EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies

Supplementary Material

Kilian Batzner[1] Lars Heckler[1,2] Rebecca König[1]
[1]MVTec Software GmbH, [2]Technical University of Munich
{kilian.batzner, lars.heckler, rebecca.koenig}@mvtec.com

Abstract

We provide the following supplementary material:

- *(1) Implementation details for EfficientAD, including the training and inference procedure of EfficientAD on the dataset of an anomaly detection scenario (1.1) and the distillation training of the patch description network (1.2).*
- *(2) Implementation and configuration details for other evaluated methods.*
- *(3) Evaluation of the anomaly detection performance of EfficientAD for different distillation backbones.*
- *(4) Results for additional anomaly detection metrics, such as the area under the precision recall curve.*
- *(5) Description of our timing methodology and results for additional computational efficiency metrics such as the number of parameters.*
- *(6) 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. Implementation Details for EfficientAD

1.1. 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 1 with that of Table 2.

Algorithm 1 EfficientAD-S Training Algorithm

```
Require: A pretrained teacher network $T : \mathbb{R}^{3\times 256\times 256} \rightarrow \mathbb{R}^{384\times 64\times 64}$ with an architecture as given in Table 1
Require: A sequence of training images $\mathcal{I}_{\text{train}}$ with $I_{\text{train}} \in \mathbb{R}^{3\times 256\times 256}$ for each $I_{\text{train}} \in \mathcal{I}_{\text{train}}$
Require: A sequence of validation images $\mathcal{I}_{\text{val}}$ with $I_{\text{val}} \in \mathbb{R}^{3\times 256\times 256}$ for each $I_{\text{val}} \in \mathcal{I}_{\text{val}}$
Randomly initialize a student network $S : \mathbb{R}^{3\times 256\times 256} \rightarrow \mathbb{R}^{768\times 64\times 64}$ with an architecture as given in Table 1
Randomly initialize an autoencoder $A : \mathbb{R}^{3\times 256\times 256} \rightarrow \mathbb{R}^{384\times 64\times 64}$ with an architecture as given in Table 3
for $c \in 1, \dots, 384$ 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 $\mu_c$ to the mean and $\sigma_c$ to the standard deviation of the elements of $X$
```

```
end for
Initialize the Adam [8] optimizer with a learning rate of $10^{-4}$ and a weight decay of $10^{-5}$ for the parameters of $S$ and $A$
for iteration $= 1, \dots, 70\,000$ do
    Choose a random training image $I_{\text{train}}$ 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 $c \in \{1, \dots, 384\}$
    $Y^{\text{S}} \leftarrow S(I_{\text{train}})$
    Set $Y^{\text{ST}} \in \mathbb{R}^{384 \times 64 \times 64}$ to the first 384 channels of $Y^{\text{S}} \in \mathbb{R}^{768 \times 64 \times 64}$
    Compute the squared difference between $\hat{Y}$ and $Y^{\text{ST}}$ 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 $D^{\text{ST}}$, denoted by $d_{\text{hard}}$
    Compute the loss $L_{\text{hard}}$ as the mean of all $D^{\text{ST}}_{c,w,h} \geq d_{\text{hard}}$
    Choose a random pretraining image $P \in \mathbb{R}^{3 \times 256 \times 256}$ from ImageNet [13]
    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 $I_{\text{train}}$ for $A$ using torchvision [10]
    Sample an augmentation coefficient $\lambda$ 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 $c \in \{1, \dots, 384\}$
    $Y^{\text{S}} \leftarrow S(I_{\text{aug}})$
    Set $Y^{\text{STAE}} \in \mathbb{R}^{384 \times 64 \times 64}$ to the last 384 channels of $Y^{\text{S}} \in \mathbb{R}^{768 \times 64 \times 64}$
    Compute the squared difference between $\hat{Y}$ and $Y^{\text{A}}$ 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 $Y^{\text{A}}$ and $Y^{\text{STAE}}$ 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 $L_{\text{AE}}$ as the mean of all elements $D^{\text{AE}}_{c,w,h}$ of $D^{\text{AE}}$
    Compute the loss $L_{\text{STAE}}$ as the mean of all elements $D^{\text{STAE}}_{c,w,h}$ of $D^{\text{STAE}}$
    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 $\phi$, 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 $c \in \{1, \dots, 384\}$
    Split the student output into $Y^{\text{ST}} \in \mathbb{R}^{384 \times 64 \times 64}$ and $Y^{\text{STAE}} \in \mathbb{R}^{384 \times 64 \times 64}$ 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 $M_{\text{ST}}$ and $M_{\text{AE}}$ to $256 \times 256$ pixels using bilinear interpolation
    $X_{\text{ST}} \leftarrow X_{\text{ST}} {}^\frown \operatorname{vec}(M_{\text{ST}})$    ▷ Append to the sequence of local anomaly scores
    $X_{\text{AE}} \leftarrow X_{\text{AE}} {}^\frown \operatorname{vec}(M_{\text{AE}})$    ▷ Append to the sequence of global anomaly scores
end for
Compute the $0.9$-quantile $q_a^{\text{ST}}$ and the $0.995$-quantile $q_b^{\text{ST}}$ of the elements of $X_{\text{ST}}$.
Compute the $0.9$-quantile $q_a^{\text{AE}}$ and the $0.995$-quantile $q_b^{\text{AE}}$ of the elements of $X_{\text{AE}}$.
return $T$, $S$, $A$, $\mu$, $\sigma$, $q_a^{\text{ST}}$, $q_b^{\text{ST}}$, $q_a^{\text{AE}}$, and $q_b^{\text{AE}}$
```

