EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies

Kilian Batzner Lars Heckler Rebecca König
MVTec Software GmbH
{kilian.batzner, lars.heckler, rebecca.koenig}@mvtec.com

Abstract

Detecting anomalies in images is an important task, especially in real-time computer vision applications. In this work, we focus on computational efficiency and propose a lightweight feature extractor that processes an image in less than a millisecond on a modern GPU. We then use a student–teacher approach to detect anomalous features. We train a student network to predict the extracted features of normal, i.e., anomaly-free training images. The detection of anomalies at test time is enabled by the student failing to predict their features. We propose a training loss that hinders the student from imitating the teacher feature extractor beyond the normal images. It allows us to drastically reduce the computational cost of the student–teacher model, while improving the detection of anomalous features. We furthermore address the detection of challenging logical anomalies that involve invalid combinations of normal local features, for example, a wrong ordering of objects. We detect these anomalies by efficiently incorporating an autoencoder that analyzes images globally. We evaluate our method, called EfficientAD, on 32 datasets from three industrial anomaly detection dataset collections. EfficientAD sets new standards for both the detection and the localization of anomalies. At a latency of two milliseconds and a throughput of six hundred images per second, it enables a fast handling of anomalies. Together with its low error rate, this makes it an economical solution for real-world applications and a fruitful basis for future research.

1. Introduction

In the past years, deep learning methods have continued to improve the state of the art across a wide range of computer vision applications. This progress has been accompanied by advances in making neural network architectures faster and more efficient [46, 62, 64, 67]. Modern classification architectures, for example, focus on characteristics such as latency, throughput, memory consumption, and the number of trainable parameters [34, 35, 57, 62, 63, 67]. This ensures that as networks become more capable, their computational requirements remain suitable for real-world applications. The field of visual anomaly detection has also seen rapid progress in the recent past, especially on industrial anomaly detection benchmarks [7, 9, 50, 53]. State-of-the-art anomaly detection methods, however, often sacrifice computational efficiency for an increased anomaly detection performance. Common techniques are ensembling, the use of large backbones, and increasing the input image resolution to up to 768×768 pixels.

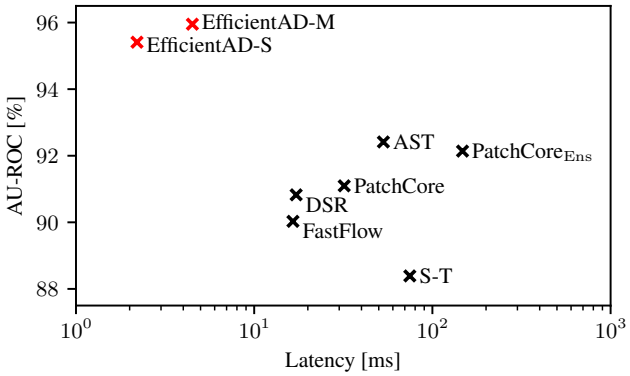


Figure 1. Anomaly detection performance vs. latency per image on an NVIDIA RTX A6000 GPU. Each AU-ROC value is an average of the image-level detection AU-ROC values on the MVTec AD [7, 9], VisA [73], and MVTec LOCO [8] dataset collections.

Real-world anomaly detection applications frequently put constraints on the computational requirements of a method. There are cases where detecting an anomaly too late can cause substantial economic damage, such as metal objects in a crop field entering the interior of a combine harvester. In other cases, even human health is at risk, for example, if a limb of a machine operator approaches a blade. Furthermore, industrial settings commonly involve strict runtime limits caused by high production rates [4]. Not adhering to these limits would decrease the production rate of the respective application and thus its economic viability. It is therefore essential to pay attention to the computational and economic cost of anomaly detection methods to keep them suitable for real-world applications.

In this work, we propose EfficientAD, a method that sets new standards for both the anomaly detection performance and the inference runtime, as shown in Figure 1. We first introduce an efficient network architecture for computing expressive features in less than a millisecond on a modern GPU. To detect anomalous features, we use a student–teacher approach [10, 53, 65]. We train a student network to predict the features computed by a pretrained teacher network on normal, i.e., anomaly-free training images. Because the student is not trained on anomalous images, it generally fails to mimic the teacher on these. A large distance between the outputs of the teacher and the student thus enables the detection of anomalies at test time. To further increase this effect, Rudolph *et al*. [53] use *architectural* asymmetry between the teacher and the student. We instead introduce *loss-induced* asymmetry in the form of a training loss that hinders the student from imitating the teacher beyond the normal images. This loss does not affect the computational cost at test time and does not restrict the architecture design. It allows us to use our efficient network architecture for both the student and the teacher, while achieving an accurate detection of anomalous features.

Identifying anomalous local features enables the detection of anomalies that are *structurally* different from the normal images, for example, contaminations or stains on manufactured products. A challenging problem, however, are violations of *logical* constraints regarding the position, size, arrangement, etc. of normal objects. To address this, EfficientAD includes an autoencoder, as proposed by Bergmann *et al*. [8]. This autoencoder learns the logical constraints of training images and detects violations at test time. We simplify the autoencoder's architecture and training protocol and show how to integrate it efficiently with a student–teacher model. Furthermore, we present a method to improve the anomaly detection performance by calibrating the detection results of the autoencoder and the student–teacher model before combining their results.

Our contributions are summarized as follows:

- We substantially improve the state of the art for both the detection and the localization of anomalies on industrial benchmarks, at a latency of 2 ms and a throughput of more than 600 images per second.
- We propose an efficient network architecture to speed up feature extraction by an order of magnitude in comparison to the feature extractors used by recent methods [50, 53, 68].
- We introduce a training loss that significantly improves the anomaly detection performance of a student–teacher model without affecting its inference runtime.
- We achieve an efficient autoencoder-based detection of logical anomalies and propose a method for a calibrated combination of the detection results with those of a student–teacher model.

2. Related Work

2.1. Anomaly Detection Tasks

Visual anomaly detection is a rapidly growing area of research with a diverse range of applications, including medical imaging [3, 19, 37], autonomous driving [13, 24, 32], and industrial inspection [7, 18, 42]. Applications often have specific characteristics, such as the availability of image sequences in surveillance datasets [31, 36, 71] or the different modalities of medical imaging datasets (MRI [5], CT [3], X-ray [27], etc.). This work focuses on detecting anomalies in RGB or gray-scale images without conditioning the prediction on a sequence of images. We use industrial anomaly detection datasets to benchmark our proposed method against existing ones.

The introduction of the MVTec AD dataset by Bergmann *et al*. [7, 9] has catalyzed the development of methods for industrial applications. It comprises 15 separate inspection scenarios, each consisting of a training set and a test set. Each training set contains only normal images, for example, defect-free screws, while the test sets also contain anomalous images. This represents a frequent challenge in real-world applications where the types and possible locations of defects are unknown during the development of the anomaly detection system. Therefore, it is a challenging yet crucial requirement that methods perform well when trained only on normal images.

Recently, several new industrial anomaly detection datasets have been introduced [8, 11, 26, 28, 38, 73]. The Visual Anomaly (VisA) dataset [73] and the MVTec Logical Constraints (MVTec LOCO) dataset [8] follow the design of MVTec AD and comprise twelve and five anomaly detection scenarios, respectively. They contain anomalies that are empirically more challenging than those of MVTec AD. Furthermore, MVTec LOCO contains not only structural anomalies, such as stains or scratches but also logical anomalies. These are violations of logical constraints, for example, a wrong ordering or a wrong combination of normal objects. We refer to MVTec AD, VisA, and MVTec LOCO as dataset collections, as each scenario is a separate dataset consisting of a training and a test set. All three provide pixel-precise defect segmentation masks for evaluating the anomaly localization performance of a method.

2.2. Anomaly Detection Methods

Traditional computer vision algorithms have been applied successfully to industrial anomaly detection tasks for several decades [61]. These algorithms commonly fulfill the requirement of processing an image within a few milliseconds. Bergmann *et al*. [7] evaluate some of these methods and find that they fail when requirements such as well-aligned objects are not met. Deep-learning-based methods have been shown to handle such cases more robustly [7, 8].

A successful approach in the recent past has been to apply outlier detection and density estimation methods in the feature space of a pretrained and frozen convolutional neural network (CNN). If feature vectors can be mapped to input pixels, assigning their outlier scores to the respective pixels yields a 2D anomaly map of pixel anomaly scores. Common methods include multivariate Gaussian distributions [16, 30, 48], Gaussian Mixture Models [38, 72], Normalizing Flows [22, 47, 51, 52, 68], and the k-Nearest Neighbor (kNN) algorithm [14, 39, 40, 50]. A runtime bottleneck for kNN-based methods is the search for nearest neighbors during inference. With PatchCore [50], Roth *et al*. therefore perform kNN on a reduced database of clustered feature vectors. They achieve state-of-the-art anomaly detection results on MVTec AD. In our experiments, we include PatchCore and FastFlow [68], a recent Normalizing-Flow-based method with a comparatively low inference runtime.

Bergmann *et al*. [10] propose a student–teacher (S–T) framework for anomaly detection, in which the teacher is a pretrained frozen CNN. They use three different teachers with different receptive field sizes. For each teacher, they train three student networks to mimic the output of the respective teacher on the training images. Because the students have not seen anomalous images during training, they generally fail to predict the teacher's output on these images, which enables anomaly detection. Bergmann *et al*. use the variance of the students' outputs together with their distance from their teacher's output to compute anomaly scores. Various modifications of S–T have been proposed [53, 56, 65]. Rudolph *et al*. [53] reach an anomaly detection performance that is competitive to PatchCore by restricting the teacher to be an invertible neural network. We compare our method to their Asymmetric Student Teacher (AST) approach and to the original S–T method [10].

Generative models such as autoencoders [6, 8, 12, 20, 33, 43, 55] and GANs [2, 21, 45, 58, 59] have been used extensively for anomaly detection. Recent autoencoder-based methods rely on accurate reconstructions of normal images and inaccurate reconstructions of anomalous images [8, 12, 20, 43]. This enables detecting anomalies by comparing the reconstruction to the input image. A common problem are false-positive detections caused by inaccurate reconstructions of normal images, e.g., blurry reconstructions. To avoid this, Bergmann *et al*. [8] let an autoencoder reconstruct images in the feature space of a pretrained network. Furthermore, they train a neural network to predict the output of the autoencoder and thus the systematic reconstruction errors on normal images. We include their proposed method, called GCAD, in our experiments, as it outperforms other methods on the logical anomalies of MVTec LOCO by a large margin. We also include DSR [70], which uses the latent space of a pretrained autoencoder and generates synthetic anomalies in it.

can be applied to arbitrary image size due to being fully convolutional

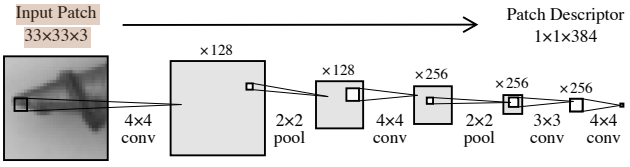


Figure 2. Patch description network (PDN) architecture of EfficientAD-S. Applying it to an image in a fully convolutional manner yields all features in a single forward pass.

3. Method

We describe the components of EfficientAD in the following subsections. It begins with the efficient extraction of features from a pretrained neural network in section 3.1. We detect anomalous features at test time using a reduced student–teacher model, as described in section 3.2. A key challenge is to achieve a competitive anomaly detection performance while keeping the overall runtime low. To this end, we introduce a novel loss function for the training of student–teacher pairs. It improves the detection of anomalies without affecting the computational cost during inference. In section 3.3, we explain how to efficiently detect logical anomalies with an autoencoder-based approach. Finally, we provide a solution for calibrating and combining the detection results of the autoencoder with those of the student–teacher model in section 3.4.

3.1. Efficient Patch Descriptors

Recent anomaly detection methods commonly use the features of a deep pretrained network, such as a WideResNet-101 [50, 69]. We use a network with a drastically reduced depth as a feature extractor. It consists of only four convolutional layers and is visualized in Figure 2. Each output neuron has a receptive field of 33×33 pixels and thus each output feature vector describes a 33×33 patch. Due to this clear correspondence, we refer to the network as a patch description network (PDN). The PDN is fully convolutional and can be applied to an image of variable size to generate all feature vectors in a single forward pass.

