozen during training. Inspired by [24]

Moreover, to further reduce memory consumption and accelerate training, we use true half-precision training with BF16 (brain floating point). It is a floating-point format that uses 16 bits to represent a number. It compromises the memory efficiency of half-precision (FP16) and the numerical stability of single-precision (FP32). BF16 can accurately represent a wide range of values while providing significant memory savings compared to FP32.

Finally, to further accelerate the training process, we use xFormers [25] used by DINOv2. xFormers is a library that provides efficient implementations of various transformer models. It is designed to improve transformer models' training and inference speed by leveraging kernel fusion, memory optimization, and parallelization techniques. By using xFormers in our code, we can benefit from its optimizations and accelerate the fine-tuning of the ViT backbone.

Combining these strategies allows us to efficiently fine-tune the ViT backbone with reduced memory consumption and faster training speed.

5. Evaluation Metrics and Tools

As briefly discussed in Section 3.4, we use the evaluation subset of the FGDS dataset defined in Section 3.1 to search for similar samples in the database subset to evaluate how well different training setups contribute to the retrieval performance. More specifically, we

- build a search index with the model on the database subset by encoding every sample in it with the model into embeddings and use the embeddings as the index of the corresponding samples in the database subset,
- encode every sample in the evaluation subset into query embeddings using the same model as the one used for building the search index,
- search for the most similar samples in the database subset with the query embeddings of the evaluation samples by computing the similarity between the query embeddings and the embeddings of the samples in the database subset,
- return a rank list of the search results for each evaluation sample and
- calculate the performance metrics with the rank lists of the search results.

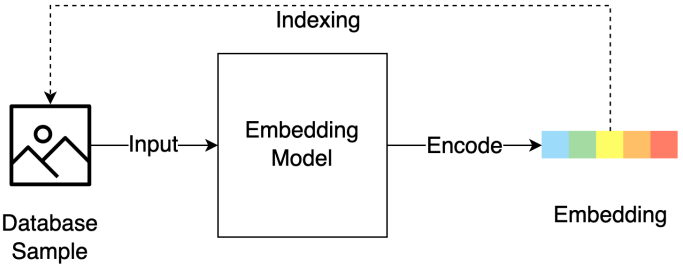
For the distance metrics used for computing similarity, we use the Euclidean distance between embeddings with the Milvus [26] vector database backend.

Figure 9a and 9b show how we build the indices on the database subset and evaluate the retrieval performance of the model using the evaluation subset of the FGDS dataset.

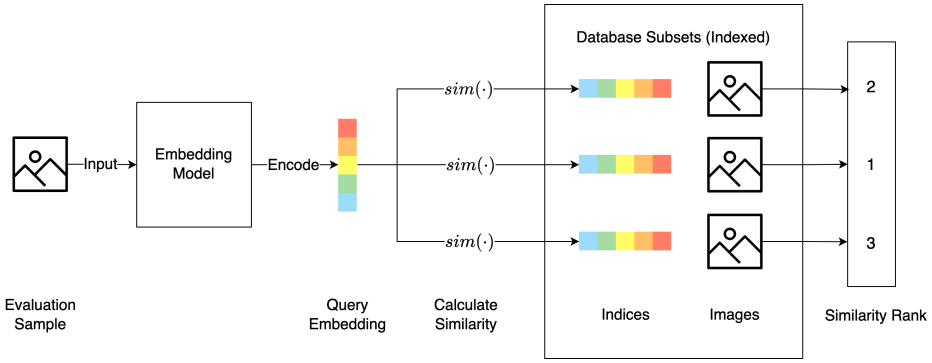
More specifically, for our retrieval system, we want to know that, given a search sample:

- how well the model can pick out the samples of the same defect category from all the samples of the same product type and
- how well the model can tell the fine-grained difference between all the database samples of the same defect category based on how visually similar they are to the search sample.

We call these two aspects label-level precision and triplet-level precision, respectively. Based on these two aspects, we can comprehensively evaluate the retrieval system's performance to match our use case and training goal.



(a) Build the search index with embedding model on the database subset



(b) Search for similar samples in the database subset with the embeddings of evaluation samples

Figure 9: Building the search index and searching for similar samples

5.1. Evaluation Metrics

5.1.1. Label-level Precision

The label-level precision is the precision of spotting the same **defect categories** when searching within the same **product type**, similar to k-nearest neighbor classification. In information retrieval, precision-based metrics are often used to evaluate search performance. Here, we use precision@k and AP@k, as briefly introduced in Section 3.4.2.

- **Precision@k** is defined as the number of relevant documents retrieved by the search engine divided by the total number of documents retrieved:

$$\text{precision@k} = \frac{\text{number of relevant documents among top } k}{\text{total number of documents } (k)}$$

It is important to note that precision@k can be suboptimal if k is greater than the total number of relevant results. Additionally, if all relevant samples are included (no false negatives), the precision@k value will decrease as k increases due to more false positives. Also, it is worth mentioning that precision@k does not take into

account the order of relevant items appearing among retrieved documents, so AP@k is introduced to address this issue.

- **AP@k** is defined as the precision of the top k documents retrieved by the search engine, averaged by the number of relevant documents in the top k search results:

$$AP@k = \frac{1}{r}\sum_{i=1}^{k} \text{precision@}i \cdot R_i$$

$$r = \text{number of relevant items}$$

$$R_i = \begin{cases} 1, & \text{if document } i \text{ is relevant} \\ 0, & \text{if document } i \text{ is not relevant} \end{cases}$$

  Compared to precision@k, AP@k considers the order of relevant items appearing among retrieved documents.

In this work, we use $k = 5$ and $k = 10$ for precision@k and AP@k.

5.1.2. Triplet-level precision

The triplet-level precision is the precision of ranking the **similarity triplets** annotated by humans when searching within a certain **defect category**. We used the two metrics, similarity precision and score-at-top-K, introduced by Wang et al. in [15]. For evaluating triplet-level precision:

- **Similarity precision** is defined as the percentage of triplets being correctly ranked.

$$\text{Similarity Precision} = \frac{\text{number of correctly ranked triplets}}{\text{total number of triplets}}$$

  Given a triplet $t_i = \left(p_i, p_i^+, p_i^-\right)$, where p_i^+ should be more similar to p_i than p_i^-. Given p_i as query, if p_i^+ is ranked higher than p_i^-, then we say the triplet t_i is correctly ranked.

- **Score-at-top-K** is defined as the number of correctly ranked triplets minus the number of incorrectly ranked ones on a subset of triplets whose ranks are higher than K.

$$\text{Score-at-top-K} = (\text{number of correctly ranked triplets at top k}) - (\text{number of incorrectly ranked triplets at top k})$$

  One triplet's rank is higher than K if its positive image p_i^+ or negative image p_i^- is among the top K nearest neighbors of the query image p_i.

  According to [15], Score-at-top-K is similar to precision-at-top-K, widely used to evaluate retrieval systems. Intuitively, score-at-top-K measures a retrieval system's performance on the K most relevant search results. This metric can better

reflect the performance of the similarity models in practical image retrieval systems because users pay most of their attention to the first few results returned.

In this work, we use $k = 5$ and $k = 10$ for score-at-top-K.

5.2. Database Design

To effectively and conveniently store and calculate the evaluation results, we designed a database to store the results of the experiments. We created three tables: `Results`, `RankListLabel`, and `RankListTriplet`, to store the aggregated evaluation results and the returned rank lists when searching at the label-level and triplet-level. In practice, we stored these tables in the same SQLite database as `TripletGTs` mentioned in Section 3.7.2. The following sections briefly describe the design of these tables, and the detailed schema is in Appendix D.

5.2.1. Results

This table stores the evaluation results of different experiments. Each row represents the performance metrics calculated by searching the database subset with the respective sample from the evaluation subset of the FGDS dataset in a single experiment.

5.2.2. RankListLabel

This table stores the rank lists of the samples when searching at the label level. Each row in the table corresponds to the rank of a database sample when searching for samples with the same defect category as the evaluation sample in a single experiment.

5.2.3. RankListTriplet

This table stores the rank lists of the samples when searching at the triplet level. Each row in the table corresponds to the rank of a database sample when searching within the same defect category with an evaluation sample in a single experiment. Note that we reconstructed the ground truth rank (RankGT) from the triplet annotations, but some of the rankings cannot be reconstructed due to the nature of the triplet annotations. This field only serves as a reference for the evaluation.

6. Experiment Design and Result Analysis

Using the training methods described in Section 4 and the evaluation metrics described in Section 5, we conducted a series of experiments to evaluate and compare the performance of different training recipes.

More specifically, we grouped our experiments into vertical and horizontal experiments. Vertical experiments aim to understand the effects of different training configurations on model performance. In contrast, horizontal experiments aim to compare the effects of different training paradigms (i.e., supervised and self-supervised contrastive learning) on model performance.

For vertical experiments, according to [2], three critical components in contrastive learning are heavy data augmentation, large batch size, and hard negative mining. As briefly introduced in Sections 4.2.1 and 4.2.2, the loss temperature hyperparameter of SupCon and InfoNCE inherently facilitates hard negative mining. Additionally, the batch size must be as large as possible to ensure hard negative samples are observed [4]. Therefore, we primarily investigate the effects of loss temperature and augmentation methods on model performance for a specific experimental setting.

6.1. Vertical Experiments on the Effects of Loss Temperatures

Hard negative samples are defined as the samples with different labels from the anchor sample but have embedding features very close to the anchor embedding [2]. It is mathematically proven in [3] that the contrastive loss provides an intrinsic mechanism for hard negative mining during training: using low temperatures is equivalent to optimizing for hard positives/negatives. However, high temperatures also have competing effects, which can result in smaller magnitude gradients (making optimization easier) and make the model more tolerant to semantically consistent samples [27]. In light of this, we need to experiment with different loss temperatures to evaluate how they affect the model performance in search results.

6.1.1. Experimental Setup

We evaluate the overall performance and the performance on the hard and non-hard defects defined in section 3.4.2, at both label and triplet levels. The parameter setting is the same as the default setting in Section 4.5.1, except for the loss temperatures, which are set to 0.01, 0.05, 0.1, 0.2, and 0.5, respectively, for both supervised and self-supervised settings.

