DinoPatch: Patch-Based Anomaly Detection Using DINOv2

Jairus Abad
Dept. of Computer Engineering University of Victoria
Victoria, Canada

Mattias Kroeze
Dept. of Computer Engineering University of Victoria
Victoria, Canada.

ABSTRACT

This paper presents DinoPatch – an anomaly detection framework that leverages META AI Research's DINOv2's Vision Transformer combined with a PatchCore-inspired memory bank. Our method effectively addresses both texture-based (pasta) and shape-based (screw) anomalies by normalizing our inputs. PCA-informed alignment and computing Euclidean distances between image patches and their nearest neighbour in a memory bank effectively classifies anomalous and nominal images. Qualitative anomaly maps and quantitative metrics demonstrate excellent performance, achieving an accuracy, precision, recall, F1-score and AUROC of 100%. Our approach highlights the viability of ViT-based embeddings coupled with spatially aware memory banks for robust anomaly detection.

Code: *DinoPatch.ipynb*

Keywords - DINOv2, Patch Core, Memory Bank

I. INTRODUCTION

The primary goal of this project was to develop a computer vision model that can classify whether an image is anomalous, using two distinct datasets: one containing images of pasta, and the other featuring a screw with a washer, as shown in Figure 1. The pasta set is a texture-based anomaly detection problem, whereas the washer and screw set are shape-based anomalies. Texture-based anomalies involve subtle, local irregularities like discolorations or a change in pattern. Conversely, shape-based anomalies involve geometric deviations such as the washer being too far from the screw head. The significant challenge addressed by this project is developing a robust model that can identify anomalies across fundamentally different problems.

Figure 1: Nominal Images (left) and Anomalous Images (right)

To address these two problems, several well-established techniques from Computer Vision and Pattern Recognition were used; however, the most significant contributor was Meta AI Research's vision transformer model, DINOv2 [1]. This powerful self-supervised feature extractor was able to pick up on subtle texture and shape-based features in distinct patches of our pre-processed images. By calculating the Euclidean distance between each patch and patches at the same spatial location within our memory bank, we generate a heatmap highlighting anomalous regions.

Our method demonstrates effectiveness through qualitative visualization of anomaly maps and quantitative evaluation metrics such as AUROC scores, which confirms its potential for versatile, accurate anomaly detection across different problems.

II. LITERATURE REVIEW

A. Literature Review of Anomaly DINO

Similarly to our approach, AnomalyDINO leverages DINOv2 as a backbone. The paper proposes a vision-only approach for one and few-shot anomaly detection [2]. Anomalous images are found by utilizing a memory bank $\mathbf{M}$ where $\mathbf{M}$ contains the DINOv2 extracted features from patches of size 14x14 from a set of nominal training images. During the testing stage, the patch features of evaluation images are extracted and checked for similarity to the features stored in $\mathbf{M}$ by calculating the *cosine* distance d described by [2, Eq. 1] as opposed to the *Euclidean* distance utilized in this report.

$$d(x, y) = 1 - \frac{< x, y >}{||x||\ ||y||} \tag{1}$$

The equation was iteratively run at each spatial location in $\mathbf{M}$, and the minimum distance d was evaluated where x is the test patch that contains the features in the evaluation image and y is the reference patch contained in the memory bank. Anomaly DINO then implements the nearest neighbour algorithm described by $d_{NN}(x; \mathbf{M})$ [2, Eq. 2].

$$d_{NN}(x;\mathbf{M}) = \min(x, y) \tag{2}$$

The patch distances are then aggregated and scored as described by [2, Eq. 3], where $S(x)$ is the score of the test image and q takes the mean average of the top 1% largest patch differences.

$$S(x) = q(\{d_{NN}(x_1;\mathbf{M}), \dots, d_{NN}(x_N;\mathbf{M})\}) \tag{3}$$

The feature patch x is found in the test feature space, and y is the reference feature patch in the memory bank $\mathbf{M}$. For post-processing and localization of potential anomalies, the 14x14 anomaly map from DINOv2 was upsampled back to the original image resolution of 448 or 672 using bilinear interpolation. This results in a dense anomaly map with rough edges. These sharp changes in intensities were then rectified by applying Gaussian smoothing with $\sigma = 4$ [2]. Enriching the memory bank to increase training data variability was done via scaling and rotations. To reduce inference time and effective RAM usage, full-shot methods were utilized to reduce the size of $\mathbf{M}$. A few-shot scenario uses a small number of domain-specific patches of normal images to populate the memory bank as opposed to using every image in the normal set.

