Chapter 5

LeMO: Online anomaly detection

5.1 Introduction to online anomaly detection

Online anomaly detection is a real-time, continuous process which identifies anomalous data points or patterns within a constantly streaming data set. It is often used in dynamic environments where data changes rapidly and constantly, such as in network security or industrial production lines. The term ”online” refers to the fact that the data is analyzed as it arrives, rather than being collected and stored for later offline analysis.

In online anomaly detection, the model is updated with each new data point, allowing it to learn and adapt to new patterns in the data as they emerge. This is in contrast to offline anomaly detection, where a model is trained on a static set of data and is not updated as new data comes in.

Anomaly detection in online environments is a challenging yet crucial task. Traditional methods for anomaly detection often rely on offline learning techniques, where a high-quality dataset of nominal samples is first constructed, then a model is trained on this dataset to understand and predict the typical pattern of normal data. These methods, although effective, encounter a few problems when deployed in real-world, dynamic environments such as industrial production lines.

The issues primarily stem from the static and offline nature of these traditional algorithms, while the data they are expected to process is dynamic and online. Firstly, offline methods require extensive collection and storage of normal samples, a process that can be resource-intensive both in terms of time and memory. Moreover, it is challenging to ascertain the sufficient size of the normal samples dataset needed for optimal performance.

Secondly, these algorithms often employ complex model structures with numerous parameters, leading to longer training times and significant computational resource consumption, which may be impractical in industrial contexts.

Lastly, offline algorithms can struggle to adapt to changing data patterns over time, such as image feature drift caused by factors like noise and equipment instability, resulting in decreased performance.

To address these challenges, a new approach to anomaly detection is proposed: an online learning image anomaly detection algorithm. Instead of collecting samples in advance, data is input as a stream, and the model conducts swift and lightweight online training for detection. This approach circumvents the need for extensive sample collection and storage, thus saving significant memory and time. It also typically results in a more lightweight model, reducing training times and the overall algorithm deployment cycle.

The model that was decided to implement to test the actual efficiency of this approach is called LeMO[14], released on 25 May 2023. The code of this model has not been released, so a scratch implementation is proposed within this thesis work.

Finally, we will compare the official and unofficial results to verify the actual efficiency and practicability of such an approach.

5.2 The method

The LeMO model in its operation consists of three main parts: an encoder, a local patch adapter and a memory bank. For the LeMO architecture overview see figure 5.1.

In a training pipeline, the image first passes through the encoder. This is nothing more than a backbone. Image features from different layers are extracted from it. Then the extracted features pass through a local patch adapter, in which they are concatenated together, interpolating them to the size of the largest feature map. Then position encoding, in the form of an x-coordinate map and a y-coordinate

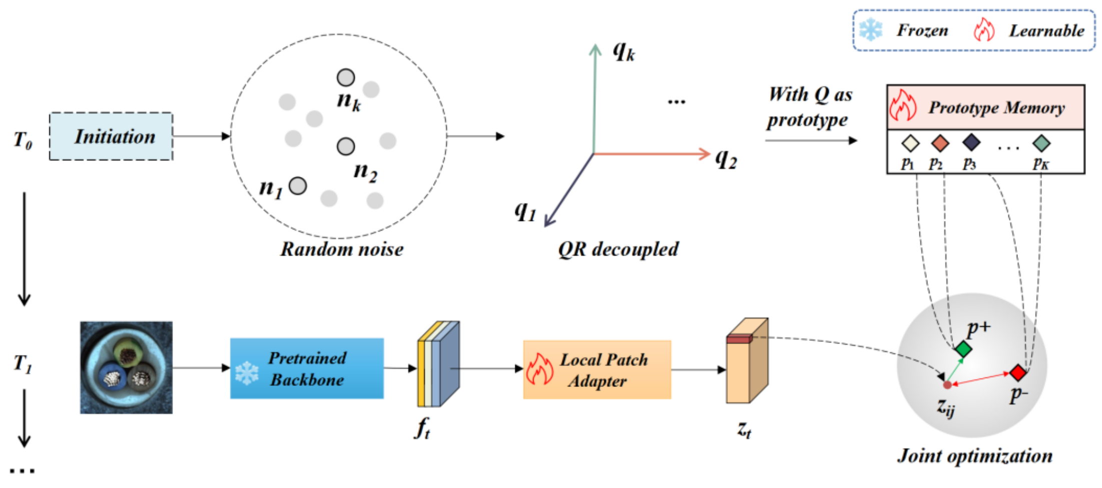

Figure 5.1: Overview of LeMO architecture

map, is appended to obtain location-aware features. Such features are then processed through a learnable 1×1 convolution layer to yield the final enhanced features.

The enhanced features thus obtained are then used to update the prototype memory bank. This is a memory bank of a predefined number of vectors and the same size as the features.

During training, we optimise a contrastive anomaly detection loss function, where each feature is drawn towards relevant prototypes and repelled from unrelated ones. In the meanwhile the memory bank is updated aswell.

5.2.1 The encoder

The first part of the network is a backbone, which takes the input image and outputs the embedding features in a similar way to how the PatchCore model does it. In our case, the backbone consists of a WideResNet50[47], which is a variant of ResNet[36] that has 50 layers of depth, meaning that it consists of a series of 50 residual blocks. In short, a residual block is a structure that allows a neural network to learn the residual errors between desired and actual predictions. This approach simplifies network optimisation and addresses the problem of gradient vanishing, a common problem in deep neural networks. In particular in the model under consideration we use a WideResNet pretrained on an image classification task on the ImageNet dataset[18], which consists of about 1.2 million images labelled on more than 1000 categories.

The encoder output consists of the outputs of the 3 deepest layers of the backbone, all interpolated to the dimension of the most superficial layer (i.e. the one with the largest dimension) and concatenated together to obtain an output tensor $f_t \in \mathbb{R}^{H \times W \times D}$ where H and W are the size of the shallowest layer from those extracted from the backbone and D the size of the embedding features after concatenation on the different layers $D = D_1 + D_2 + D_3$ where D_i is the size of the i-extimum layer.

Since each pixel location of f_t has a predetermined receptive field, patch feature $p_{t \in \{1,\ldots,HW\}} \in \mathbb{R}^D$ can be considered as semantic information at the pixel location. Next, p is input to the Local Patch Adapter.

