Towards Total Recall in Industrial Anomaly Detection

Carluccio Alex
Politecnico Di Torino
s302373@studenti.polito.it

Federico Luigi
Politecnico Di Torino
s295740@studenti.polito.it

Longo Samuele
Politecnico Di Torino
s305202@studenti.polito.it

Abstract

Nowadays the ability to recognize defective parts is very important in large-scale industrial manufacturing. The problem becomes more challenging when we try to fit a model using a nominal (non-defective) image only at training time. This problem is called "cold-start". The best approaches combine embeddings from ImageNet models with an outlier detection model. In this paper, we reproduce the state-of-the-art approach called ***PatchCore*** *[18], which uses a representative memory bank of nominal patch features to achieve high performances both on detection and localization. We developed this approach using the MVTec Anomaly Detection dataset on which PatchCore achieves an image-level anomaly detection AUROC score of up to 98.4%, almost doubling the performance compared to related works. Furthermore, we tested the patchcore by replacing the backbone with an alternative solution, trying to get better generalizable representations to leverage for the downstream task. This alternative solution is represented by CLIP: we exploited the pretrained Image Encoder of CLIP instead of the ImageNet pretrained one. The experiment results suggest that using ResNet50x16 as the architecture of the image encoder we obtain better results on a smaller training set.*

Code: PatchCore GitHub Implementatio

1. Introduction

Industrial Anomaly Detection is the process of identifying abnormal or unusual patterns in industrial data or products. This technique is commonly used in manufacturing and other industrial environments to identify potential issues with equipment or processes, prevent shipping faulty products, and increased efficiency. By detecting anomalies in a timely manner, organizations can reduce downtime, improve safety, and increase overall operational performance. There are various methods used to perform anomaly detection, including machine learning algorithms and statistical techniques. Sometimes the inspection is made by humans that are very able to differentiate between nominal samples and outliers after having only seen a small number of non-defective instances. In this work, we address the computational version of this problem, cold-start anomaly detection for visual inspection of industrial image data, in which a model must distinguish between samples extracted from the training data distribution and those outside its support. This problem arises in many industrial scenarios where it is easy to acquire images of non-defective (nominal) examples, but it is expensive and complicated to specify the expected defect variations in full. The classification of industrial visual defects is very difficult because errors can range from small scratches to entire missing components, and it is not possible to predict all cases.

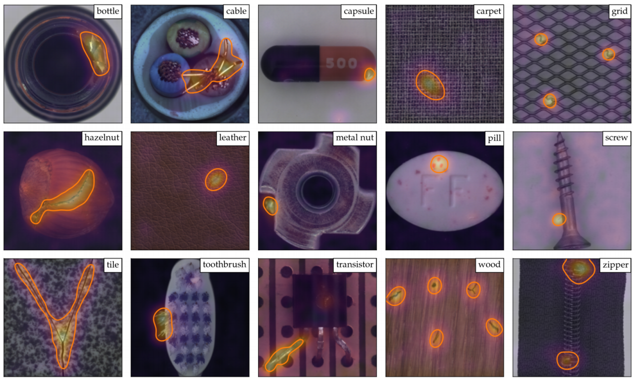

Figure 1. Examples from the MVTec benchmark datasets. The orange boundary denotes anomaly contours of actual segmentation maps for anomalies such as broken glass, scratches, burns, or structural changes in blue-orange color gradients.

Figure 1 shows some examples from the MVTec Anomaly Detection dataset used as a test base for our solution. Existing work in this area is based on learning a model of the nominal distribution by self-coding methods [7, 14, 21] , GAN [1, 15, 20], or other unsupervised methods [19, 27]. Recently [3, 5] , proposed to use common deep representations from ImageNet classification without adapting them to the target distribution for anomaly detection. This method offers strong performance and solid spatial localization of defects but has limitations such as a lack of adaptation. In this paper, we present PatchCore, as a remedy that maximizes nominal information available at test time, reduces biases towards ImageNet classes, and retains high inference speeds. PatchCore uses locally aggregated, mid-level feature patches and feature aggregation over a local neighborhood to achieve this. Experiments on the MVTec AD demonstrate the power of PatchCore for industrial anomaly detection, achieving state-of-the-art results while retaining fast inference times without requiring training on the dataset at hand. Furthermore, experiments have also shown the high sample efficiency of PatchCore, matching existing anomaly detection methods in performance while using only a fraction of the nominal training data.

