1. Appendices

A. Implementation Details for SegAD

This section describes implementation details for the final classifier (Boosted Random Forest) in SegAD and shows several examples of generating defects for the VAD one-class benchmark. Models were trained with NVIDIA A2000. More details, the code, segmentation maps, and trained models can be found in SegAD GitHub repository[1].

A.1. Final Classifier

We use the same Boosted Random Forest (BRF) for all experiments with SegAD, except for the Ablation Study, which also uses Random Forest (RF) and Boosted Tree (BT). Implementation by XGBoost [4] is used for all three of them. **BRF** in all experiments uses 10 estimators and 200 parallel trees with a learning rate of 0.3 and *binary:logitraw* objective. Maximum depth is set to 5 to avoid overfitting; for the same reason, we set the subsample ratio of columns when constructing each tree (*colsample_bytree*) to 0.6, the same as the subsample ratio per node (*colsample_bynode*) and *subsample*. We also use the L1 regularization [11] with a value of 1. For **RF**, the learning rate is 1.0, the number of estimators is 1, and the number of trees is 2000. For **BT**, the number of estimators is 2000, the number of trees is 1, *colsample_bytree* is default.

A.2. Generating Defects

Generating defects for the one-class benchmark is not optimized and is used just as a demonstration that SegAD can work with one-class tasks as well. In our case, it was sufficient to generate disturbances in the image, which caused anomaly detectors f_k to show higher anomaly scores in their output, which was sufficient to train SegAD. The code to generate defects can be found in the SegAD GitHub repository. Several examples of generated defects can be seen in Fig. 1. Advanced strategies for generating defects, which are described in [6, 12], can be investigated in future work.

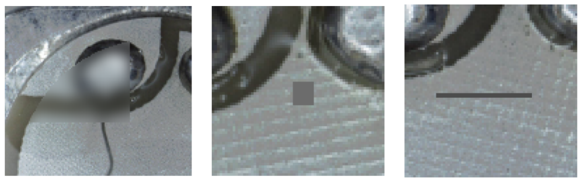


Figure 1. Generated defects, from the left: part of the segment is blurred, small rectangle, thin rectangle. (VAD)

B. Implementation Details for Other Evaluated Methods

In this section, we describe implementation details and training parameters for other methods used in this paper. All models use resolution 256×256 and pretrained feature extractor WideResNet-50-2 by TorchVision [9] unless stated otherwise. No augmentations were used. All existing augmentations were removed to have a more fair comparison and to avoid fitting a specific problem. These removed augmentations include center crop for PatchCore, which improves results as long as defects are close to the center of the image, e.g., for MvTec AD, see EfficientAD paper [2] for more details and random rotation for DRA and DevNet which made their results worse for VAD. Models were trained without early stopping.

Wide ResNet. WideResNet-50-2, pretrained weights by TorchVision [9]. Trained with a batch size of 4, the learning rate is 0.025, uses focal loss [7]. The optimizer is SGD with a momentum of 0.9.

PatchCore. Unofficial implementation by Akcay et al. [1]. The feature extractor uses layers 2 and 3. The coreset sampling ratio is 0.01, and the number of neighbors is 9.

Reverse Distillation. Unofficial implementation by Akcay et al. [1]. The feature extractor uses layers 1, 2, and 3. *Beta1* is 0.5, *beta2* is 0.999. Anomaly maps are combined through addition. Trained for 200 epochs.

FastFlow. Unofficial implementation by Akcay et al. [1]. Trained for 200 epochs (similar as in EfficientAD paper [2]), learning rate is $1e-3$, weight decay is $1e-5$.

EfficientAD. Unofficial implementation by Nelson[2], the original code was not published. Trained for 70000 iterations without an early stopping. 10% of the training dataset was used for normalization. Uses padding in convolutional layers. The

[1]https://github.com/abc-125/segad

[2]https://github.com/nelson1425/EfficientAD

model size is medium.

DevNet. Official implementation by Pang et al. [8]. The feature extractor is ResNet18 by TorchVision [9]. For VAD, the code was modified to use bad images from a separate folder instead of a testing set. For VisA, the code was modified to split bad images from the test set with predefined seeds, the same as for SegAD (ours). The parameter *n_anomaly* (number of bad images used for training) was set to 1000 for high-shot, 100 for low-shot VAD, and 10 for VisA. Trained for 50 epochs.

DRA. Official implementation by Ding et al. [5]. The feature extractor is ResNet18 by TorchVision [9]. The code was modified similarly to DevNet, with the same values for *n_anomaly*. Trained for 30 epochs.

C. Additional details on VAD

This section shows the detailed scheme of the product from VAD and the list of defects with examples. This can be useful for a better understanding of the complexity of the dataset.