Algorithm 2 EfficientAD Inference Procedure

```
Require: $T$, $S$, $A$, $\mu$, $\sigma$, $q_a^{\mathrm{ST}}$, $q_b^{\mathrm{ST}}$, $q_a^{\mathrm{AE}}$, and $q_b^{\mathrm{AE}}$, 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 $c \in \{1,\dots,384\}$
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 $M_{\mathrm{ST}}$ and $M_{\mathrm{AE}}$ to $256\times256$ 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 $m_{\mathrm{image}}$
```

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 1. 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 2. 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 [10] (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 [2, 4], VisA [18], and MVTec LOCO [3] 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 3. 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 [10]. 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.

1.2. Distillation

In the following, we describe how to distill the WideResNet-101 [16] features used by PatchCore [11] 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 1 to 3). 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 [13] as the pretraining dataset. We use the official implementation of PatchCore [1] 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 [11]. 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 1 or 2
for $c \in 1, \dots, 384$ 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 $= 1, 2, \dots, 10\,000$ do
        Choose a random training image $I_{\text{dist}}$ from $\mathcal{I}_{\text{dist}}$
        Convert $I_{\text{dist}}$ to gray scale with a probability of $0.1$
        Compute $I^{\Psi}_{\text{dist}}$ by resizing $I_{\text{dist}}$ 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 $\mu^{\Psi}_c$ to the mean and $\sigma^{\Psi}_c$ to the standard deviation of the elements of $X$
end for
Initialize the Adam [8] optimizer with a learning rate of $10^{-4}$ and a weight decay of $10^{-5}$ for the parameters of $T$
for iteration $= 1, \dots, 60\,000$ do
    $L_{\text{batch}} \leftarrow 0$
    for batch index $= 1, \dots, 16$ do
        Choose a random training image $I_{\text{dist}}$ from $\mathcal{I}_{\text{dist}}$
        Convert $I_{\text{dist}}$ to gray scale with a probability of $0.1$
        Compute $I^{\Psi}_{\text{dist}}$ by resizing $I_{\text{dist}}$ to $3 \times W \times H$ using bilinear interpolation
        Compute $I'_{\text{dist}}$ by resizing $I_{\text{dist}}$ 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 $c \in \{1, \dots, 384\}$
        $Y' \leftarrow T(I'_{\text{dist}})$
        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 $L_{\text{dist}}$ as the mean of all elements $D^{\text{dist}}_{c,w,h}$ of $D^{\text{dist}}$
        $L_{\text{batch}} \leftarrow L_{\text{batch}} + L_{\text{dist}}$
    end for
    $L_{\text{batch}} \leftarrow 16^{-1} L_{\text{batch}}$
    Update the parameters of $T$, denoted by $\theta$, 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 [10]. 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.