The S–T method [10] also uses features from networks with only few convolutional layers. The computational cost of these networks is nevertheless high because of the lack of downsampling in convolutional and pooling layers. The number of parameters of the networks used by S–T is comparably low (between 1.6 and 2.7 million per network). Yet, executing a single network takes longer and requires more memory in our experiments than a U-Net [49] with 31 million parameters, an architecture used by the GCAD method [8]. This demonstrates how the number of parameters can be a misleading proxy metric for the latency, throughput, and memory footprint of a method. Modern classification architectures typically perform downsampling early to re-

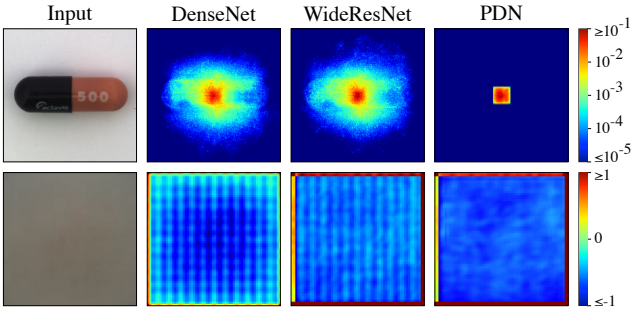


Figure 3. Upper row: absolute gradient of a single feature vector, located in the center of the output, with respect to each input pixel, averaged across input and output channels. Lower row: Average feature map of the first output channel across 1000 randomly chosen images from ImageNet [54]. The mean of these images is shown on the left. The feature maps of the DenseNet [25] and the WideResNet exhibit strong artifacts.

duce the size of feature maps and thus the runtime and memory requirements [23]. We implement this in our PDN via strided average-pooling layers after the first and the second convolutional layer. With the proposed PDN, we are able to obtain the features for an image of size 256×256 in less than 800 µs on an NVIDIA RTX A6000 GPU.

To make the PDN generate expressive features, we distill a deep pretrained classification network into it. For a controlled comparison, we use the same pretrained features as PatchCore [50] from a WideResNet-101. We train the PDN on images from ImageNet [54] by minimizing the mean squared difference between its output and the features extracted from the pretrained network. We provide the full list of training hyperparameters in Appendix A.2. Besides higher efficiency, the PDN has another benefit in comparison to the deep networks used by recent methods. By design, a feature vector generated by the PDN only depends on the pixels in its respective 33×33 patch. The feature vectors of pretrained classifiers, on the other hand, exhibit long-range dependencies on other parts of the image. This is shown in Figure 3, using PatchCore's feature extractors as an example. The well-defined receptive field of the PDN ensures that an anomaly in one part of the image cannot trigger anomalous feature vectors in other, distant parts, which would impair the localization of anomalies.

at least this feature extraction approach seems to be very usefull!

3.2. Reduced Student–Teacher

For detecting anomalous feature vectors, we reduce the S–T method [10] to only one teacher, given by our distilled PDN, and one student. Since we can execute the PDN in under a millisecond, we use its architecture for the student as well, resulting in a low overall latency. This reduced student–teacher pair, however, lacks techniques used by previous methods to increase the anomaly detection performance: ensembling multiple teachers and multiple students [10], using features from a pyramid of layers [65], and using architectural asymmetry between the student and the teacher network [53]. We therefore introduce a training loss that improves the detection of anomalies without affecting the computational requirements at test time.

distilled: dt. destiliert

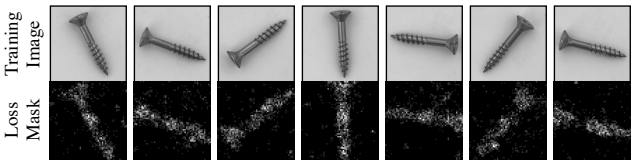


Figure 4. Randomly picked loss masks generated by the hard feature loss during training. The brightness of a mask pixel indicates how many of the dimensions of the respective feature vector were selected for backpropagation. The student network already mimics the teacher well on the background and thus focuses on learning the features of differently rotated screws.

We observe that in the standard S–T framework, increasing the number of training images can improve the student's ability to imitate the teacher on anomalies. This worsens the anomaly detection performance. At the same time, deliberately decreasing the number of training images can suppress important information about normal images. Our goal is to show the student enough data so that it can mimic the teacher sufficiently on normal images while avoiding generalization to anomalous images. Similar to Online Hard Example Mining [60], we therefore restrict the student's loss to the most relevant parts of an image. These are the patches where the student currently mimics the teacher the least. We propose a hard feature loss, which only uses the output elements with the highest loss for backpropagation.

Formally, we apply a teacher T and a student S to a training image I, which yields $T(I) \in \mathbb{R}^{C\times W\times H}$ and $S(I) \in \mathbb{R}^{C\times W\times H}$. We compute the squared difference for each tuple (c, w, h) as $D_{c,w,h} = (T(I)_{c,w,h} - S(I)_{c,w,h})^2$. Based on a mining factor $p_{\text{hard}} \in [0, 1]$, we then compute the p_{hard}-quantile of the elements of D. Given the p_{hard}-quantile d_{hard}, we compute the training loss L_{hard} as the mean of all $D_{c,w,h} \geq d_{\text{hard}}$. Setting p_{hard} to zero would yield the original S–T loss. In our experiments, we set p_{hard} to 0.999, which corresponds to using, on average, ten percent of the values in each of the three dimensions of D for backpropagation. Figure 4 visualizes the effect of the hard feature loss for $p_{\text{hard}} = 0.999$. During inference, the 2D anomaly score map $M \in \mathbb{R}^{W\times H}$ is given by $M_{w,h} = C^{-1}\sum_c D_{c,w,h}$, i.e., by D averaged across channels. It assigns an anomaly score to each feature vector. By using the hard feature loss, we avoid outliers in the anomaly scores on normal images, i.e., false-positive detections.

In addition to the hard feature loss, we use a loss penalty during training that further hinders the student from imitating the teacher on images that are not part of the nor-

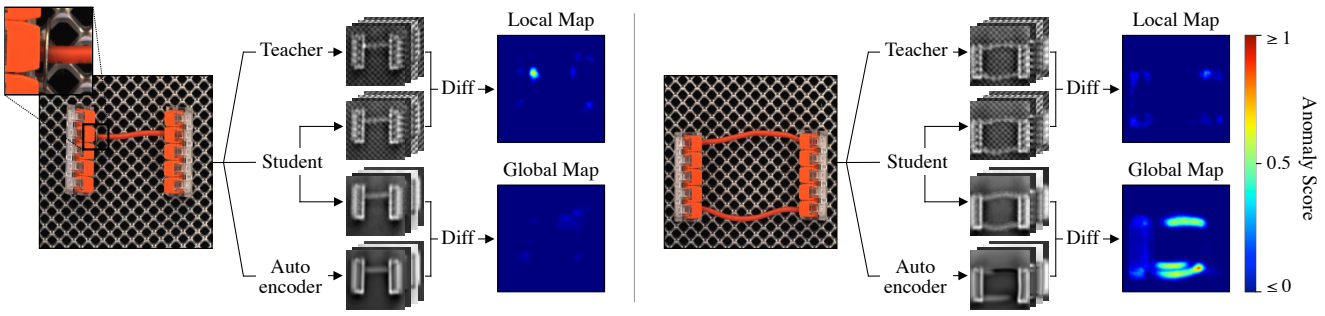


Figure 5. EfficientAD applied to two test images from MVTec LOCO. Normal input images contain a horizontal cable connecting the two splicing connectors at an arbitrary height. The anomaly on the left is a foreign object in the form of a small metal washer at the end of the cable. It is visible in the local anomaly map because the outputs of the student and the teacher differ. The logical anomaly on the right is the presence of a second cable. The autoencoder fails to reconstruct the two cables on the right in the feature space of the teacher. The student also predicts the output of the autoencoder in addition to that of the teacher. Because its receptive field is restricted to small patches of the image, it is not influenced by the presence of the additional red cable. This causes the outputs of the autoencoder and the student to differ. "Diff" refers to computing the element-wise squared difference between two collections of output feature maps and computing its average across feature maps. To obtain pixel anomaly scores, the anomaly maps are resized to the size of the input image using bilinear interpolation.

Standard: Teacher auf ImageNet vortrainieren, um Features von WideResNet101 zu imiterern. Dann Student nur mit Bildern aus der Zieldomäne füttern (und mit Teacher trainieren)

mal training images. In the standard S–T framework, the teacher is pretrained on an image classification dataset, or it is a distilled version of such a pretrained network. The student is not trained on that pretraining dataset but only on the application's normal images. We propose to also use the images from the teacher's pretraining during the training of the student. Specifically, we sample a random image P from the pretraining dataset, in our case ImageNet, in each training step. We compute the loss of the student as $L_{\text{ST}} = L_{\text{hard}} + (CWH)^{-1} \sum_c \|S(P)_c\|_F^2$. This penalty hinders the student from generalizing its imitation of the teacher to out-of-distribution images.

3.3. Logical Anomaly Detection

We cannot rule out the possibility of logical anomalies, such as misplaced objects, and therefore want our method to be able to detect them. As recommended by Bergmann *et al.* in their proposed GCAD method [8], we use an autoencoder for learning logical constraints of the training images and detecting violations of these constraints. Figure 5 depicts the anomaly detection methodology for EfficientAD. It consists of the aforementioned student–teacher pair and an autoencoder. The autoencoder is trained to predict the output of the teacher. Formally, we apply an autoencoder A to a training image I, yielding $A(I) \in \mathbb{R}^{C \times W \times H}$, and compute the loss as $L_{\text{AE}} = (CWH)^{-1} \sum_c \|T(I)_c - A(I)_c\|_F^2$.

In contrast to the patch-based student, the autoencoder must encode and decode the complete image through a bottleneck of 64 latent dimensions. On images with logical anomalies, the autoencoder usually fails to generate the correct latent code for reconstructing the image in the teacher's feature space. However, its reconstructions are also flawed on normal images, as autoencoders generally struggle with reconstructing fine-grained patterns [12, 17]. This is the case for the background grids in Figure 5. Using the difference between the teacher's output and the autoencoder's reconstruction as an anomaly map would cause false-positive detections in these cases. Instead, we double the number of output channels of our student network and train it to predict the output of the autoencoder in addition to the output of the teacher. Let $S'(I) \in \mathbb{R}^{C \times W \times H}$ denote the additional output channels of the student. The student's additional loss is then $L_{\text{STAE}} = (CWH)^{-1} \sum_c \|A(I)_c - S'(I)_c\|_F^2$. The total training loss is the sum of L_{AE}, L_{ST}, and L_{STAE}.

The student learns the systematic reconstruction errors of the autoencoder on normal images, e.g., blurry reconstructions. At the same time, it does not learn the reconstruction errors for anomalies because these are not part of the training set. This makes the difference between the autoencoder's output and the student's output well-suited for computing the anomaly map. Analogous to the student–teacher pair, the anomaly map is the squared difference between the two outputs, averaged across channels. We refer to the this anomaly map as the global anomaly map and to the anomaly map generated by the student–teacher pair as the local anomaly map. We average these two anomaly maps to compute the combined anomaly map and use its maximum value as the image-level anomaly score.

Similar to our student, GCAD includes a separate U-Net [49] model that is trained to predict the output of the autoencoder. Sharing the student instead of using a separate U-Net model reduces the computational cost and increases the asymmetry between the autoencoder and its imitator model. Like the autoencoder, GCAD's U-Net also has a bottleneck with a receptive field as large as the input image. Although its skip connections avoid this, the

U-Net has the capacity to imitate the autoencoder even on logical anomalies. Our student network, on the other hand, operates on 33×33 patches. As illustrated in Figure 5, it is not influenced by logical anomalies that involve long-range dependencies between patches. GCAD contains additional components which we do not include in our method. These are the use of skip connections for training the autoencoder, a delayed activation of L_{STAE} during training, and an upsampling module that maps the output of the autoencoder to the teacher's feature space. Removing these components simplifies the training process and improves the anomaly detection performance, as we show in our experiments.

3.4. Anomaly Map Normalization

The local and the global anomaly map must be normalized to similar scales before averaging them to obtain the combined anomaly map. This is important for cases where the anomaly is only detected in one of the maps, such as in Figure 5. Otherwise, noise in one map could make accurate detections in the other map indiscernible in the combined map. To estimate the scale of the noise in normal images, we use validation images, i.e., unseen images from the training set. For each of the two anomaly map types, we compute the set of all pixel anomaly scores across the validation images. We then compute two p-quantiles for each set: q_a and q_b, for $p = a$ and $p = b$, respectively. We determine a linear transformation that maps q_a to an anomaly score of 0 and q_b to a score of 0.1. At test time, the local and global anomaly maps are normalized with the respective linear transformation.

By using quantiles, the normalization becomes robust to the distribution of anomaly scores on normal images, which can vary between scenarios. Whether the scores between q_a and q_b are normally distributed or a mixture of Gaussians or follow another distribution has no influence on the normalization. Our experiments include an ablation study on the values of a and b. The choice of the mapping destination values 0 and 0.1 has no effect on anomaly detection metrics such as the area under the ROC curve (AU-ROC). That is because the AU-ROC only depends on the ranking of scores, not on their scale. Increasing the value of 0.1 by a factor of n would simply increase the scale of the two anomaly maps and the combined anomaly map by n. We choose 0 and 0.1 because they yield maps that are suitable for a standard zero-to-one color scale.

4. Experiments

We compare EfficientAD to GCAD [8], S–T [10], PatchCore [50], AST [53], FastFlow [68], and DSR [70], using official implementations where available. GCAD consists of an ensemble of two anomaly detection models that use different feature extractors. We find that one of the two ensemble members performs better on average than the combined ensemble and therefore report the results for this member. This reduces the latency reported for GCAD by a factor of two. For PatchCore, we include two variants: the default single model variant, for which the authors report the lowest latency, and the ensemble variant, denoted by PatchCore$_{\text{Ens}}$. We are able to reproduce the official results but disable the cropping of the center 76.6 % of input images for a fair comparison. In the case of MVTec AD, 99.9 % of the defects lie fully or partially within this cropped area. In real-world applications, anomalies can occur outside of this area as well. We disable custom cropping, as it implies knowledge about the anomalies in the test set. For FastFlow, we use the version based on the WideResNet-50-2 feature extractor, as it is similar to the WideResNet used by PatchCore and our method. We use the implementation provided by the Intel anomalib [1] but disable early stopping, i.e., the scenario-specific tuning of the training duration on test images. We provide configuration details for all evaluated methods in Appendix B.