6.1.2. Results and Analysis

Table 1 shows the overall performance of the model with different loss temperatures, and Figure 10 and 11 shows the performance of the model on hard and non-hard defects defined in Section 3.4.2, both at both label and triplet levels.

Metrics \ Temperatures	Zero-shot	0.01	0.05	0.1	0.2	0.5
PrecisionAt5	0.7292	0.7546	0.7794	0.7900	**0.7929**	0.7841
PrecisionAt10	0.5882	0.6071	0.6295	0.6389	0.6419	**0.6440**
APAt5	0.8648	0.8942	0.9009	**0.9078**	0.8989	0.9029
APAt10	0.8222	0.8461	0.8604	**0.8690**	0.8676	0.8646

(a) supervised, label-level

Metrics \ Temperatures	Zero-shot	0.01	0.05	0.1	0.2	0.5
SimilarityPrecision	0.6837	**0.7611**	0.7587	0.7571	0.7580	0.7503
ScoreAtTop5	11.2124	**15.6696**	15.6431	15.6637	15.4838	15.0442
ScoreAtTop10	12.3333	**17.7198**	17.5664	17.5723	17.5428	17.0708

(b) supervised, triplet-level

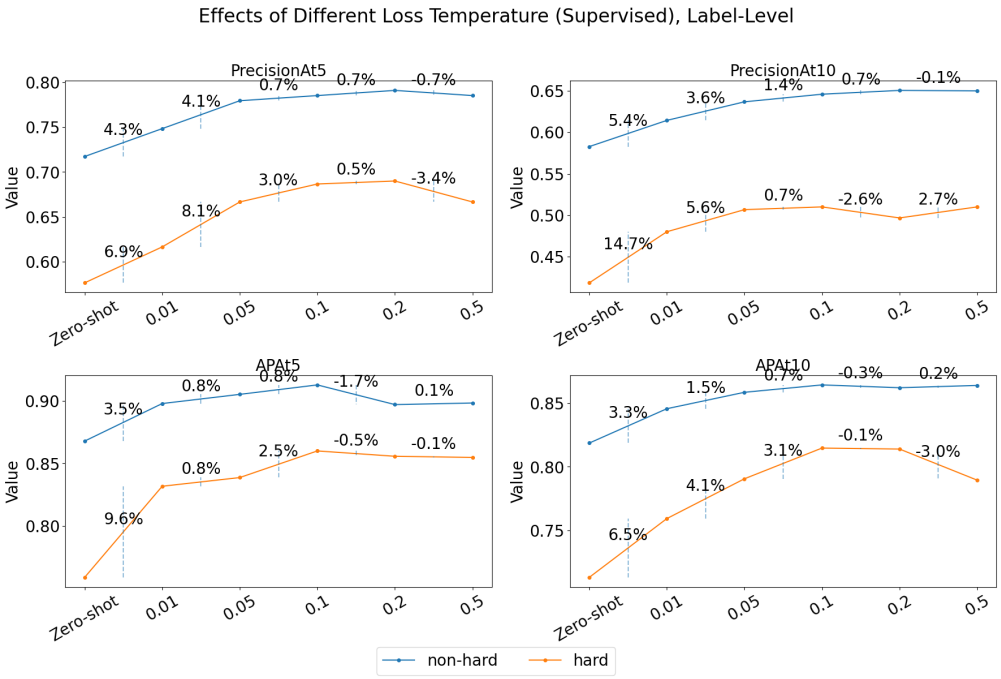
Metrics \ Temperatures	Zero-shot	0.01	0.05	0.1	0.2	0.5
PrecisionAt5	0.7292	0.7764	0.7799	0.7776	0.7811	**0.7835**
PrecisionAt10	0.5882	0.6360	0.6319	0.6354	0.6339	**0.6372**
APAt5	0.8648	0.9019	0.9090	0.9123	0.9122	**0.9152**
APAt10	0.8222	0.8561	0.8649	0.8656	**0.8689**	0.8675

(c) self-supervised, label-level

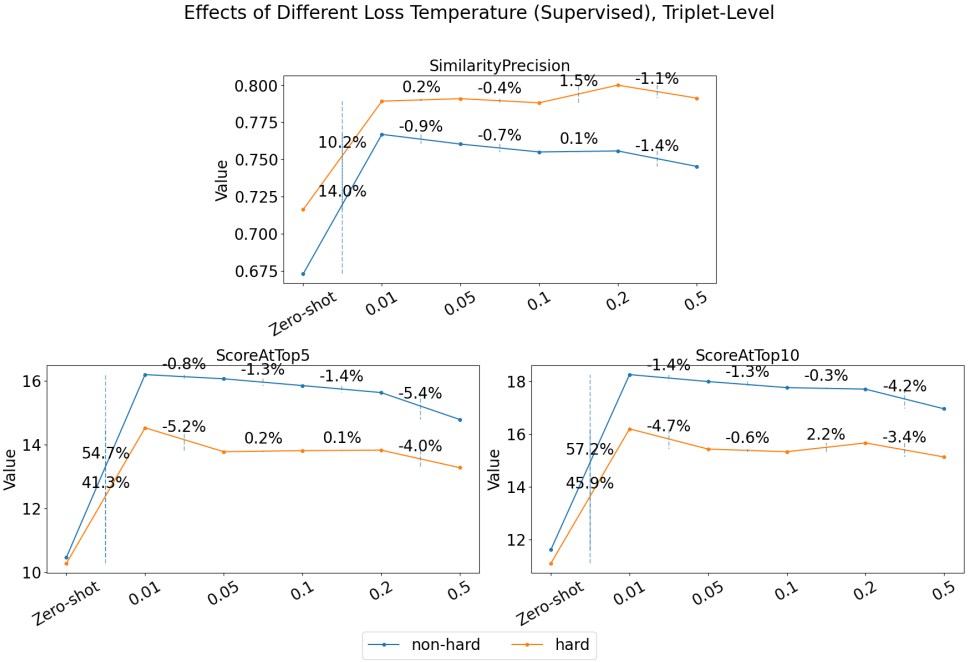
Metrics \ Temperatures	Zero-shot	0.01	0.05	0.1	0.2	0.5
SimilarityPrecision	0.6837	**0.7659**	0.7595	0.7635	0.7622	0.7575
ScoreAtTop5	11.2124	**16.0295**	15.7611	15.8938	15.8348	15.7050
ScoreAtTop10	12.3333	**18.3451**	17.7847	18.0737	18.0678	17.7847

(d) triplet-level

Table 1: Effects of different loss temperature, self-supervised. The bold numbers indicate the best performance in each metric.

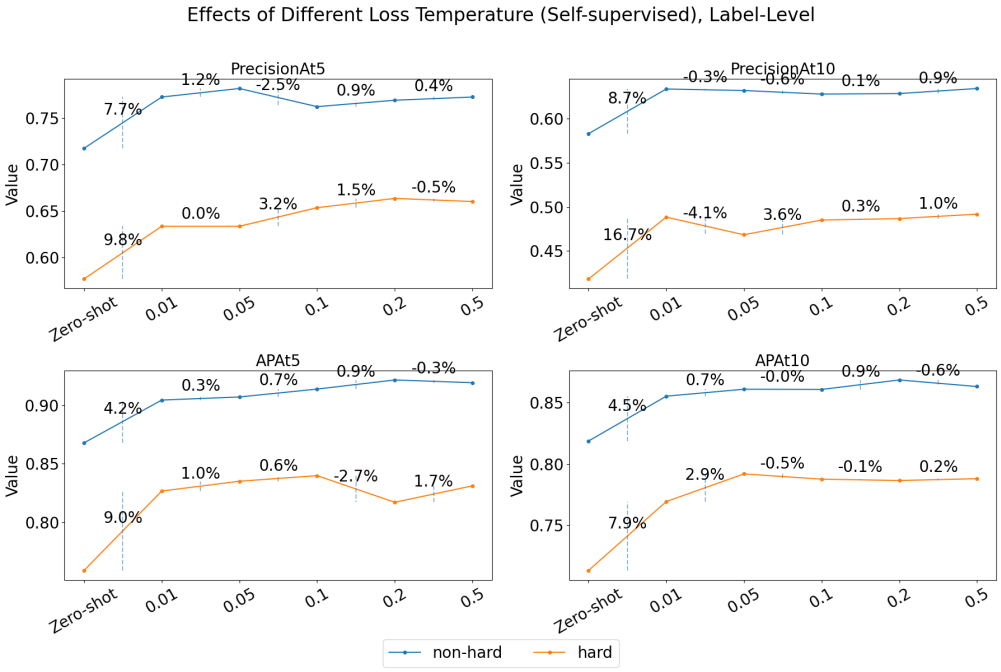


(a) label-level

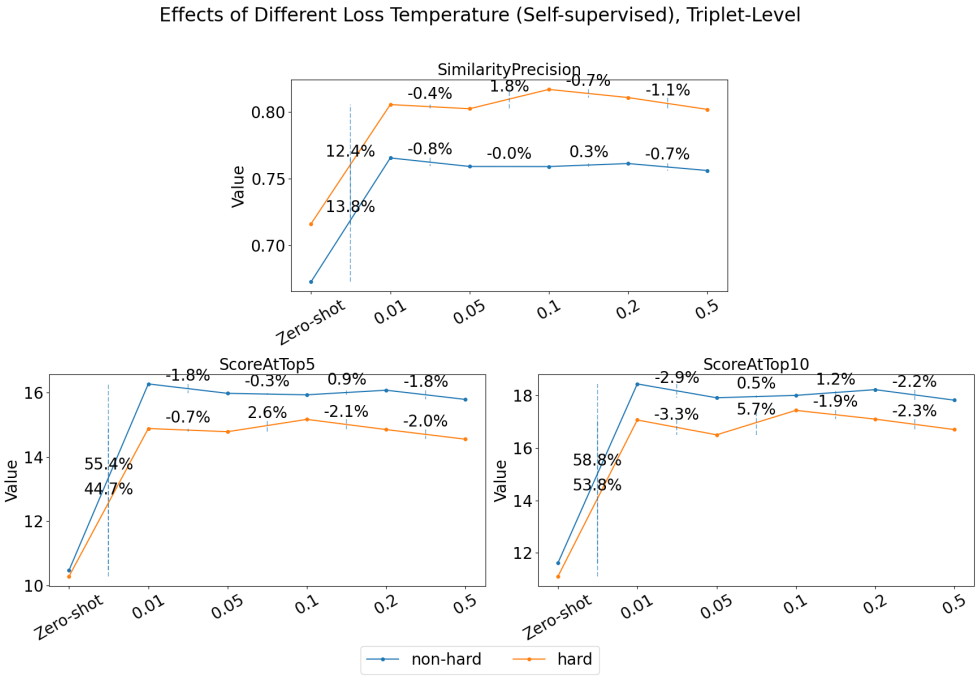


(b) triplet-level

Figure 10: Effects of different loss temperatures, supervised. The vertical dashed lines with percentages indicate the performance gain between the two adjacent experiments.



