VisionAD, a software package of performant anomaly detection algorithms, and Proportion Localised, an interpretable metric

Anonymous authors
Paper under double-blind review

Abstract

We release VisionAD, an anomaly detection library in the domain of images. The library forms the largest and most performant collection of such algorithms to date. Each algorithm is written through a standardised API. The library has a focus on fair benchmarking intended to mitigate the issue of cherry-picked results. It is designed enable rapid experimentation, and the integration of new algorithms is straightforward. In addition, we propose a new metric, Proportion Localised (PL). This reports the proportion of anomalies that are sufficiently localised via discretely classifying each anomaly as localised or not. The metric is far more intuitive as it has a real physical relation, which is attractive to industry-based professionals. We also release a thorough benchmarking of the MVTec dataset consisting of the top 15 algorithms available. We call this the VisionAD leaderboard. We are committed to hosting an updated version of this leaderboard online, and encourage researchers to add, tweak and improve algorithms to climb this leaderboard.

1 Introduction

In the real world many classification problems do not contain sufficient data of various classes. Therefore supervised training of deep learning models presents a challenge. Due to this, the field of anomaly detection has arisen. Anomaly detection refers to the process of training a model using only nominal samples, where the model is intended to classify nominal and anomalous samples during inference. The process falls into the semi-supervised category of machine learning, as the data is supervised in the manner that the training set contains only nominal samples, but unsupervised in the manner that labels are not given to the model.

The de-facto dataset in the field of visual anomaly detection is MVTec (1). This contains over 5000 images across 15 classes of various industrial and household objects. The field of anomaly detection currently enjoys growing attention, with a number of algorithms recently released (2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13).

The PapersWithCode (PWC) leaderboard can be used to judge the performance of these algorithms for MVTec (14). These publications all self-report their results, and the leaderboard is editable by anyone. This means that the positions on the leaderboard are not necessary representative of the relative performance of the algorithms. If researchers wish to experiment with many algorithms, they currently need to download many different Github repos, which contain different data loading code, metric systems, styles and environments. Despite researches best intentions to make their code as friendly to use, this overhead can be significant.

The dominant metrics in the field of anomaly detection have a number of disadvantages. Imagewise-AUC (I-AUC) works at the image level, and does not measure localisation of anomalies. Pixelwise-AUC (P-AUC) and Area Under Per-Region Overlap (AUPRO) contain noise that stems from pixel by pixel matching, cannot be compared across datasets, provide vastly optimistic estimates, and lack a physical meaning.

In light of the above issues, we make the following contributions:

1. We release a comprehensive library of anomaly detection algorithms (VisionAD). All algorithms are standardised through an API. The algorithms share the same data loading and evaluation code,

making fair benchmarking possible. The library has a focus on ease of use, with a simple system for researchers to implement their own algorithms.

2. We release a new metric, Proportion Localised (PL), for anomaly detection. This classifies each anomaly as discretely localised or not, and reports the proportion localised. This is more interpretable, and negates the noise associated with pixel-by-pixel matching. A standalone Python "PIP" package for this metric is published.

3. We undertake a thorough benchmarking of the currently implemented algorithms on the MVTec dataset. This forms the first iteration of the VisionAD leaderboard. An online version of the leaderboard will be updated as and when new algorithms are released or hyperparameters are tweaked.

1.1 Comparison of VisionAD and Anomalib

The most similar library currently available is Anomalib (15). Anomalib contains 14 anomaly detection algorithms. The library contains shared code for metric and data handling. Algorithms are also available to export in OpenVINO and ONNX, and the library comes with a number of tools such as Lightning and Gradio inferencers. The library also contains mechanics for hyperparameters sweeping. Anomalib has enjoyed community development of integrated web UIs.

However, these extensive features of Anomalib may result in costly implementation of algorithms, as algorithms must comply with all these features. This makes fast experimentation difficult. VisionAD is designed to allow researchers to add their own algorithms very easily if they wish. This is a major advantage of VisionAD (the process of adding algorithms is discussed in detail below). This is intended to allow VisionAD to grow fast. This user friendly focus of VisionAD enables quick experimentation. For instance, a researcher can dive straight into the modelling, knowing data loader and evaluation is all handled.

The algorithms of Anomalib are overall less performant and older than those of VisionAD, according to PWC leaderboard (14). Finally VisionAD contains more algorithms, meaning it is the largest, and it is hoped it will grow further as new publications are released.

The authors do not with to discredit Anomalib, which has enjoyed a lot of community attention and is a positive addition to the field. The authors propose that VisionAD can provide value to the community by overcoming the shortfalls of Anomalib. We propose that VisionAD and Anomalib have different strengths and exist for different purposes, as opposed to being in direct competition. Anomalib has rich features for analysis and deployment, whilst VisionAD provides a lightweight alternative with a focus on rapid algorithm development, benchmarking and experimentation.

1.2 Algorithms

Many different classes of algorithms exist in the anomaly detection space. Transfer learning which makes use of a backbone network pre-trained on Imagenet is found in almost all recent anomaly detection algorithms. The most performant algorithms tend to make use of one or more of the concepts described below. Note the descriptions below are not exhaustive, and we refer the reader to these review works for more information (16; 17).

Distillation algorithms make use of the concept that the difference between the predictions of a teacher and a student should highlight anomalous regions. Some algorithms of this class are CDO, RSTDN, and AST (8; 18; 13; 19).

Normalising flows map data of an arbitrary distribution to a normal distribution, using learnable bi-directional transformations. Usually, features created via a pre-trained model are mapped via normalising flows to a normal distribution. In the case of anomalous inputs, the normalising flow function should fail to create a normal distribution, as the function has not seen anomalous data before. Examples of algorithms that fall into this category are Fastflow and Cflow (2; 4; 20; 10).

