Anomaly detection on gears with BIT Machine Learning for Computer Vision Project Work

Lorenzo Scaioli - lorenzo.scaioli@studio.unibo.it
Matricola 1093722

June 2025

1 Introduction

In recent years, anomaly detection has become one of the key areas of application for Machine Learning (ML). With the rapid advancement of new technologies, ML is making significant progress across various domains, often challenging traditional methods that have long been considered state-of-the-art in their respective fields.

In this project, we focus on anomaly detection in images of mechanical gears. One of the main challenges in applying ML to anomaly detection is the scarcity of anomalous samples, as such cases are relatively rare by nature. To address this issue, we revisit a more classical approach commonly used in industrial settings. Specifically, rather than relying solely on labeled examples of anomalies, we leverage the concept of comparing each gear image to an "ideal" reference image. This ideal gear image is generated by computing the median across multiple images of gear teeth, capturing a general representation of a non-defective gear. We then compare this reference image with actual gear images using a Deep Learning (DL) model designed to highlight differences between two inputs.
For this task, we employ a Bitemporal Image Transformer (BIT), introduced in the paper *Remote Sensing Image Change Detection with Transformers* [3]. The BIT model takes two images as input and produces a change mask that identifies regions of difference, making it well-suited for our anomaly detection pipeline.
The project is available on GitHub.

2 Background

2.1 Dataset

The dataset we selected is the *Gear Computer Vision Project*, available on Roboflow [4]. It contains high-quality `.jpeg` images (resolution: 2448×2048) of both healthy gears and gears with various types of anomalies. All images are annotated with bounding boxes that specify the type and location of anomalies, provided in the COCO JSON format.

As shown in Figure 1, there are a few important considerations regarding the dataset. Firstly, the white background is ideal for image-based anomaly detection; however, the occasional presence of black strips must be addressed during preprocessing. Secondly, the most common anomalies involve damaged gear teeth or structural deformities resembling cracks or creases. However, the dataset also contains visual elements like oil stains, which are not actual anomalies but may be misclassified by the model, increasing the risk of false positives. Lastly, it is important to emphasize that the selected dataset was designed for bounding box prediction, whereas our goal is to perform a segmentation task. This intrinsic difference must be taken into account during the evaluation phase.

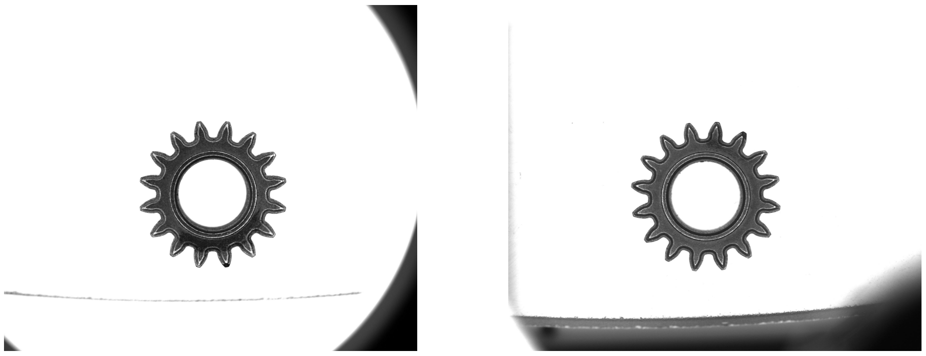


Figure 1: Examples of gears' images in the dataset

2.2 Median image method

As discussed in Section 1, a well-known approach for detecting anomalies in gear teeth involves comparing an "ideal" image to the actual image of the gear. To obtain this "ideal" image the idea is to align all the images of every single tooth and then perform the median pixel by pixel over the stack of images. If the alignment is correct, the resulting image represents an "ideal" tooth, where each pixel corresponds to the most frequent value across the aligned teeth. This median image will naturally suppress local anomalies, since it is highly unlikely that all the teeth exhibit the same defect. This method is particularly effective for gears due to their symmetrical structure and uniformity across teeth.