(a) label-level



(b) triplet-level

Figure 11: Effects of different loss temperatures, self-supervised. The vertical dashed lines with percentages indicate the performance gain between the two adjacent experiments.

Overall, the results showed that the training increases performance in all cases, but the optimal loss temperature varies depending on the training method and the precision level. Specifically:

- For supervised settings, the optimal temperature generally lies between 0.1 to 0.2 for label-level precision, while the optimal temperature is consistently 0.01 for triplet-level precision.
- For self-supervised settings, the optimal temperature is generally around 0.5 for label-level precision, while the optimal temperature is consistently 0.01 for triplet-level precision.
- As for the hard defects, at the label level, the performance on hard defects benefits from using a smaller loss temperature only for the self-supervised setting, with the two AP@Ks, while in supervised settings, the performance for the non-hard and hard defects are roughly the same. In contrast, at the triplet level, the optimal temperatures for hard defects are larger than those for non-hard defects.

From the above observation, we can see that, on the one hand, at the label level, using relatively larger loss temperatures benefits self-supervised learning more than supervised learning. This observation aligns with the findings from [27]: using a larger temperature makes the embedding space more tolerant to similar samples. As there are more positives in the supervised loss than in the self-supervised loss, one has to use a larger temperature for the self-supervised loss to accommodate the samples with the same label to achieve similar performance to the supervised loss, given that other factors are the same.

On the other hand, at the triplet level, both supervised and self-supervised settings benefit from using relatively smaller loss temperatures compared to the label level. This observation is also in line with the findings from [27]: contrastive loss with a small temperature tends to make the local structure of each sample more separated so that the similarity will be more discernable at more fine-grained level (i.e., triplet-level in our case). It is also the underlying reason that causes the effects of hard negative mining with small temperatures, as discussed at the start of this section; in other words, the triplet level task is more challenging than the label level counterpart, so it is supposed to benefit more from using a smaller temperature.

As for the performance on the hard defects, as AP@Ks consider the relative ranking of the positives in the rank list compared to Precision@K metrics, as explained in Section 5.1.1, higher AP@Ks require more discernable similarity between the samples, so here we can still see the effect of hard negative mining using a smaller temperature. However, it is still interesting to see that smaller temperature benefits the hard negative mining at the label level only by a small margin.

On the contrary, at the triplet level, the hard defects require larger loss temperatures to achieve optimal performance than non-hard defects, which goes against the above-mentioned findings. However, one possible explanation is that since the hard defects are defined based on the zero-shot performance at the label level, it can be inconsistent with

the triplet-level performance. More specifically, while the hard defects may be confused with other defects when searching within the product scope, at the triplet level, where we search within the same defect category instead, the embeddings of these defects could be more properly arranged in the embedding space – the embeddings of the more similar images are closer to that of the anchor image.

In conclusion, the results suggest we need different optimal loss temperature settings for different similarity levels. These findings align with the theoretical analysis of the contrastive loss function, even though the hard negative mining effects may not be prominent at the label level. Finally, the results also suggest that there can be inconsistencies between the "difficulty" of the defects at different similarity levels.

6.2. Vertical Experiments on the Effects of Augmentation Methods

As mentioned in Section 4.3, for SimCLR-based contrastive learning, data augmentation plays a crucial role in enforcing semantic invariances of the training images, and we have chosen five augmentation categories to achieve the best performance: `Flip`, `PixelDropout`, `OpticalDistortion`, `Affine`, and `GaussNoise`, as discussed in Section 4.3.1. In this section, we detail the evaluation of how these augmentation methods affect the model performance when used separately and in combination.

6.2.1. Experimental Setup

We evaluate the overall performance at label and triplet levels using each of the five augmentation methods separately and combine all of them. We use the temperature $\tau = 0.1$ for the supervised setting and $\tau = 0.5$ for the self-supervised setting, which are the best-performing temperatures as discussed in Section 6.1.2. The hyperparameters are the same as the default setting in Section 4.5.1.

6.2.2. Results and Analysis

Table 2 shows the overall performance of the model with different augmentation methods at label and triplet levels. Note that the `Dropout`, `Distortion`, and `Noise` in the tables refer to `PixelDropout`, `OpticalDistortion`, and `GaussNoise`, respectively.

Metrics \ Augmentations	Zero-shot	Flip	Dropout	Distortion	Affine	Noise	All
PrecisionAt5	0.7292	0.7510	0.7504	0.7658	0.7829	0.7646	**0.7900**
PrecisionAt10	0.5882	0.6159	0.6153	0.6192	0.6363	0.6215	**0.6389**
APAt5	0.8648	0.8848	0.9008	0.8933	0.9038	0.8988	**0.9078**
APAt10	0.8222	0.8403	0.8558	0.8514	0.8644	0.8557	**0.8690**

(a) supervised, label-level

Metrics \ Augmentations	Zero-shot	Flip	Dropout	Distortion	Affine	Noise	All
SimilarityPrecision	0.6837	0.7113	0.7099	0.7181	0.7366	0.7097	**0.7571**
ScoreAtTop5	11.2124	12.9027	12.6136	12.9646	14.1121	13.0914	**15.6637**
ScoreAtTop10	12.3333	14.1268	14.1917	14.5575	15.7906	14.3982	**17.5723**

(b) supervised, triplet-level

Metrics \ Augmentations	Zero-shot	Flip	Dropout	Distortion	Affine	Noise	All
PrecisionAt5	0.7292	0.7451	0.7516	0.7233	0.7546	0.7575	**0.7835**
PrecisionAt10	0.5882	0.6018	0.6047	0.5914	0.6127	0.6147	**0.6372**
APAt5	0.8648	0.8825	0.8768	0.8705	0.8815	0.8948	**0.9152**
APAt10	0.8222	0.8363	0.8342	0.8220	0.8371	0.8496	**0.8675**

(c) self-supervised, label-level

Metrics \ Augmentations	Zero-shot	Flip	Dropout	Distortion	Affine	Noise	All
SimilarityPrecision	0.6837	0.6893	0.7195	0.7029	0.7348	0.7138	**0.7575**
ScoreAtTop5	11.2124	11.6313	13.5162	12.3186	14.1947	13.1888	**15.7050**
ScoreAtTop10	12.3333	13.0826	15.0649	13.8732	16.2035	14.5752	**17.7847**

(d) self-supervised, triplet-level

Table 2: Effects of different augmentation methods. The bold numbers indicate the best performance in each metric.

Generally speaking, among individual augmentations, `Affine` and `GaussNoise` yield higher performance improvements across different settings and metrics, suggesting that most defects are invariant to affine transformations and Gaussian noises. `PixelDropout`, `OpticalDistortion`, and `Flip` also show good performance gains, but their effectiveness varies depending on the metric and the training setup. More specifically, the relative performance of different augmentation methods for overall performance:

- Supervised Learning: Affine > Noise > Distortion > Dropout > Flip
- Self-supervised Learning: Affine > Noise > Dropout > Flip > Distortion

At label-level:

- Supervised Learning: Affine > Noise > Dropout > Distortion > Flip
- Self-supervised Learning: Noise > Affine > Flip > Dropout > Distortion

At triplet-level:

- Supervised Learning: Affine > Distortion > Noise > Flip > Dropout
- Self-supervised Learning: Affine > Dropout > Noise > Distortion > Flip

Also, the additive effects are apparent when combining all the augmentation methods. The best performance is achieved when all the augmentation methods are combined and consistent across different settings and metrics. This observation suggests that these augmentation methods complement each other, and combining them can lead to better performance.

6.3. Horizontal Experiments on the Effects of Training Methods

As discussed in Section 4.1, we have two training paradigms: supervised and self-supervised contrastive learning, each with its own benefits. In this section, we compare these two paradigms to see whether and to what extent supervised and self-supervised signals benefit the model performance at label and triplet precision levels and how well they generalize over unseen product types. Here, we use the models trained with the best-performing recipes from the previous experiments.

6.3.1. Overall Performance

Table 3 shows the overall performance of the model with different training methods.

Metrics \ Methods	Zero-shot	Self-supervised (Gain)	Supervised (Gain)
PrecisionAt5	0.7292	0.7835 (7.45%)	**0.7900** (8.34%)
PrecisionAt10	0.5882	0.6372 (8.33%)	**0.6389** (8.63%)
APAt5	0.8648	**0.9152** (5.83%)	0.9078 (4.97%)
APAt10	0.8222	0.8675 (5.51%)	**0.8690** (5.69%)

(a) label-level

Metrics \ Methods	Zero-shot	Self-supervised (Gain)	Supervised (Gain)
SimilarityPrecision	0.6837	**0.7575** (10.78%)	0.7571 (10.73%)
ScoreAtTop5	11.2124	**15.7050** (40.07%)	15.6637 (39.72%)
ScoreAtTop10	12.3333	**17.7847** (44.17%)	17.5723 (42.52%)

(b) triplet-level

Table 3: Effects of different training methods (with performance gain). The bold numbers indicate the best performance in each metric.