While PatchCore shows high effectiveness for industrial anomaly detection, its applicability is limited by the transferability of the pretrained features leveraged. In order to obtain stronger and more generalizable features from the pretrained backbone, we propose a competitive solution to the ImageNet pretrained WideResNet50 used in PatchCore method: the CLIP method [16]. Specifically, we exploit the pretrained Image Encoder of CLIP instead of the ImageNet pretrained one to see if it enables obtaining better anomaly detection performance with PatchCore. Experiments have shown that for smaller training datasets, some pretrained CLIP architecture alternatives outperform the vanilla version of PatchCore.

2. Related Works

The majority of Anomaly Detection Models are based on the ability to learn information from nominal data. This goal can be achieved in different ways, like auto-encoding models [21], or using different extensions like GMM [28], generative adversarial training objectives [1, 15, 20], attention-guidance [25], and structural objectives [13, 26]. There are also other ways like unsupervised representation learning methods that can be used; an example is GANs [8]. For the majority of these approaches, anomaly localization comes naturally based on pixel-wise reconstruction errors. Other approaches such as GradCAM [22] or XRAI [12] can be used for anomaly segmentation.

The literature on general anomaly detection is large, mostly related to information security, and also the slice specialized in industrial anomaly detection using nominal representations is vast. Industrial images come with their ”problems” or ”challenges” for which recent works like [3] has shown state-of-the-art detection performance using models pretrained on large external datasets like ImageNet without adaptation. This has started a new family of methods like SPADE [5] that uses memory banks comprising various feature hierarchies, and KNN-based [11] models for anomaly segmentation or image-level anomaly detection. Similarly, the recent method PaDiM [9] uses a pretrained convolutional neural network (CNN) for patch embedding and a multivariate Gaussian distribution to get a probabilistic representation of the normal class.

The components used to develop the PatchCore method are most related to SPADE and PaDiM solutions. SPADE creates a memory bank of nominal features extracted from a pre-trained backbone network with separate approaches for image and pixel-level anomaly detection and PatchCore similarity uses a memory bank with coreset subsampling to ensure low inference cost at higher performance. More recently methods based on coreset have found their usage in deep learning approaches, mostly used for network pruning [2] or active learning [23]. Finally, the patch-level approach for both image-level anomaly and anomaly segmentation is related to PaDiM. PatchCore uses a locally aware patch feature memory bank accessible to all patches evaluated during test time; instead, PaDiM limits the detection of anomalies at patch level to Mahalanobis distance measures specific to each patch. In this way, the PatchCore method becomes more resistant to image misalignment and uses a much larger nominal context to identify anomalies. In addition, in contrast with PaDiM which requires all the images to have the same dimension at test and train time, PatchCore doesn't require this constraint and uses locally aware patch feature scores to keep the local spatial variance and to reduce bias towards ImageNet classes.

3. Method

The PatchCore method is structured in different parts that we will describe in order:

- Definition of local aware patch features than aggregated into a memory bank (Sec. 3.1)
- Coreset-reduced method to sub-sample the memory bank (Sec. 3.2)
- Full algorithm that performs anomaly detection and segmentation (Sec. 3.3).

Figure 2 provides an overview of the PatchCore method.

3.1. Locally aware patch features

We denote with $\mathcal{X}_N$ and $Y_n \in \{0, 1\}$ the set of nominal samples and their labels respectively, where the latter indicates whether an image x is anomalous $(y = 1)$ or nominal $(y = 0)$.

PatchCore is based on an ImageNet pre-trained network ϕ and we use $\phi_{i},_{j} = \phi_j(x_i)$ to denote the feature map for the image x_i at the layer j of the pre-trained network ϕ. The feature-map layer should be selected carefully, as the more the hierarchy level j is deep in a network and the worse a feature map may perform for different downstream tasks since it becomes more task-dependent losing more localized nominal information. In this case, it would