For our method, we evaluate two variants: EfficientAD-S and EfficientAD-M. EfficientAD-S uses the architecture displayed in Figure 2 for the teacher and the student. For EfficientAD-M, we double the number of kernels in the hidden convolutional layers of the teacher and the student. Furthermore, we insert a 1×1 convolution after the second pooling layer and after the last convolutional layer. We provide a list of implementation details, such as the learning rate schedule, in Appendix A.1.

We evaluate each method on the 32 anomaly detection scenarios of MVTec AD, VisA, and MVTec LOCO. The binary anomaly classification performance of a method is measured with the AU-ROC based on its predicted image-level anomaly scores. We measure the anomaly localization performance using the AU-PRO segmentation metric up to a false positive rate of 30 %, as recommended by [7]. For MVTec LOCO, we use the AU-sPRO metric [8], a generalization of the AU-PRO metric for evaluating the localization of logical anomalies. Appendix D provides the results for additional anomaly detection metrics, such as the area under the precision-recall curve and the pixel-wise AU-ROC.

When reporting the AU-ROC or AU-PRO for a dataset collection, we follow the policy of the dataset authors. For each collection, we evaluate the respective metric for each scenario and then compute the mean across scenarios. For MVTec LOCO, we use the official evaluation script, which gives logical and structural anomalies an equal weight in the computed metrics. When reporting the average AU-ROC or AU-PRO on the three dataset collections, we compute the average of the three dataset means. Thus, an overall average score weights logical anomalies and structural anomalies by roughly one-sixth and five-sixths, respectively. We provide the evaluation results for each of the 32 anomaly detection scenarios individually in the appendix to enable

Method	Classif. AU-ROC	Segment. AU-PRO	Latency [ms]	Throughput [img / s]
GCAD	85.4	88.0	11	121
S–T	88.4	89.7	75	16
FastFlow	90.0	86.5	17	120
DSR	90.8	78.6	17	104
PatchCore	91.1	80.9	32	76
$\text{PatchCore}_{\text{Ens}}$	92.1	80.7	148	13
AST	92.4	77.2	53	41
EfficientAD-S	95.4 (± 0.06)	92.5 (± 0.05)	**2.2** (± 0.01)	**614** (± 2)
EfficientAD-M	**96.0** (± 0.09)	**93.3** (± 0.04)	4.5 (± 0.01)	269 (± 1)

Table 1. Anomaly classification and anomaly localization performance in comparison to the latency and throughput. Each AU-ROC and AU-PRO percentage is an average of the mean AU-ROCs and mean AU-PROs, respectively, on MVTec AD, VisA, and MVTec LOCO. For EfficientAD, we report the mean and standard deviation of five runs.

Method	MAD	LOCO	VisA	Mean	LOCO Logic.	LOCO Struct.
GCAD	89.1	83.3	83.7	85.4	83.9	82.7
S–T	93.2	77.4	94.6	88.4	66.5	88.3
FastFlow	96.9	79.2	93.9	90.0	75.5	82.9
DSR	98.1	82.6	91.8	90.8	75.0	90.2
PatchCore	98.7	80.3	94.3	91.1	75.8	84.8
$\text{PatchCore}_{\text{Ens}}$	**99.3**	79.4	97.7	92.1	71.0	87.7
AST	98.9	83.4	94.9	92.4	79.7	87.1
EfficientAD-S	98.8	90.0	97.5	95.4	85.8	94.1
EfficientAD-M	99.1	**90.7**	**98.1**	**96.0**	**86.8**	**94.7**

Table 2. Mean anomaly classification AU-ROC percentages per dataset collection (left) and on the logical and structural anomalies of MVTec LOCO (right). For EfficientAD, we report the mean of five runs. EfficientAD matches the logical anomaly detection performance of GCAD without compromising the detection of structural anomalies. Performing method development solely on MVTec AD (MAD) becomes prone to overfitting design choices to the few remaining misclassified test images.

p_{hard}	0	0.9	0.99	**0.999**	0.9999	0.99999
AU-ROC	94.9	94.9	95.7	96.0	95.8	95.7
a (for q_a)	0.5	0.8	**0.9**	0.95	0.98	0.99
AU-ROC	95.9	95.9	96.0	95.9	95.9	95.8
b (for q_b)	0.95	0.98	0.99	**0.995**	0.998	0.999
AU-ROC	95.8	95.9	96.0	96.0	95.9	95.9

Table 3. Mean anomaly classification AU-ROC of EfficientAD-M on MVTec AD, VisA, and MVTec LOCO when varying the locations of quantiles. These are the mining factor p_{hard} and the two sampling points a and b of the quantile-based map normalization. Default values used in our experiments are highlighted in bold.

an evaluation with a custom weighting.

Table 1 reports the overall anomaly detection performance for each method. EfficientAD achieves a strong detection and localization of anomalies. Reliably localizing anomalies in an image provides explainable detection results and allows the discovery of spurious correlations in detections. It also enables a flexible postprocessing, such as excluding defect segmentations based on their size.

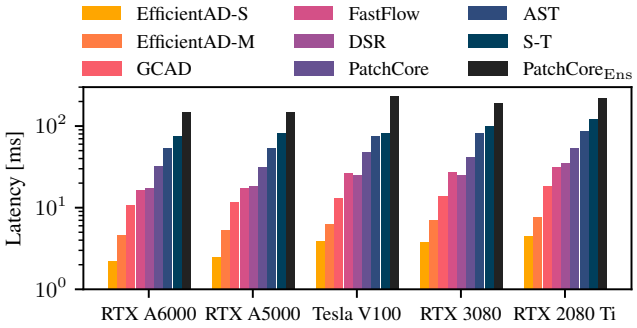


Figure 6. Latency per GPU. The ranking of methods is the same on each GPU, except for two cases in which DSR is slightly faster than FastFlow.

Table 2 breaks down the overall anomaly classification performance into the three dataset collections. It shows that the lead of EfficientAD on MVTec LOCO is in equal parts due to its performance on logical and on structural anomalies. In Table 3, we assess the robustness of EfficientAD to varying hyperparameters.

Furthermore, we measure the computational cost of each method during inference. As explained above, the number of parameters can be a misleading proxy metric for the latency and throughput of convolutional architectures since it does not consider the resolution of a convolution's input feature map, i.e., how often a parameter is used in a forward pass. Similarly, the number of floating point operations (FLOPs) can be misleading since it does not take into account how easily computations can be parallelized. For transparency, we report the number of parameters, the number of FLOPs, and the memory footprint of each method in Appendix E. Here, we focus on the metrics that are most relevant in anomaly detection applications: the latency and the throughput. We measure the latency with a batch size of 1 and the throughput with a batch size of 16. Table 1 reports the measurements for each method on an NVIDIA RTX A6000 GPU. Figure 6 shows the latency of each method on each of the GPUs in our experimental setup. Appendix E contains a detailed description of our timing methodology.

We examine the effects of our design choices in the ablation study shown in Table 4, using GCAD as a starting point. We train EfficientAD for 70 000 iterations, which is roughly 40 % of the training duration of GCAD. This allows EfficientAD-S to achieve a training time of less than twenty minutes. For a controlled comparison of subsequent changes, we report the effect on the anomaly detection performance in a separate row called "Shorter training." We

shared weights

in a CNN

	Classif. AU-ROC	Segment. AU-PRO	Latency [ms]
GCAD	85.4	88.0	10.9
Shorter training	85.6	87.5	10.9
Patch-based students	88.6	90.5	11.9
Simplified autoencoder	90.3	90.2	12.2
Upsampling network	91.0	91.0	12.9
PDN architecture	91.4	91.0	2.7
Hard feature loss	92.7	90.9	2.7
Pretraining penalty	93.1	91.3	2.7
Map normalization	94.4	92.0	2.7
Reduced augmentation	95.2	91.8	2.7
Dropout	95.4	91.9	2.7
Shared student	95.2	91.9	2.2
WRN distillation	95.4	92.5	2.2
EfficientAD-S	95.4	92.5	2.2
EfficientAD-M	96.0	93.3	4.5

Table 4. Ablation study on the methodical differences between EfficientAD and GCAD. Each AU-ROC and AU-PRO percentage is an average of the mean AU-ROCs and mean AU-PROs, respectively, on MVTec AD, VisA, and MVTec LOCO. The latency is measured on an NVIDIA RTX A6000 GPU.

then change the architecture of the two student networks of GCAD from a U-Net to the patch-based architecture of GCAD's teacher network. GCAD's local branch thus becomes a symmetrical student–teacher pair. The anomaly detection performance at this point represents the performance of simply combining GCAD with a more efficient version of S–T. Next, we simplify the architecture and training of the autoencoder. We replace the transposed convolutions in its decoder with bilinear upsampling and regular convolutional layers, as described in [41]. Detailed layer parameters are provided in Appendix A.1. This change allows us to remove the use of skip connections during the training of the autoencoder and the delay of the training of the autoencoder's student network. This further improves the anomaly detection performance ("Simplified autoencoder" row), as does the removal of the upsampling network used by GCAD. The previous changes, however, also increase the model's runtime. To reduce it again, we change the architecture of the patch-based networks to that of the PDN shown in Figure 2.

We evaluate the effect of the two proposed loss terms for training the student–teacher pair. The hard feature loss causes the student's training to focus on the features with the highest loss. These are also the most relevant features for the anomaly classification because we use the maximum pixel score as the image anomaly score. The hard feature loss significantly improves the anomaly classification performance, which is further increased by the student's penalty on pretraining images. Next, we replace GCAD's anomaly map normalization with our proposed quantile-based normalization. GCAD computes normalization parameters such that anomaly scores of normal images have a mean of zero and a variance of one. This makes it sensitive to the distribution of validation anomaly scores, which can vary between scenarios. The quantile-based normalization is independent of how the scores between q_a and q_b are distributed. It improves both the anomaly detection and the anomaly localization performance.

We switch from the scenario-specific augmentation parameters used by GCAD to a strongly reduced augmentation. We disable augmentation in the student–teacher training to avoid teaching the student how to generalize to anomalous images. For the autoencoder, we augment training images only with color jitter [44] and insert dropout layers in the decoder to avoid overfitting. To further decrease the latency of our method, we let one shared student network predict the output of both the teacher and the autoencoder. Finally, we switch from the teacher pretraining of S–T to the distillation of PatchCore's WideResNet-101, as described above, yielding EfficientAD-S. We find that the anomaly detection performance is robust to the choice of the distillation backbone and provide a comparison in Appendix C.

5. Conclusion

In this paper, we introduce EfficientAD, a method with a strong anomaly detection performance and a high computational efficiency. It sets new standards for the detection of structural as well as logical anomalies. Both EfficientAD-S and EfficientAD-M outperform other methods on the classification and the localization of anomalies by a large margin. Compared to AST, the second-best method, EfficientAD-S reduces the latency by a factor of 24 and increases the throughput by a factor of 15. Its low latency, high throughput, and high detection rate make it suitable for real-world applications. For future anomaly detection research, EfficientAD is an important baseline and a fruitful foundation. Its efficient patch description network, for instance, can be used as a feature extractor in other anomaly detection methods as well to reduce their latency.

