Supplementary Material

Unveiling the Anomalies in an Ever-ChangingWorld:
A Benchmark for Pixel-Level Anomaly Detection in Continual Learning

2024

1 Adapatation of AD Methods for CL - Proposed Approaches

1.1 PaDiM

A novel approach called PaDiM (Patch Distribution Modeling) has been proposed in paper [3]. It makes use of a pre-trained CNN for embedding extraction and has the two following properties:

- each patch position is described by a multivariate Gaussian distribution
- PaDiM takes into account the correlations between different semantic levels of a pre-trained CNN.

1.1.1 Train static scenario

PaDiM approach was introduced to rapidly adapt anomaly detection by employing pre-trained convolutional neural networks (CNNs) for extracting patch embeddings. The assumption is that each patch position can be described by a multivariate Gaussian distribution.

The normal image characteristics at position $(i; j)$ are learned by computing the set of patch embedding vectors at $(i; j)$, x_{ij}, from the N normal training images, as shown in Figure 1. To sum up the information carried by this set, it is assumed that x_{ij} is generated by a multivariate Gaussian distribution $N(\mu_{ij}; \Sigma_{ij})$ where μ_{ij} is the sample mean of x_{ij} and the sample covariance Σ_{ij} is estimated as follows:

$$\Sigma_{ij} = \frac{1}{N-1} \sum_{j=1}^{N-1} \sum_{k=1}^{N} (x_{ij}^k - \mu_{ij})(x_{ij}^k - \mu_{ij})^T + \epsilon I. \tag{1}$$

Finally, each possible patch position is associated with a multivariate Gaussian distribution as shown in Figure 1 by the matrix of Gaussian parameters. To enhance computational efficiency and reduce embedding vector size, random dimensionality reduction (Rd) is applied, proving better than commonly used PCA.

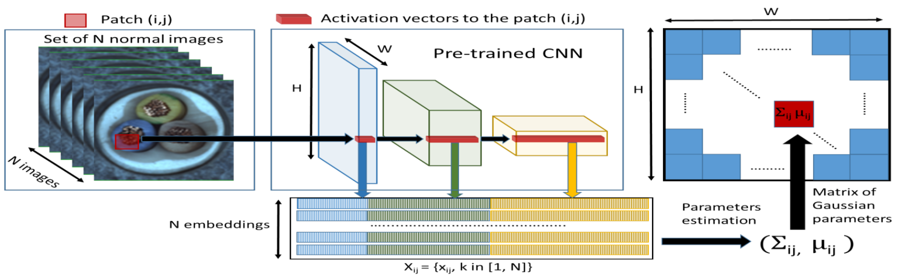
Patch (i,j)
Activation vectors to the patch (i,j)
Set of N normal images
N images
Pre-trained CNN
W
H
N embeddings
Parameters estimation
Matrix of Gaussian parameters
(Σ_{ij}, μ_{ij})
$X_{ij} = \{x_{ij}, k \text{ in } [1, N]\}$

Figure 1: Patch embedding process used in PaDiM.

1.1.2 CL adaptation

Features are produced by sampling from various depths of the CNN, yielding feature maps with distinct spatial resolutions that enable better anomaly localization. This process generates patch features of size $D \times H \times W$, with H and W representing height and width and D being the sum of dimensions of sampled feature maps. For instance, in the case explored within this work, the dimensions were $(D, H, W) = (1792, 56, 56)$, resulting in 56×56 patch features with a depth of 1792. Subsequently, random dimensionality reduction (Rd) to a depth of 350 is applied.

The method is developed purely in a continual manner with fixed-size memory. The key concept involves the incremental averaging of Gaussian parameters maps when introducing new tasks. In order to provide a clearer understanding of the pipeline and its implementation, the 1 is given.

Algorithm 1 PaDiM mean and covariance update for CL method

```
Initialize: $M \leftarrow 0$, $\Sigma \leftarrow 0$
for $i = 0$ to $N - 1$ do                                    ▷ $N$ = number of tasks
    $M_i \leftarrow$ mean of extracted features for task $i$
    $\Sigma_i \leftarrow$ covariance of extracted features for task $i$
    $M \leftarrow M \times \frac{i}{i+1} + M_i \times \frac{1}{i+1}$
    $\Sigma \leftarrow \Sigma \times \frac{i}{i+1} + \Sigma_i \times \frac{1}{i+1}$
end for
```

1.1.3 Test scenario

For the purely Continual Learning (CL) method, Mahalanobis distance:

$$M(x_{ij}) = \sqrt{(x_{ij} - \mu_{ij})^T \Sigma_{ij}^{-1} (x_{ij} - \mu_{ij})} \tag{2}$$

is used to give an anomaly score to the patch in position $(i; j)$ of a test image. $M(x_{ij})$ can be interpreted as the distance between the test patch embedding $(i; j)$ and learned distribution $N(\mu_{ij}; \Sigma_{ij})$. High scores in this map indicate the anomalous areas. The final anomaly score of the entire image is the maximum of anomaly map M.

