Advanced Machine Learning: Anomaly Detection project

Belloni Sofia (s303393) and Ferla Damiano (s305953)

Politecnico di Torino

11 September 2023

Abstract

The purpose of the project is to familiarise itself with the anomaly detection task, focusing on the industrial world. For this reason, the 'cold start' problem was tackled, i.e. building a model that is capable of detecting anomalies in industrial objects, despite only having been trained on nominal, defect-free samples. This approach is necessary because anomalies can vary from small changes such as light scratches, to larger structural defects such as missing components. This variety of anomalies makes it difficult to define in advance what types of problems might occur in industrial products, making it difficult to collect datasets of abnormal samples.
Consequently, the model is only trained on normal data and must be able to detect, during testing, all samples with defects never shown before.
Code: https://github.com/DamoFlare/PatchCore2.0

1 Introduction

The model representing the state-of-the-art in this field is PatchCore[1]. The starting point of our analysis was to implement PatchCore from scratch, trying to obtain results similar to the original one. Subsequently, modifications were made to the original model, with the aim of improving performance.
PatchCore model is based on ResNet pre-trained on Imagenet, and our improvement mainly concerns the pre-training phase.
As mentioned, the starting point is PatchCore. It offers competitive inference time while it obtains state-of-the-art performance for the anomaly detection and localization.
PatchCore is characterised by different parts: local patch features aggregated into a memory bank, a core-set reduction to increase the efficiency and the algorithm to detection and localization.
Once a coreset memory bank of nominal patch-features has been defined it is possible to exploit it to perform anomaly detection and segmentation, by comparing the patch-features of each test sample with the nominal ones.
We obtained very similar results respect the original paper. Later we will discuss about it.
Finally, we replaced the pre-trained network on ImageNet with CLIP[4]'s pretrained Image Encoder, trying to achieve better anomaly detection results.

2 Related Works

There was some past works about cold-start problem, like SPADE[2] and PaDiM[3].

2.1 SPADE

SPADE (Sub-Image Anomaly Detection with Deep Pyramid Correspondences) is an algorithm for anomaly detection in images. It uses deep pyramid correspondences to detect anomalous sub-regions within an image. The objective of SPADE is to identify significant deviations from the normal behaviour of regions. It extracts features from images using a pre-trained CNN and generates correspondences between image regions and normal reference regions. Finally, It calculates distances between features in image regions and reference regions to create an anomaly map. SPADE enables the detection of localised anomalies within images.
SPADE makes use of a memory-bank of nominal features extracted from a pretrained backbone network with separate approaches for image- and pixel-level anomaly detection. PatchCore similarly uses a memory bank, but a more nominal context is retained and a better fitting inductive bias is incorporated. In addition, the memory bank is coreset-subsampled to ensure low inference cost at higher performance.

2.2 Padim

PaDiM (Patch Distribution Modeling) is another algorithm for anomaly detection in images. Based on modelling patch distributions, PaDiM identifies anomalous regions within an image. It extracts patch features using a pre-trained CNN and builds a model of normal patch distributions. It calculates Mahalanobis distances to assess how far a region deviates from the normal distribution. Regions exceed-

ing a predefined threshold are classified as anomalies. PaDiM enables accurate anomaly detection based on modelling patch distributions.
Finally, PatchCore patch-level approach to both image-level anomaly detection and anomaly segmentation is related to PaDiM with the goal of encouraging higher anomaly detection sensitivity. It makes use of an efficient patch-feature memory bank equally accessible to all patches evaluated at test time, whereas PaDiM limits patch-level anomaly detection to Mahalanobis distance measures specific to each patch. Furthermore, unlike PaDiM, input images do not require the same shape during training and testing.

2.3 CLIP

For a long time, the most popular pretraining concerned ImageNet.
Pre-training on ImageNet focuses on general learning of the visual characteristics of images, while pretraining with CLIP combines learning from images and text to acquire a shared understanding between language and images.
CLIP (Contrastive Language-Image Pretraining) is a model that combines language and images to tackle a variety of tasks, including anomaly detection.
CLIP was trained on a large dataset of image pairs and associated texts. During training, the model learns to link the corresponding image to its textual description. This allows CLIP to acquire a shared representation of images and texts.
For anomaly detection, CLIP can be used in the following way: the algorithm is provided with an input image and a text describing normal behaviour. The model generates a representation for both inputs and calculates the similarity between the image and the text.
If the input image is abnormal, its representation is expected to be less similar to the text describing normal behaviour. Conversely, if the image is normal, more similarity is expected between the image and the reference text.
Training on a different dataset allows CLIP to gain a general understanding of images and language, enabling it to tackle anomaly detection tasks with good generalisation ability.

3 Method

PatchCore can be considered in three parts, as illustrated in Figure 1.