Limitations. The student–teacher model and the autoencoder are designed to detect anomalies of different types. The autoencoder detects logical anomalies, while the student–teacher model detects coarse and fine-grained structural anomalies. Fine-grained logical anomalies, however, remain a challenge – for example a screw that is two millimeters too long. To detect these, practitioners would have to use traditional metrology methods [61]. As for the limitations in comparison to other recent anomaly detection methods: In contrast to kNN-based methods, our approach requires training, especially for the autoencoder to learn the logical constraints of normal images. This takes twenty minutes in our experimental setup.

hier könnte noch Potential liegen

References

[1] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A deep learning library for anomaly detection. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1706–1710. IEEE, 2022. 6, 17

[2] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pages 622–637. Springer, 2019. 3

[3] Samuel G Armato III, Geoffrey McLennan, Luc Bidaut, Michael F McNitt-Gray, Charles R Meyer, Anthony P Reeves, Binsheng Zhao, Denise R Aberle, Claudia I Henschke, Eric A Hoffman, et al. The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans. *Medical physics*, 38(2):915–931, 2011. 2

[4] Donald Bailey. *Implementing Machine Vision Systems Using FPGAs*, pages 1103–1136 in "Machine Vision Handbook" by Bruce G. Batchelor. Springer London, London, 2012. 1

[5] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S. Kirby, et al. Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features. *Scientific Data*, 4(1), 2017. 2

[6] Christoph Baur, Benedikt Wiestler, Shadi Albarqouni, and Nassir Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 161–169. Springer International Publishing, 2019. 3

[7] Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, and Carsten Steger. The MVTec Anomaly Detection Dataset: A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. *International Journal of Computer Vision*, 129(4):1038–1059, 2021. 1, 2, 6, 14, 17, 19

[8] Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, and Carsten Steger. Beyond Dents and Scratches: Logical Constraints in Unsupervised Anomaly Detection and Localization. *International Journal of Computer Vision*, 130(4):947—-969, 2022. 1, 2, 3, 5, 6, 14, 17

[9] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9584–9592, 2019. 1, 2, 14

[10] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed Students: Student-Teacher Anomaly Detection With Discriminative Latent Embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4182–4191, 2020. 2, 3, 4, 6, 17

[11] Paul Bergmann, Xin Jin, David Sattlegger, and Carsten Steger. The MVTec 3D-AD Dataset for Unsupervised 3D Anomaly Detection and Localization. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*, pages 202–213. INSTICC, SciTePress, 2022. 2

[12] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving Unsupervised Defect Segmentation by Applying Structural Similarity to Autoencoders. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*, pages 372–380. INSTICC, SciTePress, 2019. 3, 5

[13] Hermann Blum, Paul-Edouard Sarlin, Juan Nieto, Roland Siegwart, and Cesar Cadena. Fishyscapes: A Benchmark for Safe Semantic Segmentation in Autonomous Driving. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2403–2412, 2019. 2

[14] Niv Cohen and Yedid Hoshen. Sub-image anomaly detection with deep pyramid correspondences. *arXiv preprint arXiv:2005.02357v1*, 2020. 3

[15] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006. 19

[16] 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. 3

[17] Alexey Dosovitskiy and Thomas Brox. Generating Images with Perceptual Similarity Metrics based on Deep Networks. In *Advances in Neural Information Processing Systems*, pages 658–666, 2016. 5

[18] Thibaud Ehret, Axel Davy, Jean-Michel Morel, and Mauricio Delbracio. Image Anomalies: A Review and Synthesis of Detection Methods. *Journal of Mathematical Imaging and Vision*, 61(5):710–743, 2019. 2

[19] Tharindu Fernando, Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Deep learning for medical anomaly detection – a survey. *ACM Computing Surveys*, 54(7), 2021. 2

[20] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton Van Den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1705–1714, 2019. 3

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014. 3

[22] Denis Gudovskiy, Shun Ishizaka, and Kazuki Kozuka. CFLOW-AD: Real-Time Unsupervised Anomaly Detection With Localization via Conditional Normalizing Flows. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 98–107, 2022. 3

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 4

[24] Dan Hendrycks, Steven Basart, Mantas Mazeika, Andy Zou, Joseph Kwon, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. Scaling out-of-distribution detection for real-world settings. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 8759–8773. PMLR, 17–23 Jul 2022. 2

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 4, 17

[26] Yibin Huang, Congying Qiu, Yue Guo, Xiaonan Wang, and Kui Yuan. Surface defect saliency of magnetic tile. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 612–617, 2018. 2

[27] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, et al. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 590–597, 2019. 2

[28] Stepan Jezek, Martin Jonak, Radim Burget, Pavel Dvorak, and Milos Skotak. Deep learning-based defect detection of metal parts: evaluating current methods in complex conditions. In *2021 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 66–71. IEEE, 2021. 2

[29] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015. 13, 16

[30] Chun-Liang Li, Kihyuk Sohn, Jinsung Yoon, and Tomas Pfister. Cutpaste: Self-supervised learning for anomaly detection and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9664–9674, 2021. 3

[31] Wei-Xin Li, Vijay Mahadevan, and Nuno Vasconcelos. Anomaly Detection and Localization in Crowded Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(1):18–32, 2013. 2

[32] Krzysztof Lis, Krishna Kanth Nakka, Pascal Fua, and Mathieu Salzmann. Detecting the unexpected via image resynthesis. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2152–2161, 2019. 2

[33] Wenqian Liu, Runze Li, Meng Zheng, Srikrishna Karanam, Ziyan Wu, Bir Bhanu, Richard J. Radke, and Octavia Camps. Towards visually explaining variational autoencoders. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8639–8648, 2020. 3

[34] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 1

[35] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 1

[36] Cewu Lu, Jianping Shi, and Jiaya Jia. Abnormal Event Detection at 150 FPS in MATLAB. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2720–2727, 2013. 2

[37] Bjoern H. Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, et al. The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, 2015. 2

[38] Pankaj Mishra, Riccardo Verk, Daniele Fornasier, Claudio Piciarelli, and Gian Luca Foresti. Vt-adl: A vision transformer network for image anomaly detection and localization. In *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pages 01–06. IEEE, 2021. 2, 3

[39] Paolo Napoletano, Flavio Piccoli, and Raimondo Schettini. Anomaly Detection in Nanofibrous Materials by CNN-Based Self-Similarity. *Sensors*, 18(1):209, 2018. 3

[40] Tiago S Nazare, Rodrigo F de Mello, and Moacir A Ponti. Are pre-trained cnns good feature extractors for anomaly detection in surveillance videos? *arXiv preprint arXiv:1811.08495v1*, 2018. 3

[41] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. 8

[42] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), mar 2021. 2

[43] Hyunjong Park, Jongyoun Noh, and Bumsub Ham. Learning memory-guided normality for anomaly detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14360–14369, 2020. 3

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 8, 13, 14, 15, 16, 22

[45] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. Ocgan: One-class novelty detection using gans with constrained latent representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2893–2901, 2019. 3

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[47] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 3

[48] Oliver Rippel., Arnav Chavan., Chucai Lei., and Dorit Merhof. Transfer learning gaussian anomaly detection by fine-tuning representations. In *Proceedings of the 2nd International Conference on Image Processing and Vision Engineering - IMPROVE,*, pages 45–56. INSTICC, SciTePress, 2022. 3

[49] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015. 3, 5

[50] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14318–14328, 2022. 1, 2, 3, 4, 6, 15, 16, 17

[51] Marco Rudolph, Bastian Wandt, and Bodo Rosenhahn. Same same but differnet: Semi-supervised defect detection with normalizing flows. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1906–1915, 2021. 3

[52] Marco Rudolph, Tom Wehrbein, Bodo Rosenhahn, and Bastian Wandt. Fully convolutional cross-scale-flows for image-based defect detection. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1829–1838, 2022. 3

[53] Marco Rudolph, Tom Wehrbein, Bodo Rosenhahn, and Bastian Wandt. Asymmetric student-teacher networks for industrial anomaly detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2592–2602, 2023. 1, 2, 3, 4, 6, 17

[54] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 4, 13, 16

[55] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, page 4–11, New York, NY, USA, 2014. Association for Computing Machinery. 3

[56] Mohammadreza Salehi, Niousha Sadjadi, Soroosh Baselizadeh, Mohammad H. Rohban, and Hamid R. Rabiee. Multiresolution knowledge distillation for anomaly detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14897–14907, 2021. 3

[57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 1

[58] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017. 3

[59] Thomas Schlegl, Philipp Seeböck, Sebastian Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. f-AnoGAN: Fast Unsupervised Anomaly Detection with Generative Adversarial Networks. *Medical Image Analysis*, 54, 2019. 3

[60] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016. 4

[61] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine Vision Algorithms and Applications*. Wiley-VCH, Weinheim, 2nd edition, 2018. 2, 8

[62] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1

[63] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021. 1

[64] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020. 1

[65] Guodong Wang, Shumin Han, Errui Ding, and Di Huang. Student-teacher feature pyramid matching for anomaly detection. In *The British Machine Vision Conference (BMVC)*, 2021. 2, 3, 4

[66] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 17

[67] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818, 2022. 1

[68] 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. *arXiv preprint arXiv:2111.07677v1*, 2021. 2, 3, 6, 17

[69] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016. 3, 15

[70] Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. Dsr–a dual subspace re-projection network for surface anomaly detection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXI*, pages 539–554. Springer, 2022. 3, 6, 17

[71] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 589–597, 2016. 2

[72] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018. 3

[73] Yang Zou, Jongheon Jeong, Latha Pemula, Dongqing Zhang, and Onkar Dabeer. Spot-the-difference self-supervised pre-training for anomaly detection and segmentation. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXX*, pages 392–408. Springer, 2022. 1, 2, 14

Appendices

We provide the following supplementary material:

- (A) Implementation details for EfficientAD, including the training and inference procedure of EfficientAD on the dataset of an anomaly detection scenario (A.1) and the distillation training of the patch description network (A.2).
- (B) Implementation and configuration details for other evaluated methods.
- (C) Evaluation of the anomaly detection performance of EfficientAD for different distillation backbones.
- (D) Results for additional anomaly detection metrics, such as the area under the precision recall curve.
- (E) Description of our timing methodology and results for additional computational efficiency metrics such as the number of parameters.
- (F) Qualitative results in the form of anomaly maps generated by EfficientAD and other methods on the evaluated datasets.
- Anomaly detection results for each method on each of the 32 scenarios from MVTec AD, VisA, and MVTec LOCO in the `per_scenario_results.json` file [1].

Appendix A. Implementation Details for EfficientAD

A.1. Training and Inference

Algorithm 1 describes the training of EfficientAD-S and Algorithm 2 explains the inference procedure. For EfficientAD-M, replace the architecture of Table 5 with that of Table 6.

Algorithm 1 EfficientAD-S Training Algorithm

Require: A pretrained teacher network $T : \mathbb{R}^{3\times256\times256} \to \mathbb{R}^{384\times64\times64}$ with an architecture as given in Table 5
Require: A sequence of training images $\mathcal{I}_{\text{train}}$ with $I_{\text{train}} \in \mathbb{R}^{3\times256\times256}$ for each $I_{\text{train}} \in \mathcal{I}_{\text{train}}$
Require: A sequence of validation images $\mathcal{I}_{\text{val}}$ with $I_{\text{val}} \in \mathbb{R}^{3\times256\times256}$ for each $I_{\text{val}} \in \mathcal{I}_{\text{val}}$
Randomly initialize a student network $S : \mathbb{R}^{3\times256\times256} \to \mathbb{R}^{768\times64\times64}$ with an architecture as given in Table 5
Randomly initialize an autoencoder $A : \mathbb{R}^{3\times256\times256} \to \mathbb{R}^{384\times64\times64}$ with an architecture as given in Table 7
for $c \in 1, \ldots, 384$ **do** ▷ Compute teacher channel normalization parameters $\mu \in \mathbb{R}^{384}$ and $\sigma \in \mathbb{R}^{384}$
    Initialize an empty sequence $X \leftarrow (\,)$
    for $I_{\text{train}} \in \mathcal{I}_{\text{train}}$ **do**
        $Y' \leftarrow T(I_{\text{train}})$
        $X \leftarrow X \frown \text{vec}(Y'_c)$ ▷ Append the channel output to X
    end for
    Set μ_c to the mean and σ_c to the standard deviation of the elements of X

[1] https://www.mydrive.ch/shares/71140/bbca60ead1194717e488313f1632504c/download/444261995-1678948469/per_scenario_results.json