Memory bank algorithms extract and process features from the training data, then filter these features before saving them into a memory bank. During inference, test features are processed and compared to the features in the memory bank via some distance or nearest neighbours measure. A larger distance indicates the likelihood a feature comes from an anomalous region of an image. Algorithms of this class are Patchcore, PNI, Memseg, and CFA (3; 11; 21; 9).

Synthetic anomalies can also be used to train a model using traditional supervision or refine the outputs using supervision. These anomalies can be entirely synthetic using a mathematical formula such as Perlin noise, or can be constructed using examples for other datasets. Algorithms which use of this are PNI, Memseg, and CDO (11; 21; 8; 22).

Generative algorithms learn to recreate the input, with the intention that during inference, the model will fail to recreate an anomalous region, therefore highlighting its existence. Algorithms which make use of this concept are PPDM (23).

Other algorithms use novel concepts altogether. For example PFM and PEFM train different models to process features to the same output (7; 6).

2 VisionAD

We release VisionAD, a wrapper comprising of many of the best anomaly detection algorithms, all written such that they communicate with the wrapper using a standardised API. The library currently comprises of 15 algorithms, metric calculation code, config files, and evaluation code.

Figure 1 shows this architecture. The wrapper is responsible for loading the config file, calling the data loading code, calling the algorithm through the various standardised API calls, calculating the metrics, and logging the results. More details of the workings of VisionAD are described below.

2.1 Implemented algorithms

The algorithms currently implemented are: Fastflow2d, Patchcore, Simplenet, Ppdm, Msflow, Pfm, Pefm, Cflow, Mspba, Memseg, Fastflow2d+AltUB, Cdo, Cfa, Reverse Distillation, EfficientAD, and Ast (2; 3; 22; 23; 20; 7; 6; 4; 5; 21; 10; 8; 9; 12; 19; 13). As of December 2023, these comprise the top 15 algorithms with code available on the following columns of the MVTec leaderboard, I-AUC, P-AUC, and AUPRO. We choose not to implement algorithms from scratch, due to the risk of introducing error and unfair representation. AltUB is the one exception to this rule, which was implemented entirely from scratch. This is due to the simplicity of the algorithm.

Algorithms were reformatted from their existing structure to fit the standardised structure necessary for VisionAD. Upmost care was taken to ensure the algorithm implementations were correct. Implementations were based on official code where possible, and unofficial code where necessary. MSPBA, Fastflow and Memseg were based on unofficial code. We give due credit to these repositories in the code base. Whilst we cannot guarantee implementations are perfect representations, algorithms were tested against the bottle and carpet class of MVTec to ensure similar results were achieved to the published results.

The authors are committed to adding more algorithms as they are published, and encourage researchers to implement their own algorithms via pull requests.

Certain aspects of the code inherit the licences of the previous code. Therefore care must be taken regarding use of the library. Due to this, the authors recommend academic and non-commercial use.

2.2 Features and use

Adding algorithms is intended to be as easy as possible, as the user needs to implement six methods in a class for integration with the wrapper. VisionAD contains efficient implementations of the standard metrics (I-AUC, P-AUC, AUPRO), alongside the introduced PL. Implementing a new metric is also simple. Due to the ease of writing complex run configurations in Python, Python configuration files control the behavior of each run. Only minimal command line arguments are used: the path to the config file, the GPU device, and an optional run description. For development purposes, an IPython notebook is provided. It includes

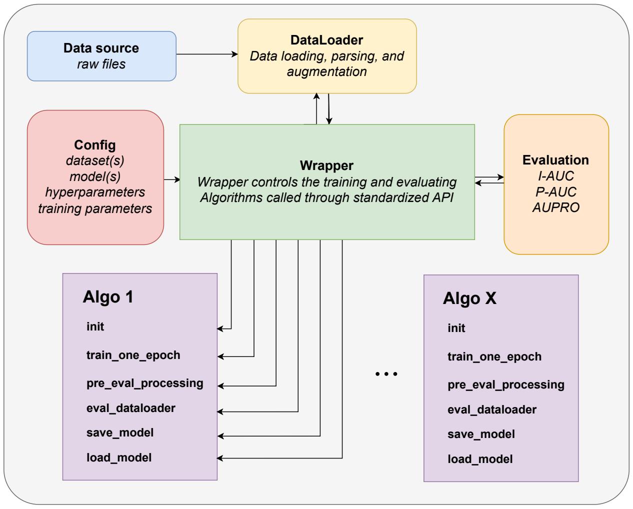


Figure 1: Flow diagram showing the architecture of the VisionAD library.

an algorithm that randomly predicts, allowing researchers to experiment in an interactive environment. A more detailed description of the above features can be found in Appendix 6, and a guide to getting started is available in the readme.md file.

3 Proportion Localised (PL) metric

We propose Proportion Localised (PL). The motivation of PL is to provide a metric that summarises the performance of an algorithm by evaluating whether it sufficiently highlights the approximate areas of the anomalies, as opposed to whether it matches each pixel. We propose that this is more suitable for industrial applications where an algorithm functions correctly if it locates the anomaly, as opposed to whether it replicates the exact shape at the pixel level. We also state the desire for a metric which reports a number with a real physical relation, so that it can be readily understood by non-machine learning professionals.

In summary, PL calculates an IoU score between each anomaly prediction heatmap and anomaly ground truth bounding boxes. To obtain the binary prediction heatmap, the continuous algorithm output needs to be put through a prediction threshold. This is done for many prediction thresholds. For each resultant binary prediction heatmap, the proportion of anomalies with a prediction and ground truth IoU greater than 0.3 is calculated, and highest score found is reported. We refer to this value of 0.3 as the IoU limit, and we justify the choice of 0.3 below. Therefore PL reports the proportion of anomalies which are sufficiently found. The calculation is described in detail in Section 4. The looping through different prediction thresholds and taking the best score means we use the best prediction threshold. This means like P-AUC and AUPRO, the algorithm is independent of prediction threshold.