In our project, we generate an ideal reference image, hereafter referred to as the *median image*, for each gear by rotating the gear image around its symmetry axis. Specifically, we first determine the gear's center, then rotate the image by an angle of $360/nr\_teeth$ to generate one view per tooth. After aligning these rotated views, we compute the pixel-wise median across them. Figure 4 shows a comparison between the input image and the corresponding median image.

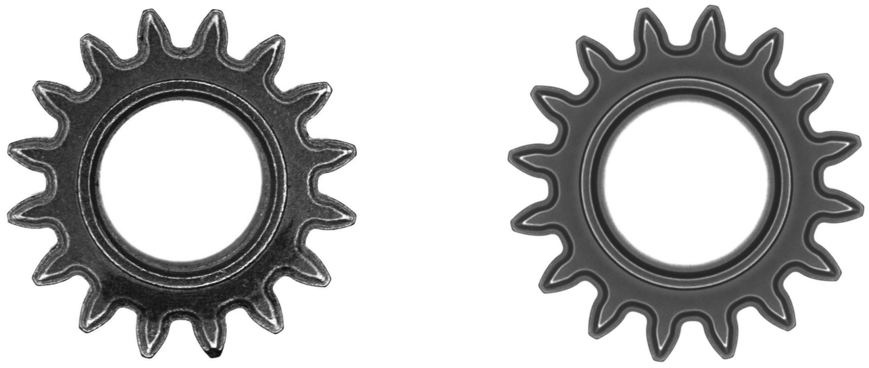


Figure 2: Comparison between input image and median "ideal" image

2.3 BIT architecture

The architecture of the BIT-based model [3] is specifically designed for change detection (CD) in high-resolution satellite images. Its main objective is to efficiently and effectively model the context within the spatiotemporal domain to identify meaningful changes while filtering out irrelevant variations.

As shown in Figure 3, the overall architecture of the BIT-based model consists of three main components:

1. **CNN Backbone:** Used to extract high-level semantic features from the pairs of input images.
2. **BIT Module:** The core of the model, responsible for refining bitemporal features by modeling global contextual relationships.
3. **Prediction Head:** A shallow CNN that uses the refined features to produce pixel-wise change predictions.

Now we describe more in depth the BIT module that characterizes the whole network. In particular, the component is composed of three key parts:

- **Siamese Semantic Tokenizer:** The core idea behind the Siamese Semantic Tokenizer is that high-level change concepts can be effectively represented using a small set of "visual words" or semantic tokens. It operates on the feature maps (X_1, X_2) extracted by the CNN backbone for each of the two temporal images. A shared tokenizer is then applied to extract compact token sets (T_1, T_2) from these feature maps. This is achieved by learning spatial attention maps that group pixels into meaningful semantic regions.
- **Transformer Encoder (TE):** The TE is designed to model the contex-