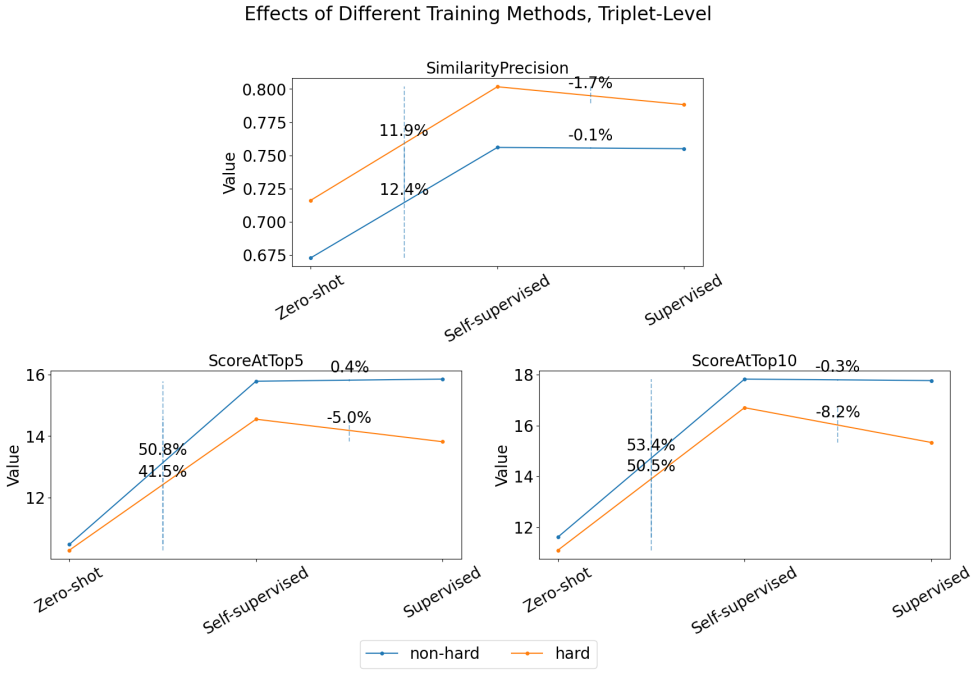
In conclusion, both self-supervised and supervised learning methods consistently outperform zero-shot across all metrics, achieving an approximately 8% performance gain in precision@k and a an approximately 5% performance gain in AP@k. At the triplet level, both methods achieve an approximately 10% performance gain in similarity precision and an approximately 40% performance gain in Score-at-top-K, showing the effectiveness of training. Moreover, the performance of the self-supervised learning method is comparable to that of the supervised learning method. This observation suggests that the self-supervision signal effectively achieves good performance in similarity search tasks, even without the explicit supervision signal, which indicates the potential of scaling up the training to larger datasets.

More specifically, supervised learning performs slightly better at the label level than self-supervised learning. In contrast, self-supervised learning performs slightly better at the triplet level than supervised learning. The performance gains are more significant at the triplet level than the label level, suggesting that the self-supervised signal is more effective in learning the fine-grained similarity between the samples than the supervised signal.

6.3.2. Performance on Hard and Non-hard Defects

Figure 12 shows the performance of the model on hard and non-hard defects defined in Section 3.4.2, both at both label and triplet levels.

(a) label-level



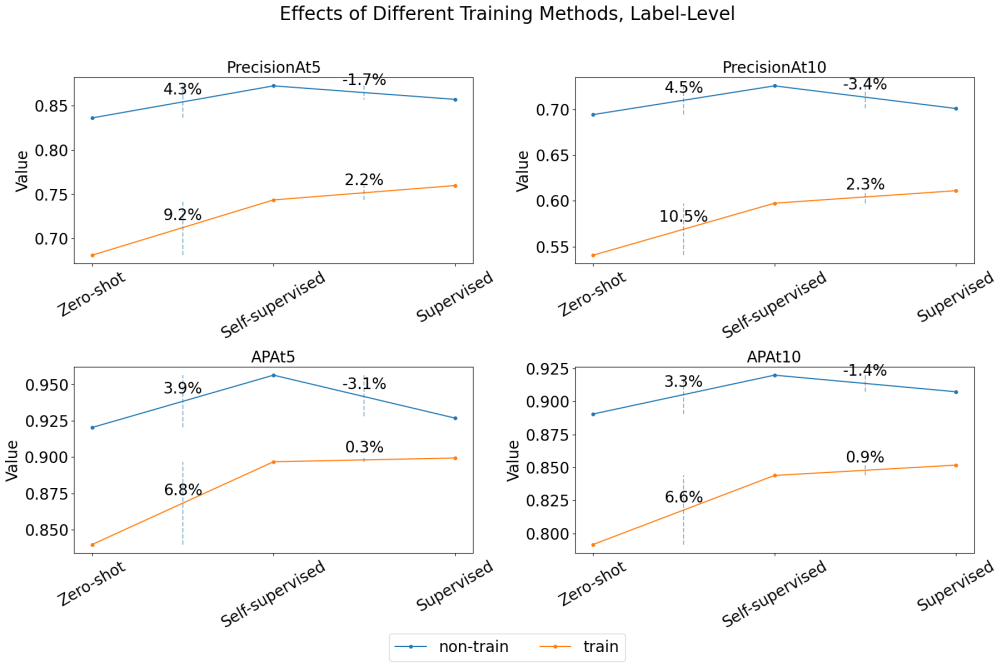
(b) triplet-level

Figure 12: Effects of different training methods on hard and non-hard defects. The vertical dashed lines with percentages indicate the performance gain between the two adjacent experiments.

The results show that the supervised signal performs better for label-wise metrics, specifically for hard defects, while the self-supervised signal performs better for triplet-wise metrics. This observation suggests that the label information can help the model better distinguish the hard defects from the non-hard ones at the defect level. However, the self-supervision signal can help the model better learn the fine-grained similarity between the samples to achieve better performance for the hard defects. Also, the performance gain over the zero-shot performance is more significant for hard defects than non-hard defects at the label level. In contrast, the performance gain is more significant for non-hard defects than hard ones at the triplet level. This observation further suggests that the supervision signals are more beneficial for the hard defects at the defect level. In contrast, the self-supervision signals are more beneficial at the triplet level.

6.3.3. Generalizability of the Training Methods

Figure 13 shows the performance of the model on the train and non-train products defined in Section 3.4.2, both at label and triplet levels.



(a) label-level

(b) triplet-level

Figure 13: Effects of different training methods on train and non-train products. The vertical dashed lines with percentages indicate the performance gain between the two adjacent experiments.

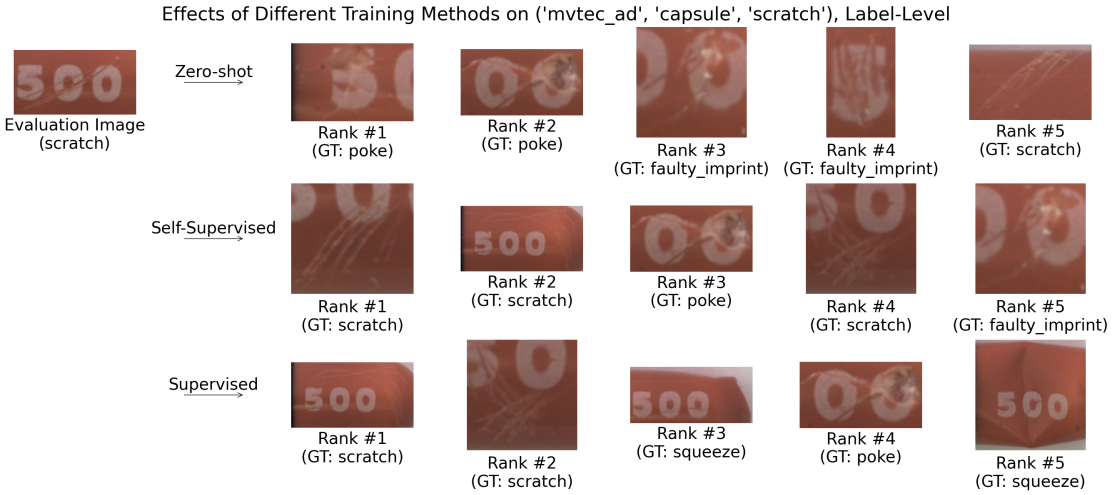
At the label level, only self-supervision signals show significant generalizability to non-trained products; supervision signals have little performance gain. Also, the performance gain over the zero-shot performance is more significant for trained products than for non-trained products, which suggests that both signals have the potential to overfit the training products. For trained products, the supervision signal performs better than self-supervision signals, suggesting that while performing better for trained products, the supervised signal may result in more severe overfitting to the training data.

At the triplet level, both supervised and self-supervised signals show generalizability to non-train products. Again, similar to the label level, the performance gain over the zero-shot performance is more significant for trained products than non-trained products, indicating overfitting potential for both signals. Additionally, the self-supervision signal performs slightly better for trained products. In contrast, the supervision signal performs slightly better for non-trained products, suggesting that the self-supervision signal has more potential to overfit the training data in the triplet-level task.

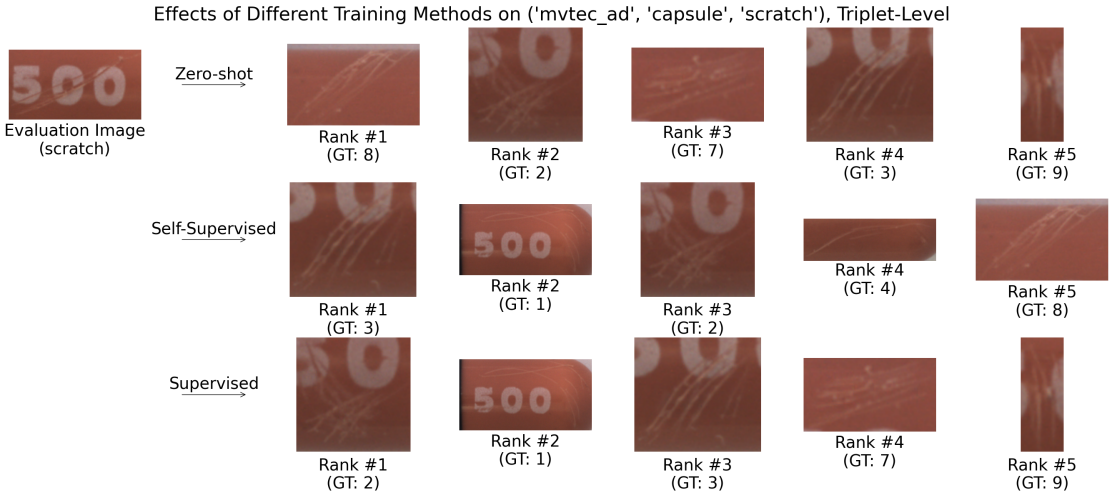
In conclusion, the self-supervision signal shows better generalizability to non-trained products at both levels than the supervision signal. However, both signals have the potential to overfit the training data. The supervision signal may result in more severe overfitting at the label-level task, and the self-supervision signal may result in more severe overfitting at the triplet level.