---

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

2. Implementation Details for Other Evaluated Methods

In the following, we provide the implementation and configuration details for Asymmetric Student–Teacher (AST) [12], DSR [17], FastFlow [15], GCAD [3], PatchCore [11], SimpleNet [9], and Student–Teacher [5].

2.1. AST

We use the official implementation of Rudolph *et al.* [12] [2]. 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 image-level detection AU-ROC of 99.2 % on MVTec AD, averaged across five runs. We obtain an AU-ROC of 98.9 % across five runs.

2.2. DSR

We use the official implementation of Zavrtanik *et al.* [17] [3]. We use the default configuration without modifications for reproducing the results on MVTec AD. We obtain a mean image-level detection 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.

2.3. FastFlow

We use the implementation of Akcay *et al.* [1] [4]. We use the FastFlow version based on the WideResNet-50-2 feature extractor, as it is similar to the WideResNet used by PatchCore, SimpleNet, 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.

2.4. GCAD

We implement GCAD as described by Bergmann *et al.* [3]. 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. On the logical anomalies of MVTec LOCO, the single model scores an image-level detection AU-ROC of 83.9 %, while the AU-ROC of the ensemble model used by the authors is 86.0 %. The overall anomaly detection performance on MVTec LOCO, however, stays the same.

2.5. PatchCore

We use the official implementation of Roth *et al.* [11] [5] and 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 for a fair comparison. For the single model variant of PatchCore, we use the 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 image-level detection AU-ROC on MVTec AD. We use a WideResNet-101, a ResNeXT-101 [14], and a DenseNet-201 [7] as backbones, set the coreset subsampling ratio to 1 %, and use images of size 320×320 pixels.

2.6. SimpleNet

We use the official implementation of Liu *et al.* [9] [6] and are able to reproduce the reported results. As explained in the main paper, we disable the scenario-specific tuning of the training duration on test images for a fair comparison.

2.7. Student–Teacher

We implement the original multi-scale Student–Teacher (S–T) method as described by Bergmann *et al.* [5]. 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 [2].

---

[2] https://github.com/marco-rudolph/AST/tree/1a157973a0e2cb23b6fbb853db8ae43537ab2568

[3] https://github.com/VitjanZ/DSR_anomaly_detection/tree/672bfb81434fd2a6c5ef00db858cef8834c54f28

[4] https://github.com/openvinotoolkit/anomalib/tree/e66a17c86489486f6bbd5099366383e8660923fd

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

[6] https://github.com/DonaldRR/SimpleNet/tree/35bf32292995842a4277a7c93431430129efccb5

3. 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 4, 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 4. Mean anomaly detection 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 the main paper.

4. Additional Anomaly Detection Metrics

In this section, we report the results for additional anomaly detection metrics. Section 4.1 evaluates image-level anomaly detection metrics. Section 4.2 evaluates pixel-level anomaly localization metrics.

For per-scenario evaluation results, see the `per_scenario_results.json` file in the supplementary material.

Following the official MVTec LOCO evaluation script [7], 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.

4.1. Anomaly Detection

In the main paper, we evaluate the image-level anomaly detection 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 [6].

Table 5 shows the anomaly detection performance of each method measured with the AU-ROC. This table contains the results reported in the main paper. Table 6 shows the results for the image-level AU-PRC.

4.2. 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 [2]. The AU-PRO metric [2] 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 defect-free pixels as anomalous is of limited use.