1.1.4 Specifics

In the PaDiM framework, a Wide ResNet-50-2 architecture is employed as the backbone. This Wide ResNet-50-2 was pre-trained on the ImageNet dataset and encompassed 68.9 million parameters. It is worth noting that PaDiM uses 24.8 million parameters due to the selective utilization of specific blocks for feature extraction.

1.1.5 Resource-utilization

In this method, we must maintain a Gaussian distribution for each patch position, with a dimensionality of 350. With 56x56 patch features in the image, this amounts to 3136 distributions. Altogether, this requires 1541MB of memory to store all the distributions.

1.2 PatchCore

The PatchCore [5] is composed of several elements. These include gathering local patch features into a memory bank, employing a coreset-reduction approach to enhance efficiency, and the holistic algorithm that guides the process of making detection and localization choices.

1.2.1 Train static scenario

It employs a pre-trained network φ that is originally trained on ImageNet to create features out of images. These features, denoted as $\varphi_{i,j}$, are obtained by extracting feature maps $\varphi_j(x_i)$ from the output of intermediate network blocks obtained on image x_i. Additionally, this approach introduces a memory bank denoted as $\mathcal{M}$, composed of patch-level features that function as intermediary or mid-level representations that effectively use the available training context.

For all nominal training samples x_i in X_N, the memory bank $\mathcal{M}$ in PatchCore is defined as follows:

$$\mathcal{M} = \bigcup_{x_i \in X_N} P_{s,p}(\varphi_j(x_i))$$

where $P_{s,p}(\varphi_j(x_i))$ represents the collection of patch-level features computed using the aggregation function and neighborhood considerations.

As the cardinality of the set X_N grows, the memory bank $\mathcal{M}$ expands in size, which leads to increased time taken for making inferences on test data. In the mentioned study [5], the authors use a coreset subsampling technique to shrink the memory bank $\mathcal{M}$, effectively reducing its size and speeding up the inference. In the context of PatchCore, which deals with nearest neighbor computations, a minimax facility location coreset selection approach is adopted.

This approach ensures that the selected coreset, referred to as $\mathcal{M}_\mathcal{C}$, covers the feature space at the patch level similarly to the original memory bank $\mathcal{M}$. To further enhance the efficiency of coreset selection, random linear projections are applied to reduce the dimensionality of the elements in $\mathcal{M}$.

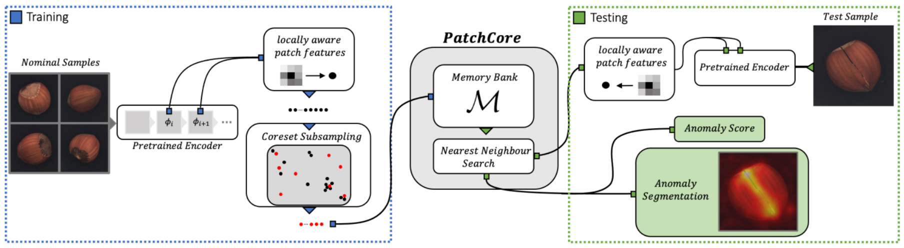

Figure 2: Overview of PatchCore.

1.2.2 Test scenario

With the nominal patch-feature memory bank $\mathcal{M}$, the image-level anomaly score $s \in \mathbb{R}$ for a test image x_{test} is estimated by calculating the maximum distance score s^* between the test patch features in its patch collection $P(x_{test}) = P_{s,p}(\varphi_j(x_{test}))$ and their respective nearest neighbor m^* in $\mathcal{M}$:

$$m^*_{\text{test}}, m^* = \arg \max_{m_{\text{test}} \in P(x_{\text{test}})} \arg \min_{m \in \mathcal{M}} \|m_{\text{test}} - m\|^2.$$

Therefore image-level anomaly score is then calculated as:

$$s^* = \|m^*_{\text{test}} - m^*\|^2.$$

1.2.3 CL adaptation

The adaptation is developed purely in a continual manner with a fixed-size memory. The key concept involves maintaining a coreset for each task encountered during training consisting of the total number of patches that can be stored in memory (30000) divided by the total number of learned tasks. This is achieved by incrementally applying coreset subsampling both to new tasks and to previous tasks saved in memory.