6.3.4. Examples of Positively and Negatively Affected Defects

Figure 14 and 15 are the two typical examples of most positively affected defects using different training methods, each from trained and non-trained products. The figures visualize the search results and relative ranks returned by models trained with different methods using a defect cropping from the evaluation subset as the query. The ground truths are attached to the respective returned defect cropping as well. Note that at the label level, the ground truth is the defect label, while at the triplet level, the ground truth is the relative rank of the defect cropping reconstructed from the annnotation triplets.

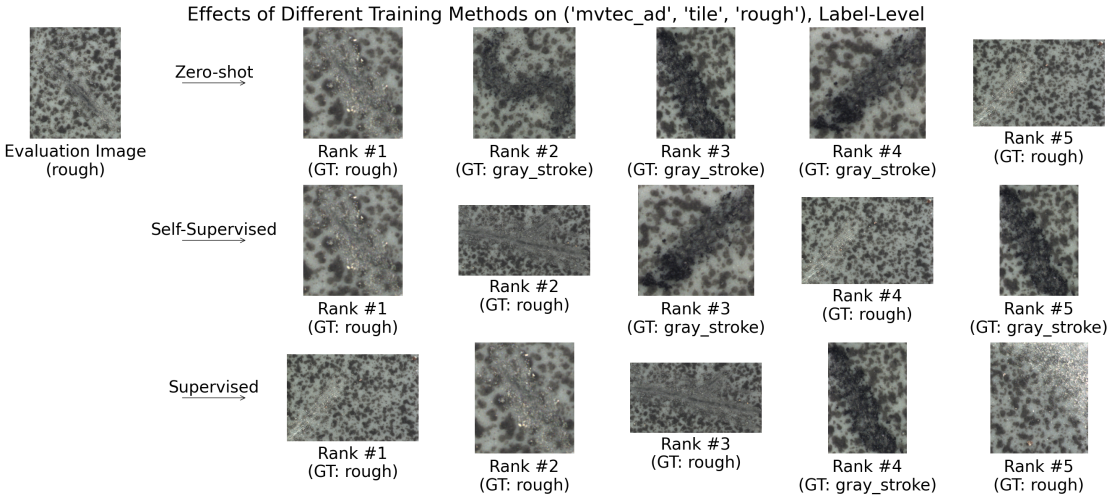


(a) label-level

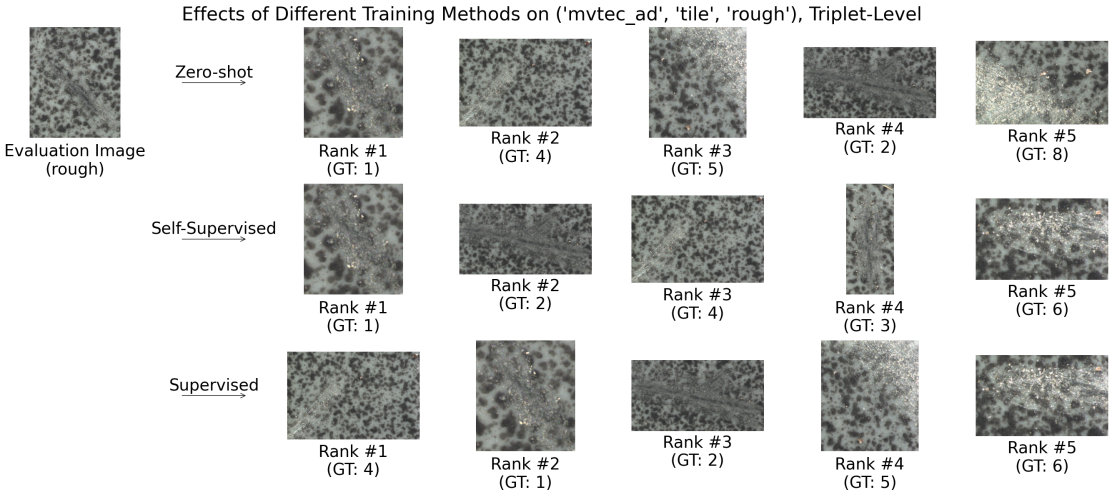


(b) triplet-level

Figure 14: An example of positively affected defects included in training (`mvtec_ad, capsule, scratch`)



(a) label-level

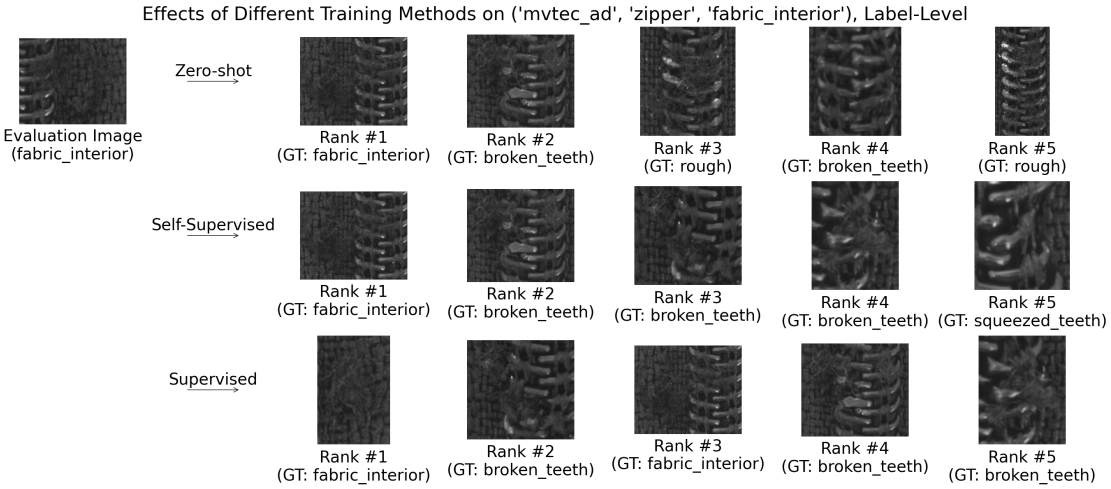


(b) triplet-level

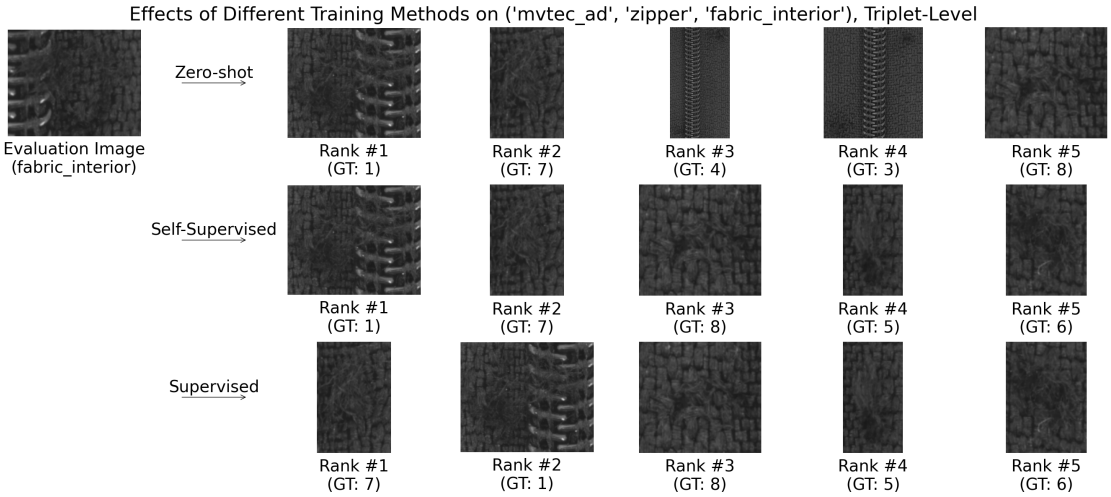
Figure 15: An example of positively affected defects not included in training (`mvtec_ad, tile, rough`)

From Figure 14 and 15, we can see that training improves the performance on the defect search task, both at the label and triplet levels. For `(mvtec_ad, capsule, scratch)`, the model trained by supervised learning is consistently better than that trained by self-supervised learning. In contrast, for `(mvtec_ad, tile, rough)`, the model trained by self-supervised learning is consistently better than that trained by supervised learning at the triplet level.

Similarly, Figure 16 and 17 are the two typical examples of the most negatively affected defects using different training methods, each from trained and non-trained products.