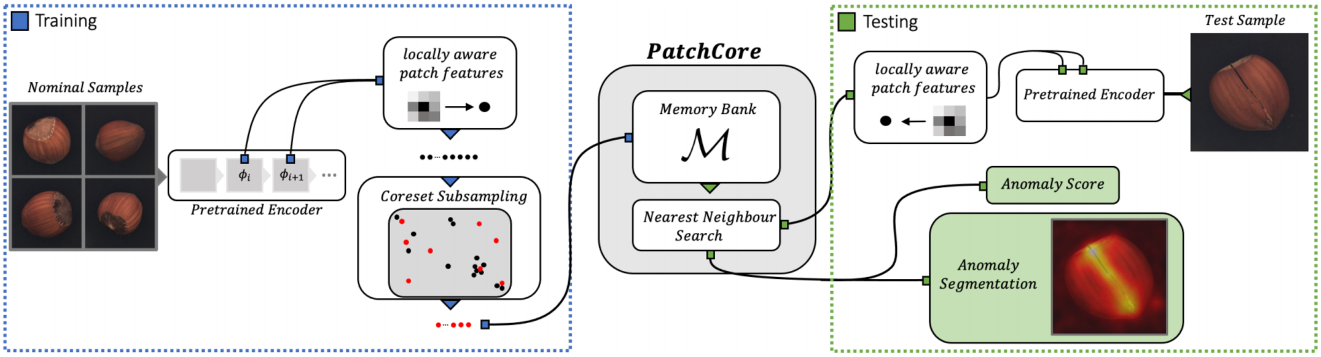

Figure 2. Overview of *PatchCore*. Nominal samples are split into a memory bank of neighborhood-sensitive patch-level features. The memory bank is downsampled via greedy coreset subsampling to reduce both redundancy and inference time. At test time, images are classified as anomalies if at least one patch is anomalous, and pixel-level anomaly segmentation is generated by scoring each patch-feature.

be biased toward the task of natural image classification. A ResNet-Like architecture is composed of 4 layers, with $j \in \{1, 2, 3, 4\}$ indicating the final output of respective spatial resolution blocks. We have selected mid-feature representations ($j = \{2, 3\}$) avoiding features too generic or too heavily biased towards ImageNet classification and then we used a memory bank $\mathcal{M}$ of patch-level features extracted at j levels.

Assuming a feature map $\phi_{i,j} \in \mathbb{R}^{c^*,h^*,w^*}$ to be a three-dimensional tensor of depth c^*, height h^* and width w^*, we denote with $\phi_{i,j}\,(h, w) = \phi_j(x_i, h, w) \in \mathbb{R}^{c^*}$ a c^*-dimensional feature slice at positions $h \in \{1, ..., h^*\}$ and $w \in \{1, ..., w^*\}$. Ideally, each patch-representation operates on a large enough receptive field size to account for meaningful anomalous context robust to local spatial variations. So a patch can be represented as a local neighbourhood aggregation of grouped points:

$$\begin{aligned}\mathcal{N}_p^{(h,w)} = \{(a, b)|a \in [h - [p/2], ..., h + [p/2]],\\ b \in [w - [p/2], ..., w + [p/2]]\},\end{aligned}$$

to increase receptive field size and robustness to small spatial deviations without losing spatial resolution or usability of feature maps and accounting for uneven patchsizes p. Additionally, we incorporate the feature vectors from the neighbourhood and locally aware features at position (h, w) with an aggregation function $\mathcal{F}_{agg}$ which is performed for all pairs (h, w). In this case, we have used adaptive average pooling as aggregation function, obtaining one single representation at (h, w) of predefined dimensionality d. Finally, we define for a feature map tensor $\phi_{i,j}$ its locally aware patch-feature collection $\mathcal{P}_{s,p}\,(\phi_{i,j}\,)$:

$$\begin{aligned}\mathcal{P}_{s,p}\,(\phi_{i,j}\,) = \{\phi_{i,j}\,(\mathcal{N}_p^{h,w})|\\ h, w \; mod \; s = 0, h < h^*, w < w^*, h, w \in \mathbb{N}\}\end{aligned}$$

with the striding parameter s set to 1. Moreover, we considered aggregated multiple feature hierarchies of mid-layers j and $j + 1$ to keep the generality of used features and to obtain further robustness. This is achieved by computing $\mathcal{P}_{s,p}(\phi_{i,j+1}\,)$ and aggregating each element with its corresponding patch feature at the lowest hierarchy level simply by rescaling $\mathcal{P}_{s,p}(\phi_{i,j+1}\,)$ such that $|\mathcal{P}_{s,p}(\phi_{i,j+1}\,)|$ and $|\mathcal{P}_{s,p}(\phi_{i,j}\,)|$ match. Finally, for all nominal training samples $x_i \in \mathcal{X}_N$, the PatchCore memory bank $\mathcal{M}$ is then simply defined as:

$$\mathcal{M} = \bigcup_{x_i \in \mathcal{X}_N} \mathcal{P}_{s,p}\,(\phi_j(x_i))$$

3.2. Coreset-reduced patch-feature memory bank

For increasing sizes of $\mathcal{X}_N$, $\mathcal{M}$ becomes exceedingly large and with it both the inference time to evaluate novel test data and required storage. Unfortunately, random subsampling, especially by several magnitudes, will lose significant information available in $\mathcal{M}$ encoded in the coverage of nominal features. A possible solution, used in this work, is the coreset subsampling mechanism to reduce $\mathcal{M}$ which keeps high performance while noticeably reducing inference time.