5.2.2 Prototypes Memory bank

The LeMO memory bank is very different from the previous methods such as PatchCore. In fact the main issue with the type of memory banks used in previous methods is that it tends to increase in proportion of the size of the target dataset $|\mathcal{X}|$. This fact leads to different problems concerning the space required to actually store the data insisde the computer memory, and the time complexity required to compare a test feature with all the features stored. So, the performance of anomaly localization depends on the size of the memory bank. Conventional methods store as many normal features of the target dataset as possible in a memory bank to accommodate unfitted features, that is, to understand the distribution of normal features. Therefore, the size of the memory bank was determined in proportion to that of the target dataset. Therefore, a great number of unfitted features in the memory bank may cause the risk of overestimated normality of abnormal features. Furthermore, a large capacity memory bank increases the inference time. Moreover traditional methods, like coreset, k-means, and patch distribution modeling, require a complete repository of normal data for building up a memory bank, which poses challenges in determining how many normal samples are sufficient or how to collect enough data efficiently. To address these issues, the LeMO paper[14] suggests using basis vectors as an alternative technique to identify optimal replacements without the need for data pre-collection where each basis vector serves as a prototype for local fine-grained normal patterns.

Specifically the "prototype memory bank" $\mathcal{M}$ is inizitialized as a set of K basis vectors $\mathcal{M} = \{m_1, m_2, ..., m_K\}$, where K is an arbitrary number, so there will be no need for nominal data for initialisation either.

In particular, the proposal of the LeMO model is to initialise memory using orthogonal random noises, effectively simulating the feature deconstruction process. The orthogonalization step ensures the decoupling of initialization vectors, guiding memory bank updates and avoiding local optimal solution. In this work the QR decomposition has been used, an algorithm that decomposes a matrix A into a matrix Q with standard orthogonal column vectors and an upper triangular matrix R. Q is denoted as the initial orthogonal prototypes memory bank $\mathcal{M}$, which makes a good foundation for the subsequent learnable optimization.

Analysis of LeMO's memory bank compared to previous models Let's consider $|\mathcal{X}|$ as the dataset scale and γ as the comparession ratio, and H, W, D as the dimension of the feature z inside the memory bank. As can be seen in the table 5.1, the LeMo memory is the only one which does not depend on $\mathcal{X}$, so it is the most lightweight among the SOTA models considered.

Table 5.1: Analysis of memory bank modeling complexity and memory bank size.

Methods	SPADE[8]	PaDiM[9]	PathCore[39]	CFA[21]	LeMO[14]
Modeling	$O(\vert\mathcal{X}\vert HWD)$	$O(\vert\mathcal{X}\vert HWD^2)$	$O(\vert\mathcal{X}\vert HWD)$	$O(HWD)$	$O(KD)$
Memory bank	$R\vert\mathcal{X}\vert \times H \times W \times D$	$RH \times W \times D^2$	$R\vert\mathcal{X}\vert \times \gamma(H \times W) \times D$	$R\gamma(H \times W \times D)$	$K \times D$

5.2.3 Local Patch Adapter

The local patch adapter $\phi(\cdot) : \mathbb{R}^D \rightarrow \mathbb{R}^{D'}$ is an auxiliary network with learnable parameters, which converts each p_t into target-oriented features $z = \phi(p_t) \in \mathbb{R}^{D'}$ following the hypersphere concept, which consists in defining a boundary around the normal data points, and any data point that fall outside this boundary is considered an anomaly. In LeMO the hypersphere concept is inspired by the so called Coupled-hypersphere-based Feature Adaptation (CFA[21]).

In order to perform a procedure that use the hypersphere concept, the first step is to build a function $\phi(\cdot)$ so that it has a high density in multiple cluster centers, i.e. multimodal, and consequently the further the points fall from the centres of the clusters, the more they are defined as anomalous. These cluster centres are the prototype memory described in the previous paragraph and CFA supervises $\phi(\cdot)$ so that p_t is embedded close to the nearest $m' \in \mathcal{M}$: $m' = \arg\min_{m \in \mathcal{M}} \|m - p_t\|_2$. Specifically, $\phi(\cdot)$ makes possible to form a high concetration between normal features by supervising p_t to embed them inside a hypersphere of radius r created with m' as the center.

The structure that allows to make this beheviour possible is the loss function.

In the LeMO architecture the Local Patch adapter is implemented as following: first the features $f_t \in \mathbb{R}^{H \times W \times D}$ extracted from the backbone are concatenated with a positional encoding $\mu \in \mathbb{R}^{H \times W \times 2}$ in the form of an x-coordinate map and a y-coordinate map which contain the normalized distances fom the center of f_t. In this way the new created features will be location-aware and have the shape of $(H \times W \times D + 2)$. After that, such features are processed through a learnable 1×1 convolution layer to yield the final enhanced features $F_t \in \mathbb{R}^{H \times W \times D'}$.

5.2.4 The LeMO loss function

The situation presented in the previous section sees on one hand a memory bank of K prototypes $\mathcal{M} = \{m_1, m_2, ..., m_K\}$, and on the other hand, for each iteration, an embedding vector $F_t = \phi(f_t) \in \mathbb{R}^{H \times W \times D'}$. The goal is to have a loss function capable of clustering each feature vector $z \in \mathbb{R}^{D'}$ around prototypes in the memory bank that will act as cluster centres. This process can be classified as contrastive learning or metric learning task.

Constrastive learning Contrastive learning is a pivotal technique in the realm of machine learning and deep neural networks. At its core, this approach centers on the concept of instance discrimination, with the primary objective of training a model to distinguish between data points that belong together and those that do not. While in supervised learning, this is straightforward, as labels can be used to identify data points belonging to the same class, the true power of contrastive learning shines in unsupervised or self-supervised scenarios, where no label information is available. The crux of contrastive loss functions is to measure the similarity or distance between embedding pairs, emphasizing high similarity for positive pairs and low similarity for negative pairs. This encourages the model to discern meaningful differences and similarities, enabling it to learn rich and transferable representations of the data.

For example, in the case considered in this work, for each $z \in \mathbb{R}^{D'}$ the prototypes $m^+ \in \mathcal{M}$ with the shortest distance can be considered as positive examples, and the prototypes $m^- \in \mathcal{M}$ with the largest distance can be considered as negative examples, given distance function $\mathcal{D}$.

In the next sections of this work we will analyse both the loss proposed by the reference paper and new losses, comparing the results with each other.

LeMO loss

The loss used by the LeMO paper [14], called AnoNCE, is a contrastive over-cluster anomaly detection loss based on Noise Contrastive Estimation (NCE)[13]. NCE is a powerful deep learning method that helps us approximate the probability distribution of a given dataset by contrasting it with a noise distribution. NCE simplifies the unsupervised learning task of probability distribution estimation into a probabilistic binary classification problem. In essence, it involves training a logistic regression classifier to distinguish between data distribution samples and noise distribution samples by comparing the likelihood ratio of a sample under the model and the noise distribution.