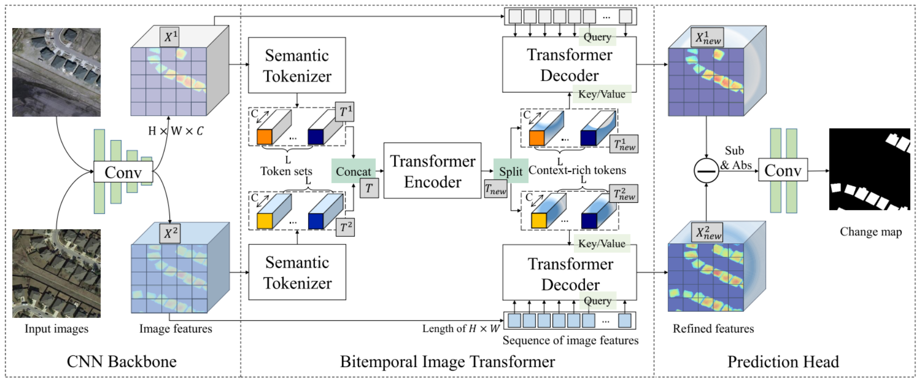


Figure 3: BIT architecture

tual relationships between the extracted semantic tokens. It receives as input a concatenated set of tokens T, formed by joining the two token sets (T_1, T_2) corresponding to the input image pair. The encoder consists of different layers, each one composed of a Multihead Self-Attention (MSA) block and a Multi-Layer Perceptron (MLP). The MSA mechanism enables the model to attend to multiple representation subspaces across different positions, capturing complex dependencies between the tokens. To preserve spatial and temporal order information a learnable positional embedding (PE) is added to the token sequence before being processed by the encoder. The TE outputs a new set of context-enriched tokens T_{new}, which are then split back into their respective updated temporal sets $(T_{1\text{new}}, T_{2\text{new}})$.

- **Siamese Transformer Decoder (TD):** The TD takes the tokens $(T_{i\text{new}})$ and projects them from the token space back into the pixel space to refine the original feature maps (X_i) at the pixel level. It is composed of 8 layers, each containing Multihead Cross-Attention (MA) and MLP blocks. In each MA block, the queries are derived from the original image features (X_i), while the keys and values are taken from the context-enriched tokens $(T_{i\text{new}})$. This structure enables each pixel to be enhanced through a combination of compact semantic tokens, improving the overall quality of the feature representation.

Actually, the model just described is called $base\_transformer\_pos\_s4\_dd8\_dedim8$, the BIT model with backbone ResNet18_S4, PE, TD with depth 8 and embedding dimension $d = 8$. In the paper, it is compared with other two simpler models: $base\_transformer\_pos\_s4\_dd8$ and $base\_resnet18$. The first one is another BIT model with a shallower backbone, ResNet18_S4. While the latter is a base model composed of a ResNet18_S5 backbone and a prediction head.

3 Implementation

In this section we describe the experimental set up for our experiment. We highlight the preprocessing on the dataset to fit the data in the BIT model and the changes we have made to the original code of the paper [1]. We specify that the whole project was developed in a Docker container to achieve a reproducible environment for our experiment.

3.1 Preprocessing

The preprocessing process is well described in the notebook in the GitHub repository of this project [2], but we will briefly report the main steps here:

- Create the groundtruth binary label from the annotation in COCO format.
- Select a crop of 1024x1024 with the gear as center using the `cv2.findContours` function and computing the centroid.
- Create the median image for each gear image.
- Divide each image into 4 images each one with dimension 512x512 to be processed by the BIT model.
- Lastly, we perform data augmentation rotating the images of 90, 180 and 270 degree.

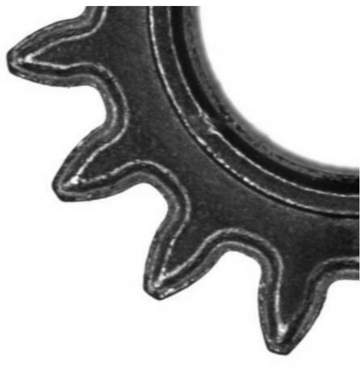
At the end of the preprocessing we have obtained train, validation and test datasets each one containing triplets of images: the original image, the median image and the binary mask label (Figure 4).

3.2 Changes to the original code

Weighted loss The most significant change that we made is to add weights value to the loss function. In fact, we believe that changing the loss function in this use-case can well prevent two possible problems. First, to weights the classes according to their frequency in the binary mask label, indeed this is a classical case with imbalanced classes. Secondly, we can adjust the weights to give more importance to the anomaly class, this could be useful to increase recall by reducing false negative predictions. The loss function of our code, a weighted binary cross entropy, is defined in equation 1, where $\hat{y}_i$ is the predicted value and w_0, w_1 are the added weights. To achieve this loss function we modify the `_backward_G(self)` in `trainer.py` and some related functions.