- Local patch features aggregated into a memory bank
- Coreset subsampling
- Anomaly detection and segmentation

We're going to analyze each point in details.

3.1 Local patch features aggregated into a memory bank

PatchCore uses a Resnet50 network pre-trained on Imagenet, from which features are extracted for the subsequent subsampling. One choice for feature representation was to take the last level of features available in the network. However, this approach caused the loss of nominal information about the samples, because the extracted features were too deep and suffered from bias towards the network's original task, image classification.
Therefore, it was decided to use the features that came from the mid to high layers of the network, especially layers 2 and layer 3. In this way, we obtained features that were not too general or too specific to the natural classification task.
In order to do this, in the forward step each sample passed through the network, done by the *timm*library, which allowed us to extract features arbitrarily. Next, the spatial dimensions of the feature maps were reduced by pooling, before undergoing a reshape and resize to obtain the patch.
Finally, we define the memory bank M (first derived from SPADE) simply as the union of all the patches obtained.

3.2 Coreset subsampling

Increasing the number of test samples, M becomes extremely large and the inference time with it, and also the amount of memory required. Therefore, it is necessary to find a way to reduce this size, unlike

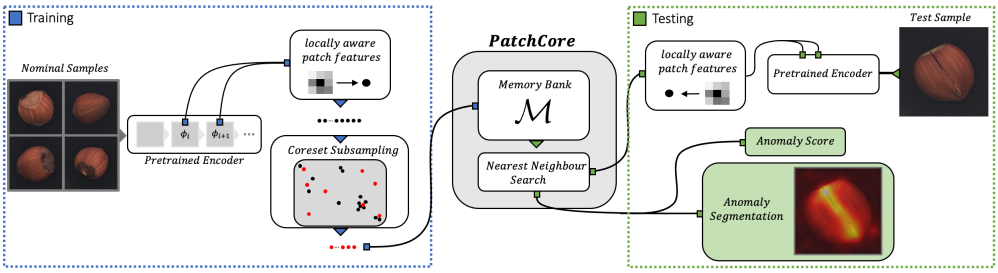

Figure 1: PatchCore architecture

SPADE which instead performed a pre-selection of features to be added to the memory bank.
Random subsampling was not a great idea because important information about nominal features were lost, so it was decided to use coreset subsampling, to reduce the size of M while affecting performance as little as possible.
This type of subsampling requires a metric to be performed. In relation to the fact that, as we're going to see later, PatchCore uses nearest neighbour computation, it was decided to use a minimax facility location metric.
This metric consists of considering the last element entered, and from that, calculating all the minimum distances to the other elements. We add the element with the minimum distance from the first sample considered to our coreset, and repeat the iteration until we have reached our predetermined coreset size.
In our tests, we tried subsampling at **1%**, **25%**and **50%** of the starting memory bank. Below is the pseudocode algorithm representing our implementation of the minimax facility location:

Algorithm 1: Coreset selection

Input : patches: array, perc_coreset: coreset %
Output: coreset_indexes: array of chosen indexes added to the coreset

```
coreset_indexes ← [];
index ← 0;
last_item ← patches[index : index + 1];
coreset_indexes.append(index);
min_distances ← torch.linalg.norm(patches -
 last_item, dim=1, keepdims=True);
while len(coreset_indexes) ≤ (len(patches) ×
 perc_coreset) do
    distances ← ——(patches - last_item)——;
    min_distances ← min(distances,
     min_distances);
    index ← argmax(min_distances);
    last_item ← patches[index : index + 1];
    min_distances[index] ← 0;
    coreset_indexes.append(index);
return coreset_indexes;
```

3.3 Anomaly detection and segmentation

We used the same methodology as in the paper to calculate the anomaly score.
With the patch-feature memory bank M, they estimate the anomaly level **s** for an image test as the maximum score of distance **s*** between test patch-features and each nearest neighbour **m*** associated to it.

$m^{test,*}, m^* = \arg\max \arg\min ||m^{test} - m||_2$
$s* = ||m^{test,*} - m*||_2$

To obtain **s**, we scale the weight **w** on **s*** to account the behaviour of neighbourhood patches. If memory bank features are closer to anomaly candidate, they are far from neighbouring samples, so we increase the anomaly score.

$(1 - \frac{exp||m^{test,*} - m^*||_2}{\sum exp||m^{test,*} - m||_2})$

Finally, we smoothed the result with a Gaussian with kernel $\sigma = 4$, just as in the paper.

4 CLIP Pretraining

In a second step, we decided to implement a variant of PatchCore using a different pre-training, in order to determine whether a different feature representation could lead to an improvement in the results.

Pre-training with ImageNet has long been a standard solution for acquiring sufficiently generic features for use in subsequent tasks. However, a number of alternatives have emerged in recent years to achieve even more generalisable representations, including the use of CLIP (Contrastive Language-Image Pretraining).