The goal of the coreset selection is to find a subset $\mathcal{S} \subset \mathcal{A}$ such that problem solutions over $\mathcal{A}$ can be most closely and especially more quickly approximated by those computed over $\mathcal{S}$ by using *minimax facility location* coreset selection as PatchCore uses nearest neighbor computations to ensure approximately similar coverage of the $\mathcal{M}$-coreset $\mathcal{M}_C$ in patch-level feature space as compared to the original memory bank $\mathcal{M}$.

$$\mathcal{M}_C^* = \underset{\mathcal{M}_C \subset \mathcal{M}}{\text{argmin}} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_C} ||m - n||_2$$

In order to compute $\mathcal{M}_C^*$, we use the iterative greedy approximation combined with Johnson-Lindenstrauss theorem [6] to further reduce the coreset selection time, where the latter is used to reduce dimensionalities of elements $m \in \mathcal{M}$ through random linear projections $\psi : \mathbb{R}^d \to \mathbb{R}^{d^*} < d$.

For notation, we use PatchCore-n% to denote the percentage n to which the original memory bank has been subsampled to, e.g., PatchCore-1% is a 100x times reduction of $\mathcal{M}$.

3.3. Anomaly detection with PatchCore

Once the memory bank is filled up with the nominal patch-features, in order to correctly recognise an anomaly, we estimate the image-level anomaly score $s \in \mathbb{R}$ for a test image x^{test} by computing the maximum distance score s^* between test patch-features in its patch collection $\mathcal{P}(x^{test}) = \mathcal{P}_{s,p}(\phi_j(x^{test}))$ to each respective nearest neighbour $m^* \in \mathcal{M}$:

$$m^{test,*}, m^* = \underset{m^{test}\in\mathcal{P}(x^{test})}{\text{argmax}} \ \underset{m\in\mathcal{M}}{\text{argmin}} ||m^{test} - m||_2$$

$$s^* = ||m^{test,*} - m^*||_2$$

To improve robustness with respect to the maximum patch distance, we use the scaling factor w on s^* to account for the behavior of neighbour patches. If the memory bank features m^* closest to anomaly candidate $m^{test,*}$ are themselves far from neighbouring samples, the anomaly score increase:

$$s = \left(1 - \frac{e^{||m^{test,*}-m^*||_2}}{\sum_{m\in\mathcal{N}_b(m^*)} e^{||m^{test,*}-m||_2}} * s^*\right)$$

with $\mathcal{N}_b(m^*)$ the b nearest patch-features in $\mathcal{M}$ for test patch-feature m*. In addition, a segmentation map can be computed in the same step, by realigning computed patch anomaly scores based on their respective spatial location. We upscaled the segmentation results by bi-linear interpolation to match the original input resolution and we smoothed the result with a Gaussian filter of kernel width $\sigma = 4$

4. Extension

4.1. CLIP

CLIP stands for Contrastive Language Image Pretraining and is a multi modal zero shot model. This means that if we give an image and a set of descriptions to the model, it's able to calculate the most pertinent description for that image without any optimization for a task. Before going deep into the architecture, let us define the meaning of the following words:

- *Multi Modal*: The architecture uses more than one domain to learn a specific task. In this case, CLIP combines text and video.
- *Zero Shot*: The model is able to work with unseen labels without having been trained to classify them.
- *Contrastive Language*: CLIP is trained to understand that similar representations should be close to the latent space, while dissimilar ones should be far apart. In the following this concept will be better clarify.

4.1.1 Contrastive Pre Training

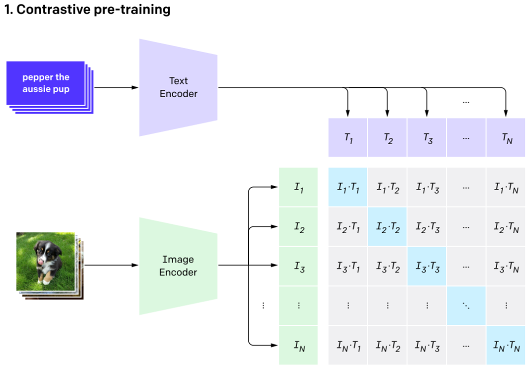
1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 ... T_N
I_1 I_2 I_3 ⋮ I_N
$I_1{\cdot}T_1$ $I_1{\cdot}T_2$ $I_1{\cdot}T_3$ ... $I_1{\cdot}T_N$
$I_2{\cdot}T_1$ $I_2{\cdot}T_2$ $I_2{\cdot}T_3$ ... $I_2{\cdot}T_N$
$I_3{\cdot}T_1$ $I_3{\cdot}T_2$ $I_3{\cdot}T_3$ ... $I_3{\cdot}T_N$
$I_N{\cdot}T_1$ $I_N{\cdot}T_2$ $I_N{\cdot}T_3$ ... $I_N{\cdot}T_N$

Figure 3. Constructive pre-training of the CLIP method.