The loss function proposed by the paper is inspired by this technique and can be summarised by the following formula:

$$L_{z_{ij},\{m^+\},\{m^-\}} = -\log \frac{\sum_{m^+} \exp(sim(z_{ij}, m^+/\tau))}{\sum_{m^+} \exp(sim(z_{ij}, m^+)/\tau) + \sum_{m^-} \exp(sim(z_{ij}, m^-)/\tau)}$$

Where $z_{ij} \in \mathbb{R}^{D'}$ is a single feature extracted, with $i \in \{0, 1, ..., W\}$ and $j \in \{0, 1, ..., H\}$,then $\{m^+\} = \{m_1, ..., m_q\}$ is the set of q similar prototypes and $\{m^+\} = \{m_1, ..., m_w\}$ is the set of w dissimilar prototypes. The way the prototypes have been chosen as similar or dissimilar is based on the Euclidean distance. Moreover the τ

parameter is the temperature, which the sharper it is, the more the model should pay attention to hard negative samples. Finally the *sim* function is a measure of similarity, which has been implemented as following:

$$sim(z, m) = \max(\mathcal{D}(p, z) - r, 0)$$

where $\mathcal{D}$ is the Euclidean distance function and r is a paramter which relaxes the similarity constraint by a margin. This ensures an enhanced robustness of the algorithm.

Feature Enhanced Memory Update

During training, the 1×1 convolution parameters are not the only one updating, but the prototype memory also goes through an upgrade. The idea is on one hand to update the mapping of the features $\phi(\cdot)$ in such a way that they form clusters in the target space, and on the other hand to update the vectors in the prototype memory (i.e. the centres of these clusters) in order to obtain a more stable result.

A possible technique to achieve this is to consider the vectors within the prototypes as parameters of the LeMO network itself, so that they are updated together with the weights of the convolutional network in the Local Patch Adapter. This tecnique will take the name of ”memory parameter update”.

Although this tecnique already leads to excellent results, there is still the possibility of degenerate solutions: ideally, the features mapped for each image should be evenly distributed over all prototypes, forming stable hypersphere. Unfortunately, however, by adopting the previously described solution, there is a risk that some clusters will be associated with more feature vectors than others, or even some clusters may be empty. To avoid this scenario, the LeMO paper proposes the use of a ”Feature Enhanced Memory Update”, or ”K-Means Memory Update”.

Feature Enhanced Memory Update To overcome the problem of collapsing clusters, LeMO propose to process and eliminate extremely small clusters in advance before they collapse.

Denoting normal clusters as $\mathcal{C}_n$ whose sizes are larger than a threshold, and small clusters as $\mathcal{C}_s$ whose sizes are not, for $c \in \mathcal{C}_s$, we first assign samples in c to the nearest centroids in $\mathcal{C}_n$ to make c empty. Next, we split the largest cluster $c_{max} \in \mathcal{C}_n$ into two sub-clusters by K-Means and randomly choose one of the sub-clusters as the new c. We repeat the process until all clusters belong to $\mathcal{C}_n$. Though this process alters some clusters abruptly, it only affects a small portion of samples which are involved in this process. The pseudocode of the algorithm is shown in Algorithm 2.

Algorithm 2: Feature Enhanced Memory Bank Update Strategy

```
1  Input: Image $x$
2  Result: Updated $z$ and $M$
3  Initialize the prototype memory bank $M = [m_1, \ldots, m_K]$ with decoupled
     random noise;
4  foreach $x \in onlinestreamingdata$ do
5      Extract the image features $z = \text{PatchAdapter}(\text{Backbone}(x))$;
6      Optimize $z$ via AnoNCE loss // Step-1: Update features $z$;
7      // Step-2: Update prototypes $P$ start;
8      foreach $z_{ij} \in z$ do
9          Assign $z_{ij}$ into the closest $m_k \in M$;
10     while $C_s$ is not empty do
11         for $c \in C_s$
12             Assign all features of $c$ to the nearest centroid in $C_n$ to
   make $c$ empty;
13             Delete $c$;
14             Apply k-means to $c_max$ operation to reformulate two new
   groups;
```

Configurations

The performance of the model was tested and compared to the official model reported in the paper with 3 different configurations. All configurations share common settings, which are as follows:

- Backbone: WideResNet 50
- Layers of extraction: [layer1,layer,2,layer3]
- Number of prototypes: 10
- Number of positive samples m^+: 3
- Number of negative samples m^-: 3

The configurations used differ in the type of memory update:

1. **Configuration 1**: Memory Parameter Update
2. **Configuration 2**: Feature Enhanced Memory Update
3. **Configuration 3**: No memory update

5.2.5 Results on the official version

As can be seen from the comparison of the results obtained from the implemented model and the official model (see table 5.2 5.3 and 5.4), there are considerable differences in performance.

In fact, the implementation of the LeMO model yields an I-AUROC of 0.668 in the best case, as opposed to the officially reported 0.972, a not insignificant difference, which forces us not to be able to use the model for any purpose due to its ineffectiveness. Moreover, configuration 2, which uses the Feature Enhanced Memory Update, reports not only the worst results, but even an I-AUROC of 0.497, i.e. a lower performance than a dummy classifier (0.5).

The discrepancies highlighted between the results obtained from our model and those reported in the LeMO official paper [14] raise doubts about the integrity and accuracy of the descriptions provided in the reference paper. It is possible that crucial parts of the model have been underestimated, poorly described or even omitted. This underlines the risk involved in implementing approaches based on papers without official validation or code publicy available.

However, in spite of the discrepancies, taking the official results as a starting point, we are going to explore alternative solutions to improve the performance of the model. One of the next directions could focus on changing the type of loss function used, exploring options that could significantly improve performance.

CONFIG	I-AUROC
LeMO Original	**0.972**
Configuration 1	0.668
Configuration 2	0.497
Configuration 3	0.666

Table 5.2: I-AUROC for different configuration settings vs. the official implementation. The table shows the difference in performance of the official model versus those implemented by reading the paper. Note also that configuration 2, i.e. the one for the enhanced update feature, shows even lower performance. This could be due to an incorrect implementation of the loss function, which exhibits the opposite behaviour to that of the kmeans operation. This behaviour is then reflected in the other metrics P-AUROC and PRO-AUROC shown in the following tables.

CONFIG	P-AUROC
LeMO Original	**0.976**
Configuration 1	0.782
Configuration 2	0.648
Configuration 3	0.784

Table 5.3: P-AUROC for different configuration settings.

CONFIG	I-AUROC
LeMO Original	**0.917**
Configuration 1	0.713
Configuration 2	0.521
Configuration 3	0.717

Table 5.4: PRO-AUROC for different configuration settings.

5.2.6 Analysis of new losses

The results obtained so far show that the architecture described and implemented does not correspond to the results presented in the reference paper. Despite the soundness of the theoretical basis of the architecture, a review of the existing literature confirms that it should perform adequately.