Table 7 contains the results reported in the main paper. Here, we report the AU-PRO for an FPR limit of 5 % as well in Table 8. For comparison, we also report the pixel-wise AU-ROC for an FPR limit of 5 % in Table 9. Furthermore, we evaluate the pixel-wise AU-PRC as an additional segmentation, and thus, localization performance metric in Table 10. The `per_scenario_results.json` file in the supplementary material also contains the AU-PRO and pixel-wise AU-ROC results for an FPR limit of 100 %.

---

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

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
SimpleNet	98.2	87.9	83.7	71.5	77.6	87.9
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 5. Mean anomaly detection 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
SimpleNet	98.5	90.1	82.5	73.5	78.0	88.9
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 6. Mean anomaly detection AU-PRC 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	91.0	83.7	89.5	89.4	89.5	88.0
SimpleNet	89.6	68.9	60.6	68.6	64.6	74.4
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 7. 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.

5. Timing Methodology and Additional Computational Efficiency Metrics

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

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
SimpleNet	61.8	37.7	36.6	36.1	36.3	45.3
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 8. 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
SimpleNet	67.9	57.1	36.4	22.1	29.2	51.4
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 9. 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.

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
SimpleNet	51.5	22.6	11.9	29.3	20.6	31.6
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 10. 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.

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 the main paper and the supplementary material.

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 11. 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).

Method	Detect. 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
SimpleNet	87.9	74.4	12	194	73	**38**	508
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
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 11. Extension of Table 1 in the main paper by additional computational efficiency metrics measured on a 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.

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 [10] (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 11 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.

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.

Latency per GPU In Table 12, 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
SimpleNet	12.0	13.3	19.2	18.1	21.9
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
$\text{PatchCore}_{\text{Ens}}$	147.6	145.0	229.2	189.0	216.9

Table 12. Latency in milliseconds per GPU, as plotted in Figure 6 in the main paper.

6. Qualitative Results

In Figures 1 to 3, 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 1. On the other hand, scores of false positive detections are similarly high, such as the predictions on the breakfast box in Figure 1.

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 1.
- 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 1 causes unseen features. Yet, the missing pushpin in the box of pushpins in Figure 1 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 4, 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 2. 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 2 or the green capsules in Figure 3.
- $\mathrm{PatchCore}_{\mathrm{Ens}}$ struggles with very small defects such as those of the printed circuit boards in Figure 3. 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.
- SimpleNet performs similar to other methods on MVTec AD, but struggles with the more challenging anomalies of MVTec LOCO and VisA, for example the defective capsules and PCBs in Figure 3.
- 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 2.