Note that a major characteristic of this metric is the use of bounding box labels. This is a deliberate decision intended to remove the noise of pixel by pixel matching. Due to this, we deliberately state that the metric evaluates anomaly *localisation*, as opposed to anomaly detection (image-wise), or anomaly segmentation (pixel-wise). Many experts can not agree on the classification of bordering pixels of anomalies, and due to this, we believe that an algorithm should be judged via whether it sufficient highlights the region of the anomaly, as opposed to classifying each pixel exactly. Using the traditional P-AUC and AUPRO, an algorithm will never classify every pixel correctly due to the ambiguity, and therefore can never achieve perfection. However, PL can report perfection if the algorithm produces an IoU greater than the required IoU limit (0.3).

P-AUC/AUPRO drawbacks	PL advantages
Pixel-level matching contains ambiguity and noise	Not effected by pixel-level noise through BB use
Reports an abstract number (no physical meaning)	Reports a number with a real physical meaning
Strong optimism, usually predict between 0.8 and 1.0	Reports linearly between 0.0 and 1.0
Lack of understandability for non-ML professionals	Readily understandable by non-ML professionals

Table 1: Drawbacks of P-AUC and AUPRO, and advantages of PL.

3.1 Weaknesses of current metrics

Table 1 summarises the drawbacks of the current metrics and the advantages of PL. I-AUC fails to properly measure images with more than one anomaly, and fails to measure any degree of anomaly localisation or segmentation. The pixel-wise metrics, P-AUC, AUPRO, amplify any ambiguity in the labelling of pixel-wise labels. Experts disagree on the individual classification of certain pixels, so therefore it is an impossible task for an anomaly detection algorithm to achieve perfect scores on these metrics. In addition to this, anomaly detection algorithms tend to output prediction heatmaps at a lower resolution that the input image, meaning it is impossible for most algorithms to extract the exact shape of an irregular anomaly. All this causes noise in the output of these metrics. We argue that anomaly detection algorithms function as intended if they highlight the approximate area of the anomaly.

The pixel-wise metrics also suffer from vastly optimistic scores due to the large imbalance of anomalous and regular pixels. The P-AUC metric has a low range of output. On an imbalanced dataset, a bad score may be 0.9 and a good score may be 0.99. This is shown with the MVTec dataset. This is undesirable, it is preferable if a bad score is 0 and a good score is 1.

The sIoU metric (24) is innovative, and recognises the drawbacks of P-AUC and AUPRO by functioning at the anomaly level. However it does have a number of downsides. It is slow to run, due to the looping where the predictions have to be separated into non-overlapping regions for many different thresholds. In addition, the multiple counting of false positive prediction area for different anomalies in a single image is over-penalisation.

Finally, none of these metrics are readily understood by users without machine learning expertise. They do not report a number with a physical relation.

3.2 Strengths of the introduced PL

PL directly reports the proportion of anomalies localised. Engineers, inspectors and doctors are likely interested in what anomalies have been 'localised' (found) by an algorithm, where localised is a binary descriptor, and is *true* if the given algorithm has highlighted enough of the defect, and *false* if not.

A major advantage of PL is that it is readily understandable by non-ML professionals. One could understand that PLs of 0.1, 0.5, and 0.99 mean that 10%, 50%, and 99% of anomalies are sufficiently located. These numbers could be used to make an informed decision in an industrial and medical setting. We emphasise that the friendliness of this metric to industry professions is a major advantage.

PL uses bounding box labels, which negates the issue of noise in pixel-wise matching, as algorithms are not judged on whether than locate the exact pixels. Instead they are judged on whether they sufficiently locate the bounding box area of the anomaly. This allows algorithms to output heatmaps at a lower resolution to the image without penalisation (many algorithms do this). It also removes the requirement that algorithms must replicate the exact shape of an anomaly. This brings anomaly detection more in line with the rest of

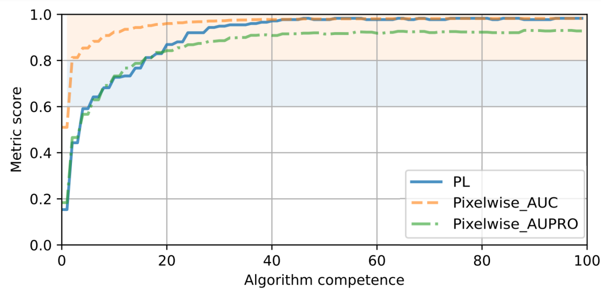


Figure 2: Desirable behaviour of PL compared to P-AUC and AUPRO. For demonstration purposes, the Fastflow algorithm is trained on increasing amounts of data from a solvable subset of the MVTec bottle dataset.

object detection, which uses IoU to measure overlap to discretely classify whether an object is found or not (mAP, AP families) (25; 26; 27).

Finally PL uses the full range of outputs [0-1]. Behaviour of the PL metric is compared to P-AUC and AUPRO in Figure 2. Metrics are plotted for each training step as the competence of the algorithm increases, from random to near perfection. In the y region between 0.6 and 0.8, P-AUC outputs optimistic scores in relation to how the algorithm is performing, which is undesirable. At around 20 training steps, in the y region between 0.8 and 1, P-AUC is very optimistic, predicting close to 1. This P-AUC curve is very nonlinear, with huge jumps early, a sharp turn, and convergent behaviour later. The AUPRO curve is more linear, but converges at 0.9, which does not represent that the algorithm has reached near perfection. PL gives the most linear increase between 5 and 40 training steps, and is able to report across the full range between 0 and 1, showing when perfection is achieved. This behaviour is far more desirable than that of P-AUC and AUPRO.