Figure 3 shows how contrastive pre-training works. Let's assume that we have a batch of images and a set of descriptions. This step jointly trains a Text Encoder and an Image Encoder that produce text $[T_1, T_2, ..., T_N]$ and image embeddings $[I_1, I_2, ..., I_N]$ in a way that, in the big matrix $N \times N$, the cosine similarities of the correct image-text pairs are maximized and the other wrong pairs are minimized. In conclusion, we can say that the Text Transfomer is a standard Transformer model with GPT2-style modifications [17] and the Image Encoder can be either a ResNet or a Vision Transformer [10].

4.1.2 Zero Shot Classification

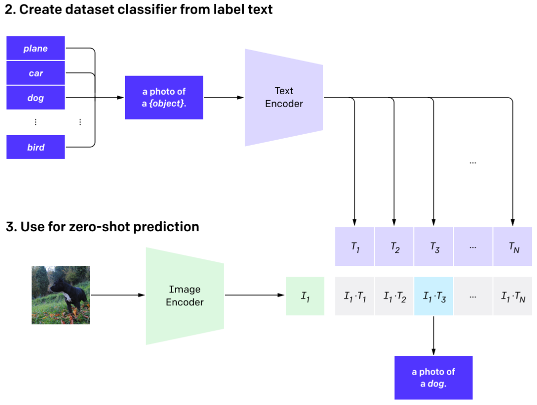
2. Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
3. Use for zero-shot prediction
Image Encoder
T_1 T_2 T_3 ... T_N
I_1
$I_1{\cdot}T_1$ $I_1{\cdot}T_2$ $I_1{\cdot}T_3$ ... $I_1{\cdot}T_N$
a photo of a dog.

Figure 4. Creation process of the dataset classifier from label text and usage of the image encoder for zero-shot prediction.

The process of how CLIP performs zero-shot classification is shown in Figure 5. First of all, we provide an image and a set of descriptions. If we provide just labels the description will be created starting from the label and asserting it into the standard phrase "A photo of a *label*". Then, descriptions and images are embedded using the Text and Image

Encoder. Finally, CLIP computes the cosine similarities between the embeddings and chooses the highest value to make the prediction.

4.1.3 Image Encoder

CLIP's authors considered two different architectures for the image encoder: a ResNet50 and a ViT. For the first one they used a ResNet50 as base architecture but with several modifications (such as replacing the global average pooling layer with an attention pooling mechanism). For the second architecture they used a Vision Transformer (ViT) (Dosovitskiy et al., 2020) with just minor modifications [16]. They trained 5 ResNets (RN50, RN50-4, RN50-16, RN-64, RN101) and 3 Vision Transformes (ViT-B/16, ViT-B/32, ViT-L/14) for 32 epochs on 30 public datasets, also using a trick for those datasets with just image and label couples by converting classes into sentences like "*bird → a photo of bird*". The modified variants RN50-4, RN50-16 and RN-64, were created respecting the EfficientNet-style [24] and use approximately 4x, 16x, and 64x the compute of a ResNet-50.

4.2. Alternative Backbones

ImageNet classification pretraining has been a standard solution to obtain generalizable features to leverage for downstream tasks for a long time. Some other alternatives have been proposed to obtain stronger and more generalizable representations: CLIP is one of those.

In our work, we exploit the pretrained image encoder of CLIP instead of the ImageNet pretrained to see if we can obtain better anomaly detection performance with PatchCore. We choose as backbone substitute for the standard WideResNet50 pretrained on ImageNet the CLIP pretrained ResNet architectures ResNet50, ResNet50-4, ResNet50-16, and ResNet101. Unfortunately, we were unable tp run the ResNet50-64 due to the high memory resources required, which we could not handle. We compared the obtained results from the other ResNets in the section 5.3.