Although the original goal of CLIP is not anomaly detection, its ability to connect images and text allows for the development of anomaly detection models that combine information from both sources.

In our extension, we propose to use CLIP's pretrained Image Encoder instead of the one pre-trained on ImageNet, to examine whether this allows for better performance in anomaly detection using PatchCore.

In particular, we trained our model from ResNet50 offered by OpenAI's Clip library. As in the standard PatchCore implementation, it was decided to use the features of the middle and upper layers of the network, in particular layer 2 and layer 3. In order to be able to achieve this without modifying the library code, we embedded a custom hook in the forward step of the model so that the intermediate features could be saved.

A hook consists of a function that is called when the forward step is taken. This was necessary because the Clip library did not allow features to be extracted arbitrarily (as is the case with the Timm library). Therefore, two forward hooks were registered (called respectively to extract features at layer2 and layer3), which merely saved these features in a data structure. It was also necessary to modify the preprocessing of the images, since CLIP requires different transformations than those used for ImageNet. In fact, unlike the ImageNet dataset in which the transformations are applied directly to the tensors, CLIP requires working with RGB images.

5 Results

Dataset. All tests were performed on the MVTec dataset, composed of high-resolution images designed for testing and evaluating anomaly detection methods, particularly in industrial settings. The dataset contains over 5000 high-resolution images divided into fifteen different object and texture categories. Each category comprises a set of nominal-only training images and a test set of images with various kinds of defects as well as images without defects.

We used the whole dataset, with the exception of the "hazelnut" class. This class contains more than 390 training samples, but we don't have enough resources to run the training phase for them. In fact, the coreset subsampling is very expensive in terms of memory. One solution could be to reduce the cardinality of that class, reaching the same number of samples as the other classes (about 300 samples).

Evaluation metrics. Image-level anomaly detection performance is measured via ROC AUC(Receiver Operating Characteristic Area Under the Curve), a metric that quantifies the overall performance of a binary classification model by measuring the area under the receiver operating characteristic curve. This metric evaluates the model's ability to discriminate between classes across different probability thresholds. Segmentation performance is measured via pixel-wise ROC AUC. This metric evaluates the model's ability to correctly classify pixels by analyzing the AUROC for each pixel independently.

Experiments. Both versions of PatchCore have been tested using different hyperparameter values to identify the best configuration. In particular, we an-

alyzed the impact of the following hyperparameters:

- **k**, i.e. the number of nearest patch-features analyzed
- **perc coreset**, i.e. the percentage to which the original memory bank has been subsampled to
- **eps**, i.e. the parameter to control the quality of the embedding according to the Johnson-Lindenstrauss lemma.

In the following, we're going to analyze the result for Vanilla PatchCore before, and Improved PatchCore later.

5.1 Vanilla PatchCore

We divide our result analysis in two parts: one for the anomaly detection and segmentation, and the other one for the inference times.
Firstly, we had to chose the value of **k**, the number of neighbors. Using the same percentage of the coreset (25%) and changing the value of k between 3 and 5, we found out that the model with $k = 5$ behaves slightly better, in terms of anomaly score and inference time. We can see the result in the following table.

Table 1: PatchCore results with different values of k

k	Anomaly segm.	Anomaly det.	Inference time
k=3	96.4%	95.6%	0.0524
k=5	96.4%	97.2%	0.0513

So, starting from now, we're going to consider the result with **k=5**.
In the following table, we're going to report the performance with a different percentage of the coreset subsampling. We was expecting that an increment of the subsampling will give a better performance, but a worst inference time.
Obviusly, the inference time increase because the number of samples that are in the memory bank are higher. We can see that, even if we're going to increase the percentage of the subsampling, the performance will be more or less the same, so the 25% of the coreset is worth enough.

Table 2: PatchCore results with different values of % coreset

%coreset	Anomaly segm.	Anomaly det.	Inference time
1%	96.2%	97.5%	0.010
25%	96.4%	97.2%	0.051
50%	96.5%	97.0%	0.105
75%	96.5%	96.7%	0.171

The last terms we want to analyze is **eps**. This is the parameter to control the quality of the embedding according to the Johnson-Lindenstrauss lemma. Smaller values lead to better embedding. The main problem is to find a tradeoff between a good embedding and a good inference time. Decreasing the eps will bring an higher inference time, and we have to consider that. We was able to use an eps value of 0.7. After that, the Google Colab runtime crashed because we used all available resources.

Table 3: PatchCore results with different values of eps

eps	Anomaly segm.	Anomaly det.	Inference time
eps=0.9	96.2%	97.5%	0.010
eps=0.7	96.4%	97.2%	0.051

Anyway, we decided to use an eps value of 0.9. The increment of 0.2% in the anomaly segmentation score is not enough the choose a smaller value.
Finally, in table 4 the result for the whole dataset.