4 Calculation of PL

Algorithm 1: Proportion Localised

```
function Calculate_PL(pred_set, GT_set)
    IoUs ← []
    for prediction_heatmap, GT_image in pred_set, GT_set do
        anomaly_BB_set ← SplitImageIntoRegions(GT_image)
        centers ← GetTargetCenters(anomaly_BB_set)
        filters ← MakeRegionFilters(centers)
        for anomaly_BB, filter in anomaly_BB_set, filters do
            scaled_anomaly_BB ← EnforceMinDimension(anomaly_BB)
            comb_filter ← LogicalOr(scaled_anomaly_BB,filter)
            pred_filtered ← comb_filter * prediction_heatmap
            IoUs_thresholds ← []
            for threshold in thresholds do
                binary_pred ← pred_filtered > threshold
                IoU ← CalcIoU(binary_pred, scaled_anomaly_BB)
                IoUs_thresholds ← append(IoU)
            IoUs ← append(IoUs_thresholds)
    IoUs ← array(IoUs)
    ret ← IoUs > 0.3
    ret ← MeanAxis0(ret)
    ret ← Max(ret)
    return ret
```

Algorithm 1 outlines the calculation. For each anomaly ground truth and corresponding prediction, the algorithm loops through 25 linearly spaced prediction heatmap thresholds, and creates a 2d array of shape (no.anomalies, no.thresholds). For each threshold, the proportion of anomalies with an IoU greater than 0.3 is calculated. The final score is the maximum of these.

There are a few nuances which make the process work. Firstly, if labels are pixel-wise, then they need to be converted to bounding boxes. This is easily achieved by creating a bounding box over every non-overlapping ground truth region.

It was found that very small anomalies cause activity on the heatmap of an area much larger than the anomaly label. This is because most anomaly detection algorithms output prediction heatmaps at resolutions far lower than the input image, *e.g.* 32×32 in comparison to 256×256. If an anomaly is very thin, it is impossible for the algorithm to achieve the desirable IoU, even if the algorithm sufficiently 'locates' the anomaly. Thus for a given image, the bounding boxes are enforced to have a minimum width and height in relation to the respective image dimension. A value of $1/8^{\text{th}}$ (12.5%) was found to work well. This means each anomaly bounding box has a minimum area of 1.56% of the image. This allows coarse prediction heatmaps to be judged fairly. This minimum dimension enforcement of $1/8^{\text{th}}$ of the image dimension is part of the introduced PL metric. This scaling is shown graphically via the difference between the first and second rows of the examples in Figure 3. This scaling gave a small boost in PL scores, which we believe is justified, as its prevents penalising relatively big predictions centered on small anomalies.

The metrics handle an arbitrary number of anomalies in one image by splitting the prediction heatmap at the boundaries between the centers of each anomaly (each pixel of the heatmap is designated to its nearest anomaly). Each anomaly and heatmap region is handled independently, meaning the metric functions at the anomaly level, as opposed to the image or pixel level. Formally, for a given anomaly x, every pixel which is not closest to its center, or inside its bounding box, is zeroed before the IoU calculation. The process is shown graphically in Figure 3 via the 3rd, 4th, and 5th rows.

PL requires the choice of an IoU limit. Only anomalies with a prediction and ground truth IoU greater than this limit are classified as located. Figure 3 shows randomly chosen images from the MVTec dataset, with the predictions and bounding boxes overlaid. This gives the reader an idea of what different IoU scores look like. We want to choose a IoU limit which excludes unsatisfactory overlap between the prediction and label, whilst including satisfactory overlap. Ultimately, the answer to this question depends on the specific use, and we recommend that the researcher makes their own decision. However, for the standard PL calculation, we set 0.3 as the default IoU limit. We believe 0.3 is the best multiple of 0.05 which separates satisfactory and non-satisfactory overlaps. The third row of Figure 3 and onwards shows satisfactory overlap between the prediction and bounding box, whilst the images before it do not. The value of 0.3 may be perceived as low, but this allows the necessary mismatch between the bounding box target and the prediction. Note that some predictions have what we perceive to be good overlap, but still have IoU values between 0.4 and 0.6. Whilst the published PL uses 0.3 as the limit, the researcher can choose any value, and this can be represented by PLxx, i.e. PL50 for 0.5 limit, and PL25 for 0.25 limit.

The metric does not require the use of any regular testing images. The reader may assume that the PL metric does not penalise false positive predictions. *PL does penalise false positive predictions.* As the anomalous images still contain a significant of non-anomalous regions, if an algorithm were to predict non-anomalous regions as anomalous, the IoU of the closest anomaly would be reduced such that it would not be classified as localised by PL, therefore reducing the final score.

A Python package to calculate PL is also released to allow researchers to quickly integrate these metrics into their code. The metric is coded efficiently. It takes approximately the same amount of time to calculate PL as it takes to calculate P-AUC or AUPRO.