Therefore, the hypothesis for improving the performance of the model explored in this thesis work focus on adopting a different approach to the loss function. In particular, we will explore different types of loss functions, starting with the conventional ones for contrastive learning and moving on to examine experimental or less conventional losses.

Triplet central loss

Triplet central loss[19] is a technique widely used in contrastive learning to improve the separability of objects or data representations. It is based on triplet loss, which is commonly used in feature learning for face and object recognition problems. Triplet central loss introduces an extra component, the central loss, to improve learning efficiency.

Triplet loss[56] was introduced as a solution for embedding learning that maintains a margin between similar and dissimilar image representations. This margin helps to ensure that the representations are well separable. However, the standard triplet loss can be computationally expensive and may require a considerable amount of training data.

Triplet central loss was proposed to address some of the limitations of standard triplet loss. Introduced [19] in 2014, this loss adds a centroid for each object class in the embedding space and pushes the data representations to cluster around these centroids, improving class separability.

In the triplet centroid loss, we have the following elements:

1. An embedding space, in which we represent data.
2. Centroids for each class of object.
3. A set of sample triplets, each of which contains an 'anchor' (an example to be classified), a 'positive' (an example of the same class as the anchor) and a 'negative' (an example of a different class).

The triplet centre loss minimises the following objective function:

$$L_{\text{triplet}} = \sum \left[\|f(\text{anchor}) - f(\text{positive})\|^2 - \|f(\text{anchor}) - f(\text{negative})\|^2 + \alpha \right] \tag{5.1}$$

Where:

- $f(x)$ represents the embedding function for the example x.
- $\|\|$ denotes the Euclidean distance between the vectors.
- α is a positive margin term that controls the desired separation between triplet representations.

To implement the triplet central loss, a term is added that penalises the distance between the anchor and the centroid of the anchor class. This term is given by:

$$L_{\text{center}} = \sum \|f(\text{anchor}) - c(\text{anchor\_label})\|^2 \tag{5.2}$$

Where $c(\text{anchor\_label})$ represents the centroid of the anchor class.

The total loss function is given by the combination of the triplet loss and the central loss:

$$L_{\text{total}} = L_{\text{triplet}} + \lambda L_{\text{center}} \tag{5.3}$$

Where λ is a coefficient that balances the importance of triplet loss and centre loss.

Triplet central loss improves contrastive learning by pushing representations to cluster around class centroids, thus increasing the separability of classes in the embedding space. This can lead to more discriminative representations and better classification performance.

In the case treated by this work, the embedding of such a loss sees as an anchor the feature z_{ij}, as a positive example the nearest in-memory feature $m^+ \in \mathcal{M}$ and as a negative example the most distant in-memory feature $m^- \in \mathcal{M}$. So the final triplet central loss formula implemented will be as follows:

$$L_{z_{ij},\{m^+\},\{m^-\}} = \|z_{ij} - m^+\|_2 - \|z_{ij} - m^-\|_2 + \alpha + \lambda\|z_{ij} - m^+\|_2$$

Where λ is set to 0.1 and α to 1, without having been optimised.

NCE loss

As previously mentioned in section 5.2.4, LeMO loss draws its inspiration from the Noise-Contrastive Estimation (NCE)[13] technique. In fact, LeMO loss is sometimes referred to as AnoNCE loss in the paper. Therefore, in this section we will examine the NCE technique in depth in order to develop a new working loss function inspired by this methodology.

Noise-Contrastive Estimation is a method used for training and learning the parameters of probabilistic models, particularly in the context of language modeling and word embeddings. It was introduced by A. Mnih and Y. Teh in their paper [13] in 2012.

The primary purpose of NCE is to estimate and optimize the parameters of a neural network, such as a language model, without having to compute the full likelihood of the data. This is particularly useful when dealing with large datasets and complex probabilistic models.

The key idea behind NCE is to transform the estimation of the model's parameters into a binary classification problem. Instead of directly estimating the probability distribution over a large vocabulary, NCE turns it into a problem of distinguishing between the actual data and "noise" samples. Noise samples are typically drawn from a predefined noise distribution, like a uniform distribution over the vocabulary.

In the context of training word embeddings, for example, NCE can be used to learn word vectors w by training a model to distinguish between real word-context pairs (positive examples) and randomly sampled word-context pairs (negative examples). This makes the training process computationally efficient and scalable.

So, as was just described, let's analyze the NCE loss function example in the context of training word embeddings in an NLP task:

$$\mathcal{L}_{NCE}(w^+, \{w^-\}, c, \Delta s_\theta) = \log\left(\delta\left(\Delta s_\theta(w^+, c)\right)\right) + \sum_{i=0}^{k} \log\left(\delta\left(-\Delta s_\theta(w_i^-, c)\right)\right)$$

In the formula, Δs_θ is a scoring function for the model θ of the similarity between a word w and its context c and $\delta(\cdot)$ is a function which trasforms the input score into a probability, which could be for example a softmax or a sigmoid function.

As the NCE paper suggests and as can be seen from the formula, it is advisable to consider for each positive pair of examples (w^+, c), more than one negatively correlated word (w^-, c) in order to have more stable solutions.

If we want to adapt the NCE loss to the problem of anomaly detection that is treated in this work and reformulate a new version of AnoNCE loss, we could consider the context c as the feature z_{ij}, the positive word embedding w^+ as the closest feature vector of the prototype memory $m^+ \in \mathcal{M}$ and the k negative word emebddings as the k feature vectors in the memory more far after m^+. Finnaly the Δs_θ scoring function could be rapresented by the Euclidean distance $\|\|_2$, as we used in the others losses. So in summuary the final loss function AnoNCE$_2$ will be:

$$\mathcal{L}_{\text{AnoNCE}_2}(m^+, \{m^-\}, z_{ij}) = \log\left(\delta\left(\|m^+, z_{ij}\|_2\right)\right) + \sum_{i=0}^{k} \log\left(\delta\left(-\|m_i^-, z_{ij}\|_2\right)\right)$$

CFA loss

Of all the papers from which LeMO draws inspiration for its method, CFA[21]'s is certainly one of the most important due to its similar architecture. It proposes within its work a very interesting loss function $\mathcal{L}_{CFA}$, completely compatible with our problem. The loss $\mathcal{L}_{CFA}$, similar to the NCE, it consists of two components, an attractive part $\mathcal{L}_{att}$ and a repulsive part $\mathcal{L}_{rep}$:

$$\mathcal{L}_{CFA} = \mathcal{L}_{att} + \mathcal{L}_{rep}$$

The main objective of the attractive component of the loss is to gradually bring the emebddings z_{ij} closer to the hypersphere of radius r centred in $m^+ \in \mathcal{M}$. More specifically:

$$\mathcal{L}_{att, z_{ij}} = \frac{1}{U} \sum_{u=1}^{U} \max\{0, \mathcal{D}(z_{ij}, m_u^+) - r^2\}$$