5. Experiments

5.1. Dataset and metrics

The main dataset used for benchmarking anomaly detection methods with a focus on industrial inspection is the **MVTec Anomaly Detection dataset** [4]. It contains over 5000 high-resolution images divided into 15 different object and texture categories. Each category includes a set of non-defective training images and a test set of both images without defects as well as images with various kinds of defects, with the respective anomaly ground truth masks.

For vanilla PatchCore, the one with WideResNet-50 as backbone, images are resized and center cropped to 256x256 and 224x224, respectively. Regarding CLIP PatchCore, which uses the clip image encoder as backbone, we resized and center cropped images based on the architecture used. We obtained the size used from the clip transformer and the size are the same for both resize and center crop image pre-processing:

- ResNet50: 224x224
- ResNet50x4: 288x288
- ResNet50x16: 384x384
- ResNet101: 224x224

As evaluation metric to measure image-level anomaly detection performance we used the area under the receiver operating characteristic curve (AUROC) using produced anomaly scores. In order to have a single evaluation metric we compute the class-average AUROC. We used the same metric to evaluate the segmentation performance.

5.2. Anomaly Detection using vanilla PatchCore

Table 1 shows the results for image-level and pixel-level (segmentation) anomaly detection on MVTec classes. We report the results obtained by PatchCore-10%, i.e. with a level of memory bank subsampling of 10%. We choose this subsampling value because it guarantees the best compromise between score and inference time, according to the experiments conducted by *Roth et al.* [18]. We can observe that the AUROC values are similar for each class. It follows that regardless of the class analyzed the patchcore achieves state-of-the-art performances. Comparing the average AUROC of 0.984 that we obtained with the one computed by *Roth et al* [18] (0.990) we can conclude that the results are almost the same and that PatchCore confirms to achieve state-of-the-art image-level anomaly detection performances. The comparison can be done for anomaly segmentation: we obtained 0.974 AUROC value, similar to the 0.981 obtained by *Roth et al* [18].

5.3. CLIP PatchCore experiments

In this section, we analyze if exploiting the pretrained Image Encoder of CLIP instead of the ImageNet pretrained one enables better AD performances with PatchCore. We tested four pretrained architectures based on ResNet as Image Encoder: ResNet101, ResNet50, ResNet50x4, and ResNet50x16.

Table 2 shows image-level performances on MVTec both for ResNet101 and ResNet50 as PatchCore backbone. The average results suggest that using WideResNet50 pretrained on ImageNet guarantees better results. Overall, the performances are high for both the CLIP image encoders. In Table 3 we can observe the segmentation results, comparing the same architectures. The average performance of CLIP

MVTec Class	image-level	pixel-level
Average	**0.984**	**0.974**
Bottle	1.000	0.979
Cable	0.986	0.980
Capsule	0.973	0.984
Carpet	0.987	0.984
Grid	0.952	0.957
Hazelnut	1.000	0.981
Leather	1.000	0.986
Metal Nut	0.997	0.968
Pill	0.941	0.988
Screw	0.966	0.985
Tile	0.980	0.948
Toothbrush	1.000	0.982
Transistor	0.998	0.979
Wood	0.994	0.931
Zipper	0.985	0.974

Table 1. Image-Level and Pixel-Level (segmentation) anomaly detection performance on MVTec AD classes usign AUROC as metric.

image encoder with ResNet50 is comparable with the results obtained with the PatchCore vanilla version.

MVTec Class	RN101	RN50	Vanilla
Average	0.969	0.978	**0.984**
Bottle	1.000	1.000	1.000
Cable	0.945	0.977	**0.986**
Capsule	0.961	0.969	**0.973**
Carpet	**0.998**	0.990	0.987
Grid	0.966	**0.974**	0.952
Hazelnut	**1.000**	0.997	**1.000**
Leather	1.000	1.000	1.000
Metal Nut	0.978	**0.998**	0.997
Pill	0.880	0.888	**0.941**
Screw	0.936	0.960	**0.966**
Tile	**0.999**	0.997	0.980
Toothbrush	0.947	0.964	**1.000**
Transistor	0.984	0.992	**0.998**
Wood	0.988	0.991	**0.994**
Zipper	0.950	0.978	**0.985**

Table 2. Image-Level AD performances on MVTec classes using AUROC as metric and using ResNet101 or ResNet50 as CLIP Image Encoder for PatchCore's backbone. The last column shows the vanilla patchcore results (Tab. 1).