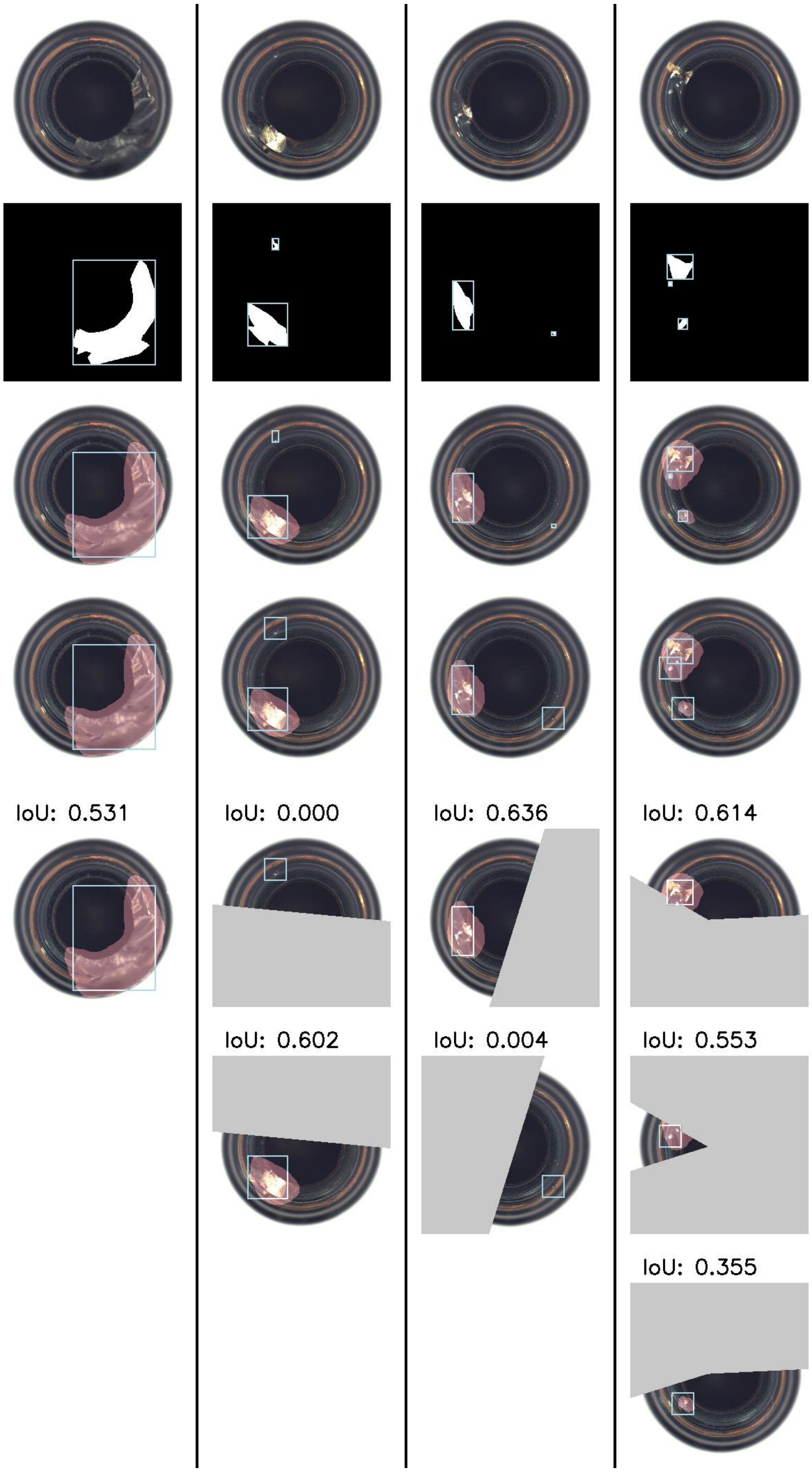


Figure 3: Demonstration of splitting of images with more than one defect. The first row shows the input image, the second row shows the pixelwise ground truth and the corresponding Bounding Boxes (BBs), the third row shows the prediction (red) and BB ground truth, the fourth row shows the equivalent with minimum width enforced BB, and the following rows show the individual anomalies, with the ignored area in grey.