For more robust results, preprocessing using masking was implemented by also using DINOv2 to reduce false-positive results. Principal Component Analysis (PCA) on the patch features was used to produce the mask. The first PCA component of the patch was thresholded but was supervised since failure cases occurred when the objects of interest accounted for over 50% of the patches [2]. The application of these masks can be seen in [2, Fig. 2] below.

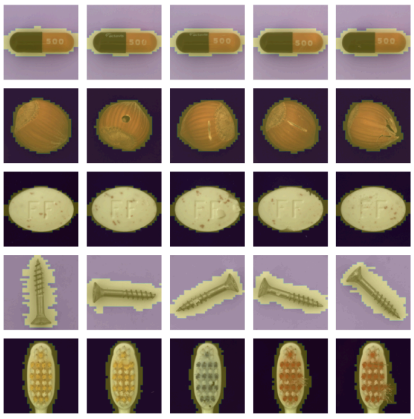

Figure 2: Masked Objects of Interest from MVTec-AD

After optimizing the memory bank by testing multiple shots, it was found that 16 shots (16 nominal images in the memory bank) with a resolution of 672 performed the best. AnomalyDINO (672) at 16-shots was able to yield impressive AUROC scores of 98.4±0.1% and 97.5±0.0% for Classification and Segmentation, respectively.

AnomalyDINO performs better than other modern zero and few-shot Anomaly Detection methods, such as SPADE, PaDiM, PatchCore and APRIL-GAN, to name a few, since it uses DINOv2's powerful self-supervised Vision Transformer (ViT) features (Section III, C). Furthermore, AnomalyDINO is a training-free method as opposed to APRIL-GAN and AnomalyCLIP.

AnomalyDINO was also able to optimally bound the memory bank to a limited set of images to reduce RAM usage and inference time. Our proposed solution does not apply few-shot methods since our training set was already quite limited. AnomalyDINO is more robust with few-shot scenarios or with limited training data since each evaluation patch is compared against every patch in the memory bank rather than just patches at the same spatial location. This works well for texture-based anomalies but can struggle with shape-based anomalies where the texture of the anomalous region is still valid; for example, in the cases of the missing washer or the misplaced washer within our anomalous screws dataset.

B. Literature Review of Patch Core

Patchcore leverages spatial context and retains spatial information in the memory bank, which makes it better for catching shape-based anomalies. This paper aims to detect anomalies by extracting patch-level embeddings using pre-trained Convolutional Neural Network (CNN) backbones (eg. WideResNet-50) instead of ViTs such as DINOv2 [3]. These patches populate a memory bank $\mathbb{M}$, efficiently capturing what is considered "nominal" from feature embeddings.

During evaluation, anomalies are identified by calculating the Euclidean distances from test patches to their nearest neighbours in the memory bank. Notably, Patchcore enhances anomaly localization by aggregating the nearest-neighbour (NN) distances within its local spatial neighbourhoods as opposed to globally across all patches like AnomalyDINO. This spatial sensitivity allows it to excel at detecting shape-based anomalies, such as misaligned or structurally defective objects, compared to purely texture-focused approaches.

Patchcore and DinoPatch are similar in their fundamental approach in detecting anomalies by utilizing a memory bank, and both use nearest-neighbor patch comparisons. These two approaches differ in how image features are extracted – DinoPatch uses a ViT, and Patchcore uses a CNN. In addition to feature extraction, our problem scope demands explicit preprocessing steps such as PCA-based alignment (Section III. B.) for consistent orientation and scale since, unlike Patchcore, our datasets lacked spatial consistency. Additionally, our approach avoided Patchcore's neighborhood aggregation strategy, which could be implemented in the future, as highlighted in Section IV C.

III. IMPLEMENTED APPROACH

A. Data Set Expansion

Due to our implementation's rotation, scale, and position invariant nature, standard image augmentations such as rotations, crops and translations are simply undone by our preprocessing step (Section III.B). If these images are added to our memory bank after the augmentations are reversed, they will only increase the RAM usage and redundant information.