end for
Initialize Adam [29] with a learning rate of 10^{-4} and a weight decay of 10^{-5} for the parameters of S and A
for iteration $= 1, \ldots, 70\,000$ **do**
    Choose a random training image I_{train} from $\mathcal{I}_{\text{train}}$
    $Y' \leftarrow T(I_{\text{train}})$ ▷ Forward pass of the student–teacher pair
    Compute the normalized teacher output $\hat{Y}$ given by $\hat{Y}_c = (Y'_c - \mu_c)\sigma_c^{-1}$ for each $c \in \{1, \ldots, 384\}$
    $Y^{\text{S}} \leftarrow S(I_{\text{train}})$
    Set $Y^{\text{ST}} \in \mathbb{R}^{384\times64\times64}$ to the first 384 channels of $Y^{\text{S}} \in \mathbb{R}^{768\times64\times64}$
    Compute the squared difference between $\hat{Y}$ and Y^{ST} for each tuple (c, w, h) as $D^{\text{ST}}_{c,w,h} = (\hat{Y}_{c,w,h} - Y^{\text{ST}}_{c,w,h})^2$
    Compute the 0.999-quantile of the elements of D^{ST}, denoted by d_{hard}
    Compute the loss L_{hard} as the mean of all $D^{\text{ST}}_{c,w,h} \geq d_{\text{hard}}$
    Choose a random pretraining image $P \in \mathbb{R}^{3\times256\times256}$ from ImageNet [54]
    Compute the loss $L_{\text{ST}} = L_{\text{hard}} + (384 \cdot 64 \cdot 64)^{-1} \sum_{c=1}^{384} \|S(P)_c\|_F^2$
    Randomly choose an augmentation index $i_{\text{aug}} \in \{1, 2, 3\}$ ▷ Augment I_{train} for A using torchvision [44]
    Sample an augmentation coefficient λ from the uniform distribution $U(0.8, 1.2)$
    if $i_{\text{aug}} == 1$ **then** $I_{\text{aug}} \leftarrow$ `torchvision.transforms.functional_pil.adjust_brightness`$(I_{\text{train}}, \lambda)$
    else if $i_{\text{aug}} == 2$ **then** $I_{\text{aug}} \leftarrow$ `torchvision.transforms.functional_pil.adjust_contrast`$(I_{\text{train}}, \lambda)$
    else if $i_{\text{aug}} == 3$ **then** $I_{\text{aug}} \leftarrow$ `torchvision.transforms.functional_pil.adjust_saturation`$(I_{\text{train}}, \lambda)$
    end if
    $Y^{\text{A}} \leftarrow A(I_{\text{aug}})$ ▷ Forward pass of the autoencoder–student pair
    $Y' \leftarrow T(I_{\text{aug}})$
    Compute the normalized teacher output $\hat{Y}$ given by $\hat{Y}_c = \sigma_c^{-1}(Y'_c - \mu_c)$ for each $c \in \{1, \ldots, 384\}$
    $Y^{\text{S}} \leftarrow S(I_{\text{aug}})$
    Set $Y^{\text{STAE}} \in \mathbb{R}^{384\times64\times64}$ to the last 384 channels of $Y^{\text{S}} \in \mathbb{R}^{768\times64\times64}$
    Compute the squared difference between $\hat{Y}$ and Y^{A} for each tuple (c, w, h) as $D^{\text{AE}}_{c,w,h} = (\hat{Y}_{c,w,h} - Y^{\text{A}}_{c,w,h})^2$
    Compute the squared difference between Y^{A} and Y^{STAE} for each tuple (c, w, h) as $D^{\text{STAE}}_{c,w,h} = (Y^{\text{A}}_{c,w,h} - Y^{\text{STAE}}_{c,w,h})^2$
    Compute the loss L_{AE} as the mean of all elements $D^{\text{AE}}_{c,w,h}$ of D^{AE}
    Compute the loss L_{STAE} as the mean of all elements $D^{\text{STAE}}_{c,w,h}$ of D^{STAE}
    Compute the total loss $L_{\text{total}} = L_{\text{ST}} + L_{\text{AE}} + L_{\text{STAE}}$ ▷ Backward pass
    Update the union of the parameters of S and A, denoted by ϕ, using the gradient $\nabla_\phi L_{\text{total}}$
    if iteration $> 66\,500$ **then**
        Decay the learning rate to 10^{-5}
    end if
end for
Initialize empty sequences $X_{\text{ST}} \leftarrow (\,)$ and $X_{\text{AE}} \leftarrow (\,)$ ▷ Quantile-based map normalization on validation images
for $I_{\text{val}} \in \mathcal{I}_{\text{val}}$ **do**
    $Y' \leftarrow T(I_{\text{val}})$, $Y^{\text{S}} \leftarrow S(I_{\text{val}})$, $Y^{\text{A}} \leftarrow A(I_{\text{val}})$
    Compute the normalized teacher output $\hat{Y}$ given by $\hat{Y}_c = (Y'_c - \mu_c)\sigma_c^{-1}$ for each $c \in \{1, \ldots, 384\}$
    Split the student output into $Y^{\text{ST}} \in \mathbb{R}^{384\times64\times64}$ and $Y^{\text{STAE}} \in \mathbb{R}^{384\times64\times64}$ as above
    Compute the squared difference $D^{\text{ST}}_{c,w,h} = (\hat{Y}_{c,w,h} - Y^{\text{ST}}_{c,w,h})^2$ for each tuple (c, w, h)
    Compute the squared difference $D^{\text{STAE}}_{c,w,h} = (Y^{\text{A}}_{c,w,h} - Y^{\text{STAE}}_{c,w,h})^2$ for each tuple (c, w, h)
    Compute the anomaly maps $M_{\text{ST}} = 384^{-1} \sum_{c=1}^{384} D^{\text{ST}}_c$ and $M_{\text{AE}} = 384^{-1} \sum_{c=1}^{384} D^{\text{STAE}}_c$
    Resize M_{ST} and M_{AE} to 256×256 pixels using bilinear interpolation
    $X_{\text{ST}} \leftarrow X_{\text{ST}} \frown \text{vec}(M_{\text{ST}})$ ▷ Append to the sequence of local anomaly scores
    $X_{\text{AE}} \leftarrow X_{\text{AE}} \frown \text{vec}(M_{\text{AE}})$ ▷ Append to the sequence of global anomaly scores
end for
Compute the 0.9-quantile q_a^{ST} and the 0.995-quantile q_b^{ST} of the elements of X_{ST}.
Compute the 0.9-quantile q_a^{AE} and the 0.995-quantile q_b^{AE} of the elements of X_{AE}.
return T, S, A, μ, σ, q_a^{ST}, q_b^{ST}, q_a^{AE}, and q_b^{AE}

Algorithm 2 EfficientAD Inference Procedure

Require: T, S, A, μ, σ, q_a^{ST}, q_b^{ST}, q_a^{AE}, and q_b^{AE}, as returned by Algorithm 1
Require: Test image $I_{\mathrm{test}} \in \mathbb{R}^{3\times256\times256}$
$Y' \leftarrow T(I_{\mathrm{test}})$, $Y^{\mathrm{S}} \leftarrow S(I_{\mathrm{test}})$, $Y^{\mathrm{A}} \leftarrow A(I_{\mathrm{test}})$
Compute the normalized teacher output $\hat{Y}$ given by $\hat{Y}_c = (Y'_c - \mu_c)\sigma_c^{-1}$ for each $c \in \{1, \ldots, 384\}$
Split the student output into $Y^{\mathrm{ST}} \in \mathbb{R}^{384\times64\times64}$ and $Y^{\mathrm{STAE}} \in \mathbb{R}^{384\times64\times64}$ as above
Compute the squared difference $D^{\mathrm{ST}}_{c,w,h} = (\hat{Y}_{c,w,h} - Y^{\mathrm{ST}}_{c,w,h})^2$ for each tuple (c, w, h)
Compute the squared difference $D^{\mathrm{STAE}}_{c,w,h} = (Y^{\mathrm{A}}_{c,w,h} - Y^{\mathrm{STAE}}_{c,w,h})^2$ for each tuple (c, w, h)
Compute the anomaly maps $M_{\mathrm{ST}} = 384^{-1}\sum_{c=1}^{384} D^{\mathrm{ST}}_c$ and $M_{\mathrm{AE}} = 384^{-1}\sum_{c=1}^{384} D^{\mathrm{STAE}}_c$
Resize M_{ST} and M_{AE} to 256×256 pixels using bilinear interpolation
Compute the normalized $\hat{M}_{\mathrm{ST}} = 0.1(M_{\mathrm{ST}} - q_a^{\mathrm{ST}})(q_b^{\mathrm{ST}} - q_a^{\mathrm{ST}})^{-1}$
Compute the normalized $\hat{M}_{\mathrm{AE}} = 0.1(M_{\mathrm{AE}} - q_a^{\mathrm{AE}})(q_b^{\mathrm{AE}} - q_a^{\mathrm{AE}})^{-1}$
Compute the combined anomaly map $M = 0.5\hat{M}_{\mathrm{ST}} + 0.5\hat{M}_{\mathrm{AE}}$
Compute the image-level score as $m_{\mathrm{image}} = \max_{i,j} M_{i,j}$
return M and m_{image}

Layer Name	Stride	Kernel Size	Number of Kernels	Padding	Activation
Conv-1	1×1	4×4	128	3	ReLU
AvgPool-1	2×2	2×2	128	1	-
Conv-2	1×1	4×4	256	3	ReLU
AvgPool-2	2×2	2×2	256	1	-
Conv-3	1×1	3×3	256	1	ReLU
Conv-4	1×1	4×4	384	0	-

Table 5. Patch description network architecture of the teacher network for EfficientAD-S. The student network has the same architecture, but 768 kernels instead of 384 in the Conv-4 layer. A padding value of 3 means that three rows, or columns respectively, of zeros are appended at each border of an input feature map.

Layer Name	Stride	Kernel Size	Number of Kernels	Padding	Activation
Conv-1	1×1	4×4	256	3	ReLU
AvgPool-1	2×2	2×2	256	1	-
Conv-2	1×1	4×4	512	3	ReLU
AvgPool-2	2×2	2×2	512	1	-
Conv-3	1×1	1×1	512	0	ReLU
Conv-4	1×1	3×3	512	1	ReLU
Conv-5	1×1	4×4	384	0	ReLU
Conv-6	1×1	1×1	384	0	-

Table 6. Patch description network architecture of the teacher network for EfficientAD-M. The student network has the same architecture, but 768 kernels instead of 384 in the Conv-5 and Conv-6 layers. A padding value of 3 means that three rows, or columns respectively, of zeros are appended at each border of an input feature map.

Comments on Algorithm 1 and Algorithm 2:

- We use the default initialization method of PyTorch [44] (version 1.12.0) for the convolutional layers.
- We apply the teacher and the student to both the original and the augmented training image. That is necessary because the student–teacher model is trained without augmentation, while the autoencoder is trained with augmentation. During inference, we do not need these second forward passes because images are not augmented at test time.
- The sizes of the images of MVTec AD [7, 9], VisA [73], and MVTec LOCO [8] differ. We resize each input image to 256×256 and resize the anomaly map M back to the original image size using bilinear interpolation.
- We use a batch size of one.

Layer Name	Stride	Kernel Size	Number of Kernels	Padding	Activation
EncConv-1	2×2	4×4	32	1	ReLU
EncConv-2	2×2	4×4	32	1	ReLU
EncConv-3	2×2	4×4	64	1	ReLU
EncConv-4	2×2	4×4	64	1	ReLU
EncConv-5	2×2	4×4	64	1	ReLU
EncConv-6	1×1	8×8	64	0	-
Bilinear-1	Resizes the 1×1 input features maps to 3×3				
DecConv-1	1×1	4×4	64	2	ReLU
Dropout-1	Dropout rate = 0.2				
Bilinear-2	Resizes the 4×4 input features maps to 8×8				
DecConv-2	1×1	4×4	64	2	ReLU
Dropout-2	Dropout rate = 0.2				
Bilinear-3	Resizes the 9×9 input features maps to 15×15				
DecConv-3	1×1	4×4	64	2	ReLU
Dropout-3	Dropout rate = 0.2				
Bilinear-4	Resizes the 16×16 input features maps to 32×32				
DecConv-4	1×1	4×4	64	2	ReLU
Dropout-4	Dropout rate = 0.2				
Bilinear-5	Resizes the 33×33 input features maps to 63×63				
DecConv-5	1×1	4×4	64	2	ReLU
Dropout-5	Dropout rate = 0.2				
Bilinear-6	Resizes the 64×64 input features maps to 127×127				
DecConv-6	1×1	4×4	64	2	ReLU
Dropout-6	Dropout rate = 0.2				
Bilinear-7	Resizes the 128×128 input features maps to 64×64				
DecConv-7	1×1	3×3	64	1	ReLU
DecConv-8	1×1	3×3	384	1	-

Table 7. Network architecture of the autoencoder for EfficientAD-S and EfficientAD-M. Layers named "EncConv" and "DecConv" are standard 2D convolutional layers.

- We use the image normalization of the pretrained models of torchvision [44]. That means we subtract 0.485, 0.456, and 0.406 from the R, G, and B channel, respectively, for each input image and divide the channels by 0.229, 0.224, and 0.225, respectively. We perform this normalization directly before applying a network to an image, i.e., after augmentation. At test time, this can also be done by adjusting the weights and bias of the first convolutional layer of a network accordingly.
- The parameters of the autoencoder A are not only affected by the gradient of L_{AE}, but also by the gradient of L_{STAE}.
- We obtain an image $P \in \mathbb{R}^{3\times256\times256}$ from ImageNet by choosing a random image, resizing it to 512×512, converting it to gray scale with a probability of 0.3, and cropping the center 256×256 pixels.

A.2. Distillation

In the following, we describe how to distill the WideResNet-101 [69] features used by PatchCore [50] into the teacher network T. The distillation training algorithm is presented in Algorithm 3. The process in analogous for other pretrained feature extractors.

There are only few requirements regarding the output shape of the feature extractor. The feature extractors used by PatchCore output features of shape $384 \times 64 \times 64$ for an input image size of 512×512 pixels. Therefore, the teacher and the autoencoder also output 384 channels (as described in Tables 5 to 7). If a pretrained feature extractor outputs a different number of channels, this default of 384 output channels of the teacher and the autoencoder can be adjusted flexibly. During distillation, we resize input images to 512×512 for the pretrained feature extractor and to 256×256 for the teacher network that is being trained. This results in an output shape of $384 \times 64 \times 64$ for the teacher network as well. If a feature extractor outputs feature maps of a size other than 64×64, we can adjust its input image size to achieve an output feature map size

of 64×64. Alternatively, we can adjust the input image size of the teacher network because it is fully convolutional and operates separately on patches of size 33×33. A feature map size of 53×71, for example, can be achieved by applying the teacher network to images of size 212×284.

We use a batch size of 16 for the distillation training and use ImageNet [54] as the pretraining dataset. We use the official implementation of PatchCore [2] and its default values if not stated otherwise. We use the feature postprocessing of PatchCore as well, which includes pooling features from two layers and projecting each feature vector to a reduced dimensionality of 384 dimensions, as described in [50]. The features used for our distillation training are the final features used by PatchCore, i.e., those given to the coreset subsampling algorithm when training PatchCore. We denote the WideResNet-101-based feature extractor, including the feature postprocessing, as $\Psi : \mathbb{R}^{3\times512\times512} \to \mathbb{R}^{384\times64\times64}$.