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[w_1 \cdot y_i \cdot \log(\hat{y}_i) + w_0 \cdot (1-y_i) \cdot \log(1-\hat{y}_i)\right] \tag{1}$$

Evaluation metrics Given the mismatch between the dataset's original purpose (bounding box detection) and our segmentation objective, standard object


(a) Input image

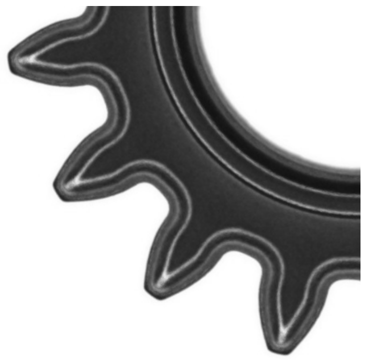
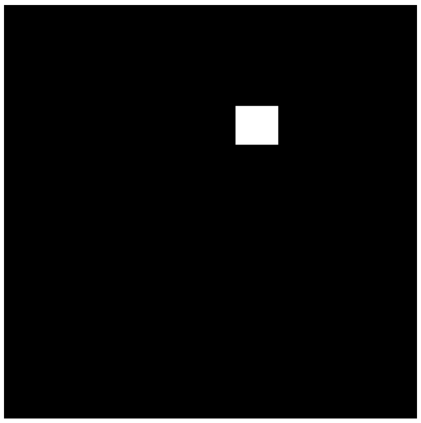

(b) Median "ideal" image

Figure 4: Comparison between the input gear image and the computed median "ideal" image.


(a) Binary mask label

detection metrics like Intersection over Union (IoU) or mean Average Precision (mAP) at the box level may not fully reflect segmentation quality. A more suitable and representative evaluation metric would be precision, recall and F1 score computed on the predicted segmentation masks. These metrics evaluate the overlap between predicted and ground truth pixel regions, making them better suited for assessing the spatial accuracy of segmented anomalies.
Our custom `eval_instances.py` code evaluates instance-level segmentation performance by comparing predicted masks with ground truth masks using a IoU-based matching strategy. It processes binary masks by first performing connected component analysis to extract individual object instances, filtering out small components below a specified area threshold to eliminate noise. Each predicted instance is then matched with an unmatched ground truth instance if

their IoU exceeds a given threshold, resulting in counts of True Positives (TP), False Positives (FP), and False Negatives (FN). These counts are used to compute standard evaluation metrics: Precision, Recall, and the F1 Score. Unlike pixel-wise accuracy, this evaluation strategy focuses on instance-level matching, providing a more realistic assessment of object-level segmentation quality. Moreover, our script offers optional visualizations by overlaying predicted and ground truth regions on the original images, highlighting correct and incorrect predictions for qualitative inspection (Figure 6).

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

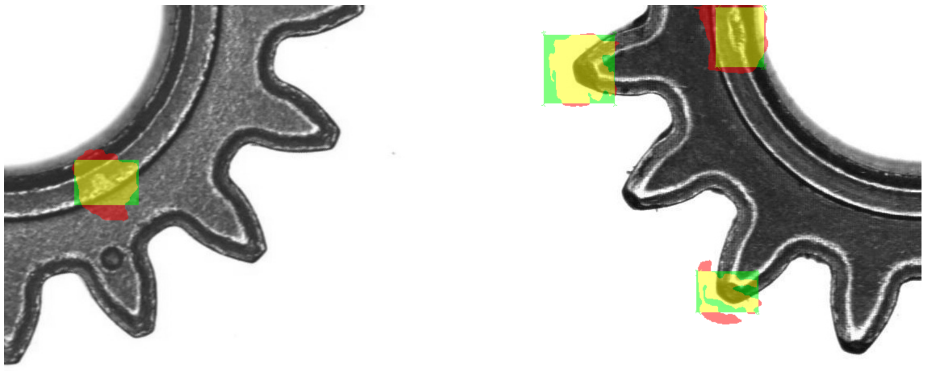


Figure 6: Images to compare the predicted output (red) and the groundtruth label (green).