Where $\mathcal{D}$ is a predefined distance function such as the Euclidean distance used for the experiments in this work and the hyper-parameter U is the number of nearest neighbours matching z_{ij}

However, the ambiguous z_{ij} belonging to multiple hyperspheres at the same time still leaves room for the normality of abnormal features to be overestimated. To address this, hard negative features are additionaly used to perform contrastive supervision to obtain a more discriminative $\phi(z_{ij})$. Hard negative features are defined as the $K + j$-th nearest neighbor m_j of z_{ij} matched through NN search with $\mathcal{M}$. Thus, we define L_{rep} that supervises $\phi(\cdot)$ contrastively so that the hypersphere created with m_j as the center repels z_{ij} as follows:

$$\mathcal{L}_{rep,z_{ij}} = \frac{1}{V}\sum_{v=1}^{V} \max\{0, r^2 - \mathcal{D}(z_{ij}, m_v^{+}) - \alpha\}$$

where the hyperparameter V is the total number of hard negative features to be used for contrastive supervision, and the parameter α is used to control the balance between $\mathcal{L}_{att,z_{ij}}$ and $\mathcal{L}_{rep,z_{ij}}$ and α set to 0.1.

5.2.7 Results analysis

All experiments were conducted following the same set-up as the official configuration, changing only the type of loss used with the ones previously described.

It is clear from the tables 5.5 5.6 5.7 presented that the adoption of different types of loss functions produces results that are clearly superior and more closely aligned with the official benchmarks. The comparison between the 'official' and experimental configurations shows a significant increase in the performance of I-AUROC of 43% over the 'official' implementation. This significant results improvment suggests the possibility that the hypotesis that the loss function used in LeMO[14] may contain some error is valid.

Table 5.5: Configurations and results on I-AUROC

Declared Results	
K-means (Feature Enhanced)	**0.972**
Memory Parameter Update	**0.967**
Original Loss	
K-means (Feature Enhanced)	0.497
Memory Parameter Update	0.668
No Memory Update	0.666
CFA Loss	
K-means (Feature Enhanced)	0.9556
Memory Parameter Update	0.9529
No Memory Update	0.9557
(new)NCE Loss	
K-means (Feature Enhanced)	0.9382
Memory Parameter Update	0.9520
No Memory Update	0.9445
Triplet Central Loss	
K-means (Feature Enhanced)	0.8796
Memory Parameter Update	0.9321
No Memory Update	0.9177

Analysing the results in the table, it emerges that the configuration that achieves the best performance in terms of I-AUROC is surprisingly the one that adopts the loss CFA without any memory update. This indicates that memory initialisation with decoupled-noise is extremely efficient. This configuration, due to its lack of memory updates, is faster in training than the one in which the memory is considered as a parameter. It is also significantly faster than the feature enhanced memory update configuration, which uses K-means operations between iterations, resulting in significant slowdowns.

In contrast, the loss NCE and triplet central show slightly lower performance in that order. In particular in the figure 5.2 can be observed how the Triplet Central

Loss performance are much lower then the ones of the other two losses.

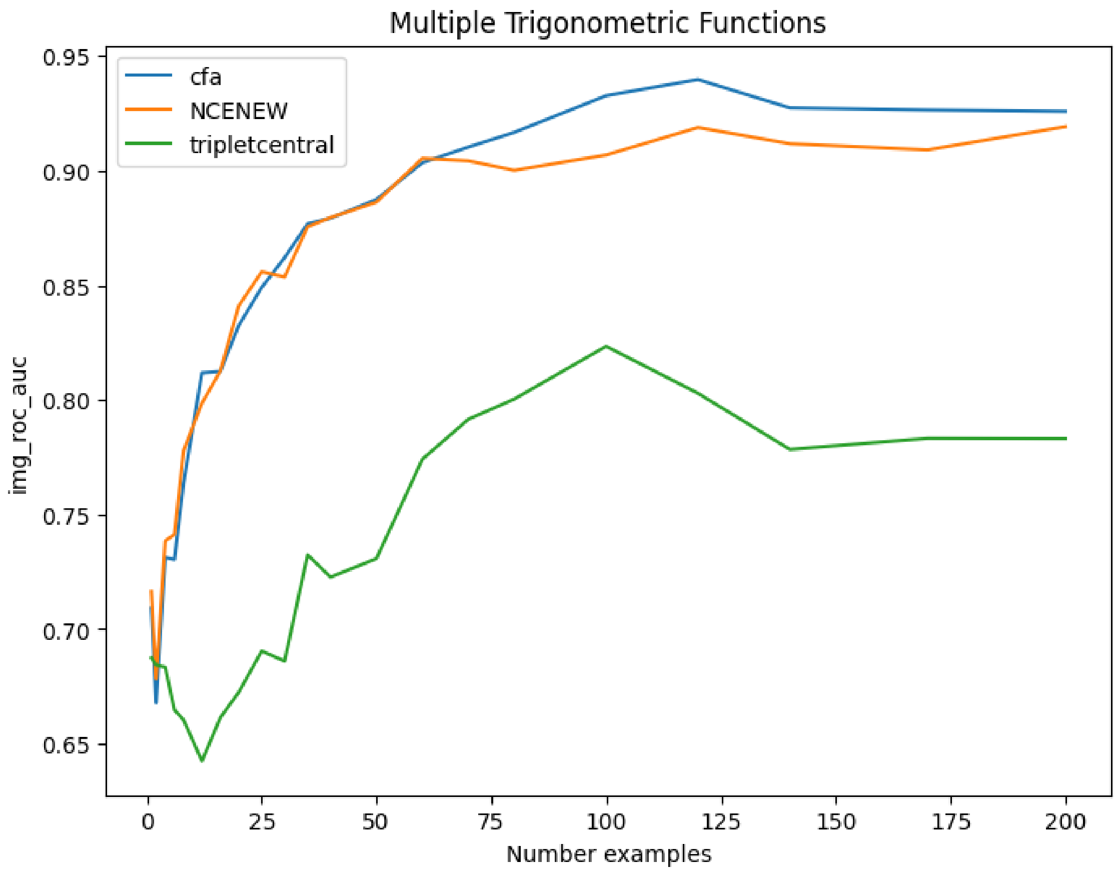

Figure 5.2: In the figure, the performance graph for the I-AUROC metrics during the first 200 iterations of the first training epoch of the various impmemented losses can be observed. For each iteration, all the classes of the MVTec dataset and all memory update methods were averaged. As in the previous figures, the toothbrush class was excluded. As can be seen, the curve relating to triplet central loss performs much less well than the other curves.