Algorithm 3 Distillation Training Algorithm

Require: A pretrained feature extractor $\Psi : \mathbb{R}^{3\times W\times H} \to \mathbb{R}^{384\times64\times64}$.
Require: A sequence of distillation training images $\mathcal{I}_{\text{dist}}$
Randomly initialize a teacher network $T : \mathbb{R}^{3\times256\times256} \to \mathbb{R}^{384\times64\times64}$ with an architecture as given in Table 5 or 6
for $c \in 1, \ldots, 384$ **do** ▷ Compute feature extractor channel normalization parameters $\mu^\Psi \in \mathbb{R}^{384}$ and $\sigma^\Psi \in \mathbb{R}^{384}$
    Initialize an empty sequence $X \leftarrow (\,)$
    for iteration $= 1, 2, \ldots, 10\,000$ **do**
        Choose a random training image I_{dist} from $\mathcal{I}_{\text{dist}}$
        Convert I_{dist} to gray scale with a probability of 0.1
        Compute I^Ψ_{dist} by resizing I_{dist} to $3 \times W \times H$ using bilinear interpolation
        $Y^\Psi \leftarrow \Psi(I^\Psi_{\text{dist}})$
        $X \leftarrow X \frown \text{vec}(Y^\Psi_c)$ ▷ Append the channel output to X
    end for
    Set μ^Ψ_c to the mean and σ^Ψ_c to the standard deviation of the elements of X
end for
Initialize the Adam [29] optimizer with a learning rate of 10^{-4} and a weight decay of 10^{-5} for the parameters of T
for iteration $= 1, \ldots, 60\,000$ **do**
    $L_{\text{batch}} \leftarrow 0$
    for batch index $= 1, \ldots, 16$ **do**
        Choose a random training image I_{dist} from $\mathcal{I}_{\text{dist}}$
        Convert I_{dist} to gray scale with a probability of 0.1
        Compute I^Ψ_{dist} by resizing I_{dist} to $3 \times W \times H$ using bilinear interpolation
        Compute I'_{dist} by resizing I_{dist} to $3 \times 256 \times 256$ using bilinear interpolation
        $Y^\Psi \leftarrow \Psi(I^\Psi_{\text{dist}})$
        Compute the normalized features $\hat{Y}^\Psi$ given by $\hat{Y}^\Psi_c = (Y^\Psi_c - \mu^\Psi_c)(\sigma^\Psi_c)^{-1}$ for each $c \in \{1, \ldots, 384\}$
        $Y' \leftarrow T(I'_{\text{dist}})$
        Compute the squared difference between $\hat{Y}^\Psi$ and Y' for each tuple (c, w, h) as $D^{\text{dist}}_{c,w,h} = (\hat{Y}^\Psi_{c,w,h} - Y'_{c,w,h})^2$
        Compute the loss L_{dist} as the mean of all elements $D^{\text{dist}}_{c,w,h}$ of D^{dist}
        $L_{\text{batch}} \leftarrow L_{\text{batch}} + L_{\text{dist}}$
    end for
    $L_{\text{batch}} \leftarrow 16^{-1} L_{\text{batch}}$
    Update the parameters of T, denoted by θ, using the gradient $\nabla_\theta L_{\text{batch}}$
end for
return T

Comments on Algorithm 3: We use the image normalization of the pretrained models of torchvision [44]. That means we subtract 0.485, 0.456, and 0.406 from the R, G, and B channel, respectively, for each input image and divide the channels by 0.229, 0.224, and 0.225, respectively. We perform this normalization directly before applying a network to an image, i.e., after augmentation.

[2] https://github.com/amazon-science/patchcore-inspection/tree/6a9a281fc34cb1b13c54b318f71e6f1f371536bb

Appendix B. Implementation Details for Other Evaluated Methods

In the following, we provide the implementation and configuration details for Asymmetric Student–Teacher (AST) [53], DSR [70], FastFlow [68], GCAD [8], PatchCore [50], and Student–Teacher [10].

B.1. AST

We use the official implementation of Rudolph *et al*. [53] [3]. We use the default configuration without modifications, but are not able to fully reproduce the results reported in the AST paper. The AST paper reports a mean classification AU-ROC of 99.2 % on MVTec AD, averaged across five runs. We obtain an AU-ROC of 98.9 % across five runs.

B.2. DSR

We use the official implementation of Zavrtanik *et al*. [70] [4]. We use the default configuration without modifications for reproducing the results on MVTec AD. We obtain a mean classification AU-ROC of 98.1 % on MVTec AD, which is close to the 98.2 % reported by the authors. On the scenarios from VisA, which contain more training images than those of MVTec AD, we change the number of epochs to 50 to keep the total number of training iterations in a similar range.

B.3. FastFlow

We use the implementation of Akcay *et al*. [1] [5]. We use the FastFlow version based on the WideResNet-50-2 feature extractor, as it is similar to the WideResNet used by PatchCore and our method. We use the default configuration, but disable early stopping, i.e., the scenario-specific tuning of the training duration on test images. Instead, we choose a constant training duration (200 steps) that works well on average for all evaluated datasets. The mean classification AU-ROC on MVTec AD differs from the results reported in the original work, but is in line with those reported by Akcay *et al*. for the released implementation in the Intel anomalib [1].

B.4. GCAD

We implement GCAD as described by Bergmann *et al*. [8]. We are able to reproduce the results reported by the authors, but adapt GCAD to a configuration that performs better in our experiments. GCAD consists of an ensemble of two anomaly detection models that use different feature extractors. The first member uses a feature extractor that operates on patches of size 17×17 while the feature extractor used by the second member operates on patches of size 33×33. We find that the second member performs better on average than the combined ensemble and therefore report the results for this member in the main paper. This reduces the latency reported for GCAD by a factor of two. Furthermore, we change the size of the autoencoder's bottleneck from 32 to 64, which slightly improves the average anomaly detection performance on MVTec AD, VisA, and MVTec LOCO. On the logical anomalies of MVTec LOCO, the single model scores a classification AU-ROC of 83.9 %, while the AU-ROC of the ensemble model used by the authors is 86.0 %. The overall anomaly classification performance on MVTec LOCO, however, stays the same.

B.5. PatchCore

We use the official implementation of Roth *et al*. [50] [6]. With the default configuration, we are able to reproduce the results reported for MVTec AD. As described in the main paper, we disable the cropping of the center 76.6 % of input images. Apart from that, we do not alter the hyperparameters of PatchCore. We report the results for two variants of PatchCore. For the single model variant, we use the default configuration of PatchCore for which the authors report the lowest latency. Specifically, this means setting the coreset subsampling ratio to 1 %, the image size to 224×224 pixels, and the feature extraction backbone to a WideResNet-101. For the ensemble variant, we use the configuration for which the authors report the best classification AU-ROC on MVTec AD. We use a WideResNet-101, a ResNeXT-101 [66], and a DenseNet-201 [25] as backbones, set the coreset subsampling ratio to 1 %, and use images of size 320×320 pixels.

B.6. Student–Teacher

We implement the original multi-scale Student–Teacher (S–T) method as described by Bergmann *et al*. [10]. We use the default hyperparameter settings without modification. Our implementation achieves better anomaly localization results on MVTec AD than those reported by the authors but matches those reported in [7].

[3] https://github.com/marco-rudolph/AST/tree/1a157973a0e2cb23b6fbb853db8ae43537ab2568
[4] https://github.com/VitjanZ/DSR_anomaly_detection/tree/672bfb81434fd2a6c5ef00db858cef8834c54f28
[5] https://github.com/openvinotoolkit/anomalib/tree/e66a17c86489486f6bbd5099366383e8660923fd
[6] https://github.com/amazon-science/patchcore-inspection/tree/6a9a281fc34cb1b13c54b318f71e6f1f371536bb

Appendix C. Robustness to the Distillation Backbone Architecture

In the main paper, we use the features from a WideResNet-101 for training a teacher network in Algorithm 3. The default configuration of PatchCore uses the same features. In Table 8, we evaluate the anomaly detection performance for other backbones. Specifically, we evaluate the two additional backbones that PatchCore$_{\text{Ens}}$ uses, i.e., a ResNeXt-101 and a DenseNet-201. On the three evaluated dataset collections, the anomaly detection performance of EfficientAD is similarly robust to the choice of the backbone in comparison to the robustness of PatchCore. On MVTec AD, both methods perform very similarly across backbones, while their performance on MVTec LOCO varies more. On VisA, the gap between the structural anomaly detection performance of PatchCore and that of EfficientAD becomes evident.

	Method	WideResNet-101	ResNeXt-101	DenseNet-201
MVTec AD	PatchCore	98.7	98.8	98.7
	EfficientAD-S	98.8	98.9	98.8
	EfficientAD-M	99.1	99.0	99.2
MVTec LOCO	PatchCore	80.3	78.9	76.5
	EfficientAD-S	90.0	90.1	90.6
	EfficientAD-M	90.7	89.9	88.3
VisA	PatchCore	94.3	95.2	94.8
	EfficientAD-S	97.5	97.3	97.1
	EfficientAD-M	98.1	98.0	97.7

Table 8. Mean anomaly classification AU-ROC percentages for different backbones. For EfficientAD, each listed architecture is used as the distillation backbone in Algorithm 3. The "WideResNet-101" column contains the results reported in Table 2 in the main paper.

Appendix D. Additional Anomaly Detection Metrics

In this section, we report the results for additional anomaly detection metrics. Section D.1 evaluates image-level anomaly classification metrics. Section D.2 evaluates pixel-level anomaly localization metrics.

For per-scenario evaluation results, see the `per_scenario_results.json` file [7].

Following the official MVTec LOCO evaluation script [8], we evaluate each performance metric separately on the structural and on the logical anomalies of MVTec LOCO. Then, we compute the mean of the two scores to compute the overall performance of a method on a scenario of MVTec LOCO.

D.1. Anomaly Classification

In the main paper, we evaluate the anomaly classification performance with the area under the ROC curve (AU-ROC). Here, we report the results for the area under the precision recall curve (AU-PRC) as well. For information on the differences between the AU-ROC and the AU-PRC, we refer to Davis and Goadrich [15].

Table 9 shows the anomaly classification performance of each method measured with the AU-ROC. This table contains the results reported in the main paper. Table 10 shows the results for the image-level AU-PRC.

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	89.1	83.7	82.7	83.9	83.3	85.4
S–T	93.2	94.6	88.3	66.5	77.4	88.4
FastFlow	96.9	93.9	82.9	75.5	79.2	90.0
DSR	98.1	91.8	90.2	75.0	82.6	90.8
PatchCore	98.7	94.3	84.8	75.8	80.3	91.1
$\text{PatchCore}_{\text{Ens}}$	**99.3**	97.7	87.7	71.0	79.4	92.1
AST	98.9	94.9	87.1	79.7	83.4	92.4
EfficientAD-S	98.8	97.5	94.1	85.8	90.0	95.4
EfficientAD-M	99.1	**98.1**	**94.7**	**86.8**	**90.7**	**96.0**

Table 9. Mean anomaly classification AU-ROC percentages per dataset collection. For EfficientAD, we report the mean of five runs.

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	95.7	87.1	81.0	84.9	83.0	88.6
S–T	95.7	94.6	87.9	70.7	79.3	89.9
FastFlow	95.3	94.7	79.5	76.2	77.9	89.3
DSR	98.1	93.8	88.2	76.6	82.4	91.4
PatchCore	98.9	95.2	84.6	77.7	81.2	91.8
$\text{PatchCore}_{\text{Ens}}$	**99.0**	97.8	88.3	74.7	81.5	92.8
AST	98.9	95.3	84.5	80.5	82.5	92.2
EfficientAD-S	98.7	97.5	93.6	86.2	89.9	95.4
EfficientAD-M	98.9	**98.0**	**93.9**	**86.8**	**90.3**	**95.7**

Table 10. Mean anomaly classification AU-PRC percentages per dataset collection. For EfficientAD, we report the mean of five runs.

D.2. Anomaly Localization

To evaluate the anomaly localization performance, we use the area under the PRO curve (AU-PRO) up to a false positive rate (FPR) of 30 % in the main paper, as recommended by [7]. The AU-PRO metric [7] is similar to the pixel-wise AU-ROC. The difference is that the pixel-wise AU-ROC gives each ground truth defect *pixel* the same weight in its computation. The AU-PRO gives each ground truth defect *region* the same weight. The FPR limit of 30 % is due to the fact that a method that segments, on average, more than 30 % of every defect-free image as anomalous is of limited use.

[7] https://www.mydrive.ch/shares/71140/bbca60ead1194717e488313f1632504c/download/444261995-1678948469/per_scenario_results.json

[8] https://www.mvtec.com/company/research/datasets/mvtec-loco

Table 11 contains the results reported in the main paper. Here, we report the AU-PRO for an FPR limit of 5 % as well in Table 12. For comparison, we also report the pixel-wise AU-ROC for an FPR limit of 5 % in Table 13. Furthermore, we evaluate the pixel-wise AU-PRC as an additional segmentation, and thus, localization performance metric in Table 14. The `per_scenario_results.json` file [9] also contains the AU-PRO and pixel-wise AU-ROC results for an FPR limit of 100 %.

