AML Project 5 - Anomaly Detection

Thibaut Rouet[1](s307013), Stanick Le Gal[1](s307340), Thomas Osorio[1](s307107)
[1]Politecnico di Torino

Abstract

This report aims to present the work done for the completion of the Anomaly Detection Project in the course of Advanced Machine Learning (AML) of Politecnico di Torino (PoliTo) during the first semester of the 2022/2023 year. The main contribution to this project is the reimplementation of ***PatchCore****. The Github of our work can be found at the following link : https://github.com/tonatiu92/AMLProject_Patchcore*

1. Introduction

As mentioned above, this paper presents the work done for an Anomaly Detection project. The anomaly detection is a data exploration problem where the purpose is to find or detect the components that are presenting some defects or which are really different from same-class members. This project is based on the previous work presented in [1] where researchers present a method working as **cold-start problems**. Their method is to detect anomalies on images issued from industrial dataset without processing the learning phase on defective images but on without-default ones. This approach was used due to the low number of defective images present in the dataset but also in general : it is easier to find a large amount of nominal images than defective ones. The hardest part of detecting errors in an industrial environment lies in the large variety of errors, as they can go from tiny changes to larger structural defects.

PatchCore is the main approach to solve this problem. It is an effective remedy based on characteristics, allowing us to ensure good performances. The first one is the utilization of **mid-level network patch features** allowing to reduce biases and to retain high inference speeds. By reading the paper on PatchCore, we find that the networks from which the best results can be obtained are from the WideResNet class of architecture. They provide an architecture of four layers which are extracted from the middle ones. The second one is the **coreset-reduced patch-features memory-bank**, allowing us to evaluate the images during the testing phase, by comparing them with the features extracted and saved from the nominal ones. It also increases inference time by reducing the number of data saved.

Multiple datasets have been used, but as the paper introduces, the aim is to develop an effective approach to resolving anomaly detection problems in an industrial field. One of the recent and most complete dataset related to this subject had been proposed in [2] and called **MVTecAD** and the results provided by the usage of PatchCore on it were excellent. This dataset is composed of 15 different classes of images, each one divided in multiple levels of degradation from *good* to *broken_large*.

2. Related Works

The development of PatchCore and its components have been mainly inspired by two other approaches for anomaly detection on images : **SPADE** proposed in [3] and **PaDiM** proposed in [4]. Each one of them has characteristics from which PatchCore got inspired. SPADE has been developed with the use of

pretrained networks which provide state-of-the-art performances without any adaptation to the data, and add to this usage memory banks of nominal features and kNN for both anomaly detection and segmentation. As the previous approach, PaDiM also uses pretrained networks but in this case they are locally constrained features allowing the estimation of the patch-level specifications used in the approach.

PatchCore is mainly inspired from SPADE and PaDiM : memory banks from SPADE are reused as they permit to extract nominal features from the pretrained network, however to achieve higher performances, neighborhood-aware of the levels of the patch are also saved. To ensure low inference cost the memory bank is then corset-subsampled. The patch-level approach of PatchCore is mainly similar to the PaDiM one for the anomaly detection and anomaly segmentation.

3. Method

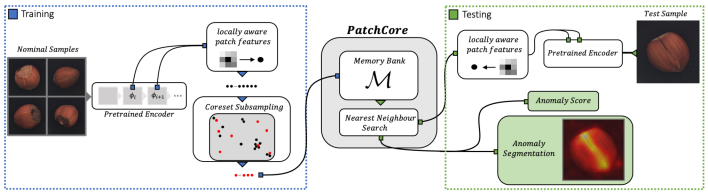


Figure 1. PatchCore Design

PatchCore implementation is done in two main phases: the training and the testing, with both of them being divided in a similar way. Effectively, the patch-features retrieved from the testing phase will allow determination, by applying a kNN on the patch-features memory-bank, if the image is abnormal or not. The following parts explain the implementation of PatchCore in our project.

3.1 Training - Fit function

As we already mentioned, the training phase works as a cold-start problem. In this phase, nominal images (without-defaults pictures) of the industrial dataset are used to train on. This dataset has been created with defective and non-defective images. We use non-defectives ones, stored in the *good* folder of each dataset category for training. In this part we also need a pretrained network on ImageNet available to extract features. According to the original paper the networks giving the best results are WideResNet type networks. In our implementation we use the WideResNet50 architecture to implement a PatchCore version.

The main function designed for the training phase is the ***fit*** function that is composed of several actions : features-patches extraction from the "good" images, preprocessing on them and finally definition of the memory-bank throughout a **coreset** subsampling.