Furthermore, with regard to the pixel-related metrics, i.e. P-AUROC 5.6 and PRO-AUROC 5.7, the same pattern is observed as in the I-AUROC metric, with the model adopting loss CFA as the best performing. Within the results of the CFA loss, we again see that the performances are comparable. Specifically, as far as P-AUROC is concerned, the configuration without memory update is confirmed as the best, whereas for PRO-AUROC the configuration using enhanced memory update is the one that performs the best.

Table 5.6: Configurations and results on P-AUROC

Declared Results	
K-means (Feature Enhanced)	**0.976**
Memory Parameter Update	**0.971**
Original Loss	
K-means (Feature Enhanced)	0.648
Memory Parameter Update	0.782
No Memory Update	0.784
CFA Loss	
K-means (Feature Enhanced)	0.9690
Memory Parameter Update	0.9658
No Memory Update	0.9696
(new)NCE Loss	
K-means (Feature Enhanced)	0.9596
Memory Parameter Update	0.9579
No Memory Update	0.9598
Triplet Central Loss	
K-means (Feature Enhanced)	0.9497
Memory Parameter Update	0.9559
No Memory Update	0.9497

Table 5.7: Configurations and results on PRO-AUROC

Declared Results	
K-means (Feature Enhanced)	**0.917**
Memory Parameter Update	**0.909**
Original Loss	
K-means (Feature Enhanced)	0.521
Memory Parameter Update	0.713
No Memory Update	0.717
CFA Loss	
K-means (Feature Enhanced)	0.9067
Memory Parameter Update	0.9008
No Memory Update	0.9039
(new)NCE Loss	
K-means (Feature Enhanced)	0.8958
Memory Parameter Update	0.8936
No Memory Update	0.8894
Triplet Central Loss	
K-means (Feature Enhanced)	0.8498
Memory Parameter Update	0.8871
No Memory Update	0.8736

Analysing the results in detail, especially considering the context of online anomaly detection in which the LeMO model operates, it is interesting to examine its performance during training, as highlighted in figures 5.5 5.3 and 5.4. A closer look at this performance, in fact, reveals that in the early stages of training, approximately in the first 75 iterations, the NCE and CFA configurations, which employ the K-means algorithm for memory updating, show a steeper learning curve. This aspect intuitively favours better solutions in scenarios where the number of elements in the dataset is limited.

A particularity that must be noted is how, in iteration 2, the graphs for the algorithms corresponding to the use of memory update through K-means show very low scores, only to rise again in the immediately following iterations.

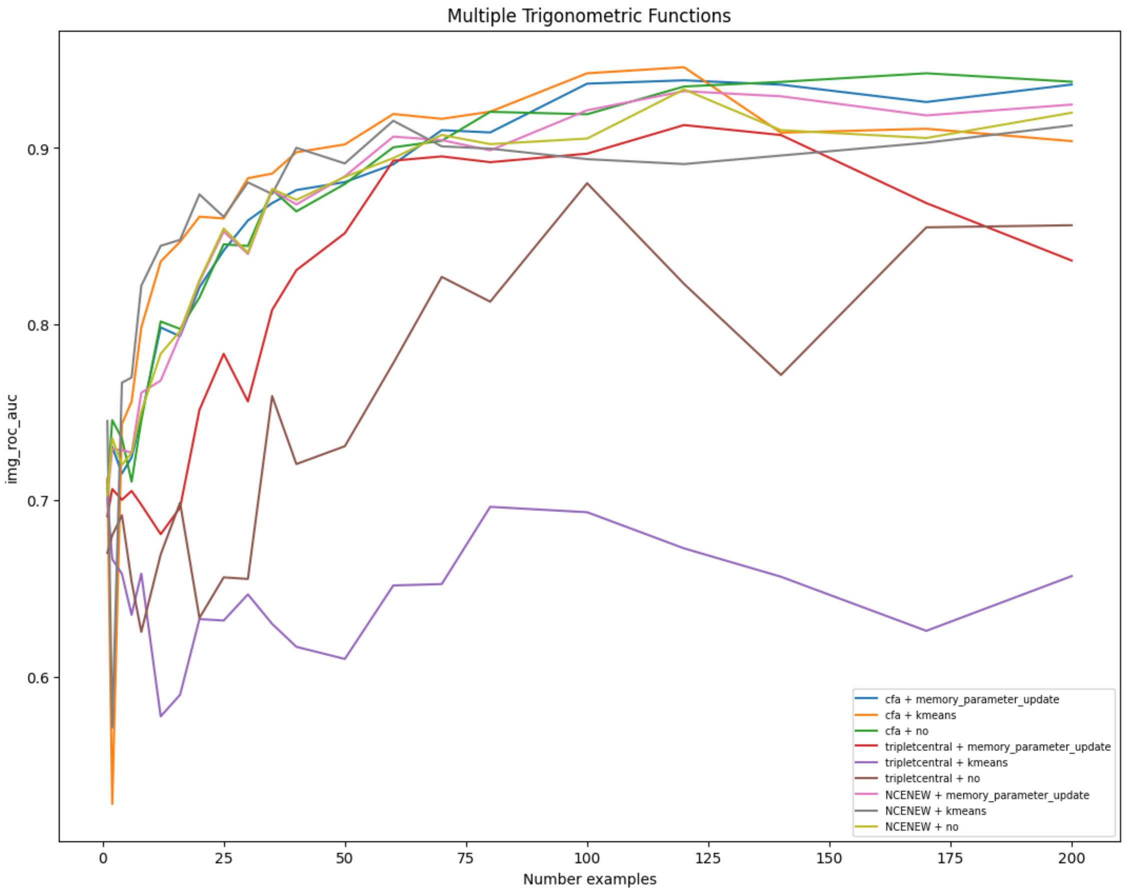

Figure 5.3: The figure shows the performance graph for the I-AUROC metrics during the first 200 iterations of the first training epoch of the various configurations. For each iteration, all the classes of the MVTec dataset were averaged, with the exception of the 'toothbrush' class, which has only 60 examples.