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	91.0	83.7	89.5	89.4	89.5	88.0
S–T	92.4	93.0	90.8	76.4	83.6	89.7
FastFlow	92.5	86.8	84.2	76.5	80.3	86.5
DSR	90.8	68.1	81.3	72.3	76.8	78.6
PatchCore	92.7	79.7	64.3	76.6	70.4	80.9
$\text{PatchCore}_{\text{Ens}}$	**95.6**	79.3	62.0	72.6	67.3	80.7
AST	81.2	81.5	75.4	62.6	69.0	77.2
EfficientAD-S	93.1	93.1	92.6	90.1	91.3	92.5
EfficientAD-M	93.5	**94.0**	**93.7**	**91.3**	**92.5**	**93.3**

Table 11. Mean anomaly localization performance per method and dataset collection, measured with the AU-PRO up to a FPR of 30 %. For EfficientAD, we report the mean of five runs.

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	68.8	52.6	68.8	67.1	68.0	63.1
S–T	73.4	75.0	75.6	49.7	62.6	70.4
FastFlow	71.6	63.4	64.5	49.1	56.8	63.9
DSR	78.9	49.5	67.1	49.8	58.5	62.3
PatchCore	68.6	49.4	37.9	41.5	39.7	52.6
$\text{PatchCore}_{\text{Ens}}$	**79.5**	55.1	37.8	35.3	36.5	57.1
AST	42.1	48.0	50.1	35.3	42.7	44.3
EfficientAD-S	78.2	73.4	80.8	74.8	77.8	76.5
EfficientAD-M	78.4	**75.9**	**83.2**	**76.5**	**79.8**	**78.0**

Table 12. Mean anomaly localization performance per method and dataset collection, measured with the AU-PRO up to a FPR of 5 %. For EfficientAD, we report the mean of five runs.

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	72.1	73.1	73.1	32.0	52.5	65.9
S–T	74.3	82.7	69.8	20.6	45.2	67.4
FastFlow	72.1	78.9	63.4	33.7	48.6	66.5
DSR	76.1	66.5	66.0	25.5	45.7	62.8
PatchCore	74.1	65.0	43.5	24.1	33.8	57.6
$\text{PatchCore}_{\text{Ens}}$	79.4	65.7	38.7	20.4	29.6	58.2
AST	41.1	67.4	52.4	30.9	41.7	50.1
EfficientAD-S	**79.7**	86.3	80.6	33.8	57.2	74.4
EfficientAD-M	79.4	**86.9**	**82.1**	**35.3**	**58.7**	**75.0**

Table 13. Mean anomaly localization performance per method and dataset collection, measured with the AU-ROC up to a FPR of 5 %. For EfficientAD, we report the mean of five runs.

[9] https://www.mydrive.ch/shares/71140/bbca60ead1194717e488313f1632504c/download/444261995-1678948469/per_scenario_results.json

Method	MAD Mean	VisA Mean	LOCO Structural	LOCO Logical	LOCO Mean	Overall Mean
GCAD	59.3	27.8	41.4	38.7	40.1	42.4
S–T	59.9	36.2	43.5	27.4	35.4	43.8
FastFlow	57.6	33.4	35.1	41.3	38.2	43.1
DSR	**69.2**	**41.1**	50.4	32.7	41.5	50.6
PatchCore	57.6	27.8	17.8	32.5	25.2	36.8
$\text{PatchCore}_{\text{Ens}}$	64.1	28.3	15.1	28.9	22.0	38.2
AST	29.7	22.9	17.0	35.6	26.3	26.3
EfficientAD-S	65.9	40.4	**54.0**	40.2	**47.1**	**51.1**
EfficientAD-M	63.8	40.8	51.9	**42.0**	46.9	50.5

Table 14. Mean anomaly localization performance per method and dataset collection, measured with the pixel-wise AU-PRC. For EfficientAD, we report the mean of five runs.

Appendix E. Timing Methodology and Additional Computational Efficiency Metrics

Method	Classif. AU-ROC	Segment. AU-PRO	Latency [ms]	Throughput [img / s]	Number of Parameters [$\times 10^6$]	FLOPs [$\times 10^9$]	GPU Memory [MB]
GCAD	85.4	88.0	11	121	65	416	555
S–T	88.4	89.7	75	16	26	4468	1077
FastFlow	90.0	86.5	17	120	92	85	404
DSR	90.8	78.6	17	104	40	267	314
PatchCore	91.1	80.9	32	76	83 + 3	41 + kNN	637 + kNN
$\text{PatchCore}_{\text{Ens}}$	92.1	80.7	148	13	150 + 8	159 + kNN	1335 + kNN
AST	92.4	77.2	53	41	154	199	618
EfficientAD-S	95.4 (± 0.06)	92.5 (± 0.05)	**2.2** (± 0.01)	**614** (± 2)	**8** (± 0)	76 (± 0)	**100** (± 0)
EfficientAD-M	**96.0** (± 0.09)	**93.3** (± 0.04)	4.5 (± 0.01)	269 (± 1)	21 (± 0)	235 (± 0)	161 (± 0)

Table 15. Extension of Table 1 in the main paper by additional computational efficiency metrics measured on an NVIDIA RTX A6000 GPU. For EfficientAD, we report the mean and standard deviation of five runs. For PatchCore, we report the computational requirements of the feature extraction during inference separately from the nearest neighbor search.

In the following, we describe how we measure the latency and the throughput of each anomaly detection method. Latency refers to the inference runtime, i.e., how long it takes a method to generate the anomaly detection result for a single image. Throughput refers to how many images can be processed per second when allowing a batched processing of images. In settings in which latency constraints are fulfilled or not present, a high throughput is relevant for using computational resources efficiently and thus for reducing the economic cost of an application.

All evaluated methods are implemented in PyTorch. All of them, including the nearest neighbor search of PatchCore, run faster on each of the GPUs in our experimental setup than on a CPU. We therefore execute each method on a GPU. For a test image, our timing begins with the transfer of the image from the CPU to the GPU. We include the transfer to regard the benefit of a method that would run exclusively on a CPU. Our timing stops when the anomaly detection result, which for all evaluated methods is an anomaly map, is available on the CPU. For each method, we remove unnecessary parts for the timing, such as the computation of losses during inference, and use float16 precision for all networks. Switching from float32 to float16 for the inference of EfficientAD does not change the anomaly detection results for the 32 anomaly detection scenarios evaluated in this paper. In latency-critical applications, padding in the PDN architecture of EfficientAD can be disabled. This speeds up the forward pass of the PDN architecture by 80 µs without impairing the detection of anomalies. We time EfficientAD without padding and therefore report the anomaly detection results for this setting in the experimental results of this paper. We perform 1000 forward passes as warm up and report the mean runtime of the following 1000 forward passes. For the latency, we report the average runtime of 1000 forward passes with a batch size of 1. We compute the throughput by dividing 16 000 by the sum of the runtimes of 1000 forward passes with a batch size of 16. In addition to the latency and the throughput, we report the number of parameters, the number of floating point operations (FLOPs), and the GPU memory consumption

for each method in Table 15. Analogously to the latency, we measure these metrics for the processing of one image during inference and report the mean of 1000 forward passes. The number of parameters and the FLOPs remain constant across forward passes, while the GPU memory consumption varies slightly (less than one MB difference between forward passes).

Technical Details For methods that use features from hidden layers of a pretrained network, we exclude the layers that are not required for computing these features, i.e., classification heads etc. We measure the number of FLOPs using the official profiling framework of PyTorch [44] (version 1.12.0). Specifically, we wrap the inference function of a method into a call of `with torch.profiler.profile(with_flops=True) as prof:`. For measuring the GPU memory consumption, we also use the official profiling framework of PyTorch. We obtain the peak of the reserved GPU memory during inference with `torch.cuda.memory_stats()['reserved_bytes.all.peak']`.

Interpretability of Efficiency Metrics In the main paper, we focus on the latency and the throughput of the evaluated anomaly detection methods. The number of parameters and the number of FLOPs are often used as proxy metrics for the runtime, but can be misleading. For example, the number of parameters of FastFlow in Table 15 is roughly 2.5 times larger than that of S–T. Yet, the latency of FastFlow is substantially lower and its throughput is 6.5 times higher.

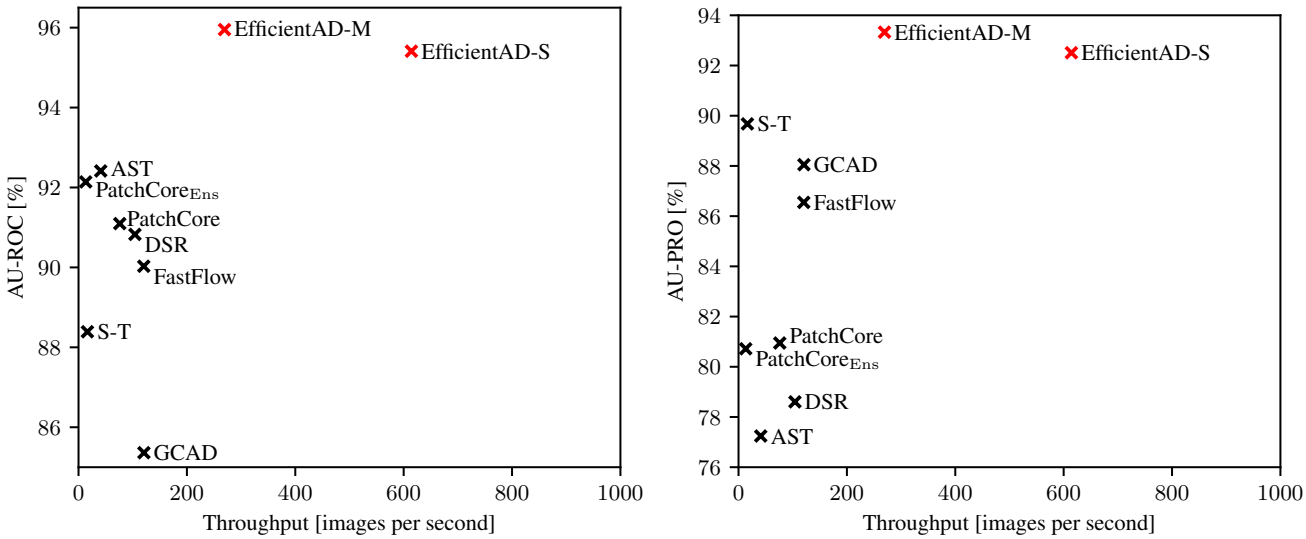
FLOPs nicht aussagekräftig, weil zB Convolutions auf GPU einfach parallelisiert werden können

With 4.5 trillion FLOPs, S–T exceeds the FLOPs of other methods by a large margin. The high number of FLOPs, however, comes from the fact that S–T uses convolutions that operate on large feature maps. This means that these convolutions can be parallelized well on a GPU, while implementing them naively on a CPU would indeed cause a prohibitively long runtime. FLOPs measurements do not account for this, because they do not consider how well operations can be parallelized. The number of FLOPs can therefore be an unreliable metric for efficiency. For example, the number of FLOPs of S–T is more than 2000 % higher than that of AST, but the latency is only 42 % higher.

The GPU memory footprint of a method can theoretically be reduced drastically by freeing obsolete GPU memory segments after each layer's execution during a forward pass. In the extreme case, one could even directly free the memory of individual input activation values directly after the output activation of a neuron in a convolutional layer has been computed. This, however, would worsen the runtime of a forward pass, which generally improves when reserved GPU memory segments can be reused. Therefore, the GPU memory footprint of a method needs to be reported and analyzed jointly with the latency and throughput. We focus on the GPU memory required for achieving the reported latency and throughput and therefore measure the peak of the reserved GPU memory during a forward pass.

PatchCore For PatchCore, we distinguish between the backbones used to compute features and the kNN algorithm itself. For example, the part of the WideResNet-101 backbone until the layer used for computing features has 83 million parameters. During training, PatchCore computes the feature vectors of all training images. The coreset subsampling phase of PatchCore reduces the number of feature vectors to 1 % of the computed feature vectors. These are then indexed and stored in GPU memory to enable a fast search for nearest neighbors during inference. This, however, means that the number of parameters, the FLOPs, and the GPU memory footprint of PatchCore depend on the training images. We therefore benchmark PatchCore on the "cashew" scenario of VisA, which contains 450 training images and is thus closest to the average 439 training images of the 32 scenarios of MVTec AD, VisA, and MVTec LOCO. We do not report the FLOPs and the GPU memory consumption of the kNN search, as we were not able to measure it with the kNN library used by the official PatchCore implementation. The number of parameters of the kNN search is given by the number of values stored in the GPU memory during inference. In the case of $\text{PatchCore}_{\text{Ens}}$, for example, the search database contains 8 million values.

Anomaly Detection and Throughput Figure 7 shows the anomaly detection performance together with the throughput of each evaluated method, analogous to Figure 1 in the main paper. Apart from EfficientAD, the ranking of methods changes drastically between the image-level and the pixel-level detection of anomalies.