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.**

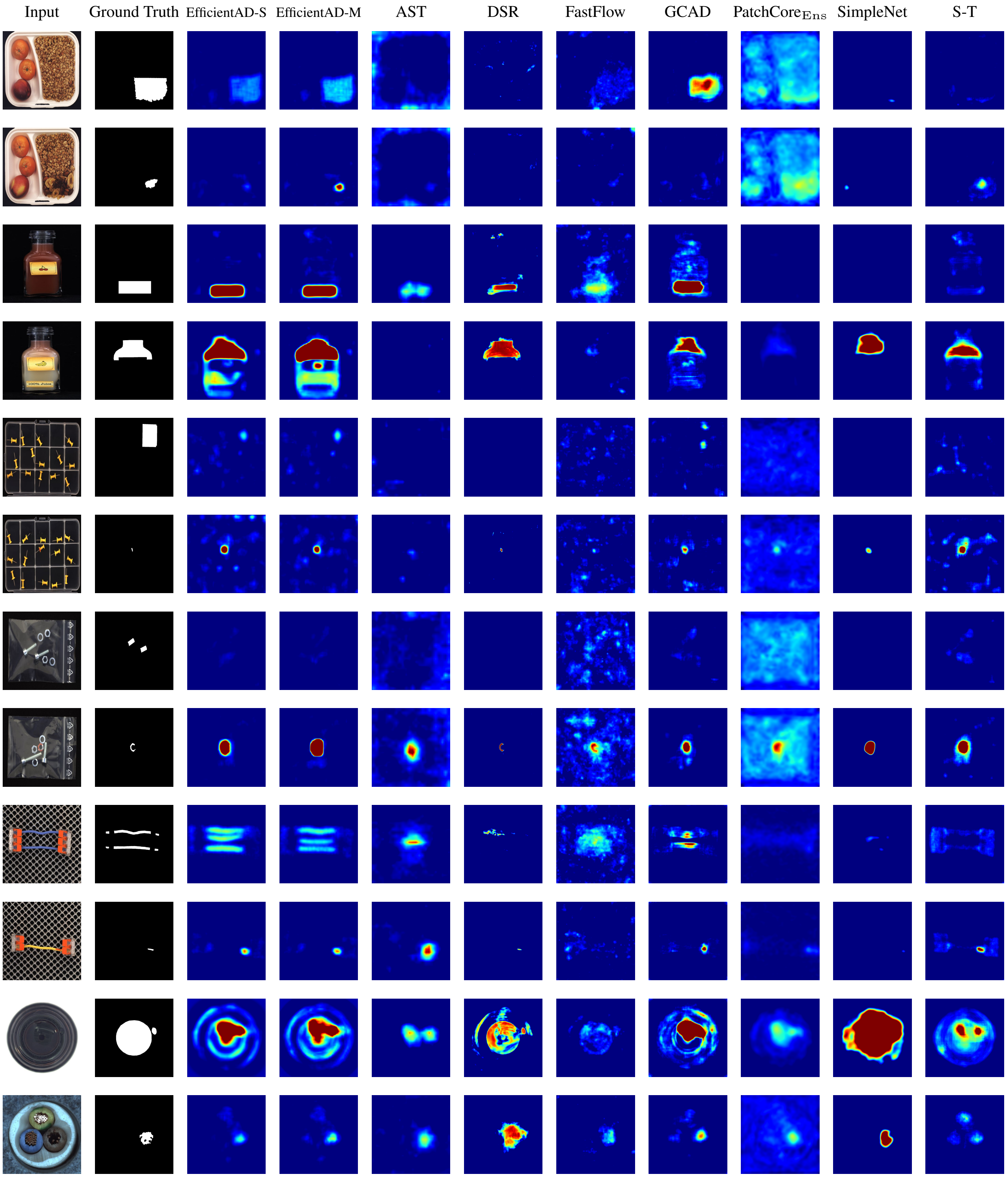

Figure 1. Anomaly maps on anomalous images from MVTec LOCO and MVTec AD. For MVTec LOCO, we show a logical anomaly (upper row) and a structural anomaly (lower row) for each scenario. The receptive field of AST's features is large enough to detect some logical anomalies, while PatchCore$_{\text{Ens}}$ and S–T struggle with logical anomalies.