First of all, feature representations are extracted from the spatial resolution blocks of the pretrained network. As we are looking for mid-level and intermediate patch-level features, and in concordance with the paper, the layers extracted are **the outputs of the second and the third blocks** : it allows to avoid features too generic or too biased, situated respectively in the beginning and the end of the networks. Then the memory banks of the image corresponding to each patch are collected by the call of ***patchify()*** function.

This approach is based on the unfolding technique. It consists of applying a filter with a determined kernel size (3 in our case) on every channel of the targeted block of the tensor. After application, it aligns all the pixels of this block in the same column. That is why we get a column of size kernel_size * kernel_size * nb_channels. In the final output, each column will match with one block or in other word: one patch-feature.

As we want to compare those features, we need to have them in the same scale. That is why we also apply an interpolation in order to have all the intermediate layers with the same resolution as the lowest one. So, we scale the third layer features to get the same resolution as the second one.

The spatial resolution and generality of the features is an important point of the patchcore algorithm. To avoid deviation and loss of resolution, we apply the equations 1 to 3 of the reference paper in order to get all features on the same dimensionality.

Finally the memory bank is reduced with a hyper parameter determining the percentage of features to be kept. This reduction is done with a **greedy coreset subsampling.** As mentioned in the SPADE article, the size of the memory can grow quickly and with it the inference time will also grow during the testing phase. In order to reduce the computation time, we apply the Johnson-Lindenstrauss theorem on the memory bank. Then, using a minimax algorithm on the distances between the different distances between all the features, we select the indices we want for the final memory bank. This kind of algorithm provides a more representative spatial selection.

3.2 Testing - Predict function

Patchcore aims to give a score to an image according to its anomaly level. It manages to do it by comparing the features of the tested image and the memory bank built in the previous section. Therefore, it extracts all patches in a similar way as seen previously. This approach consists in calculating many distances between patches in which we look for the maximal score value (***maximum_distance_score()***). It relies on a k-nearest neighbor approach.

Among all those distances, the algorithm needs to highlight the one with the minimal distance. It identifies the patches considered as nearest neighbors. Then, we extract inside it the maximal score of the top k nearest neighbors. This logic is based on the sixth equation from the reference paper[1].

$$m^{test,*}, m^{*} = argmax\ argmin\ ||m^{test} - m^{*}||_2$$

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

As we have different levels of anomaly score, we need to normalize the anomaly score for each anomaly and avoid some bias towards the rarity of some anomalies. We built the equation 7, which is a softmax normalization. (***increasing_score()***)

$$s = (1 - \frac{exp||m^{test} - m^{*}||_2}{\sum_{m \in N_b(m^*)} exp||m^{test} - m^{*}||_2}) \times s^{*}$$

For the segmentation representation, it provides an interpolation on the minimal distance patch-feature. After rescaling this patch, we can apply a gaussian filter (radius = 4) on it in order to highlight the region as an anomalous region.

3.3 Contrastive Language-Image Pre training (CLIP)

After implementing a functional version of Patchcore, we decided to try an extension of another pre-training approach, the *Contrastive Language-Image Pre-training (CLIP)* ***[Ref].*** Instead of training on a specific label category, this encoder was pre-trained using the image description over a wide range of images found on the internet. Using NLP supervision on the text and Image encoder as ResNet or Vision Transformer (ViT), this offers a more generalizable and flexible approach for object visualization. Training on a wide variety of images with a wide variety of texts gives us better results for image recognition on images representing specific situations.

In our case, we trained our model on the several ResNet models offered by CLIP. Those models were modified from the original ones to improve efficiency and performance. To fit our implementation, it is needed to hard-code the forward function of the encoder instead of the already made feature extractor of the *timm* library. In order to do it, we will reuse the feature extractor from the reference paper. As we trained our model with mid-level features, we chose the layer 2 and 3 as in the previous implementation. At the end of each one, we added an activation hook. This kind of object registers the layer activation at each time the forward function is applied on the layer.

Finally, we run the model using the same methodology as seen in the previous sections.