3.3 Executed runs

For this experiment, we conducted multiple runs to investigate two key questions. The first was whether the BIT model is suitable for addressing this type of problem; to assess this, we performed three training processes, each using one of the model variants described in Section 2.3. The second question was whether altering the weights of the loss function affects model performance, and if so, how. To explore this, we carried out a total of seven training runs, each lasting 100 epochs with batch size of 1, SGD optimizer and learning rate set to 0.01 with linear decay. All experiments were executed on an NVIDIA RTX 2080 GPU with 12 GB of RAM.

4 Evaluation of results

In Table 1 we can see that the BIT model designed by the authors of the paper achieves the best performances in both Recall and F1 score, but the base model *base_resnet18* achieved the best Precision. These values align with the fact that *base_resnet18* has the lowest number of FP, but we have to consider that in anomaly detection problems it is way more important to detect anomalies even if we increment the number of False Positive. Indeed it is well-know that allowing a defective product to pass can cause significant damage to the company.

Backbone	TP	FP	FN	Precision	Recall	F1 Score
base_resnet18	88	**29**	34	**0.7521**	0.7213	0.7364
transformer_pos_s4_dd8	96	46	26	0.6761	0.7869	0.7273
transformer_pos_s4_dd8_dedim8	**98**	36	**24**	0.7313	**0.8033**	**0.7656**

Table 1: Performance metrics with different backbones

In Table 2, we compare the results of four different runs using the *transformer_pos_s4_dd8_dedim8* model, each with different weight configurations in the loss function. The first column of the table shows the pair of weights used, where the first value, set to 1, corresponds to non-anomalous pixels, and the second value is the increased weight assigned to anomalous pixels. As expected, increasing the anomaly weight generally leads to an increase in the number of TP. The number of FP almost always increase as the second weight grows.
The fourth weight pair, $[1, 36]$, reflects the actual class imbalance ratio in the training set, corresponding to the proportion between non-anomalous and anomalous pixels.
In summary, Table 2 shows that changing the loss function weights does not lead to a clear or consistent improvement in overall model performance. The main effect is a trade-off between Recall and Precision. Therefore, the choice of weights should be guided by the specific application context. For instance, if FN are particularly costly in the industrial setting, assigning a higher weight to anomalies might be justified, even if it results in a higher number of FP.

weights	TP	FP	FN	Precision	Recall	F1 Score
$[1, 1]$	98	**36**	24	**0.7313**	0.8033	**0.7656**
$[1, 10]$	99	178	23	0.3574	0.8115	0.4962
$[1, 20]$	102	119	20	0.4615	0.8361	0.5948
$[1, 36]$	102	174	20	0.3696	0.8361	0.5126
$[1, 5000]$	**107**	387	**15**	0.2166	**0.8770**	0.3474

Table 2: Performance metrics with transformer_pos_s4_dd8_dedim8 and different weights on the loss function

To conclude we would like to emphasize some extreme cases analysed with the evaluation mapping of *transformer_pos_s4_dd8_dedim8* with weights $[1, 1]$:

- There are some cases where we have found some cracks that seems mislabeled in the groundtruth, therefore we can suppose that the whole dataset was not perfectly classified (Figure 7).
- As we can see from Figure 8 oil stains seem to be well classified by the model.
- Lastly, we want to emphasize an important limitation of this method. In Figure 9 we can see a gear where all the teeth are ruined, this has led to a median image where all the teeth are damaged and, therefore, the model is unable to correctly classify the anomalies.