During the experiments conducted on the ResNet50x4 and ResNet50x16 architectures, we encountered memory management problems since we ran out of memory when we trained on too many training images. To address this problem, we opted for reducing the training set over which the locally aware patch features are extracted. We use half training samples for each class when the clip image-encoder architecture is ResNet50x4 and a quarter when the architecture is ResNet50x16. In order to compare the results, we trained the vanilla PatchCore over the same training subset obtaining the results shown in Table 4 and Table 5 for ResNet50x4 and in Table 6 and Table 7 for ResNet50x16. Comparing image-level and segmentation results using ResNet50x4 as PatchCore backbone with the ones obtained running vanilla PatchCore, the CLIP extension slightly outperforms the vanilla results.

MVTec Class	RN101	RN50	Vanilla
Average	0.969	**0.975**	0.974
Bottle	0.976	**0.980**	0.979
Cable	0.976	**0.980**	**0.980**
Capsule	0.975	0.980	**0.984**
Carpet	0.982	**0.985**	0.984
Grid	0.932	**0.957**	**0.957**
Hazelnut	0.985	**0.986**	0.981
Leather	0.982	0.985	**0.986**
Metal Nut	0.970	**0.974**	0.968
Pill	0.976	0.982	**0.988**
Screw	0.976	**0.986**	0.985
Tile	**0.955**	0.951	0.948
Toothbrush	0.987	**0.989**	0.982
Transistor	0.970	0.976	**0.979**
Wood	0.920	**0.934**	0.931
Zipper	0.970	**0.975**	0.974

Table 3. Pixel-Level AD performances on MVTec classes using AUROC as metric and using ResNet101 or ResNet50 as CLIP Image Encoder for PatchCore's backbone. The last column shows the vanilla patchcore results (Tab. 1).

Table 6 and Table 7 confirms this trend also for ResNet50x16 as backbone. In this case the difference of performance are higher, suggesting that the pretrained CLIP image-encoder based on ResNet50x16 architecture as PatchCore backbone performs better over a smaller training set.

The expectation of having better results using the pretrained Image Encoder of CLIP instead of the ImageNet pretrained one has been confirmed specifically when we use ResNet50x16 as architecture over a smaller MVTec training set. This suggests that in case the training dataset with images of non-defective objects is small, using the CLIP image encoder can guarantee better performance than patchcore with the backbone pre-trained on ImageNet.

MVTec Class	RN50x4	Vanilla
Average	**0.970**	0.968
Bottle	1.000	1.000
Cable	0.941	**0.984**
Capsule	**0.957**	0.953
Carpet	0.981	**0.988**
Grid	**0.980**	0.910
Hazelnut	1.000	1.000
Leather	1.000	1.000
Metal Nut	**0.998**	0.996
Pill	0.896	**0.941**
Screw	**0.958**	0.887
Tile	**0.996**	0.990
Toothbrush	0.875	**0.894**
Transistor	0.988	**0.997**
Wood	0.979	**0.990**
Zipper	**0.996**	0.990

Table 4. Image-Level AUROC results obtained using ResNet50x4 as architecture for PatchCore's CLIP Image-Encoder backbone. The MVTec training set has been halved.

MVTec Class	RN50x4	Vanilla
Average	**0.977**	0.972
Bottle	**0.988**	0.979
Cable	0.974	**0.980**
Capsule	**0.985**	0.983
Carpet	**0.987**	0.985
Grid	**0.978**	0.946
Hazelnut	**0.989**	0.982
Leather	**0.993**	0.987
Metal Nut	**0.969**	0.967
Pill	0.977	**0.987**
Screw	**0.993**	0.981
Tile	**0.961**	0.949
Toothbrush	**0.983**	0.965
Transistor	0.937	**0.978**
Wood	**0.952**	0.933
Zipper	**0.989**	0.979

Table 5. Pixel-Level AUROC results obtained using ResNet50x4 as architecture for PatchCore's CLIP Image-Encoder backbone. The MVTec training set has been halved.

MVTec Class	RN50x16	Vanilla
Average	**0.967**	0.946
Bottle	1.000	1.000
Cable	0.927	**0.967**
Capsule	**0.931**	0.919
Carpet	**0.995**	0.985
Grid	**0.990**	0.906
Hazelnut	0.999	**1.000**
Leather	1.000	1.000
Metal Nut	**0.997**	0.983
Pill	**0.928**	0.924
Screw	**0.935**	0.752
Tile	**0.999**	0.989
Toothbrush	**0.892**	0.789
Transistor	0.934	**0.998**
Wood	**0.995**	0.991
Zipper	0.983	**0.987**

Table 6. Image-Level AUROC results obtained using ResNet50x16 as architecture for PatchCore's CLIP Image-Encoder backbone. The training set used is one-fourth of the original MVTec training set.