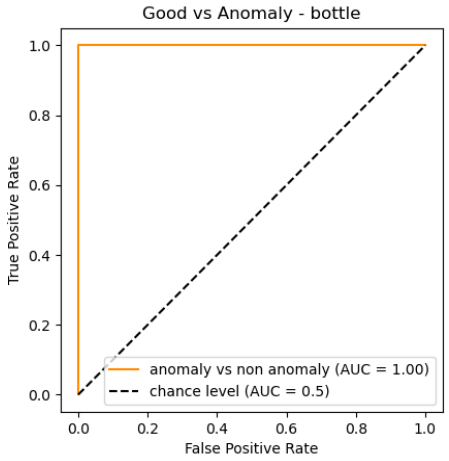
4. Experiments

In this part, we will present different results from the experiments we have made, either by changing the hyper parameters of the program or by comparing the results obtained between the implementation of PatchCore using WideResNet or extending it via CLIP. The main assessment factor considered in the PatchCore article was the AUROC score (the Area Under the Receiver Operator Curve). A ROC curve shows the trade-off between true positive rate (TPR) and false positive rate (FPR). The score could be between 0.5 and 1.0 where this-last one is the maximum score.

4.1 Original PatchCore

This part presents the results obtained by the re-implementation of the original work presented in the paper for the anomaly detection and segmentation. The best result proposed in the original work was a mean of **99.1 anomaly AUROC detection score out of 100** with a 25% utilization of the memory bank on the MVTecAD dataset. The best result proposed in our work is a mean of **98.1** score for the same memory-bank utilization. Even if our outcomes are a bit lower than the ones from the original approach, they are still at a very high level and we can consider that the detection of an error is most of the time done. But AUROC results depend on the category of images. For example, on figure 2a and 2b, a gap can be observed between *bottle* and *cable* categories : the first one shows a 100% AUROC detection score while the

second one only shows a 93% AUROC detection score. This can be explained by the fact that on cable images there are a lot more different patterns and they seem less homogeneous than the one from other classes like bottle.



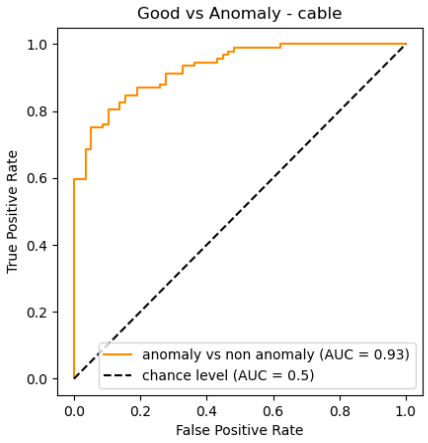


Figure 2a&2b. Example of ROC-AUC graph

The table 1 shows the comparison between the coreset sampling percentage value of the memory bank. We tried to confirm the results of the reference paper. We observe better results for 25% of the memory bank. According to the paper, it is the best value for the hyperparameter. Running the application for the entire memory bank leads to a very big computational time.

Percentage	**10%**	**25%**	**100%**
Bottle	1.00	1.00	1.00
Cable	0,93	0,95	0.94
Capsule	0,98	0,98	0.98
Carpet	0,99	0,98	0.98
Wood	0,99	0,99	0.99
Zipper	0,99	0,99	0.99
Toothbrush	1.00	1.00	1.00
Metal Nut	0,99	0,99	0.99
Screw	0,97	0,98	0.98
Pill	0,92	0,94	0.93
Transistor	0,98	0,98	0.97
Tile	0,99	0,99	0.99
Leather	1.00	0,99	1.00

Table 1: Comparison between 10% , 25%, 100% of the memory bank

The segmentation of the different anomalies has also been done in our implementation. The following figure shows the ability of the program to detect either big or tiny scratches (respectively the second and third horizontal part of the figure) and on several object types.

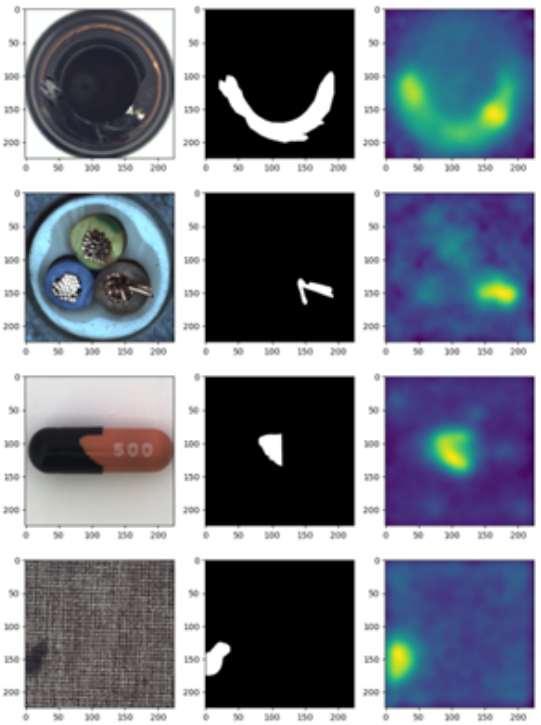


Figure 3. Implementation anomaly segmentation and heatmap

We also trained our algorithm trying different Hyperparameters. First we wanted to try different, k-nearest neighbors values for k in [2,3,4,5] (Figure 4). We calculate the average score of 14 MVTEC Dataset on many different k values. We could see that the best value would probably be between 3 and 4.