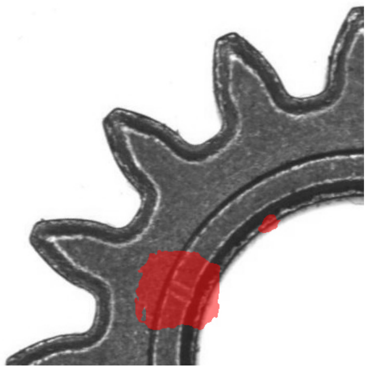


Figure 7: Example of mislabeled crack.

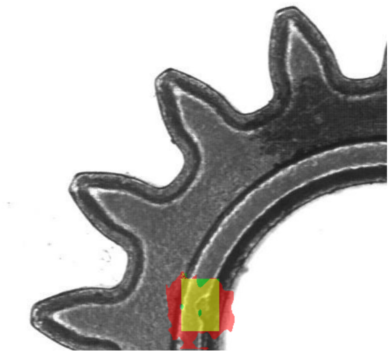


Figure 8: Example of correctly classified oil stain.

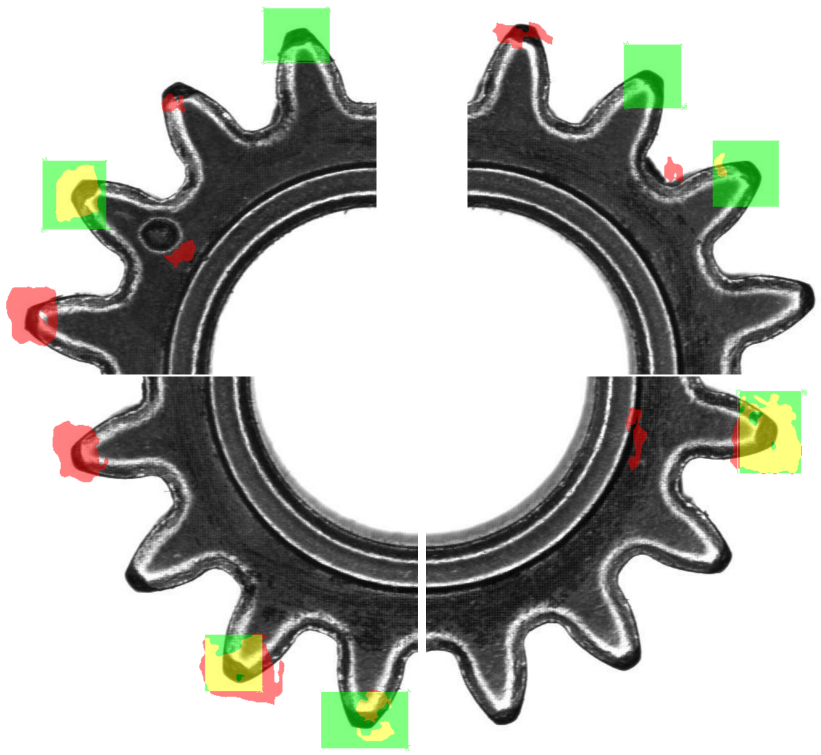


Figure 9: The four images of a gear with anomalies on the majority of teeth.

References

[1] GitHub. *GitHub repository of the paper Remote Sensing Image Change Detection with Transformers*. 2025. URL: `https://github.com/justchenhao/BIT_CD.git` (visited on 06/2025).

[2] GitHub. *GitHub repository of the Project Work*. 2025. URL: `https://github.com/LorenzoScaioli/Gear-Anomaly-Detection` (visited on 06/2025).

[3] Zipeng Qi Hao Chen and Zhenwei Shi. "Remote Sensing Image Change Detection with Transformers". In: *IEEE Transactions on Geoscience and Remote Sensing* (2021), pp. 1–14. DOI: `10.1109/TGRS.2021.3095166`.

[4] Roboflow. *Roboflow dataset gear Computer Vision Project*. 2025. URL: `https://universe.roboflow.com/senquire-6kfzs/gear-4lmaa` (visited on 06/2025).