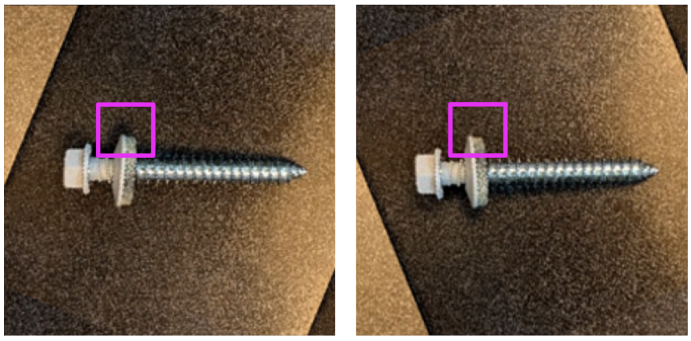

Figure 3: Original (left) and Mirrored (right) Preprocessed Training Images

Image mirror augmentations, however, introduce an effectively unseen image to our memory bank. After correcting the perspective with our preprocess step, the features at a patch location will be different between the original and mirrored, as shown in *Figure 3* above.

B. Pre-Processing

In the first stage of our pre-processing step, we convert the input image to grayscale and apply a Gaussian blur to reduce noise. Note that σ was automatically calculated by OpenCV based on the Kernel Size [3]. We then extract edges using OpenCV's Canny Edge Detector function *cv2.Canny()*, followed by contour detection with *cv2.findContours()*. The contour points are then stacked and passed to sklearn.decomposition.PCA to compute the principal components of the object(s) [3]. Performing PCA analysis allows us to extract parameters that help rotate and scale the images such that they are of consistent orientation and scale before feature extraction with DINOv2.

Algorithm 1: Principal Component Extraction

1: **Input**: A D-dimensional training set $X = \{x_1, x_2, \ldots, x_N\}$ and the new (lower) dimensionality d (with $d < D$).

2: Compute the mean

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

3: Compute the covariance matrix

$$\mathrm{Cov}(x) = \frac{1}{N}\sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T.$$

4: Find the spectral decomposition of $\mathrm{Cov}(x)$, obtaining the eigenvectors $\xi_1, \xi_2, \ldots, \xi_D$ and their corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_D$. Note that the eigenvalues are sorted, such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$.

PCA is a linear dimensionality reduction technique and a linear feature extraction method [6]. The features are extracted by using OpenCV's PCA decomposition function but can also be implemented using [6, Algo. 1] below, where x is the Canny Edges of our training set and N is the number of Edges detected.

The relevant output for our preprocessing method is the explained variance, which can be extracted from the Covariance Matrix's eigenvector and eigenvalue pair. The eigenvector with the highest eigenvalue is the first principal component and defines the direction of the maximum variance in the data, and the eigenvalue defines how much variance is captured along that direction.

From these principal components, we extract the mean position (estimated center), the orientation angle of the primary axis (from the eigenvector) and a scale proportional to the square root of the largest explained variance (from the eigenvalue).

Plotting these extracted features, as shown in *Figure 4,* allows us to visualize how accurately we are determining a consistent centerpoint, angle, and scale across a sample of images.

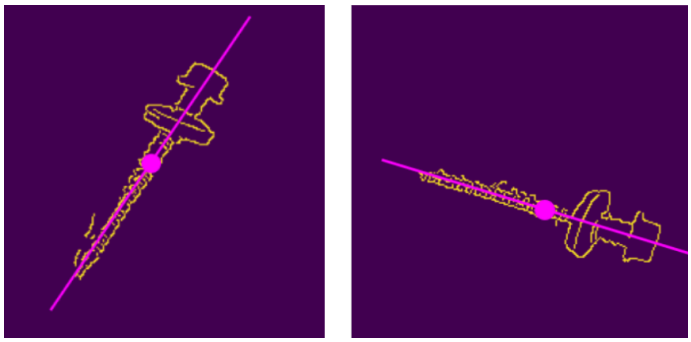

Figure 4: Plotting the extracted PCA info

With the information extracted during our PCA step, we then perform a series of transformations to our input image so that the object(s) are consistently positioned, scaled, and oriented within the image viewport, such that the memory bank can expect features in specific locations across various images.

Before any transformations are applied, we first pad the image by 100% using cv2.BORDER_WRAP. This ensures that when we transform and crop the image, we don't introduce any artificial black borders. Using wrap padding effectively "tiles" the image, so edge regions are filled with wrapped image content rather than a black fill. This prevents our feature extractor from encountering unnatural black regions and helps maintain the quality and relevance of features stored in the memory bank.