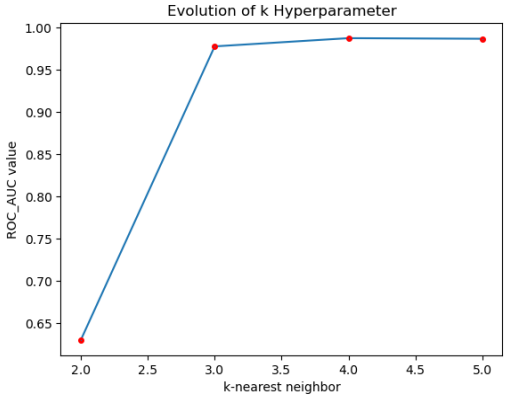


Figure 4. Evolution according to k-nearest neighbors

4.2 CLIP Extension

As we mentioned we try to implement an extension to the patchcore approach by using a more powerful pre-trained network which is CLIP. CLIP offers 5 different models compared to the ResNet One. We didn't work with the Visual Transformer one as it already shows worse performance than ResNet. That is why we used the RN50, RN101, RN50x4, RN50x16 and RN50x64 neural networks to analyze the results. Those networks are based on the natural ResNet Network with some improvement. We obtain the following result table for 6 datasets.

	RN 50	RN 101	RN 50x4	RN 50x16	RN 50x64
cable	0.89	0.86	0.82	0.92	0.92
capsule	0.97	0.99	0.99	0.97	0.98
bottle	1.00	1.00	1.00	1.00	1.00
carpet	0.99	0.97	1.00	0.99	0.99
wood	0.99	0.99	0.98	0.99	0.99
zipper	0.97	0.96	0.97	0.96	0.98

Table 2: Comparison between different networks of CLIP

We observe that sometimes CLIP networks lead with bad results for the MVTEC Datasets. It shows less stability than ResNet. For those reasons, we cannot say that CLIP gets better results for all situations.

5. Discussion

Patchcore shows incredible results compared to the other anomaly recognition algorithm, working very well with many types of objects. Our reimplementation results are very similar to the ones from the original article. Maybe we could have trained on more hyperparameters, for example more k-nearest neighbors, to see when the performance would begin to decrease. However, because of the computational time it was hard to compute all the desired comparisons.

The segmentation implemented by us shows some difficulties to detect anomalies for some datasets, such as for capsules. We wanted to try thinking about other segmentation filters that maybe could show other results, but ended up lacking the time to do so.

Regarding the CLIP extension, we didn't manage to get any real improvement. Our thoughts are that CLIP provides well pre-trained networks that offer a better generalization to classify an object, and that maybe we should rethink the training dataset. As it provides only good images, the CLIP networks will tend to identify pieces with little defects as a good piece instead of detecting the anomaly, generalizing the image identification. In our opinion, the task here is not the best one to use for a CLIP pre-trained network. Maybe learning what is an anomaly in the train set instead of what is a good piece would increase the performance of our algorithm.

The authors of PatchCore define as some limitations the "transferability of the pretrained features leveraged" (*p. 8, pt. 5*) for applicability.

6. Conclusion

The PatchCore implementation is an effective way to resolve the problem of it has been designed first for : the high mean overall anomaly score detection on datasets such as MVTecAD establish that PatchCore is an adapted solution for an application in an industrial environment for several reasons. The way it was conceived as a cold-start problem provides a more easy solution because only nominal images and data are used. Moreover, the inspirations from SPADE and PaDiM have allowed PatchCore to go further in multiple domains : due to the memory-bank reduction, the inference time is reduced and the efficiency is

increased. The way they have been designed to use pre-trained networks helps a new user to not have to develop a whole new one, specific for the task of anomaly detection and segmentation.

In conclusion, we can say that PatchCore is well designed and efficient for its work of Anomaly detection and segmentation, performing at a level no other architecture has done before. The official Github repository helps the implementation for the same task on other items.

7. References

[1] “Towards Total Recall in Industrial Anomaly Detection”, Roth and al.

[2] “MVTecAD - a comprehensive real-world dataset for unsupervised anomaly detection”, Bergmann and al.

[3] “Sub-image anomaly detection with deep pyramid correspondences”, Cohen and al.

[4] “Pattern Recognition”, Defard and al.