(a) Anomaly classification (b) Anomaly localization

Figure 7. Anomaly detection performance vs. throughput on an NVIDIA RTX A6000 GPU. We report the binary anomaly classification performance on the left using the image-level AU-ROC. On the right, we report the anomaly localization performance using the pixel-level AU-PRO segmentation metric up to a FPR of 30 %. Each AU-ROC and AU-PRO value is an average of the values on MVTec AD, VisA, and MVTec LOCO. We measure the throughput using a batch size of 16.

Latency per GPU In Table 16, we provide the values for Figure 6 in the main paper.

Method	RTX A6000	RTX A5000	Tesla V100	RTX 3080	RTX 2080 Ti
EfficientAD-S	**2.2**	**2.5**	**3.9**	**3.8**	**4.5**
EfficientAD-M	4.5	5.3	6.3	7.0	7.6
GCAD	10.7	11.7	12.9	13.7	18.0
FastFlow	16.5	17.1	26.1	27.5	31.0
DSR	17.2	18.0	24.8	24.6	34.5
PatchCore	32.0	31.5	47.1	41.1	53.2
AST	53.1	53.4	75.6	82.3	87.1
S–T	74.7	81.0	82.2	99.6	121.7
PatchCore$_{\text{Ens}}$	147.6	145.0	229.2	189.0	216.9

Table 16. Latency in milliseconds per GPU, as plotted in Figure 6 in the main paper.

Appendix F. Qualitative Results

In Figures 8 to 10, we display anomaly maps for each of the 32 scenarios of MVTec AD, VisA, and MVTec LOCO. For MVTec LOCO, we show both logical and structural anomalies. We visualize the anomaly maps using a different scale for each method, since the anomaly score scales differ between methods. Across scenarios, however, we use the same color scale per anomaly detection method. A consistent anomaly score scale across applications is an important requirement for a method. Otherwise, the scale of scores on anomalies is hard to forecast if no or only few defect images are present during the development of the anomaly detection system. Knowing the scale is important for choosing a robust threshold value that ultimately determines whether an image or a pixel is anomalous or not. Furthermore, a consistent scale facilitates the interpretation of anomaly maps. For the evaluated methods, we choose the start and end values of the color scales so that true positive and true negative detections become clearly visible. For example, the color scale of AST ranges from 2 to 10. Scores outside of this range are visualized with the minimum and maximum color value, respectively. For PatchCore, choosing the range of the color scale is difficult. On the one hand, scores of true positive detections are low, such as the contamination of the banana juice bottle in Figure 8. On the other hand, scores of false positive detections are similarly high, such as the predictions on the breakfast box in Figure 8.

Overall, the evaluated anomaly detection methods succeed on the anomalies of MVTec AD, but leave room for improvement on MVTec LOCO and VisA.

- EfficientAD responds to both logical and structural anomalies in the images. The strength of its response sometimes leaves room for improvement, for example, on the logical anomalies of the breakfast box and the box of pushpins in Figure 8.
- AST detects some logical anomalies, but lacks an approach that detects logical anomalies by design. For example, it detects that the additional blue cable connecting two splicing connectors in Figure 8 causes unseen features. Yet, the missing pushpin in the box of pushpins in Figure 8 is also an unseen feature and does not cause a response in the anomaly map of AST. This highlights the importance of a reliable approach to logical anomalies. Furthermore, it shows the dependence of anomaly detection methods on the choice of the feature extractor. As shown in Table 8, EfficientAD is robust to this choice.
- DSR produces very precise segmentations, but also suffers from false positives, for example on the grid and the wood image of MVTec AD in Figure 9. At times, it furthermore shows no response at all to defects.
- FastFlow's anomaly maps contain a large amount of noise, i.e. false positive detections. This hinders the interpretability of its detection results.
- GCAD succeeds at detecting logical anomalies, but has difficulty with some structural anomalies that other methods detect reliably, such as the scratches on the metal nut in Figure 9 or the green capsules in Figure 10.
- PatchCore$_{\text{Ens}}$ struggles with very small defects such as those of the printed circuit boards in Figure 10. Small defects are challenging, but highly relevant for practical applications. A small contamination can cause a high economic damage if it goes unnoticed, for example, in a pharmaceutical application.
- S–T is a patch-based anomaly detection approach and therefore can only detect anomalies if they involve patches that are anomalous per se, i.e., without putting them in the global context of the respective image. While AST's feature vectors have a receptive field that spans across the entire image, S–T's receptive field is limited to 65×65 pixels. Therefore, it does not detect anomalies such as the missing transistor in Figure 9.

The qualitative results show tendencies of each method regarding the behavior on anomalous images. While these results are informative, they should not be used exclusively for evaluating the anomaly detection performance of a method or for comparing methods. For that, metrics such as the AU-ROC and the AU-PRO are well-suited, since they are evaluated objectively on thousands of test images across dataset collections.

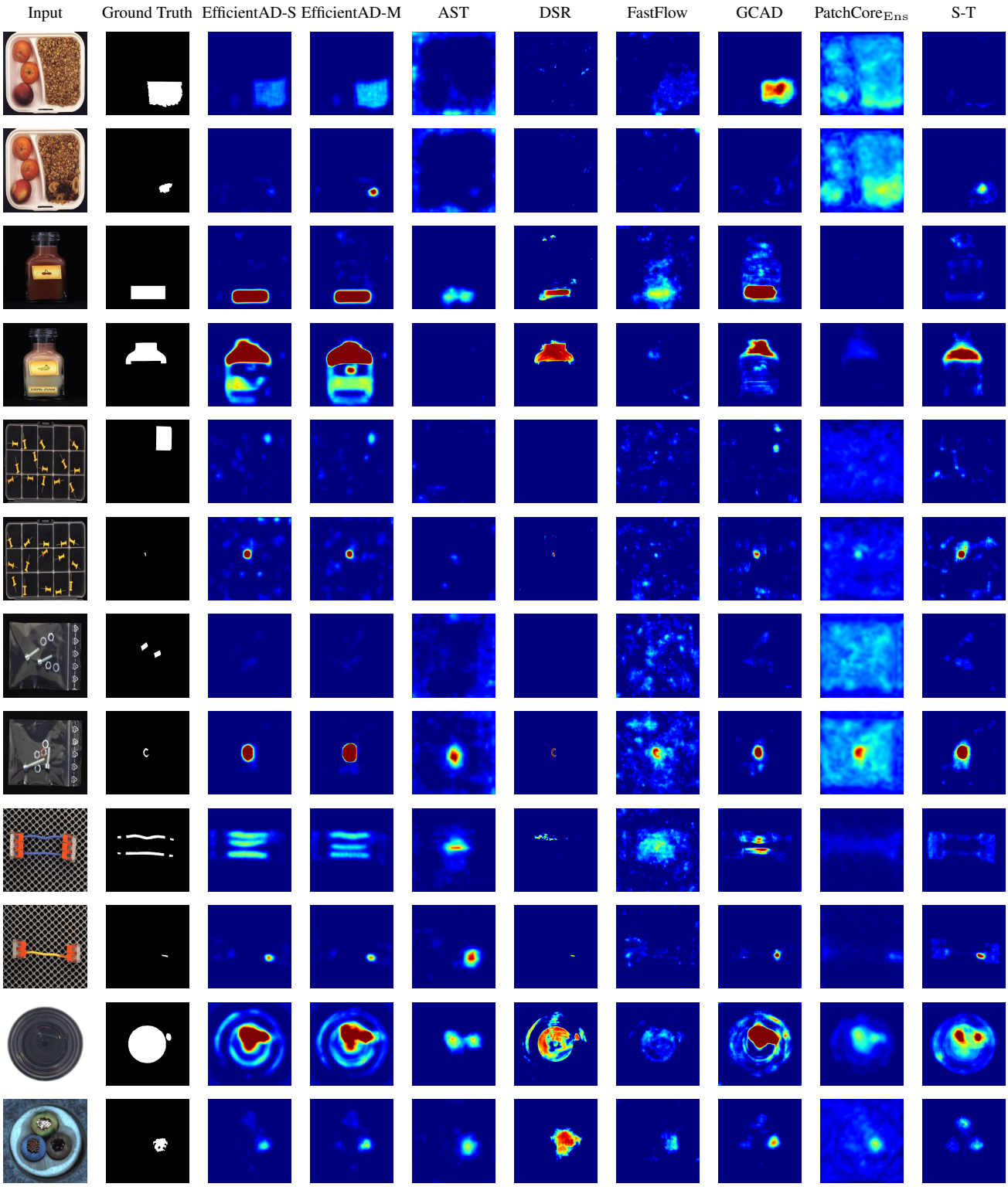


Figure 8. Anomaly maps on anomalous images from MVTec LOCO and MVTec AD. For MVTec LOCO, we show a logical anomaly (upper row) and a structural anomaly (lower row) for each scenario. The receptive field of AST's features is large enough to detect some logical anomalies, while PatchCore$_{\text{Ens}}$ and S–T struggle with logical anomalies.

Figure 9. Anomaly maps on anomalous images from MVTec AD. Almost all anomalies are detected by every method, but the separability of pixel anomaly scores varies between methods. For example, PatchCore$_{\text{Ens}}$ detects the anomaly on the capsule in the first row but the pixel anomaly scores are in a similar range as the false positive detections in the background of the screw image.

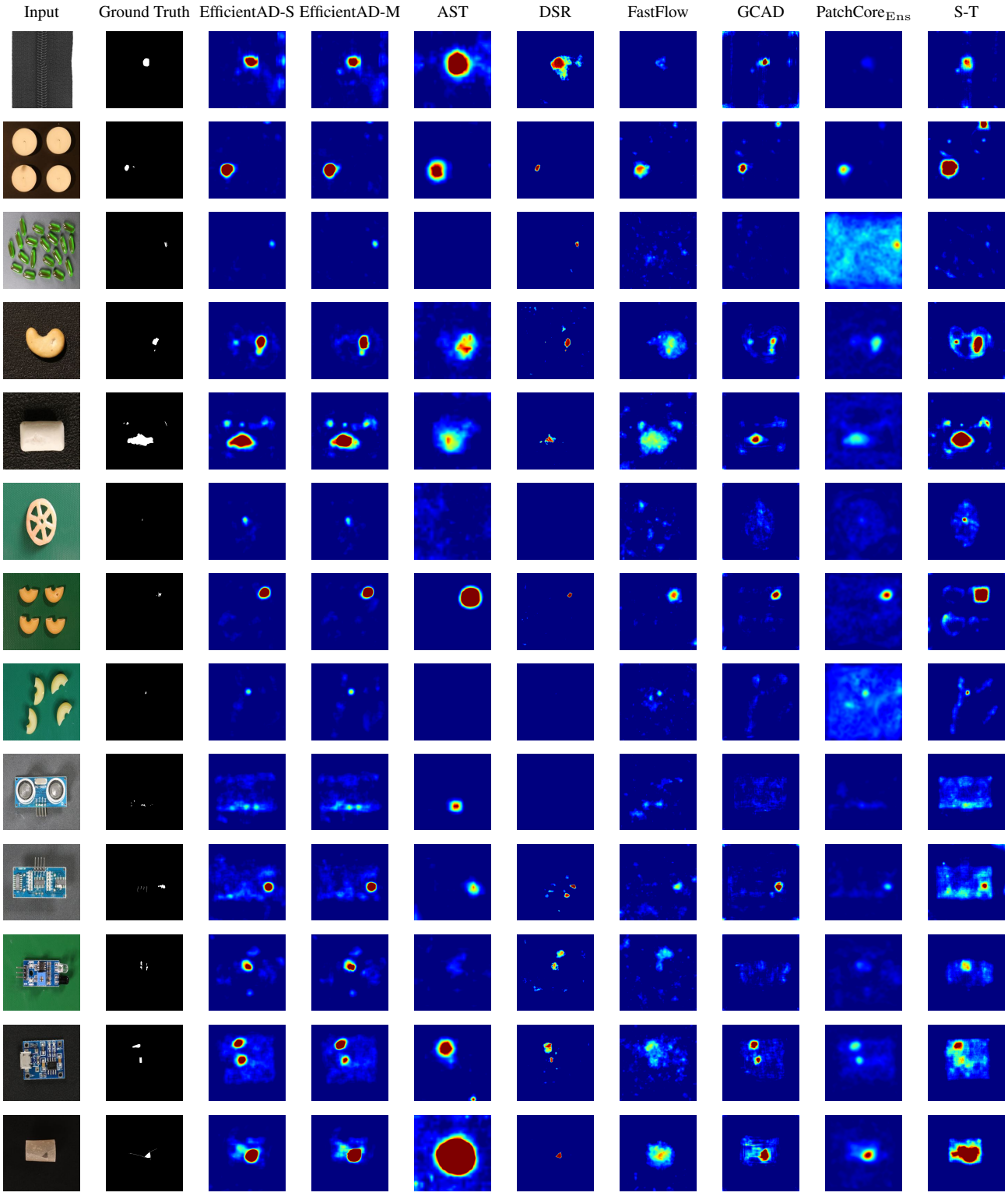


Figure 10. Anomaly maps on anomalous images from MVTec AD and VisA. VisA contains challenging, small anomalies, such as the defect on the non-aligned macaronis or the defect on the fryum two rows above.