While adding this padding is important to efficiently fill out our memory bank, we don't want artificially padded regions to influence the anomaly score during image evaluation. Therefore, in the preprocessing step, we duplicate all transformations onto a corresponding mask image. This allows us to isolate only organic regions of the image during scoring.

C. Main Feature Extraction

Using a feature extractor allows us to represent image patches as high-dimensional feature embeddings that capture texture and shape information. This leads to a more robust and semantically meaningful representation of each patch compared to raw pixel values alone.

To do this feature extraction, rather than attempt to train our own Convolutional Neural Network (CNN) from our very limited input datasets, we opted to use Meta AI Research's DINOv2 ViT model, which has a strong semantic understanding of various textures and shapes with no additional fine-tuning.

DINOv2 transforms image patches of shape (14,14,3) into a (1, 768) vector embedding, resulting in an image feature matrix of (16,16,768). However, to obtain a higher-resolution feature matrix, we upscale the input image by a factor of 3.5, such that each DINOv2 patch effectively captures patches of shape (4,4,3), yielding a resulting feature matrix of (56,56,768).

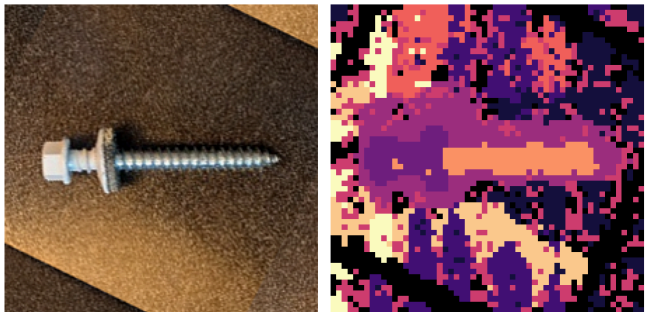

Figure 5: KNN grouped visualization of extracted DINOv2 feature matrix

While our preprocessing step (Section III.B) effectively aligns the object's principal axis to the horizontal, PCA alone cannot determine the object's polarity. To address this, we sample the extracted feature embeddings along the primary axis. We construct a reference orientation vector by subtracting the first half of the axis's embeddings from the second half.

This reference orientation vector is then used to align the polarity of future feature matrices. For each new image, we extract an orientation vector along the same axis and compute its dot product with the orientation reference. If the result is negative, indicating opposite polarity, we horizontally flip the feature matrix to ensure consistent alignment.

D. Memory Bank

Iterating through each image in the training dataset, we apply the preprocessing augmentations to normalize the object's scale, position, and orientation. The preprocessed image is then passed into our feature extractor along with a reference orientation vector to ensure consistent polarity. All resulting feature matrices are stacked to form a feature memory bank. Each memory bank must be loaded into RAM to perform the nearest neighbour search required for anomaly map construction. Due to the relatively small size of our training dataset, explicit memory management was not necessary for our implementation. However, if a larger dataset was provided and memory management was required, we discuss strategies and considerations in detail in Section IV.C.

E. Anomaly Map Construction

During evaluation, each image is passed through the same preprocessing and feature extraction pipeline used to construct the memory bank. The image's orientation vector is compared to the reference orientation vector to correct for polarity; if a flip is required, the feature matrix and padding mask are flipped accordingly.

We then iterate through each spatial location in the evaluation image's feature matrix, computing the Euclidean distance (described by Eq. 4 below) between its embedding and each of the corresponding embeddings in the memory bank.

$$d(x,y) = \sqrt{\sum_{i=1}^{N} (y_i - x_i)^2} \quad (4)$$

This L2 norm function has parameters x and y, which are the patch features in $\mathbb{M}$ and the patch features of the test set, respectively, and N is the length of the feature embedding. Similarly to AnomalyDINO, the Cosine distance was also used but discarded since it yielded worse results.

An anomaly heatmap, as shown in *Figure 7,* is constructed using the minimum distance at each location, representing the best-case anomaly score across all training samples. The mask generated during the pre-processing stage is used to remove regions that were artificially padded.