In order to provide a clearer understanding, the 2 is given.

Algorithm 2 PatchCore memory bank update for CL method

```
Initialize: $M \leftarrow$ empty list
memory_size $\leftarrow$ 30000
for $i = 0$ to $N - 1$ do                                  ▷ $N$ = number of tasks
    $p \leftarrow$ patches extracted for task $i$
    for $j = 0$ to length($M$) do
        $M[j] \leftarrow coreset\_subsampling(M[j]$, memory_size/$(i + 1))$
    end for
    $m \leftarrow coreset\_subsampling(p$, memory_size/$(i + 1))$
    $M.append(m)$
end for
```

1.2.4 Adapted test scenario

During inference, anomaly maps are created for each learned class by calculating the distance between each patch of a test image and its nearest neighbor in the corresponding memorized coreset. The coreset with the lowest average distance over its corresponding map is automatically selected for the test-image class. The image-level anomaly score is obtained by using the test patch and memorized patch, both corresponding to the maximum value across the anomaly map, in addition to its 3 nearest memorized neighbors.

1.2.5 Specifics

In the PatchCore method, a Wide ResNet-50-2 architecture is employed as the backbone. It is already pre-trained on the ImageNet dataset, consisting of a total of non-trainable 68.9 million parameters. However, in this particular approach, only three consecutive blocks from the overall structure are employed, amounting to a utilization of 24.8 million parameters.

1.2.6 Resource-utilization

In this approach, the training process does not involve neural network parameters updates, but only memorizing proper patch coreset. Essentially, the frozen pre-trained Wide ResNet-50-2 is employed as the feature extractor for all train images. This process yields patch features of size $D \times H \times W$ per image, where H and W represent height and width, and D is the sum of dimensions of sampled feature maps. In that regard, the specification is $(D, H, W) = (1536, 28, 28)$, which results in 28×28 patch features with a depth of 1536.

To further enhance coreset selection efficiency, random linear projections with compression factor ($\epsilon = 0.9$) are applied to reduce the dimensionality of the elements in $\mathcal{M}$. This directly reduces computation time during the coreset selection process.

When considering a memory size 30000, we obtain a total of 184.3MB.

1.3 CFA

The paper [4] proposes a novel approach called Coupled-Hypersphere-Based Feature Adaptation (CFA) which aims to obtain discriminative normal features to be processed by a learnable patch descriptor whose output is compared with a memorized patch features in a scalable memory bank. A key point is that memory size is independent of the target dataset.

1.3.1 Train static scenario

CFA contains a frozen Convolutional Neural Network (CNN) that extracts patch features from the target dataset, each carrying semantic information about the corresponding pixel location. These features are generated by sampling from various depths of the CNN, resulting in feature maps with distinct spatial resolutions that are then interpolated to achieve uniform resolution before concatenation. As such, they are processed by the learnable patch descriptor architecture, consisting of trainable parameters that embed target-oriented features and a memory bank. The descriptor's main task is to learn to densely cluster normal features and evade their multiple hyperspheres using a novel loss function based on soft-boundary regression. The main idea behind the CFA approach is to optimize the hypersphere radius and center of the adapted features to minimize the distance between the adapted features and the target features. (Figure 3)

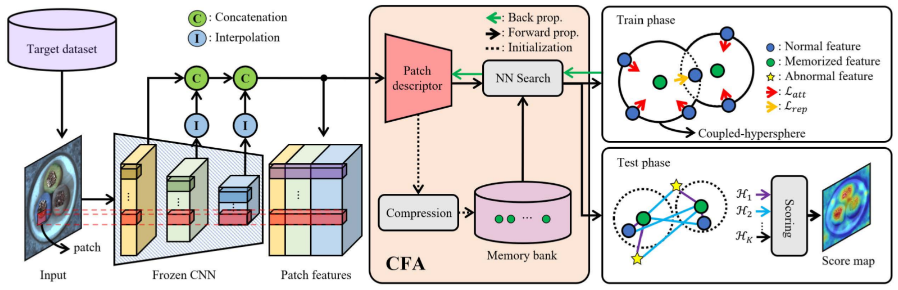

Figure 3: Overall structure of CFA method.

1.3.2 CL adaptation

The specification of produced patch features denoted as $F \in \mathbb{R}^{D\times H\times W}$ is the following$(D, H, W) = (1792, 56, 56)$, which leads to 56×56 patch features with a depth of 1792. This tensor of patch features is subsequently fed through the Patch Descriptor network (PDN), whose objective is to learn features extracted from normal samples within the target dataset, ensuring a high density around the memorized features.