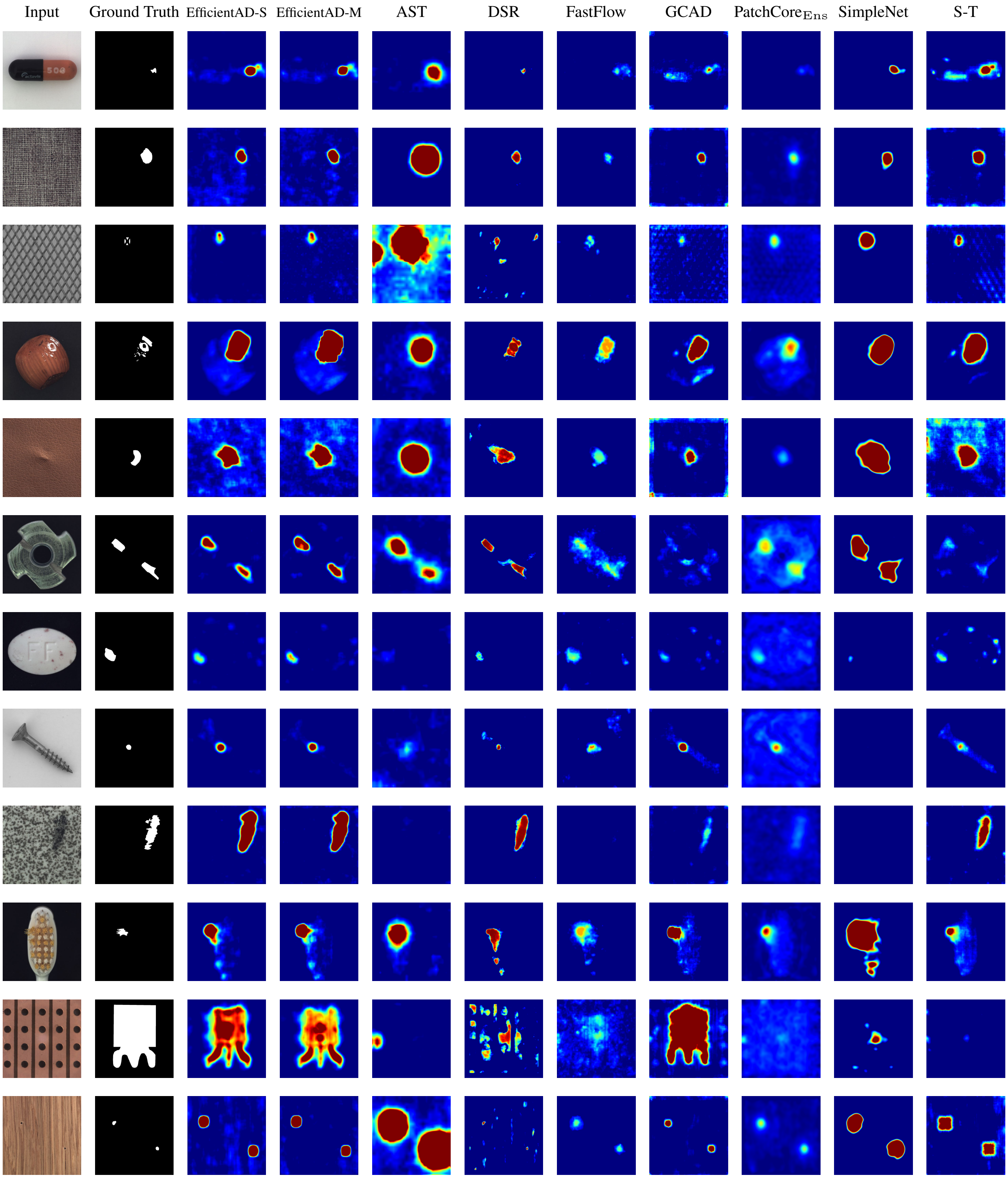

Figure 2. Anomaly maps on anomalous images from MVTec AD. Almost all anomalies are detected by every method, but the separability of pixel anomaly scores varies between methods. For example, $\text{PatchCore}_{\text{Ens}}$ detects the anomaly on the capsule in the first row but the pixel anomaly scores are in a similar range as the false positive detections in the background of the screw image.