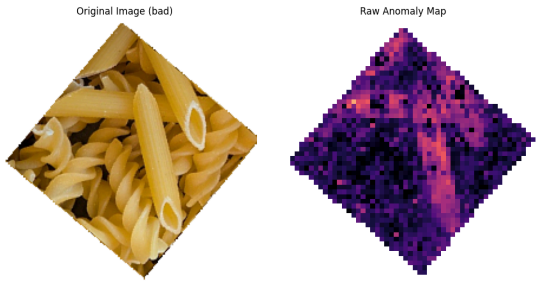

Figure 7: Anomalous evaluation image and corresponding raw anomaly heatmap

F. Post-Processing

Once the anomaly map is constructed, we can qualitatively identify anomalous regions through visual inspection. However, the raw anomaly map contains too much noise and fine detail to reliably quantitatively classify anomaly presence. To address this, we apply a post-processing step that reduces noise and highlights meaningful regions according to predetermined hyperparameters, as seen in *Figure 8*. This includes applying a minimum and maximum threshold, followed by morphological filtering to isolate clusters above a minimum size and fill small holes. We also experimented with Gaussian and median filtering, although we found these approaches degraded performance.

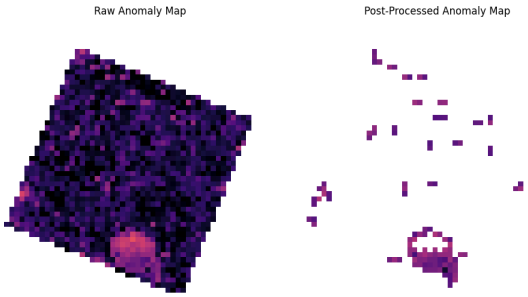

Figure 8: Post-Processed Anomaly Map

G. Scoring & Optimizations

Once we obtain a post-processed anomaly heatmap, we consider three potential methods for determining an overall anomaly score for the evaluated image: the maximum value in the heatmap, the mean value across the heatmap, and the 99th percentile value.

With the anomaly detection pipeline complete, we process all evaluation images to generate their raw anomaly maps. These maps are then passed through an optimization loop that randomly generates post-processing hyperparameters within defined ranges and computes anomaly scores for each image using all specified scoring methods (max, mean, and 99th percentile). The loop runs for a fixed duration, updating the optimal hyperparameter configuration and scoring method whenever a higher AUROC score is achieved.

IV. RESULTS & EVALUATION

A. Evaluation of the Results

Across multiple runs of the optimizer loop, we consistently see similar optimal hyperparameter configurations, such as: *{'min_threshold': 39.386, 'max_threshold': 41.942, 'min_object': 2, 'min_hole': 15, 'method': '99p'}*. After optimizing the hyperparameters, DinoPatch was able to fully classify between an anomalous and a nominal image. As can be seen from *Table 1*. Our method was able to yield an accuracy, precision, recall, F1 and AUROC score of 1.00. The histograms seen in *Figure 9* show that the bad and good images are fully separable, with a few samples lying very close to the threshold.

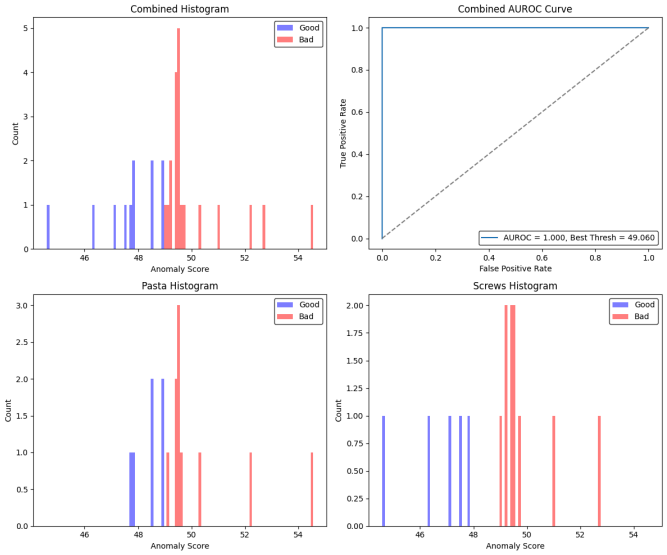

Figure 9: Histograms and AUROC Curve of DinoPatch

Table 1: Quantitative Contribution of each Step

Method	Accuracy	Precision	Recall	F1 Score	AUROC
No Pre & Post Processing (49.772 Threshold)	0.839	0.800	1.000	0.889	0.889
No Pre-Processing (48.144 Threshold)	0.968	0.952	1.000	0.976	0.973
No Post-Processing (49.453 Threshold)	0.903	1.000	0.850	0.919	0.895
DinoPatch (49.092 Threshold)	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

It was found that without the preprocessing step, the AUROC score of 97.3% was of a result of an anomalous pasta being classified as nominal and a nominal screw being classified as anomalous, as seen in *Figure 10* below.