The process begins with updating memorized patch features using an incremental average across batches of the current class's non-anomalous training dataset, before moving on to updating network parameters. In order to provide clearer understanding of the pipeline and its computational implementation, the 3 is given.

Algorithm 3 CFA memory bank update for replay method

```
Initialize: $C \leftarrow 0$
for $i = 0$ to $N - 1$ do                    ▷ $N$ = number of tasks
    $C_i \leftarrow$ memory bank modeling for task $i$
    $C \leftarrow C \times \frac{i}{i+1} + C_i \times \frac{1}{i+1}$
end for
```

The same procedure is followed for every subsequent task, which relies on a previously updated memory bank and PDN. The aim is to bring patch features closer to the memorized ones, achieved through a ”soft boundary loss” driven by two parameters, K and J, both set to 3. In essence, distances from each patch feature to its K nearest memorized neighbors are considered in the first part of the total loss, provided they exceed the radius size r. Similarly, the next J nearest neighbors (specifically, the $(K + j)$-th nearest) are considered if their distances are less than r. The total loss formula is shown below:

$$L_{CFA} = L_{att} + L_{rep} = \frac{1}{TK}\sum_{t=1}^{T}\sum_{k=1}^{K}\max\{0, D(\phi(p_t), c_t^k) - r^2\} + \frac{1}{TJ}\sum_{t=1}^{T}\sum_{j=1}^{J}\max\{0, r^2 - D(\phi(p_t), c_t^j) - \alpha\}$$

In broad terms, the underlying concept for this loss function is to ensure that local content associated with neighboring pixels in the images is grouped. In contrast, other content should be situated at a greater distance in the feature space. This approach allows the model to identify anomalous regions accurately at the pixel level.

1.3.3 Test scenario

During the test phase, CFA leverages patch features obtained from a given test sample. It matches these features with their K nearest neighbors (in our case, $K = 3$) in the memory bank, generating heatmaps that depict the degree of anomaly. In the final step, an anomaly score map is formulated by applying softmin function on previously generated K heatmaps.

1.3.4 Specifics

In this work, CFA leverages a fixed-pretrained Wide ResNet-50-2 on ImageNet as its backbone, containing 68.9 million non-trainable parameters, and patch descriptor network with 6.4 million trainable weights. The training is conducted over 30 epochs for each successive task with a batch size of 4, with an additional 4 samples in the replay sampling strategy.

1.3.5 Resource-utilization

The total memory required to accommodate memorized patch features can be calculated by performing the following calculation: $56 \times 56 \times 1792 \times 4$ bytes. Furthermore, additional memory is required to store images used in the Replay strategy. Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics. To store the images of size $224 \times 224 \times 3$ channels $\times$ 1 byte, it is necessary to have 45.2MB for memory size 300.

1.4 STPFM

This paper incorporates the Student-Teacher Feature Pyramid Matching (STFPM) strategy, introduced in [6], within the CL framework.

1.4.1 Train static scenario

In the study [6], the authors employ the student-teacher learning framework belonging to the domain of embedding similarity-based methods. The teacher, in this context, is a powerful ResNet-18 network pre-trained on ImageNet. To mitigate information loss and address the complexity of handling scaling, the student network shares an identical architecture with the teacher. The precise position of knowledge distillation within deep neural networks is pivotal in this approach. Deep neural networks generate a feature pyramid for each input image, which facilitates the enhancement of the scale robustness by integrating multi-scale feature alignment between the student and teacher networks. Consequently, the student network gains a comprehensive blend of multi-level knowledge, enabling it to effectively identify anomalies of varying sizes.

Given the varying receptive fields associated with different layers in deep neural networks, this approach involves leveraging features extracted from three consecutive lower-layer groups of the teacher network to guide the student's learning process. For a visual representation of this method using images from the MVTec AD dataset, refer to Figure 4.

The goal of the training phase is to create a student network capable of replicating the feature maps generation from a pre-trained teacher network when processing normal images.

Throughout the training process, both the teacher and student networks generate three feature pyramid blocks of maps for each input image. These blocks are subsequently compared using an L2-loss computation based on the following formula:

$$l^l(I_k)_{ij} = \frac{1}{2}\left|\left|F_t^l(I_k)_{ij} - (F_s^l(I_k)_{ij})\right|\right|_{l_2}^2$$

where I_k is input image, F_t^l and F_s^l are feature maps generated in l-th layer (block) of a teacher and student, respectively.