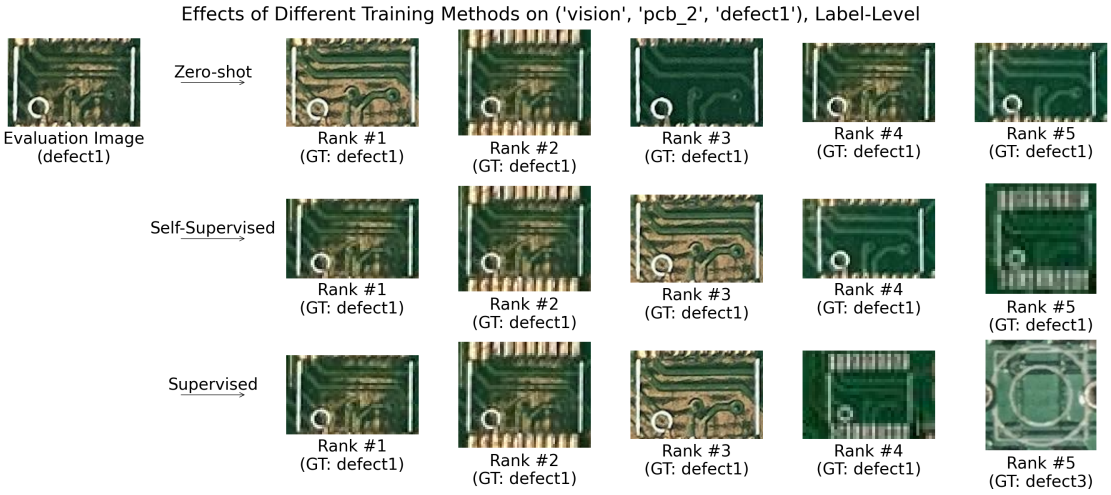


(a) label-level

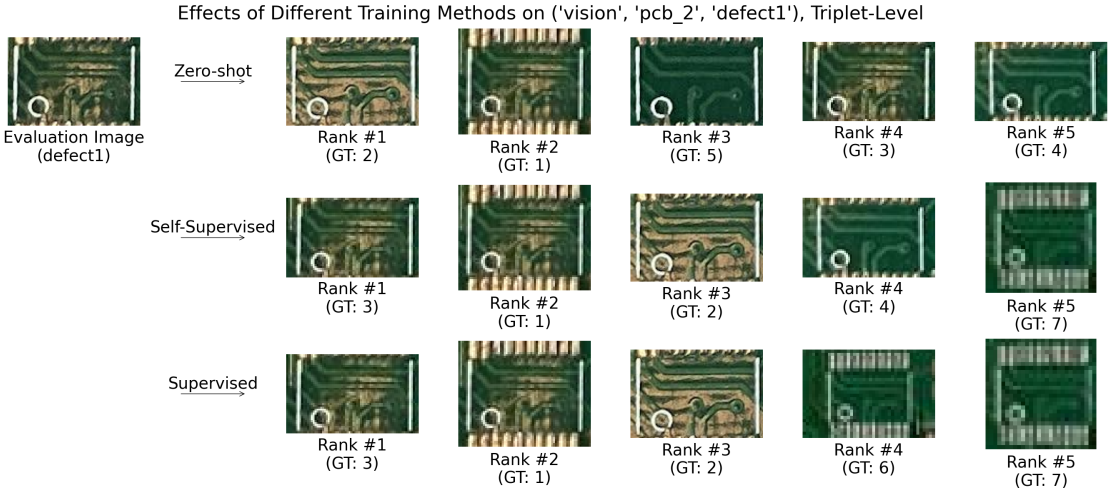


(b) triplet-level

Figure 16: An example of negatively affected defects included in training (`mvtec_ad, zipper, fabric_interior`)



(a) label-level



(b) triplet-level

Figure 17: An example of negatively affected defects not included in training `(vision, pcb_2, defect1)`

From Figure 16 and 17, we can see that the models may result in worse performance on the defect search task, both at the label and triplet levels. For `(mvtec_ad, zipper, fabric_interior)`, there are almost no improvements in the performance of the model trained by supervised learning, and the model trained by self-supervised learning is consistently worse than that zero-shot performance. It may partially be because the defect is very hard to distinguish from the other defects, and the gradient signal generated by the training dataset is not strong enough to improve the performance of this specific defect. For `(vision, pcb_2, defect1)`, there is a slight degradation in the model's performance trained by supervised learning at the label level. At the triplet level, the model trained by supervised learning is consistently worse than that trained by self-supervised learning. However, at the triplet level, the cropping ranked at three is apparently more similar to the query cropping than the cropping ranked at 2, which suggests that there can be faults or inconsistencies within the human annotations.

6.4. Conclusions

In this section, we have outlined a series of experiments to evaluate and compare the performance of different training methods. We have proved the effectiveness of the training in improving the model's performance for fine-grained similarity search tasks, achieving a 5-8% performance gain in label-level precision and a $> 10\%$ performance gain in triplet-level precision that aligns with human perception. More specifically, we have found the optimal loss temperature to be contingent upon the training paradigm and the level of precision, aligning with the theoretical analysis of the contrastive loss function. We have also observed that affine and Gaussian noise augmentation strategies create the invariances that most defects agree upon, and the combination of different augmentation methods results in a synergistic improvement.

Furthermore, our research has unveiled the remarkable potential of self-supervised learning, which has shown more scalability in fine-grained similarity search than supervised learning, reducing the need for supervision signals during training. Notably, the self-supervision signal has proven more effective in learning the fine-grained similarity between the samples than the supervised signal. These findings reassure us about the self-supervision signal's effectiveness and display its exciting potential for larger-scale and unlabeled domain datasets.

Finally, we have demonstrated the generalizability of the training methods, with approximately 5-7% performance gain over unseen products and defect types in training. Additionally, the self-supervision signal showed better generalizability to non-trained products at both levels than the supervision signal, albeit both tend to overfit. This observation further indicates the potential of using the self-supervised learning paradigm of fine-grained similarity search tasks.

7. Prototype

This section describes a prototypical search application that implements the basic functionalities of the one used in an actual industrial setting, with a defect database and the embedding models we trained with the methods described in Section 4.

7.1. Functionalities

As mentioned in Section 4, to perform searches within the defect database, we need to

- build an embedding-based search index with the model on the defect database,
- encode every input sample with the model into an embedding and
- search for the most similar samples in the defect database with the embedding

Also, as briefly discussed in Section 3.3, only the defect areas are interesting for the search application. Therefore, the search application should at least have the following functionalities:

- first, the user can upload an image and crop out any interested regions via the UI,
- then, the cropped region is sent to an embedding model specified by the user to generate region embedding
- the embeddings will be used to query the database with the distance measures specified when building the search index, and finally,
- the entries with the most similar region embeddings will be retrieved.

It is important to note that, as a prerequisite, we must prepare the defect database and build a compound search index to map a defect image with all the defect croppings from it. For simplicity, in the prototype, we will directly search for the defect croppings in the database subset of FGDS we constructed in Section 3.4 and build the index upon this subset.

7.2. Implementation

Since the database subset of FGDS is managed by FiftyOne [18], we leverage its app and plugin system to build the prototypical search application. FiftyOne App is a web-based application that provides a visual interface for exploring datasets and models. At the same time, the FiftyOne Plugin system, which lives in the app, allows users to extend the functionality of the app with custom components. The plugins can be built with Python, Javascript/Typescript, and various external packages and frameworks in these languages.

FiftyOne allows for the storage of embeddings and the building of search indices with various vector database backends. It can store the embeddings and indices in the dataset and provides convenient APIs that allow the user to perform similarity-based searches with such info. Our prototype uses Milvus [26] as the vector database backend.

On the other hand, we get the inspiration for the plugin from the `reverse-image-search` plugin [28] by FiftyOne's official team member. The plugin provides a simple interface for uploading an image and searching for similar images in the dataset. For the image cropping functionality, we use the `react-easy-crop` package [29], which provides a simple and easy-to-use image cropping component.

7.3. User Interface

Figure 18, 19, 20, and 21 show the user interface of the prototype and how users would interact with it.

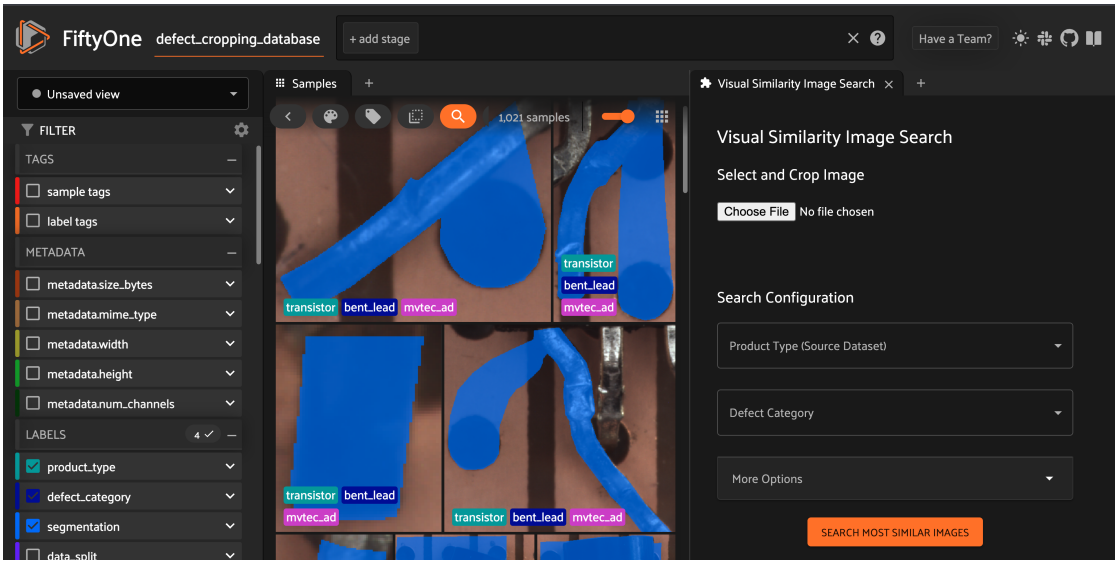
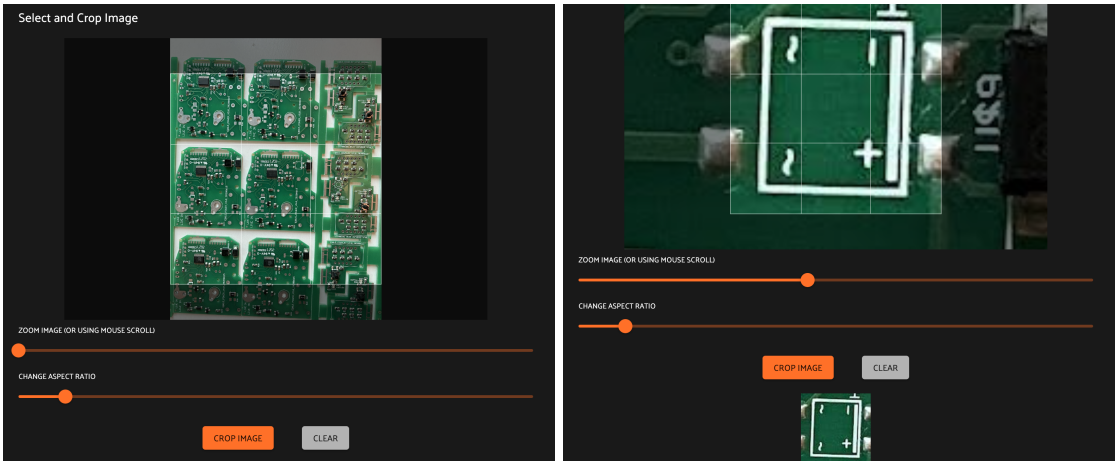


Figure 18: The initial interface

Figure 18 shows the initial interface of the prototypical search application. The panel has two main sections: the `Select and Crop Image` section and the `Search Configuration` section. The `Select and Crop Image` section allows users to upload an image and crop out the defect areas; the `Search Configuration` section allows users to adjust the settings for the search operation.