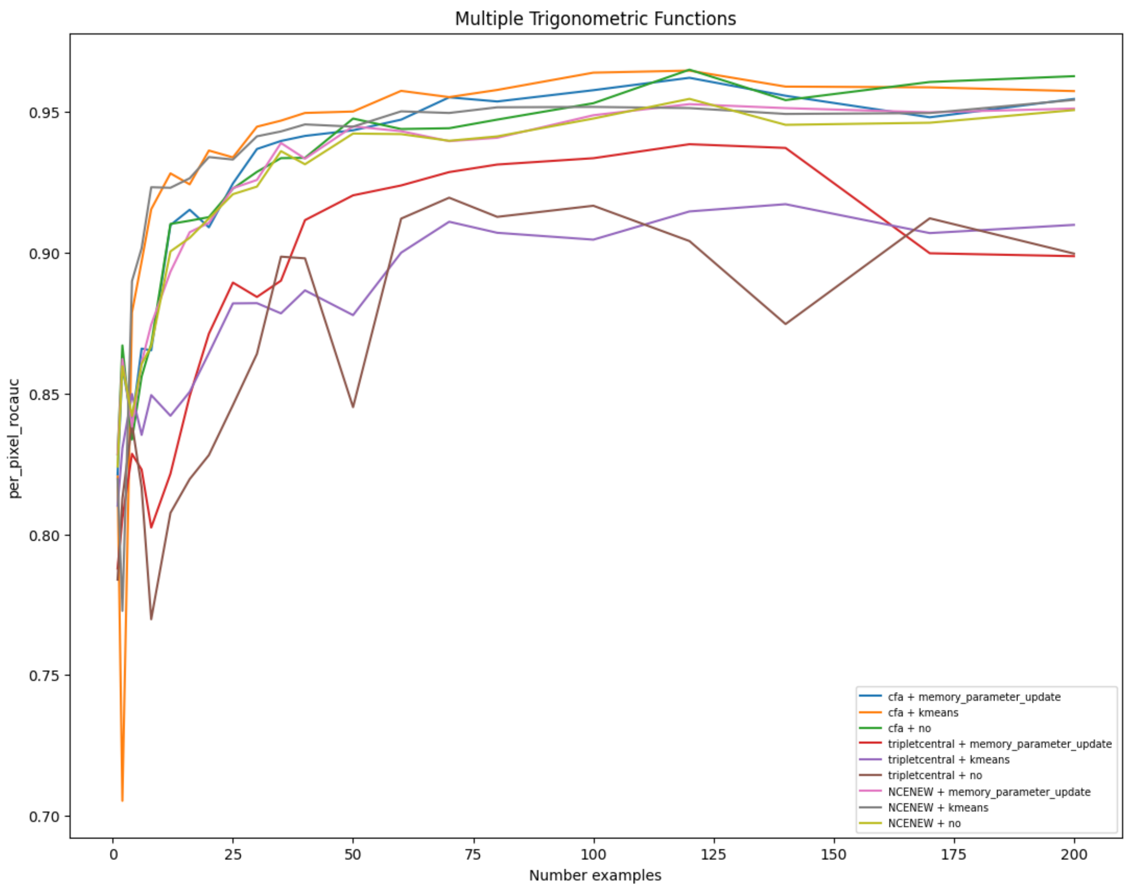

Figure 5.4: In the figure, the performance graph for the P-AUROC metrics during the first 200 iterations of the first training epoch of the various configurations can be observed. For each iteration, all the classes of the MVTec dataset were averaged, with the exception of the 'toothbrush' class, which has only 60 examples.

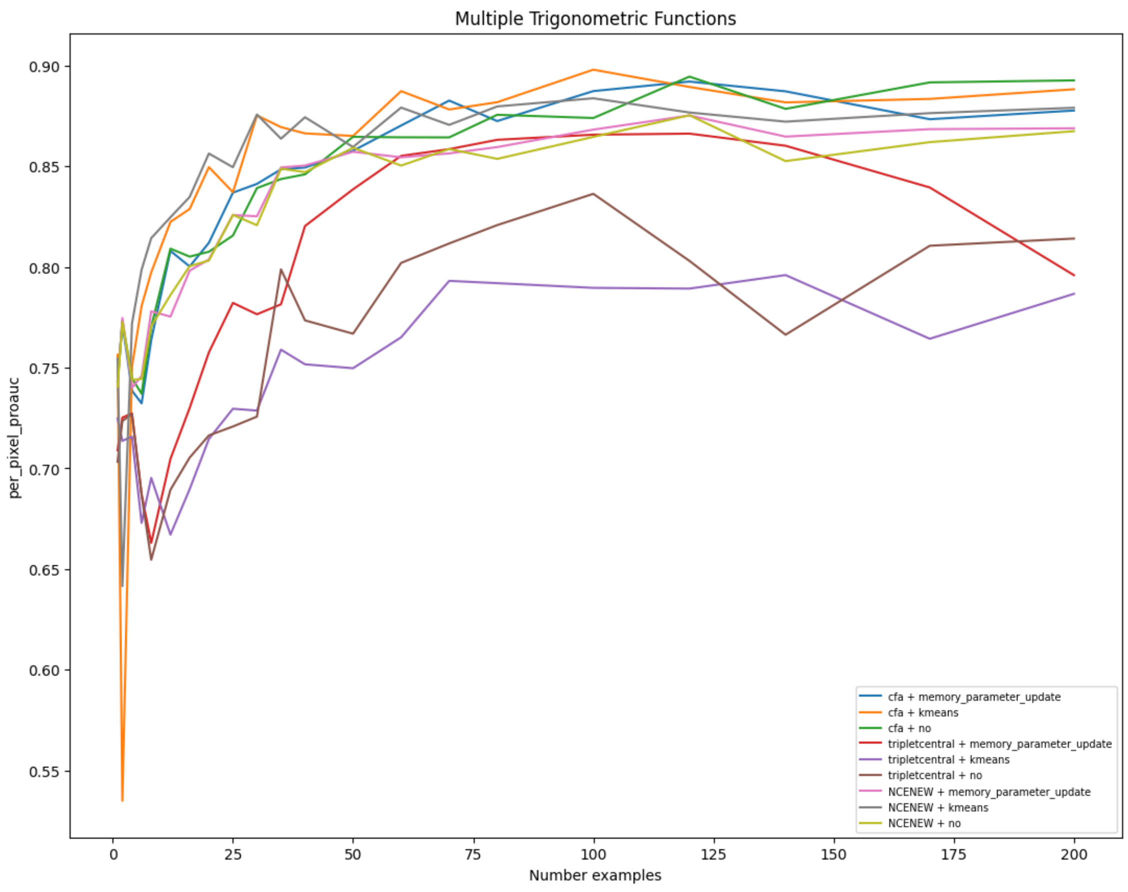

Figure 5.5: The figure shows the performance graph for the PRO-AUROC metrics during the first 200 iterations of the first training epoch of the various configurations. For each iteration, all the classes of the MVTec dataset were averaged, with the exception of the 'toothbrush' class, which has only 60 examples.

On the other hand, as far as the Triplet Central Loss is not considered, memory updating via the K-means algorithm seems to have the opposite effect, making learning more complex for the model. In the figure 5.6 it can clearly be seen that, concerning the first iterations, the curves for the algorithms employing the enhanced memory update are superior in performance compared to the other models.

However, beyond 75 iterations, the performance of the configurations tends to converge more for those that treat memory updating as a parameter or even do not update memory at all.