The total loss is then calculated as the sum of the losses from all the blocks and is employed to update the parameters of the student's network.

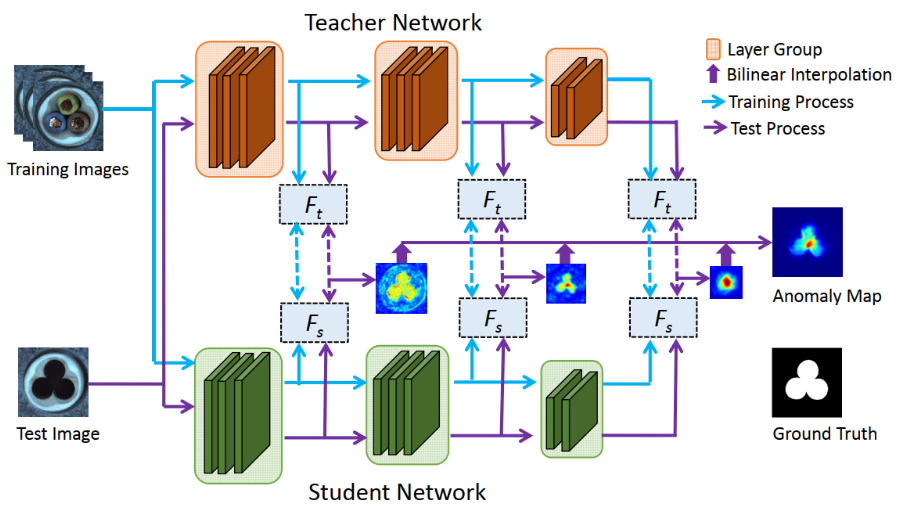

Figure 4: Schematic overview of Student-Teacher Feature Pyramid Matching.

1.4.2 CL adaptation

In this work, the training employs a batch size of 8 (8+8 in replay sampling strategy) over 100 epochs, with early stopping included. Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics.

1.4.3 Test scenario

Essentially, student network's capability of replicating feature maps generated by a pre-trained teacher network is utilized for revealing discrepancies across various network levels by comparing their corresponding intermediate maps during the test phase on anomalous images.

During the test phase, the losses generated by each block, which can also be regarded as intermediate anomaly maps, are multiplied on a pixel-level basis. Subsequently, the final anomaly map is derived using the provided formula:

$$\Omega(J) = \prod_{l=1}^{L} \text{Upsample}\Omega^{l}(J)$$

where J is test image, $\Omega^l(J)$ is anomaly map generated after l-th block, L is total number of blocks and $\Omega(J)$ is final anomaly map.

1.4.4 Specifics

In this Student-Teacher setting both networks leverage fixed-pretrained ResNet-18 on ImageNet as its backbone. The teacher network consists of 11.7 million non-trainable parameters, while the student network has an identical number of trainable parameters. This results in a total of 23.4 million parameters.

1.4.5 Resource-utilization

Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics. In order to make replay strategy feasible, images of size 256 × 256 × 3 channels × 1 byte need to be stored in memory for which it is necessary to have 59.0 for memory size 300.

1.5 EfficientAD

Paper [1] proposes EfficientAD, a novel approach that emphasizes both computational efficiency and economic feasibility in anomaly detection methods. The proposed method is characterized by a multi-network architecture consisting of three components:

1. Teacher Network
2. Student Network
3. Autoencoder

1.5.1 Train static scenario

EfficientAD employes efficient extraction of features from a pretrained neural network with drastically reduced depth, comprising only four convolutional layers, as a feature extractor. This network, known as a patch description network (PDN) Figure 5, generates descriptive 33×33 patches per output feature vector. The PDN is trained on images from ImageNet by minimizing the mean squared difference between its output and the features extracted from the pretrained network.

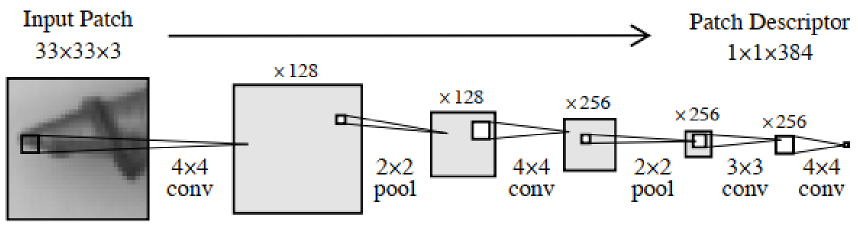

Figure 5: Patch description network (PDN) architecture.