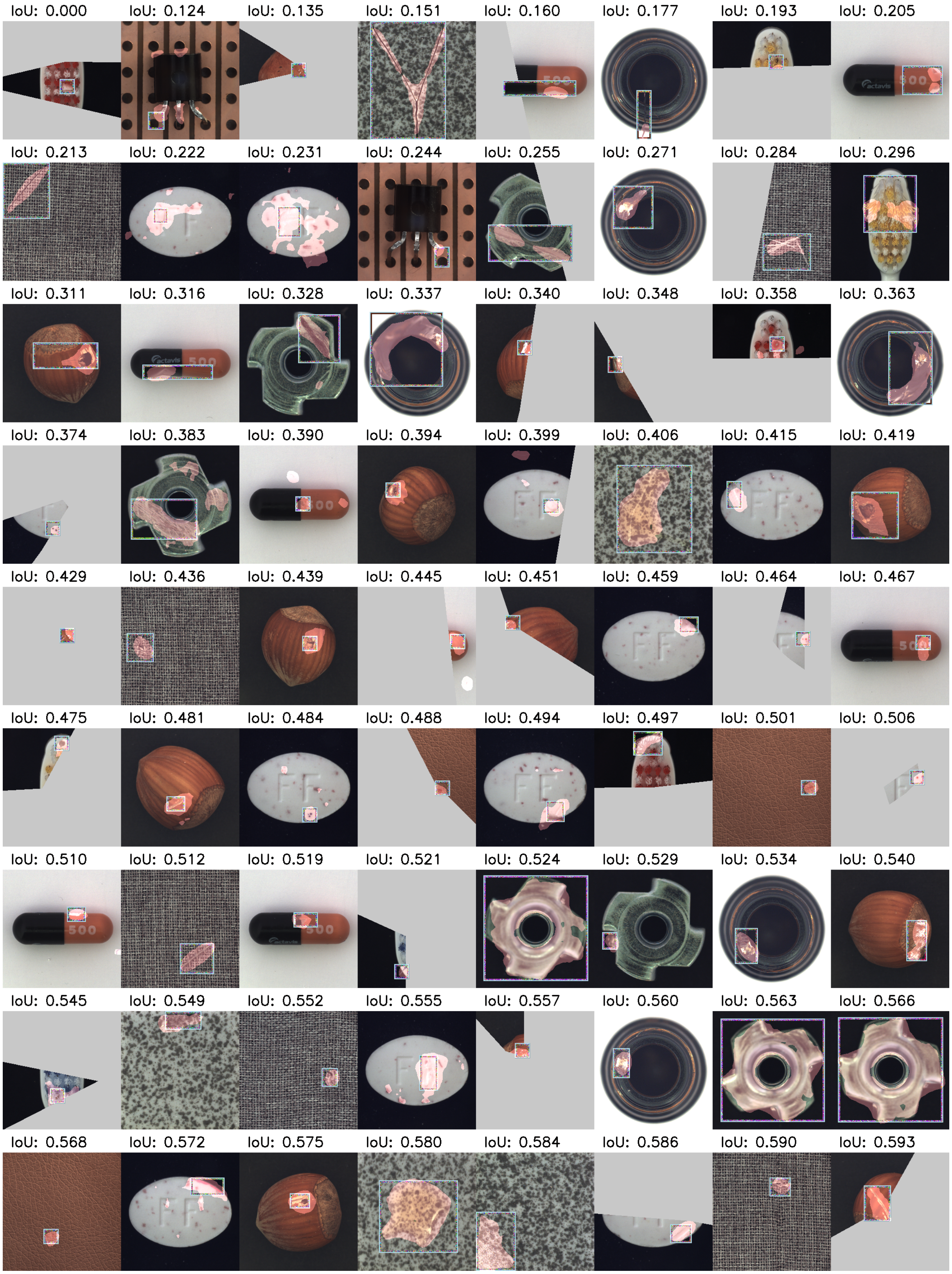


Figure 4: Demonstration of ordered IoU values between prediction (red) and bounding box label (blue). Random images from different MVTec classes, with predictions created using the Patchcore algorithm (3). We recommend an IoU limit of 0.3, as we propose that from this point onwards (the third row), the prediction and bounding box have satisfactory overlap. Note the gray area means ignored pixels, due to there being other closer anomalies in the given image (see above for explanation).

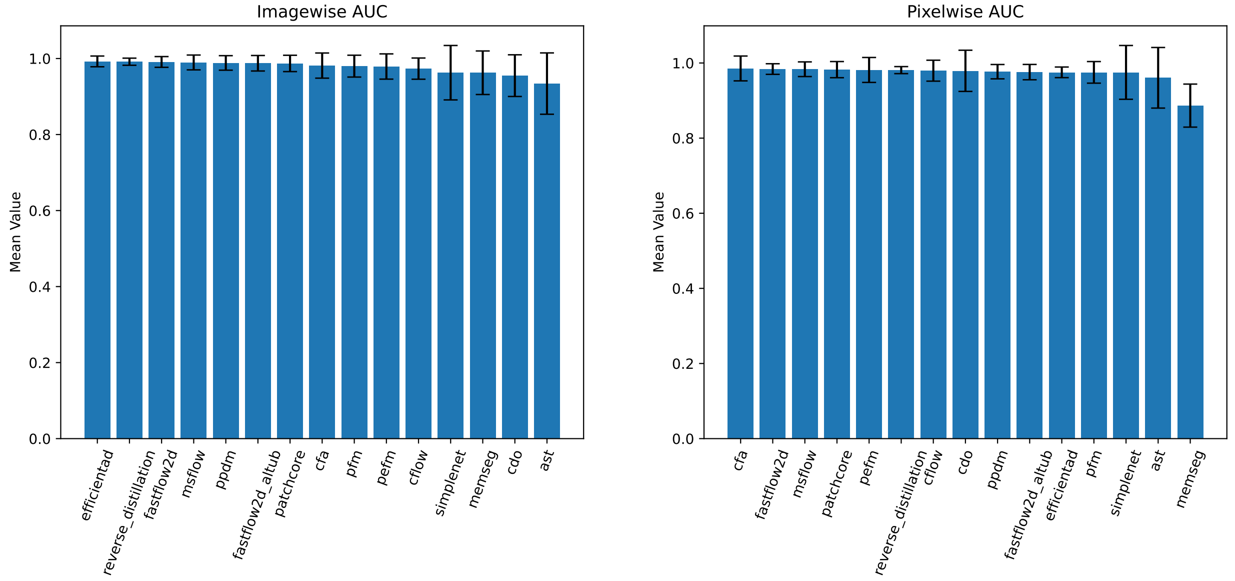
5 MVTec benchmark

5.1 Results

Below we present what we believe are the most comprehensive and fair set of results currently available for the MVTec dataset. We trialled the top 15 anomaly detection algorithms which were selected by the criteria described in Section 2.1.