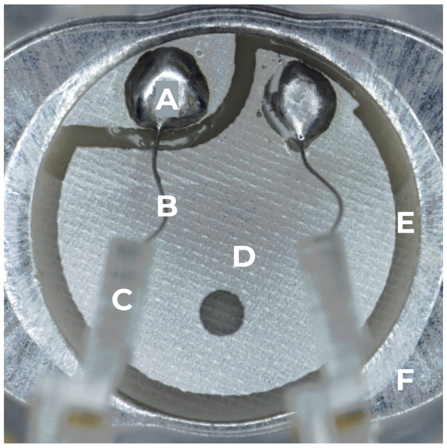
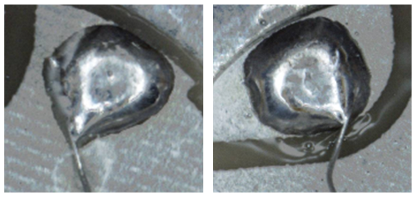
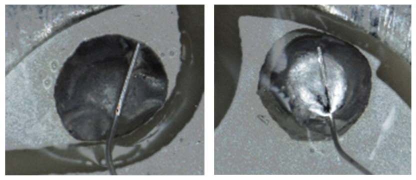
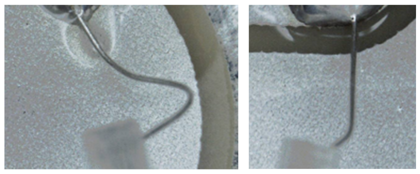
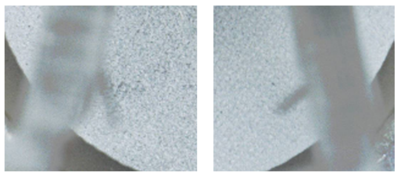
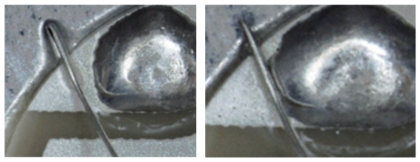
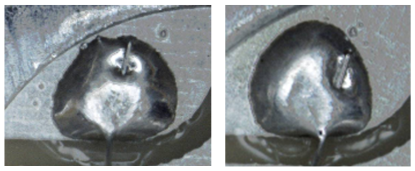
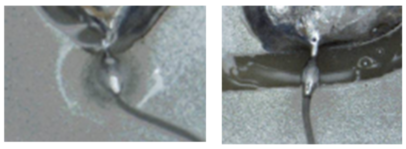
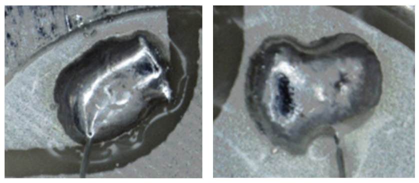
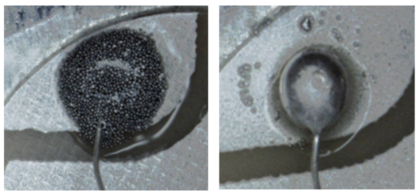
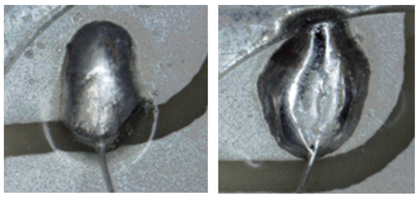
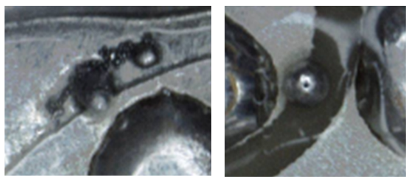
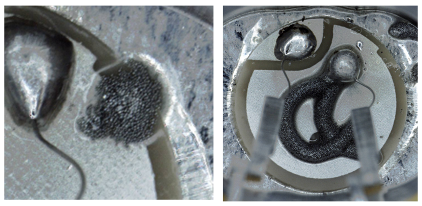
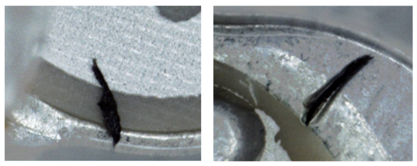
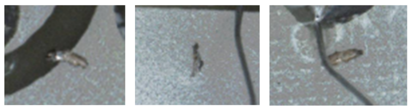
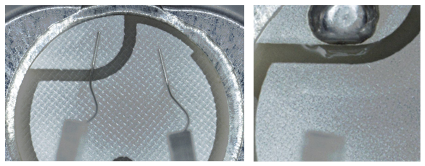
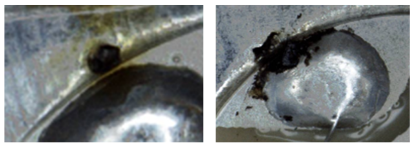
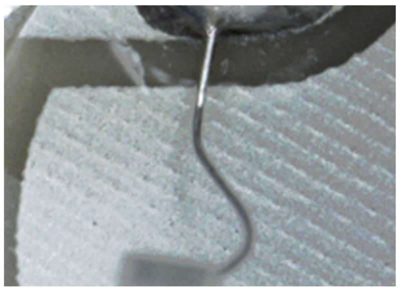
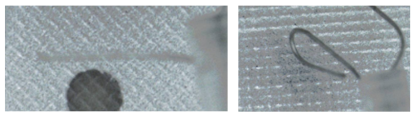
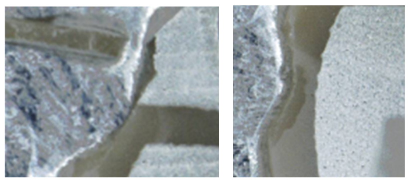