MVTec Class	RN50x16	Vanilla
Average	**0.976**	0.969
Bottle	**0.988**	0.979
Cable	0.966	**0.978**
Capsule	0.978	**0.983**
Carpet	**0.991**	0.984
Grid	**0.988**	0.940
Hazelnut	**0.989**	0.981
Leather	**0.995**	0.987
Metal Nut	**0.972**	0.965
Pill	0.967	**0.985**
Screw	**0.990**	0.973
Tile	**0.966**	0.948
Toothbrush	**0.981**	0.948
Transistor	0.920	**0.977**
Wood	**0.959**	0.933
Zipper	**0.992**	0.977

Table 7. Pixel-Level AUROC results obtained using ResNet50x16 as architecture for PatchCore's CLIP Image-Encoder backbone. The training set used is one-fourth of the original MVTec training set.

6. Conclusion

This paper reimplemented the *PatchCore* method for cold-start anomaly detection, in which only non-defective samples are used at training time to detect, leveraged to detect and segment anomalous data at test-time. *PatchCore* guarantees state-of-the-art cold-start image anomaly detection and localization system with low computational cost through the coreset subsampling and using nominal patch-level feature representations extracted from an ImageNet pretrained WideResNet50 network. We achieve an image anomaly detection AUROC over 98%, confirming the results retrieved by *Roth et al.* [18].
We also prosed an extension hoping to obtain better AD performances, using an innovative and stronger solution to obtain generalizable features from the backbone. We replaced the ImageNet pretrained network with the pretrained Image Encoder of CLIP, testing different architectures. The most performing CLIP image encoder was ResNet50-16 which outperforms the vanilla backbone for a smaller training set. Regardless of the backbone used, PatchCore shows high effectiveness for both industrial anomaly detection and segmentation. Using the CLIP image encoder as backbone to retrieve generalized features from the nominal samples could provide better results for a small training set, compared to the vanilla version.

References

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training, 2018. 1, 2

[2] Ben Mussay andMargarita Osadchy, Vladimir Braverman, Samson Zhou, and Dan Feldman. Data-independent neural prun- ing via coresets, 2020. 2

[3] Liron Bergman, NivCohen, and Yedid Hoshen. Deepnearest neighbor anomaly detection, 2020. 1, 2

[4] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Ste. Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection, 2019. 5

[5] Niv Cohen and Yedid Hoshen. Sub-image anomaly detection with deep pyramid correspondences, 2020. 1, 2

[6] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss, 2003. 3

[7] Diana Davletshina, Valentyn Melnychuk, Viet Tran, Hitansh Singla, Max Berrendorf, Evgeniy Faerman, Michael Fromm, and Matthias Schubert. Unsupervised anomaly detection for x-ray images, 2020. 1

[8] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. Image anomaly detection with generative adversarial networks, 2019. 2

[9] Thomas Defard, Aleksandr Setkov, Angelique Loesch, and Romaric Audigier. Padim: A patch distribution modeling framework for anomaly detection and localization, 2021. 2

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 4

[11] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection, 2002. 2

[12] Andrei Kapishnikov, Tolga Bolukbasi, Fernanda Viegas, and Michael Terry. Xrai: Better attributions through regions, 2019. 2

[13] Paul Bergmann Sindy Lowe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders, 2019. 2

[14] Duc Tam Nguyen, Zhongyu Lou, MichaelKlar, and Thomas Brox. Anomaly detection with multiple-hypotheses predictions, 2019. 1

[15] Stanislav Pidhorskyi, Ranya Almohsen, Donald A Adjeroh, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders, 2018. 1, 2

[16] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 2, 5

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. 4

[18] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection, 2022. 1, 5, 8

[19] Marco Rudolph, Bastian Wandt, and Bodo Rosenhahn. Same same but differnet: Semi-supervised defect detection with normalizing flows, 2021. 1

[20] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, and Ehsan Adeli. Adversarially learned one-class classifier for novelty detection, 2018. 1, 2

[21] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction, 2014. 1, 2

[22] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization, 2017. 2

[23] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018. 2

[24] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. 5

[25] Shashanka Venkataramanan, Kuan-Chuan Peng, Rajat Vikram Singh, and Abhijit Mahalanobis. Attention guided anomaly localization in images, 2020. 2

[26] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity, 2004. 2

[27] Jihun Yi and Sungroh Yoon. Patch svdd: Patch-level svdd for anomaly detection and segmentation, 2020. 1

[28] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoen-

coding gaussian mixture model for unsupervised anomaly detection, 2018. 2