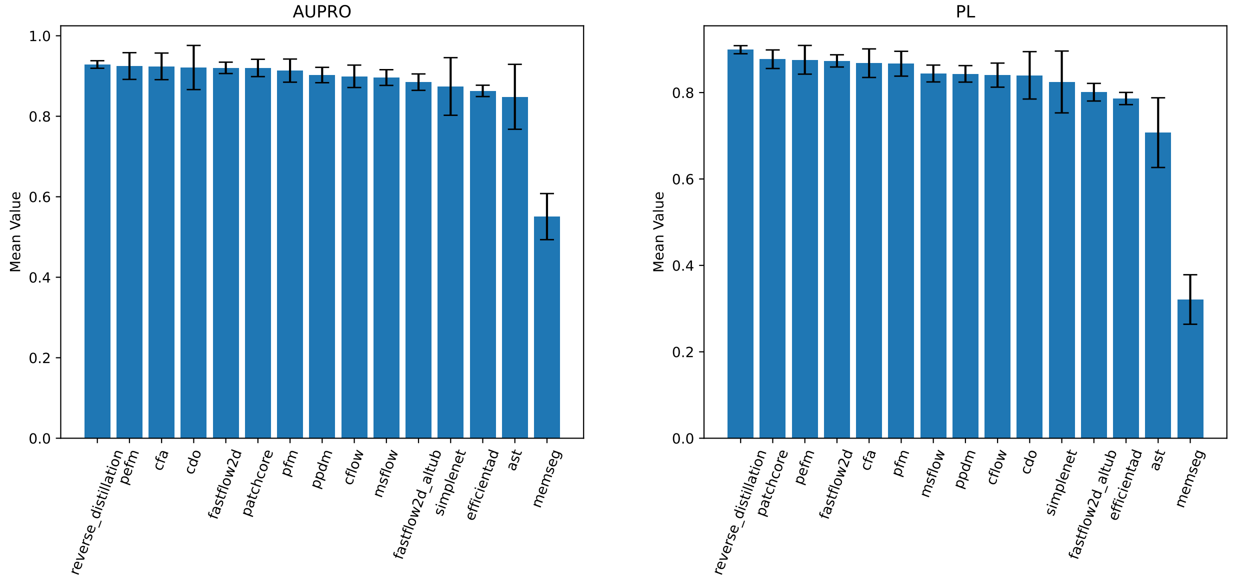
Each algorithm was given a training time of two hours (evaluation and metric calculation is not counted). There were no extra image augmentations carried out in training (we leave a similar run with maximal training augmentations for future work). Each algorithm was run using the default parameters from the publication. The hyperparameters were kept the same across each MVTec class (this may disadvantage some algorithms which change hyperparameters based on MVTec category, e.g. texture or object).

Figure 5 shows the mean value across the MVTec sub datasets for each algorithm and metric. Tables presenting the class breakdown of these results across all algorithms are shown in Appendix 6.



(a) Algorithm rankings based on the Imagewise AUC metric. The mean of the best algorithm is within the error bars of 13 of the other algorithms (all).

(b) Algorithm rankings based on the Pixelwise AUC metric. The mean of the best algorithm is within the error bars of 13 of the other algorithms.



(c) Algorithm rankings based on the AUPRO metric. The mean of the best algorithm is within the error bars of 8 of the other algorithms.

(d) Algorithm rankings based on the PL metric. The mean of the best algorithm is within the error bars of 2 of the other algorithms.

Figure 5: We present the mean of the MVTec dataset for each algorithm. As a measure of variance, we also show the standard deviation of the result across each class as error bars. Each error bar represents one standard deviation above and one standard deviation below the mean.

The best scoring algorithms are EfficientAD and CFA for I-AUC and P-AUC respectively, and Reverse Distillation for both AUPRO and PL. We note that according to the PWC publicly available leaderboard (14), Reverse Distillation is in the 14th position for AUPRO (December 2023), despite being found to be the best by VisionAD for both AUPRO and PL. We use this example to show the importance of not relying on self-reported results to judge algorithms.

For each metric, Figure 5 shows the mean of each algorithm for each metric. The error bars show the standard deviation taken across the classes of the MVTec dataset. The first two graphs, based on the traditional metrics I-AUC and P-AUC, Figures 5a and 5b, show less variation in the means. For each metric, we present how many of the non-top performing algorithm's error bars encapsulate the mean of the top performing algorithm. In the absence of a better calculation, we use these error bars to represent a lack of significance for a given result. The respective top performing algorithms for the I-AUC and P-AUC metrics are encapsulated by the error bars of 14 (all) and 13 of the other algorithms respectively. This implies we should have little confidence in these comparisons, as it implies there is a large degree of random chance in the results. For this measure, AUPRO produces a better value of 8. However, the introduced metric PL produces a value of 2. This implies a researcher can put far more confidence in the significance of the PL results. We present the above as evidence that PL is able to compare algorithms more meaningfully.

Finally, we emphasise the PL results of Table 5 and Figure 5d. These values are interpretable, the mean score of 0.899 for Reverse Distillation means that 89.9% of anomalies are sufficiently located. We can see that the next best algorithm, Patchcore, localises 87.7% of the anomalies. These results demonstrate the value in the explainability of the PL metric.

6 Conclusion and Future Work

We release the VisionAD anomaly detection library intended for fair benchmarking and algorithm development. We also release a novel metric Proportion Localised, intended to give researchers a more intuitive metric by classifying each anomaly as localised or not. We run the VisionAD leaderboard for MVTec, ranking the top 15 algorithms available on the traditional and introduced metric(s).

A limitation of VisionAD is that many algorithms are formatted to work in one system. This can mean certain algorithms are written less naturally than they may be written when standing alone. Formatting different algorithms to fit into one API pattern is a necessary compromise to achieve the benefits VisionAD offers. A limitation of the MVTec VisionAD leaderboard is the time limit of two hours per algorithm. Certain algorithms may perform better with more time. Another limitation is the lack of hyperparameter sweeping and training augmentations (although we used the recommended hyperparameters for each algorithm). We leave longer experiments, training augmentations, and hyperparameter tuning to future iterations of the VisionAD leaderboard.

Future work includes adding more algorithms to VisionAD. More features could be added such as memory logging, and a results visualisation dashboard. VisionAD is backed by a research group committed to improving and maintaining the library.

References

[1] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad - a comprehensive real-world dataset for unsupervised anomaly detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2019.

[2] 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. *CoRR*, abs/2111.07677, 2021.

[3] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14298–14308, 2022.

[4] Denis Gudovskiy, Shun Ishizaka, and Kazuki Kozuka. Cflow-ad: Real-time unsupervised anomaly detection with localization via conditional normalizing flows. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1819–1828, 2022.

[5] Chin-Chia Tsai, Tsung-Hsuan Wu, and Shang-Hong Lai. Multi-scale patch-based representation learning for image anomaly detection and segmentation. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3065–3073, 2022.

[6] Qian Wan, Cao YunKang, Liang Gao, Shen Weiming, and Xinyu Li. Position encoding enhanced feature mapping for image anomaly detection. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022.