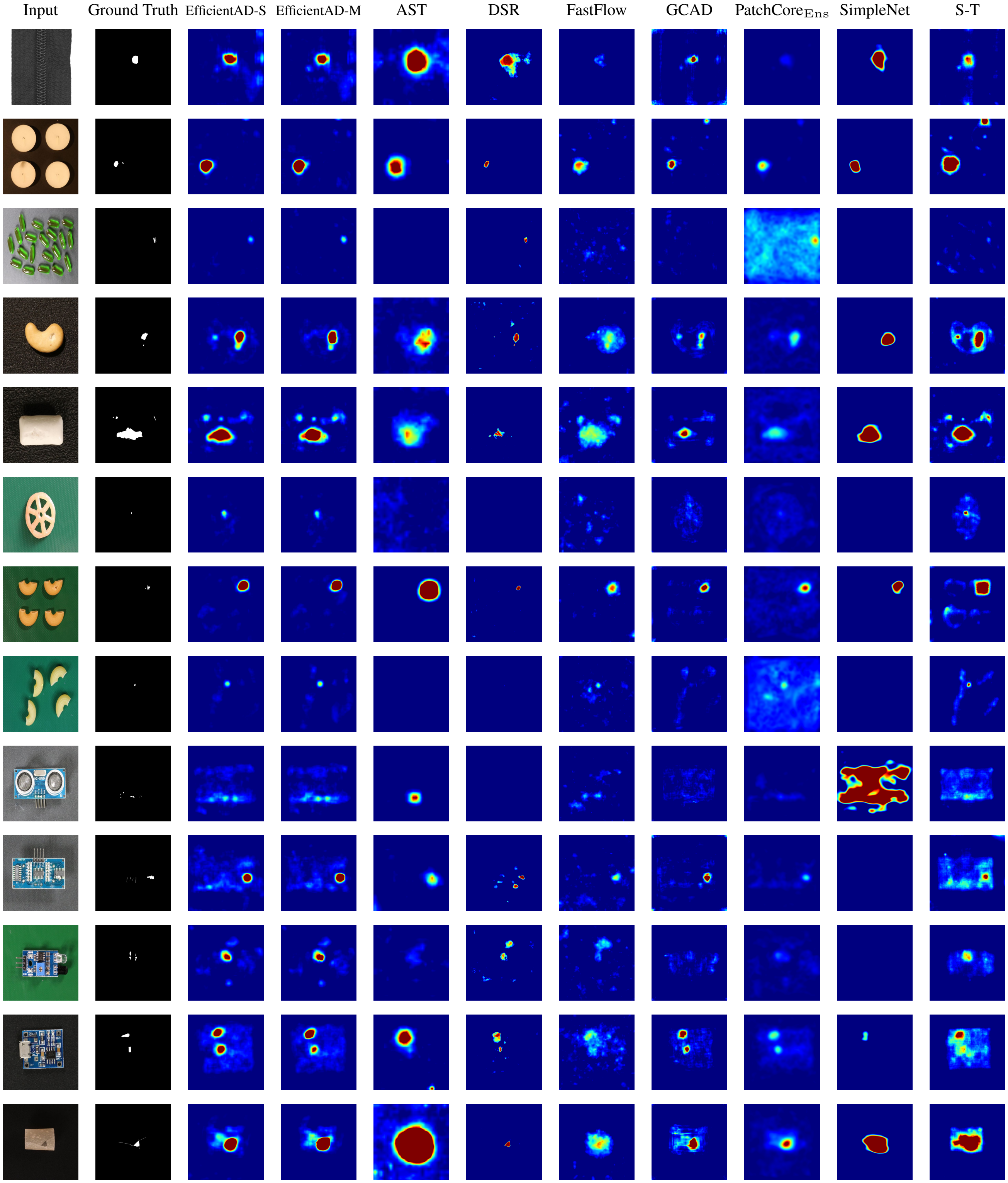

Figure 3. Anomaly maps on anomalous images from MVTec AD and VisA. VisA contains challenging, small anomalies, such as the defect on the non-aligned macaronis or the defect on the fryum two rows above.

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

[2] 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. 3, 6, 7

[3] 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. 3, 6

[4] 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. 3

[5] 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. 6

[6] 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. 7

[7] 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. 6

[8] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015. 2, 5

[9] Zhikang Liu, Yiming Zhou, Yuansheng Xu, and Zilei Wang. Simplenet: A simple network for image anomaly detection and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20402–20411, June 2023. 6

[10] 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. 2, 3, 4, 5, 10

[11] 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. 4, 5, 6

[12] 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. 6

[13] 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. 2, 5

[14] 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. 6

[15] 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. 6

[16] 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. 4

[17] 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. 6

[18] 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. 3