To perform the search operation, the users can start uploading the product image by clicking the `Choose File` button in the `Select and Crop Image` section. Then, the cropping section will be shown as displayed in Figure 19.

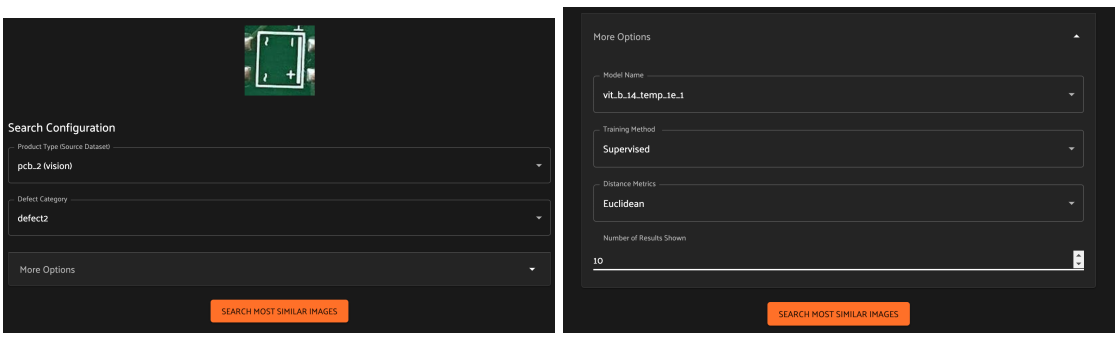


(a) Before cropping (b) After cropping

Figure 19: The cropping section

The cropping section is initialized with a default cropping area, as shown in Figure 19a. Then, the users can adjust it using either the mouse/touchpad scroll or the slide bar named `ZOOM IMAGE (OR USING MOUSE SCROLL)` below the cropping area to zoom the image. The other slide bar named `CHANGE ASPECT RATIO` can be used to adjust the aspect ratio. Once the users are satisfied with the cropping area, they can click the `CROP IMAGE` button to create the defect cropping from the image, and a thumbnail will be displayed right below the button, as shown in Figure 19b. Throughout the process, the users can always discard the existing image and start over with another one by clicking the `CLEAR` button anytime.

After successfully obtaining the defect cropping, the user can adjust the configurations in the `Search Configuration` section as shown in Figure 20.



(a) Basic config (b) Advanced config

Figure 20: The search config section

The basic configuration shown in Figure 20a allows the user to specify the search scope,

including the product type and the defect category, over the database. The default is set to search the whole product range and all defect categories. Users can narrow the search scope by selecting a product or defect category. Here, we use the same product and defect names as the ground truth for these two options.

Moreover, the advanced configuration is folded within the accordion region `More Options` and can be expanded by clicking the accordion header. As shown in Figure 20b, the advanced configuration allows the user to specify the model for the embeddings. More specifically, it allows the user to choose the model names that specify its essential training config, the training methods (i.e., Supervised or Self-Supervised), and the distance metrics for the search index that can uniquely identify an embedding model. The user can also adjust the number of results shown in the search result. These configurations are offered with default values, and the user can adjust them as needed. Here, we leave them as default.

Finally, after clicking the `SEARCH MOST SIMILAR IMAGES` button, the search result will be displayed in the `Samples` panel as shown in Figure 21.

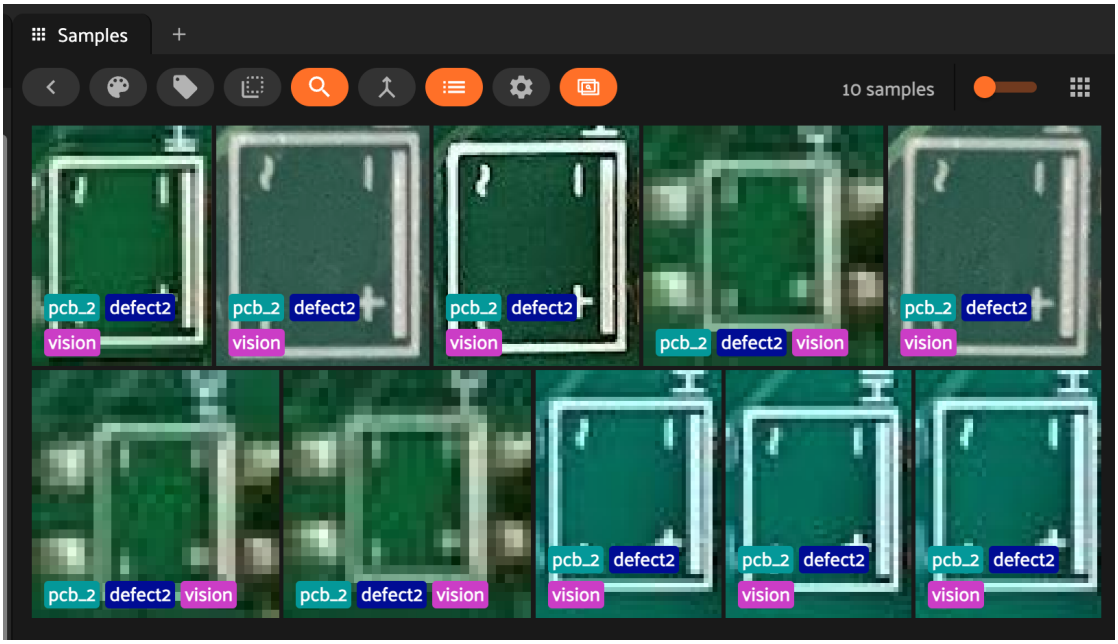


Figure 21: the search result

Here, the prototype returns the top 10 most similar defect croppings from the database subset. The users can also always click the `CLEAR` button in 19b to clear the search result and start over with another search operation.

8. Conclusion

In conclusion, this thesis has presented an effective solution for distinguishing fine-grained visual similarity among extensive industrial defect images through several critical achievements and methodologies. First, we curated the Fine-Grained Defect Similarity (FGDS) dataset, which includes defect images from open-source domain datasets and is annotated with human preference data using triplets, serving as a training resource for our embedding model and for evaluating fine-grained similarity performance. Then, we fine-tuned a ViT embedding model using a SimCLR-based contrastive learning framework in both supervised and self-supervised modes on the FGDS training subset. The embedding model was then used to generate search indices for similarity-based retrieval. Furthermore, we established a comprehensive evaluation protocol with detailed metrics to assess similarity-based retrieval performance at both defect category and fine-grained levels using triplet-based evaluation metrics to leverage human-annotated triplet data.

The experimental results demonstrate significant performance gains from zero-shot learning using the annotated data, demonstrating the proposed training methods' effectiveness in aligning model performance with human perception. These gains, contingent on the optimal loss temperature, validate the theoretical analysis of the contrastive loss function. The results further demonstrate the critical role of augmentation methods in creating robust invariances, which suggests that combining different augmentation strategies can lead to synergistic performance enhancement. Most notably, these findings emphasize the remarkable potential of self-supervised learning, which reduces the dependency on supervision signals and outperforms supervised learning in terms of capturing fine-grained similarities. These implications advocate preferring self-supervised paradigms in fine-grained similarity search, which fits into the context of the manufacturing quality control domain where labeled data are scarce and expensive to obtain.

Overall, these contributions pave the way for future research and development, offering a solid foundation for developing advanced image search applications for the manufacturing quality control industry. This research not only provides a solution for distinguishing fine-grained visual similarity among industrial defect images but also opens up new possibilities for the industry, promising more efficient and accurate quality control processes in the future.

9. Future Work

In the future, we have the potential to enhance the current search application in several ways:

Firstly, we can enhance the pre-training process by extracting defect croppings from a broader range of open-source domain datasets beyond MVTec AD and VISION. Pre-training on a more diverse dataset can result in a more robust embedding backbone, providing a superior starting point for fine-tuning the model on industry-specific datasets from customers.

Another area for potential enhancement is the introduction of different region contexts. Currently, we only use the tightest bounding box around the defect to create the croppings. However, some defects require more contexts to be easily identified. Conversely, some non-defect regions could be misleading, so the industry professional may want to remove them. Therefore, experimenting with different region contexts and even introducing a segmenter like a Segment Anything Module (SAM) could significantly improve the application's adaptability and its potential to identify defects.

Moreover, the current training configuration is limited by the available computing resources. For future experiments, we could use more computing to try larger batch sizes and backbone models to see if they can further improve the retrieval performance. It also gives room to refine the fine-tuning schedule. For example, we could try different learning rate schedules for different attention layers or even use a learning rate finder to find the optimal learning rate for the fine-tuning process. Alternatively, we could use a different contrastive learning framework instead of the SimCLR [5] one to see which works better for our task.

Furthermore, to evaluate the retrieval performance comprehensively, we could consider introducing user-centric metrics, such as nDCG (normalized Discounted Cumulative Gain), in addition to precision-based metrics. This approach would provide a more holistic view of the retrieval performance in an interactive setting, emphasizing its usability and its ability to meet industry needs.

Lastly, as explained in Section 7.1, in the actual search application, we need to design the database containing the actual product images multi-indexed with all of the associated defect croppings, along with a user interface that could adequately display the cropping-based search results on top of the product image.

References

[1] X. Wei, Y. Song, O. Aodha, J. Wu, Y. Peng, J. Tang, J. Yang, and S. Belongie, "Fine-grained image analysis with deep learning: A survey," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 44, no. 12, pp. 8927–8948, dec 2022.