In this method, Student-Teacher (S-T) architecture for detecting anomalous feature vectors uses just one teacher (distilled PDN) and one student. The

PDN's architecture serves as the student's model, ensuring low overall latency due to its fast execution.

Images from the teacher's pretraining dataset (ImageNet) are incorporated during student training. By randomly sampling a pretraining image in each training step, the student is penalized from overly generalizing the teacher's imitation to out-of-distribution images. This measure improves the student's ability to detect anomalies accurately while maintaining efficiency in the detection process.

To account for logical anomalies, such as misplaced objects, an autoencoder is incorporated into EfficientAD architecture. Figure 6 illustrates the anomaly detection process used in EfficientAD.

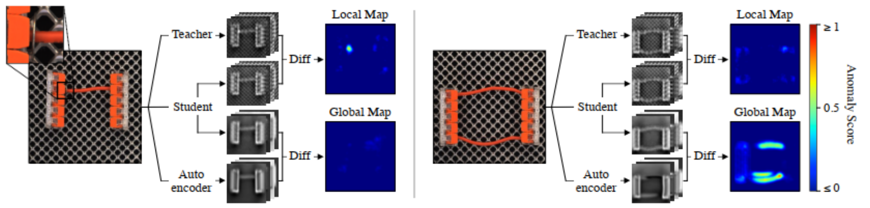

Figure 6: Anomaly detection methodology for EfficientAD.

While the patch-based student relies on generating descriptive patches for anomaly detection, the autoencoder encounters challenges in reconstructing fine-grained patterns, leading to flawed reconstructions on both normal and anomalous images. To avoid false-positive detections, the student network is modified by doubling the number of output channels. The student is then trained to predict both the output of the teacher and the output of the autoencoder, enabling effective detection of anomalies while considering the limitations of the autoencoder's reconstruction capabilities.

1.5.2 CL adaptation

Within the scope of CL, training in this work utilizes replay sampling strategy with batch size 1 (1+1 in replay) over 70 epochs, including early stopping. Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics.

1.5.3 Test scenario

The student network learns the systematic reconstruction errors of the autoencoder on normal images, such as blurry reconstructions. However, it remains unaware of the reconstruction errors for anomalies, as they are not part of its training set. This difference between the autoencoder's output and the student's output is well-suited for computing the anomaly map. Analogous to the anomaly map generated by the student–teacher pair reffered to as the "local anomaly map", the "global anomaly map" is obtained by squaring the differences between the outputs of the student and the autoencoder, and then averaging the differences across channels. To create the final "combined anomaly map", both the "local" and "global anomaly maps" are averaged together. The image-level anomaly score is determined by selecting the maximum value from this combined map, facilitating effective anomaly detection.

1.5.4 Specifics

EfficientAD employs a multi-network architecture consisting of three components: a teacher network, a student network, and an autoencoder. The teacher network comprises 8.0 million non-trainable parameters, while the student network has 11.6 million trainable parameters. Additionally, the autoencoder component containes 1.1 million trainable parameters. In total, the architecture of EfficientAD composes of 20.7 million parameters.

1.5.5 Resource-utilization

The required memory for storing images for replay is calculated as follows: $256 \times 256 \times 3$ (image size) multiplied by 300 for the case of memory 300.

1.6 DRÆM

The DRÆM method [8] is introduced to address a common drawback of generative anomaly detection methods. These methods only learn from anomaly-free data and lack optimization for discriminative anomaly detection since positive examples (anomalies) are unavailable during training. Training with synthetic anomalies leads to overfitting to synthetic appearances, resulting in poor generalization to real anomalies. To reduce overfitting, DRÆM proposes training a discriminative model that considers the joint appearance of both reconstructed and original data, including the reconstruction subspace. The architecture of DRÆM network comprises a reconstructive subnetwork followed by a discriminative sub-network Figure 7.

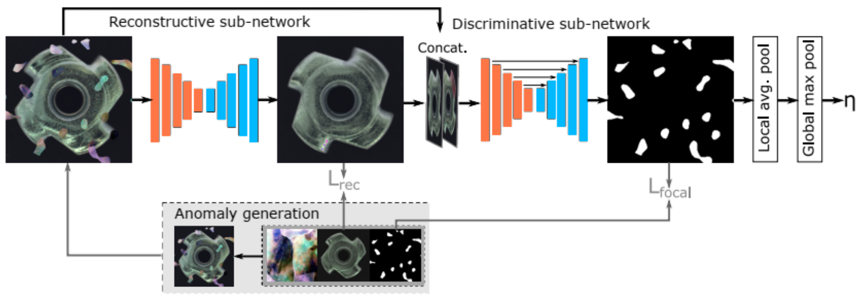

Figure 7: The anomaly detection process of the DRÆM method.