Figure 2. Product (VAD). A - soldering dot, B - wire, C - pin, D - piezo, E - piezo border, F - membrane.

Examples	Description
	The wire is too close to the side of the soldering
	The wire is on the top of the soldering
	The wire is poorly shaped, too bent or too straight

	The wire is not properly connected to the pin
	The wire is out of soldering entirely
	The end of the wire is out
	Solder paste is on the wire
	Soldering is not complete
	Poor quality soldering
	Poorly shaped soldering
	Smaller solder balls pollution

	Misplaced soldering paste
	Plastic pollution
	Metal pollution
	Missing parts
	Burned soldering or wires
	The upper part of the wire is too long
	Additional wires
	Piezo is broken

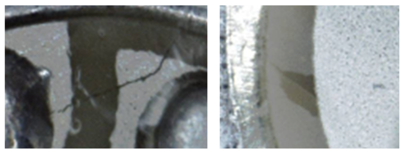
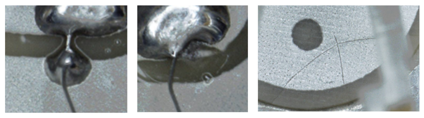
	Cracks in piezo
	Various rare defects

Table 1. Defects in VAD.

D. Detailed results for VisA

In this section, we put the results for our supervised benchmark for VisA per class. It is worth noticing that the VisA paper [13] already has two supervised benchmarks, which were not widely adopted by the anomaly detection community. Their first supervised benchmark includes 60% of all images, good and bad, used for training and the rest for testing. VisA paper already reports results of 99.7 Cl. AUROC with the supervised classifier used together with their method SPD. Such a high result for a supervised method can be explained by the fact that VisA contains a low number of anomaly types per class (4.7 on average), so 60 bad images per class for training were sufficient to recognize the rest. Their second supervised benchmark uses just 10 good and 10 bad images, and it also has not become too popular because anomaly detection methods usually try to utilize many good images, which are much easier to get than bad for real-world problems. There are very few methods that can tackle such problems successfully, such as PatchCore [10]. Our supervised benchmark for VisA is an attempt to fix these problems. We have both a high number of good parts, as well as 10 randomly selected bad parts for training, in a similar way to the popular supervised MVTec AD benchmark [3]. Results per class are shown in Table 3.

Method	candle	capsules	cashew	chewinggum	fryum	macaroni1	macaroni2	pcb1	pcb2	pcb3	pcb4	pipe_fryum	Mean
One-class Anomaly Detection													
RD4AD	94.6	85.4	96.3	98.8	93.7	96.5	84.7	96.0	97.1	95.9	**99.9**	97.5	94.7
EfficientAD	98.3	**92.3**	96.0	**100.0**	98.5	99.1	96.7	98.6	**99.9**	**98.4**	98.3	**99.4**	98.0
Supervised Anomaly Detection													
DRA	96.0	70.6	94.6	91.4	**99.2**	92.2	74.4	92.1	91.6	77.7	90.9	96.5	88.9
DevNet	94.7	78.2	97.4	91.9	98.3	92.1	72.3	94.4	89.5	73.4	95.7	94.3	89.3
SegAD (ours)													
RD4AD + Ours	98.4	80.1	98.9	99.3	96.2	97.4	90.6	96.4	96.3	94.1	**99.9**	95.8	95.3
EfficientAD + Ours	98.7	89.8	98.6	99.9	98.7	**99.6**	98.0	**99.5**	99.7	**98.4**	99.4	99.2	98.3
All AD + Ours	**99.1**	90.1	**99.1**	99.9	98.6	**99.6**	**98.4**	99.2	**99.9**	98.1	99.8	**99.4**	**98.4**

Table 2. Results on VisA dataset, Cl. AUROC values for different classes. The best results are shown in bold. All AD means RD4AD and EfficientAD.