5.2 Improved PatchCore

After that, we proceeded to run the PatchCore version with Clip pre-training using the previously identified best hyperparameter values. Also in this case we used resnet50 as backbone in order to better compare the performance of the two models. In Table 5

Table 4: PatchCore results with k=5, eps=0.9, %corset=25

Class	Anomaly segm.	Anomaly det.	Inference time
BOTTLE	97.0%	100%	0.046
CABLE	97.0%	98.0%	0.050
CAPSULE	98.0%	96.0%	0.049
CARPET	98.0%	99.0%	0.060
GRID	95.0%	91.0%	0.056
LEATHER	98.0%	100%	0.053
METAL NUT	95.0%	98.0%	0.049
PILL	98.0%	90.0%	0.059
SCREW	99.0%	95.0%	0.070
TILE	93.0%	99.0%	0.050
TOOTHB.	98.0%	100%	0.019
TRANSI.	97.0%	99.0%	0.048
WOOD	90.0%	99.0%	0.056
ZIPPER	97.0%	98.0%	0.054

Table 5: PatchCore pre-trained on Clip results with k=5, eps=0.9, perc corset=25%

Class	Anomaly segm.	Anomaly det.	Inference time
BOTTLE	98.0%	100%	0.015
CABLE	98.0%	96.0%	0.040
CAPSULE	98.0%	96.0%	0.039
CARPET	98.0%	99.0%	0.044
GRID	96.0%	97.0%	0.035
LEATHER	98.0%	100%	0.040
METAL NUT	97.0%	99.0%	0.029
PILL	98.0%	88.0%	0.036
SCREW	99.0%	94.0%	0.038
TILE	95.0%	99.0%	0.034
TOOTHB.	99.0%	98.0%	0.016
TRANSI.	97.0%	99.0%	0.038
WOOD	93.0%	99.0%	0.041
ZIPPER	98.0%	96.0%	0.032

the results achieved by PatchCore with Clip.

We can see how the performance changes according to the different classes of the dataset, sometimes behaving better and sometimes worse than the classic version of PatchCore.

From the Table 5 we can see how the model with Clip pretraining brings a 0.9% improvement for segmentation problem, although the performance on anomaly detection is about 0.1% worse on average. Another interesting data is the average inference time: the latter model is in fact much faster.
Finally, in Table 6, we consider the average values of both models.

Table 6: Comparison between two PatchCore versions

Model	Anomaly segm.	Anomaly det.	Inference time
Classic	96.4%	97.2%	0.0513
Clip	97.3%	97.1%	0.0345

6 Conclusion

This project has allowed us to improve our knowledge and skills on the anomaly detection task, especially the "cold-start" problem, in which knowledge of only nominal examples has to be leveraged to detect and segment anomalous data at test-time.

PatchCore shows remarkable performance respect to other anomaly detection approach, demonstrating its effectiveness across a wide range of different objects. Our re-implementation results are very similar to the ones from the original article and we are very proud of it, especially for the limited computational resources that didn't allow us to carry out tests with all the values we wanted to analyze.
Regarding the CLIP version, we can say that the extension of PatchCore with a different pre-training is very interesting. In the future it could be used together with new changes of the classic PatchCore model to bring even more significant improvements. For example it would be interesting to add a fine-tuning of the network on the images of the specific

task, in order to reduce the bias towards the pre-training dataset features and to obtain better results. In fact, one of the main problems of using a network pre-trained on a different dataset than the test one is the fact that when going towards deepest network layers the features become more object-centrics. In both versions of PatchCore we tried to avoid this by choosing the features of the middle and upper layers of the network, in particular layer 2 and layer 3. Nevertheless, fine-tuning the network on task-specific images could reduce the bias towards ImageNet or Clip object features and potentially enhance the approach to achieve better results.

References

[1] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, P. Gehler. *Towards Total Recall in Industrial Anomaly Detection*, In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 14318–14328.

[2] N. Cohen, Y. Hoshen. *Sub-image anomaly detection with deep pyramid correspondences* . arXiv preprint arXiv:2005.02357, 2020.

[3] T. Defard, A. Setkov, A. Loesch, and R. Audigier. *Padim: A patch distribution modeling framework for anomaly detection and localization.* In A. Del Bimbo, R. Cucchiara, S. Sclaroff, G. Farinella, T. Mei, M. Bertini, H. Jair Escalante, and R. Vezzani, editors, Pattern Recognition. ICPR International Workshops and Challenges, pages 475–489, Cham, 2021. Springer International Publishing.

[4] A. Radford, J. Wook Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. *Learning transferable visual models from natural language supervision.* In International Conference on Machine Learning, pages 8748–8763. PMLR, 2021.