Figure 10: Misclassified Images of Pasta and Screw

Without our preprocessing step, the optimizer loop tends to select very aggressive thresholding hyperparameters, effectively masking off the entire string anomaly within the pasta image above since it's considered "too anomalous." The nominal screw being classified as anomalous, on the other hand, is likely due to the slightly angled position of the washer, where the top surface of the washer is more visible compared to the other nominal samples. We have consistently seen this image flagged across multiple implemented approaches. DINOv2 likely sees the top surface of the washer as a noticeably different texture from the rubberized side edge.

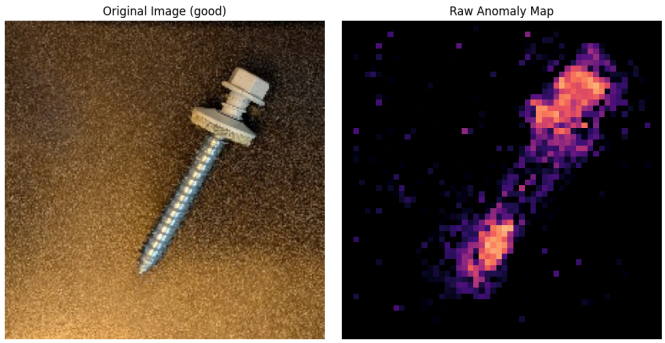

Figure 11: No Preprocessing Anomaly Map of Nominal Image

Furthermore, when the pre-processing step is disabled, a sample raw anomaly map (as seen in *Figure 11*) shows highly anomalous regions despite the input image being nominal. This can be attributed to the memory bank having no record of screw features at that spatial location during training. However, despite these anomalous regions, after applying our post-processing step, this heatmap is classified as nominal. This indicates that our optimizer is aggressively tuning the post-processor for optimal evaluation performance. This may suggest that real-world performance will likely be lower than what was measured. This disparity could have been captured with a separate validation and testing set.

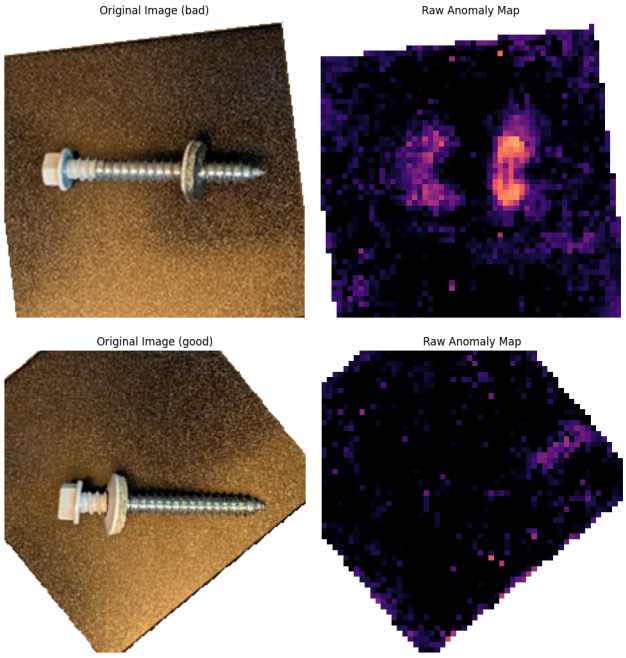

Figure 12: Heatmaps of Preprocess (top) vs. no preprocessed (bottom) images

This observation, in combination with the similar AUROC scores between the "No Post-Processing" and "No Pre or Post-Processing" trials, somewhat implies that our pre-processing technique has a relatively insignificant effect; however, visual analysis contradicts this. When utilizing our pre-processor, visual inspection of a nominal and anomalous image shown in *Figure 12* demonstrates that our has an impressive capability to localize anomalistic regions, however, noise imparted by the variance within the background necessitates a post-processing step. The success of our "No Pre-Processing" runs can be attributed to the efficiency of the post-processor optimizer alone.