Our method falls short with one class, capsules. This might be explained by some of the bad images having dark spots in the background, which are not defects, but together with image-level labels they provide SegAD with misleading information about the importance of the background. In any case, it shows that our method can sometimes make the results worse if the training data has semantic differences from the testing data. The supervised method from the original VisA high-shot supervised benchmark, mentioned in their paper, also shows lower results on the capsules class, 97.2 Cl. AUROC compared to the average result of 99.7, which might also indicate a difference between test and training data.

[3] https://paperswithcode.com/sota/supervised-anomaly-detection-on-mvtec-ad

E. Additional experiments with MVTec AD

This section shows the results of additional experiments with MVTec AD per class. The task is to investigate how our method will work on another dataset, it is not a benchmark. 20 random bad images from the test set are selected for training SegAD over ten different seeds for the selection process. 10% of good images from the training dataset are used to train SegAD as well. MVTec AD includes 15 classes, 5 of which are textures and 10 are simple objects [3]. Training datasets consist of 242 good images per class on average (compared with 720 for VisA and 2000 for VAD), which leaves us with fewer good images for training SegAD. EfficientAD + Ours shows worse results compared to EfficientAD. This displays that some anomaly detectors might be not working with our method in some cases for reasons that require further investigation. Nevertheless, other anomaly detectors show improvement.

M	bottle	cable	capsule	carpet	grid	hazelnut	leather	metal_nut	pill	screw	tile	toothbrush	transistor	wood	zipper	Mean
One-class Anomaly Detection																
F	**100.0**	88.2	95.0	99.2	**100.0**	78.4	99.6	97.6	94.3	74.7	**100.0**	88.1	89.8	99.4	97.7	93.5
R	99.8	94.7	97.5	99.2	95.3	**100.0**	**100.0**	**100.0**	96.2	**98.4**	**100.0**	93.8	97.1	**99.6**	97.9	98.0
E	**100.0**	91.1	97.6	97.1	98.2	99.6	99.4	99.9	97.9	92.4	99.7	**99.8**	**99.8**	**99.6**	98.8	98.1
SegAD (ours)																
F+O	99.7	71.6	94.4	96.9	97.5	98.2	99.4	96.6	95.2	74.9	99.9	99.2	95.4	98.5	**99.4**	94.5
R+O	99.6	**95.7**	97.9	**98.6**	99.7	**100.0**	**100.0**	**100.0**	97.5	95.7	**100.0**	97.3	95.9	99.0	98.4	**98.4**
E+O	99.5	88.6	95.9	95.6	98.1	99.5	98.5	99.2	97.3	90.4	99.9	95.2	96.7	99.1	96.3	96.7
A+O	**100.0**	92.5	**98.3**	98.5	99.6	**100.0**	**100.0**	99.5	**98.2**	95.2	**100.0**	96.3	99.3	99.0	99.3	**98.4**

Table 3. Results on MVTec AD dataset, Cl. AUROC values for different classes. The best results are shown in bold. M = *Method*, F = *FastFlow*, R = *RD4AD*, E = *EfficientAD*, O = *Ours*, A = *All AD* (includes FastFlow, RD4AD, and EfficientAD).

References

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

[2] Kilian Batzner, Lars Heckler, and Rebecca König. Efficientad: Accurate visual anomaly detection at millisecond-level latencies, 2023. 1

[3] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection. In *CVPR*, pages 9584–9592, 2019. 6

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794, 2016. 1

[5] Choubo Ding, Guansong Pang, and Chunhua Shen. Catching both gray and black swans: Open-set supervised anomaly detection. In *CVPR*, 2022. 2

[6] Songqiao Han, Xiyang Hu, Hailiang Huang, Mingqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. In *Neural Information Processing Systems (NeurIPS)*, 2022. 1

[7] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *TPAMI*, 42(2): 318–327, 2020. 1

[8] Guansong Pang, Chunhua Shen, and Anton van den Hengel. Deep anomaly detection with deviation networks. In *SIGKDD*, pages 353–362, 2019. 2

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *In NeurIPS-W*, 2017. 1, 2

[10] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *CVPR*, pages 14318–14328, 2022. 5

[11] Robert Tibshirani. Regression shrinkage and selection via the lasso. *JRSS*, 58(1):267–288, 1996. 1

[12] Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. DrÆm – a discriminatively trained reconstruction embedding for surface anomaly detection. pages 8310–8319, 2021. 1

[13] Yang Zou, Jongheon Jeong, Latha Pemula, Dongqing Zhang, and Onkar Dabeer. Spot-the-difference self-supervised pre-training for anomaly detection and segmentation. In *ECCV*, pages 392–408. Springer, 2022. 5