1.6.1 Train static scenario

The reconstructive sub-network is trained to implicitly detect and reconstruct anomalies with semantically plausible anomaly-free content, while keeping the non-anomalous regions of the input image unchanged. It is formulated as an encoder-decoder architecture that converts the local patterns of an input image into patterns closer to the distribution of normal samples. The network is trained to reconstruct the original image I from an artificially corrupted version I_a obtained by a simulator. The simulated anomaly generation process Figure 8 involves generating a binary anomaly mask M_a from Perlin noise P. The anomalous regions are sampled from a set A based on the values in M_a and overlaid on the anomaly-free image I to create the anomalous image I_a.

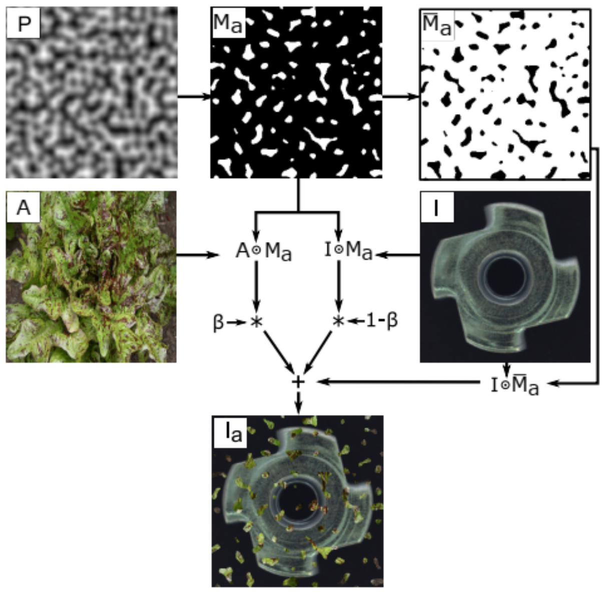

Figure 8: Simulated anomaly generation process.

The reconstruction loss is defined as:

$$L_{rec}(I, I_r) = \lambda \cdot L_{SSIM}(I, I_r) + l_2(I, I_r)$$

where image I represents the input image, I_r is the reconstructed image output by the network, and λ is a loss balancing hyper-parameter.

The discriminative sub-network utilizes a U-Net-like architecture. The input of the sub-network, denoted as I_c, is formed by the channel-wise concatenation of the output from the reconstructive sub-network (I_r) and the input image (I_a). As the reconstructive sub-network restores the normality of the image, the joint appearance of I_a and I_r exhibits significant differences in anomalous images. These differences in joint appearance provide crucial information for accurate anomaly segmentation. To enhance accurate segmentation of challenging examples, the output of the discriminative sub-network is subjected to Focal Loss (L_{seg}).

Taking into account the objectives of both the segmentation and reconstructive sub-networks, the total loss used in training DRÆM is defined as:

$$L(I, I_r, M_a, M) = L_{rec}(I, I_r) + L_{seg}(M_a, M),$$

where M_a and M represent the ground truth and output anomaly segmentation masks, respectively.

1.6.2 CL adaptation

Within the scope of CL, training in this study employs replay sampling strategy with batch size of 4 (4+4 in replay) over 50 epochs. The training process runs for 50 epochs. Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics.

1.6.3 Test scenario

The output of the discriminative sub-network is a pixel-level anomaly detection mask, M_o, which directly indicates the presence of anomalies in the image. To estimate the image-level anomaly score, M_o is smoothed using a mean filter convolution layer. The final image-level anomaly score, denoted as η, is computed by taking the maximum value from the smoothed anomaly score map:

$$\eta = \max(M_o * f_{sf \times sf}),$$

where $f_{sf \times sf}$ represents a mean filter of size $sf \times sf$, and $*$ denotes the convolution operator.

1.6.4 Specifics

DRÆM incorporates both a reconstructive network and a discriminative network. The reconstructive network consists of 69.0 million parameters, while the discriminative network has 28.4 million parameters. In total, the architecture of DRÆM is composed of 97.4 million trainable parameters. Additionally, the Describable Textures Dataset [2] is used as the anomaly source dataset.

1.6.5 Resource-utilization

The required memory for storing images for replay is calculated as follows: $256 \times 256 \times 3$ (image size) multiplied by 300 for the case of memory 300.