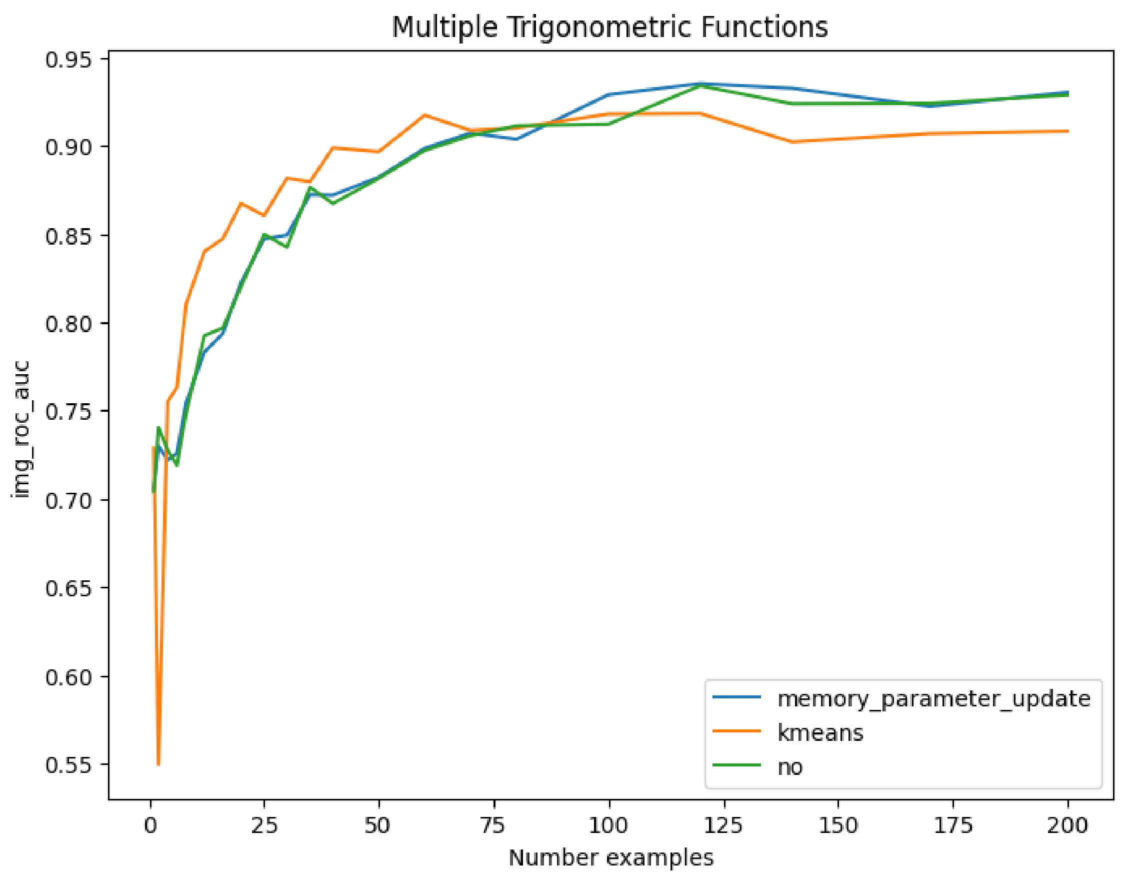

Figure 5.6: The figure shows the performance graph for the I-AUROC metrics during the first 200 iterations of the first training epoch of the various implemented memory update methods. For each iteration, all the classes of the MVTec dataset and all losses were averaged, with the exception of the triplet central loss. As in the previous figures, the toothbrush class was excluded. As can be seen, the curve for the configuration using the enhanced memory update is much steeper in the first 75 iterations than those for the memory parameter update and no memory update.

With respect to the losses, instead, it is clear how the performances of the triplet centeral loss score is lower with respect to the ones of the NCE and CFA loss 5.2.

With regard to the network's ability to classify and detect anomalies in learning situations with only a few examples (Few Shot Learning), the results in the table 5.8 indicate that the approach proposed in this study does not actually perform better than other similar networks.

Table 5.8: Comparison of Four Models on Few-Shot Learning Task

Model	**1-shot**	**2-shot**	**4-shot**	**8-shot**
Patchcore[39]	0.619	0.721	0.817	0.864
CFA[21]	0.813	0.839	0.879	**0.923**
FastFlow[67]	0.552	0.552	0.729	0.801
DREAM[54]	0.685	0.777	0.820	0.883
Declared[14]	**0.855**	**0.879**	**0.887**	0.908
Ours	0.696	0.517	0.747	0.793

Offline settings As for the results in offline configurations, the experiments were performed by training the network for 10 epochs in the same configurations as the experiments in online mode. In this case, however, given the results obtained in online mode, it was decided not to analyse the performance of the configurations using triplet central loss given its poor performance compared to the other configurations.

As can be seen from the table 5.9, and as we might have expected, the worst results are obtained with Feature Enhanced Memory Update. This was already anticipated by the graphs 5.6 in the online settings, where it could already be observed that the K-means operation involved in the Feature Enhanced Memory Update led to worse convergences for large numbers of iterations.

Table 5.9: Comparison on I-AUROC of different LeMO configurations in offline settings

Declared Results	
K-means (Feature Enhanced)	**0.990**
CFA Loss	
K-means (Feature Enhanced)	0.913
Memory Parameter Update	0.975
No Memory Update	0.962
(new)NCE Loss	
K-means (Feature Enhanced)	0.931
Memory Parameter Update	0.978
No Memory Update	0.973

Moreover, as might be expected, although in the offline settings (0.978 I-AUROC), the network performs slightly better than in the online setting (0.956 I-AUROC), the network fails to match the results reported in the paper (0.990 I-AUROC), which appear to be slightly better.

As far as pixel accuracy is concerned, the official paper LeMO[14] does not report the performance obtained in offline settings. Instead, in this work we report the tables for P-AUROC (see table 5.10) and PRO-AUROC (see table 5.11) precision. In both of them we can see, in contrast to the online setting, that the NCE loss performs slightly better than the CFA loss, which performs better in I-AUROC.

In addition, we can see how, for pixel accuracy, the feature enhanced memory update performs comparable to the other memory update configurations, whereas the I-AUROC performs drastically lower.

Table 5.10: Comparison on P-AUROC of different LeMO configurations in offline settings

CFA Loss	
K-means (Feature Enhanced)	0.960
Memory Parameter Update	0.963
No Memory Update	0.961
(new)NCE Loss	
K-means (Feature Enhanced)	0.973
Memory Parameter Update	0.973
No Memory Update	0.974

Table 5.11: Comparison on PRO-AUROC of different LeMO configurations in offline settings

CFA Loss	
K-means (Feature Enhanced)	0.886
Memory Parameter Update	0.899
No Memory Update	0.894
(new)NCE Loss	
K-means (Feature Enhanced)	0.908
Memory Parameter Update	0.910
No Memory Update	0.910