[2] L. Weng, "Contrastive representation learning," *lilianweng.github.io*, May 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-05-31-contrastive/

[3] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 18 661–18 673. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/d89a66c7c80a29b1bdbab0f2a1a94af8-Paper.pdf

[4] R. Balestriero, M. Ibrahim, V. Sobal, A. S. Morcos, S. Shekhar, T. Goldstein, F. Bordes, A. Bardes, G. Mialon, Y. Tian, A. Schwarzschild, A. G. Wilson, J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum, "A cookbook of self-supervised learning," *ArXiv*, vol. abs/2304.12210, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258298825

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html

[6] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent - a new approach to self-supervised learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 271–21 284. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf

[7] S. Pan, Y. Qin, T. Li, X. Li, and L. Hou, "Momentum contrastive pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2022, pp. 2647–2656.

[8] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9630–9640.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[10] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9912–9924. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/70feb62b69f16e0238f741fab228fec2-Paper.pdf

[11] A. Bardes, J. Ponce, and Y. LeCun, "VICReg: Variance-invariance-covariance regularization for self-supervised learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=xm6YD62D1Ub

[12] J. Zhou, C. Wei, H. Wang, W. Shen, C. Xie, A. Yuille, and T. Kong, "Image BERT pre-training with online tokenizer," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=ydopy-e6Dg

[13] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2024. [Online]. Available: https://arxiv.org/abs/2304.07193

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[15] J. Wang, Y. song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 04 2014.

[16] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "The mvtec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1038–1059, 2021. [Online]. Available: https://doi.org/10.1007/s11263-020-01400-4

[17] H. Bai, S. Mou, T. Likhomanenko, R. G. Cinbis, O. Tuzel, P. Huang, J. Shan, J. Shi, and M. Cao, "Vision datasets: A benchmark for vision-based industrial inspection," *arXiv preprint arXiv:2306.07890*, 2023.

[18] Voxel51, "Fiftyone: The open-source tool for building high-quality datasets and computer vision models," https://github.com/voxel51/fiftyone, 2024.

[19] Wikipedia contributors, "Evaluation measures (information retrieval) — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)?oldformat=true, 2024, accessed: 2024-07-19.

[20] L. Weng, "Self-supervised representation learning," *lilianweng.github.io*, 2019. [Online]. Available: https://lilianweng.github.io/posts/2019-11-10-self-supervised/

[21] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *ArXiv*, vol. abs/1807.03748, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49670925

[22] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/125

[23] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1708.03888

[24] H. Touvron, M. Cord, A. El-Nouby, J. Verbeek, and H. Jégou, "Three things everyone should know about vision transformers," 2022. [Online]. Available: https://arxiv.org/abs/2203.09795

[25] B. Lefaudeux, F. Massa, D. Liskovich, W. Xiong, V. Caggiano, S. Naren, M. Xu, J. Hu, M. Tintore, S. Zhang, P. Labatut, D. Haziza, L. Wehrstedt, J. Reizenstein, and G. Sizov, "xformers: A modular and hackable transformer modelling library," https://github.com/facebookresearch/xformers, 2022.

[26] M. Contributors, "Milvus: A cloud-native vector database, storage for next generation ai applications," https://github.com/milvus-io/milvus, 2024.

[27] F. Wang and H. Liu, "Understanding the behaviour of contrastive loss," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 2495–2504.

[28] J. Marks and E. Hofesmann, "Reverse image search plugin," https://github.com/jacobmarks/reverse-image-search-plugin, 2024.

[29] V. Hintz, "react-easy-crop: A react component to crop images/videos with easy interactions," https://github.com/ValentinH/react-easy-crop, 2024.

A. Hard Defects and Non-train Products

A.1. Hard Defects

Source Dataset	Product Type	Defect Category
mvtec_ad	cable	bent_wire
mvtec_ad	cable	cut_outer_insulation
mvtec_ad	cable	poke_insulation
mvtec_ad	capsule	crack
mvtec_ad	capsule	poke
mvtec_ad	grid	metal_contamination
mvtec_ad	grid	thread
mvtec_ad	pill	faulty_imprint
mvtec_ad	screw	thread_side
mvtec_ad	wood	color
mvtec_ad	zipper	broken_teeth
vision	console	dirty
vision	hemisphere	defect_a
vision	hemisphere	defect_c
vision	lens	flash_particle
vision	lens	hole
vision	lens	surface_damage
vision	lens	tear
vision	pcb_1	short
vision	pcb_1	spur

Table 4: Hard Defects

A.2. Non-train Products

Source Dataset	Product Type
mvtec_ad	carpet
mvtec_ad	leather
mvtec_ad	tile
mvtec_ad	transistor
vision	cable
vision	casting
vision	pcb_2
vision	ring
vision	wood

Table 5: Non-train Products

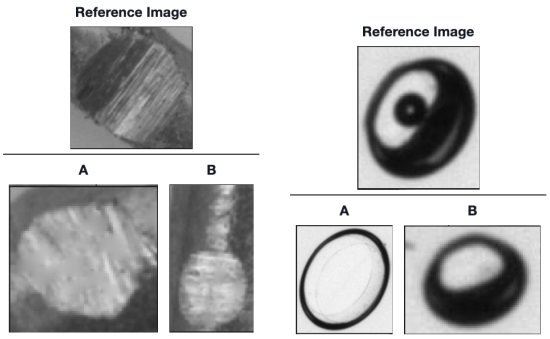
B. Annotation Instructions

B.1. Short Instructions

Based on pure visual appearance, tell us which of the two images at the bottom is more similar to the image at the top.

Note that we're considering the visual factor only, so choose the first thing that comes into your mind the moment you see these images. This process shouldn't take you more than a couple of seconds.

Two possible examples are:



In the first case, image A is more similar to the reference image than B.
In the second case, image B is more similar to the reference image than A.

B.2. Full Instructions

```
The images presented to you are some manufacturing defect samples. We're
interested in how visually similar they are to humans comparing to
one another.

As we're considering the visual appearances only, your choices are
supposed to be subjective, so you don't have to go through a deliberate
thought process or seek objective verifications for your choices.
```

C. Data Augmentation Configuration

The data augmentation parameters used in the training are as follows. Note that the parameters that are not listed are set to default values in the library.

- `Flip`: We use horizontal and vertical flips, each with a probability of 0.5.
- `PixelDropout`: We use random pixel dropout with a probability of 0.2, along with 10% of the pixels dropped and filled with 0.
- `OpticalDistortion`: We use optical distortion with a probability of 0.5, a distortion and shift limit range of (-0.1, 0.1), and a constant border value of 0.
- `Affine`: We use affine transformations with a probability of 0.5, along with translation fraction, rotation angle, and scaling factor limits of (-0.1, 0.1), (-15, 15), and (0.8, 1.2), respectively.
- `GaussNoise`: We use Gaussian noise with a probability of 0.5 and variation range of (10, 50).

D. Evaluation Database Schema

This section describes the table schemas to store evaluation results.

D.1. Results

- `ExperimentRunName (Primary Key)`: The name of the MLflow experiment and run.
- `EvalFilePath (Primary Key)`: The path to the evaluation file of the sample.
- `ProductType`: The product type of the sample.
- `SourceDataset`: The source dataset of the sample.

- `DefectCategory`: The defect category of the sample.
- `PrecisionAt5`: The precision@5 of searching for samples with the same defect category in the database subset with this sample.
- `PrecisionAt10`: The precision@10 of searching for samples with the same defect category in the database subset with this sample.
- `APAt5`: The AP@5 of searching for samples with the same defect category in the database subset with this sample.
- `APAt10`: The AP@10 of searching for samples with the same defect category in the database subset with this sample.
- `SimilarityPrecision`: The similarity precision of ranking samples within the same defect category in the database subset with this sample as the anchor of the triplet.
- `ScoreAtTop5`: The score-at-top-5 of ranking samples within the same defect category in the database subset with this sample as the anchor of the triplet.
- `ScoreAtTop10`: The score-at-top-10 of ranking samples within the same defect category in the database subset with this sample as the anchor of the triplet.

D.2. RankListLabel

- `ExperimentRunName (Primary Key)`: The name of the MLflow experiment and run.
- `EvalFilePath (Primary Key)`: The path to the evaluation file of the sample.
- `Rank (Primary Key)`: The rank of the database sample in the search results.
- `DatabaseFilePath`: The path to the database sample file.
- `DefectCategory`: The defect category of the database sample.

D.3. RankListTriplet

- `ExperimentRunName (Primary Key)`: The name of the MLflow experiment and run.
- `EvalFilePath (Primary Key)`: The path to the evaluation file of the sample.
- `Rank (Primary Key)`: The rank of the database sample in the search results.
- `DatabaseFilePath`: The path to the database sample file.
- `RankGT`: The ground truth rank of the database sample, if available.

D.4. ER Diagram

Figure 23 shows the ER diagram for the database to store evaluation results.

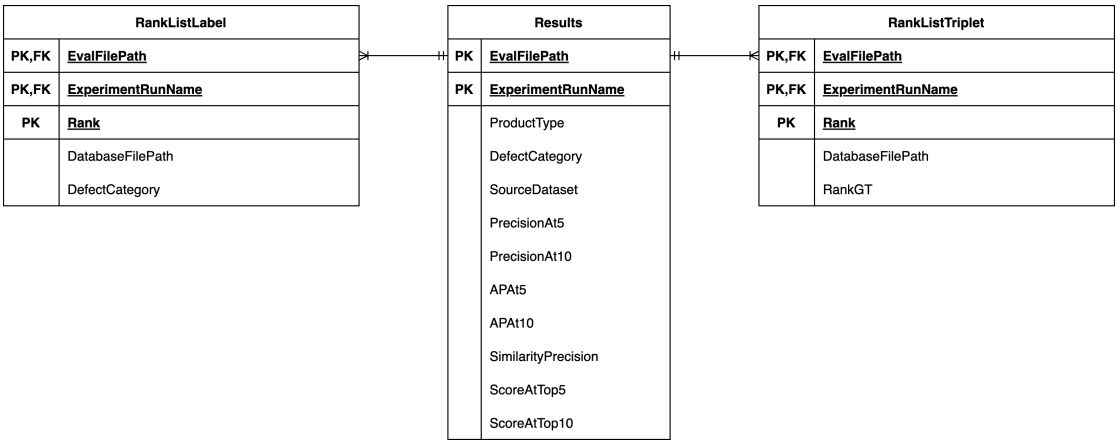


Figure 23: the ER diagram for the database to store evaluation results