B. DinoPatch vs Existing Methods Summary

Our approach leverages PCA-informed pre-processing, PatchCore framework and DINOv2 feature extraction, effectively handling both texture-based anomalies (pasta) and shape-based anomalies (screws). Like AnomalyDINO, our method benefits from DINOv2's robust feature embeddings, but unlike their *global* nearest-neighbor aggregation, we maintain explicit spatial context, which allows us to enhance sensitivity to geometric abnormalities. Compared to Patchcore, which uses CNN-based embeddings with explicit neighborhood aggregation, our ViT implementation provides a rich semantic description of the features within each patch. Our technique integrates strengths from both PatchCore and AnomalyDINO, optimizing performance across diverse anomalies.

C. Future Considerations

In a future, more refined implementation, the goal would be to eliminate the need for a post-processing step entirely. Instead, the raw anomaly map would be of sufficiently high quality and free from noise, allowing it to be directly used for scoring and classifying an image as anomalous.

Improvements not implemented that may have significant performance gains include: Implementing a combined weighted local and global patch feature distance anomaly scoring method, allowing for spatially consistent and image-wide patch differences to influence patch anomaly scores. We could also implement semantic segmentation to attempt to mask off irrelevant background regions. Additionally, we could implement the memory bank neighbourhood analysis employed by Patchcore. Finally, we could train a simple CNN on each dataset that will predict an Affine transformation matrix, rather than relying on information extracted by PCA alone.

Each memory bank must be loaded into RAM to perform the nearest neighbour search required for anomaly map construction. If this algorithm was employed on larger datasets than the two we were tasked with, the memory bank quickly becomes cumbersome and memory intensive as it would use additional memory proportional to the number of training images. To address this, we could adopt the N-shot methodology used by AnomalyDINO, where the memory bank is constructed from N sample images rather than the full dataset. Alternatively, we could build the memory bank from the full training set and perform a distillation step using KNN grouping to reduce redundancy at each patch location. In addition, the dimensionality of our (1, 768) feature embeddings could be reduced using PCA without significant loss in specificity.

V. CONCLUSION

In this paper, we presented DinoPatch, an effective anomaly detection framework combining DINOv2 ViT embeddings with a PatchCore-inspired memory bank. Our method addressed both texture (pasta) and shape-based (screws) anomaly detection problems using PCA-based image alignment and Euclidean nearest-neighbour distance computations. DinoPatch was used to demonstrate impressive performance, achieving accuracy, precision, recall, F1-score and AUROC scores of 100%. However, we are unsure of real-world performance without a testing dataset. Our optimizer may have overfit to the evaluation set to maximize AUROC, and the final post-processing hyperparameters may generalize poorly.

Qualitative anomaly maps also demonstrated our method's strong localization capability. Compared to AnomalyDINO, our method maintains spatial context, which is crucial for accurately detecting shape anomalies where AnomalyDINO primarily performs global NN textural comparisons. Additionally, unlike PatchCore's CNN-based feature extraction, our ViT-drive approach provides richer information. While DinoPatch demonstrated a bright future, future work involving larger and more varied datasets and rigorous validation and testing procedures is essential to better assess generalization and practical implementation.

VI. REFERENCES

[1] M. Oquab et al., "DINOv2: Learning Robust Visual Features without Supervision," Meta AI Blog, Apr. 14, 2023. [Online]. Available: https://ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/(arXiv) https://ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/

[2] S. Damm, M. Laszkiewicz, J. Lederer, and A. Fischer, "AnomalyDINO: Boosting Patch-based Few-shot Anomaly Detection with DINOv2," arXiv preprint arXiv:2405.14529, Mar. 2025. [Online]. Available: https://arxiv.org/abs/2405.14529(arXiv)

[3] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler, "Towards Total Recall in Industrial Anomaly Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. [Online]. Available: https://arxiv.org/pdf/2106.08265

[4] OpenCV Development Team, "OpenCV 4.x Documentation," OpenCV, 2025. [Online]. Available: https://docs.opencv.org/4.x/index.html

[5] Scikit-learn Developers, "sklearn.decomposition.PCA," *scikit-learn Documentation*, 2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[6] J. Wu, Essentials of Pattern Recognition: An Accessible Approach, Cambridge University Press, 2020.