1.7 FastFlow

While current methods for unsupervised anomaly detection and localization produce satisfactory results, they fall short in effectively mapping image features to a tractable base distribution, which is crucial for identifying anomalies. To address this, the FastFlow approach was proposed in [7], which utilizes 2D normalizing flows as the probability distribution estimator. FastFlow takes as input the features obtained from arbitrary deep feature extractors.

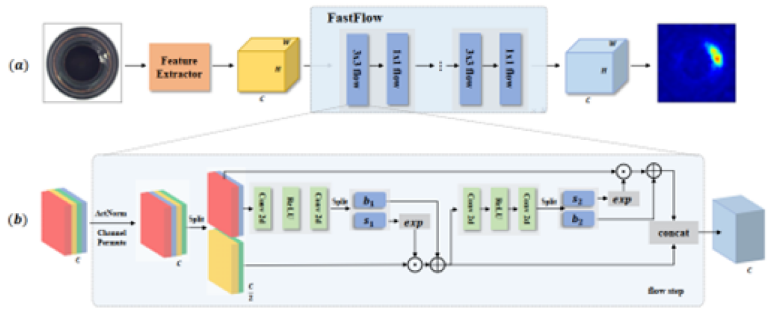

Figure 9: Simulated anomaly generation process.

1.8 Train static scenario

FastFlow extends the original normalizing flow to two-dimensional space to retain the inherent spatial positional relationship of the two-dimensional image, which a one-dimensional normalizing flow cannot do. A fully convolutional network is used as the subnet in the flow model, as it can maintain the relative position of the space to improve the performance of anomaly detection. During training, FastFlow is trained with normal images to transform the original distribution to a standard normal distribution in a 2D manner.

1.8.1 CL adaptation

Within the scope of CL, training in this study employs a replay sampling strategy with a batch size of 32 (32+32 in replay) over 15 epochs. The training process runs for 15 epochs. Additionally, different replay memory sizes were employed to analyze the impact of the number of images on performance metrics.

1.8.2 Test scenario

During the test phase, the high-dimensional visual features extracted from typical backbone networks are projected into probability density maps representing the likelihood of each pixel in the image under the learned distribution. Anomalies are detected as regions with low density, which correspond to regions that are far away from the distribution of normal data.

1.8.3 Specifics

FastFlow uses 20-step flows for CaiT and DeiT, and 8-step flows for ResNet18 and Wide-ResNet50-2. FastFlow takes as input the features obtained from arbitrary deep feature extractors, such as ResNet and Vision Transformer (ViT), all initialized with pre-trained weights from ImageNet, and with their parameters frozen. For ResNet18 and Wide-ResNet50-2, the features of the last layer in the first three blocks are directly used and put into the 2D flow model to obtain their respective anomaly detection and localization results, and the average value is taken as the final result. For Vision Transformer, the method only uses feature maps of a specific layer, and does not require the design of complicated multi-scale features manually. We consider the Wide-ResNet50-2 architecture during the experiments.

1.8.4 Resource-utilization

The required memory for storing images for replay is calculated as follows: $256 \times 256 \times 3$ (image size) multiplied by 300 for the case of memory 300 Additionally, the memory required by FastFlow is estimated based on the reported values of [7].

References

[1] Kilian Batzner, Lars Heckler, and Rebecca König. EfficientAD: Accurate visual anomaly detection at millisecond-level latencies. *arXiv:2303.14535*, 2023.

[2] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[3] Thomas Defard, Aleksandr Setkov, Angelique Loesch, and Romaric Audigier. PaDiM: A patch distribution modeling framework for anomaly detection and localization. In *Pattern Recognition. ICPR International Workshops and Challenges*, pages 475–489. Springer International Publishing, 2021.

[4] Sungwook Lee, Seunghyun Lee, and Byung Cheol Song. Cfa: Coupled-hypersphere-based feature adaptation for target-oriented anomaly localization. *arXiv:2206.04325*, June 2022.

[5] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. *arXiv:2106.08265*, 2022.

[6] Guodong Wang, Shumin Han, Errui Ding, and Di Huang. Student-teacher feature pyramid matching for anomaly detection. *arXiv:2103.04257*, 2021.

[7] Jiawei Yu, Ye Zheng, Xiang Wang, Wei Li, Yushuang Wu, Rui Zhao, and Liwei Wu. Fastflow: Unsupervised anomaly detection and localization via 2d normalizing flows, 2021.

[8] Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. DRAEM - A discriminatively trained reconstruction embedding for surface anomaly detection. *arXiv:2108.07610*, 2021.