[7] Qian Wan, Liang Gao, Xinyu Li, and Long Wen. Unsupervised image anomaly detection and segmentation based on pre-trained feature mapping. *IEEE Transactions on Industrial Informatics*, 2022.

[8] Yunkang Cao, Xiaohao Xu, Zhaoge Liu, and Weiming Shen. Collaborative discrepancy optimization for reliable image anomaly localization. *IEEE Transactions on Industrial Informatics*, pages 1–10, 2023.

[9] Sungwook Lee, Seunghyun Lee, and Byung Cheol Song. Cfa: Coupled-hypersphere-based feature adaptation for target-oriented anomaly localization. *IEEE Access*, 10:78446–78454, 2022.

[10] Yeongmin Kim, Huiwon Jang, DongKeon Lee, and Ho-Jin Choi. Altub: Alternating training method to update base distribution of normalizing flow for anomaly detection, 2022.

[11] Jaehyeok Bae, Jae-Han Lee, and Seyun Kim. Pni : Industrial anomaly detection using position and neighborhood information. *arXiv preprint 10.48550/arXiv.2211.12634*, 2023.

[12] Hanqiu Deng and Xingyu Li. Anomaly detection via reverse distillation from one-class embedding. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9727–9736, 2022.

[13] Marco Rudolph, Tom Wehrbein, Bodo Rosenhahn, and Bastian Wandt. Asymmetric student-teacher networks for industrial anomaly detection. *arXiv preprint 10.48550/ARXIV.2210.07829*, 2022.

[14] MetaAI. Papers with code - mvtec ad benchmark (anomaly detection). `https://paperswithcode.com/sota/anomaly-detection-on-mvtec-ad`.

[15] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A deep learning library for anomaly detection, 2022.

[16] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. Deep learning for anomaly detection: A review. *CoRR*, abs/2007.02500, 2020.

[17] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7, 07 2020.

[18] Shinji Yamada, Satoshi Kamiya, and Kazuhiro Hotta. Reconstructed student-teacher and discriminative networks for anomaly detection, 2022.

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

[20] Cool-Xuan. Cool-xuan/msflow: The official code for “msflow: Multi-scale normalizing flows for unsupervised anomaly detection”.

[21] Minghui Yang, Peng Wu, Jing Liu, and Hui Feng. Memseg: A semi-supervised method for image surface defect detection using differences and commonalities, 2022.

[22] 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.

[23] W. Liu, H. Chang, B. Ma, S. Shan, and X. Chen. Diversity-measurable anomaly detection. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12147–12156, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society.

[24] Robin Chan, Krzysztof Lis, Svenja Uhlemeyer, Hermann Blum, Sina Honari, Roland Siegwart, Pascal Fua, Mathieu Salzmann, and Matthias Rottmann. Segmentmeifyoucan: A benchmark for anomaly segmentation. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. cite arxiv:1405.0312.

[26] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[27] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587. IEEE, 2014.

Appendix A

Adding algorithms

The ease of adding algorithms is intended to be one of the major strengths of VisionAD. The process is described below.

Each algorithm is built from a class called {*algo_name*}Wrapper, which inherits the ModelWrapper class. Using the provided boilerplate code, a researcher only needs to implement six methods for the algorithm to work. Each method is discussed below:

Initialisation: ___init___(self)
Initialise the model and optimisers.

Training: train_one_epoch(self)
Train the model using the self.dataloader_train (automatically attached via the wrapper).

Pre-eval processing: prev_eval_processing(self)
Optionally undertake any necessary pre-processing before evaluation, using access to self.dataloader_train. Left as 'pass' in most algorithms. This cannot be done at the start of eval_outputs_dataloader, as eval_outputs_dataloader may be called on a new class instance in the case of loading the model from disk, where the train generator would not be available.

Evaluation: eval_outputs_dataloader(self, generator, len_generator)
This method is called twice, once with a generator of regular test images, and once with a generator of anomalous test images. To avoid data leakage, this method does not have access to ground truths or paths. The method must iterate over the passed generator, saving to memory an anomaly map and score for each image. Returns a tuple of two dictionaries: a dictionary of anomaly maps: {'*anomaly_map_method_a*': *torch.tensor/np.array of shape (no.images, height, width)*}, and a similar dictionary of scoring methods. Notably, the system allows any number of different methods for creating scores and heatmaps. For instance, the researcher may wish to trial reducing a feature channel reduction via mean, standard deviation, and max, without wishing to rewrite three algorithms. The metric code will provide results for each key and tensor passed through.

Saving: save_model(self, location)
Save the parameters of the model to a given location.

Loading: load_model(self, location)
Load the parameters of the model from a given location.

A trivial example of a randomly classifier is included to help researchers with the small but necessary boilerplate code.

Metrics

VisionAD has a comprehensive list of metrics, included the standard image-wise IoU, pixel-wise IoU, AUPRO, alongside the introduced PL. However, researchers can add metrics as they desire. Researcher needs only to implement a function which accepts an array of targets, predictions, labels and scores, which returns a score.

Run configuration files

Config files are Python files. Each file contains a model_list variable, which is a list of dictionaries each containing the information to run a model (algo_name, model_parameters, epoch, n_epochs_test). Each file also contains a dataset_list, which contains strings which match the dataset keys (discussed below). The wrapper will run each model on each dataset in a given config file.

The choice of Python config file over Yaml is done because it gives the researcher the expressiveness of Python when creating experiments. For instance, each algorithm contains a default set of model_parameters, and these can be imported into different config files whenever this algorithm is ran (with changes made when necessary). Looping can also allow one parameter to be changed whilst other are kept the same via copying the model dictionary inside model_list X times, but changing the target hyperparameter and description each time. Many other similar tricks are available which save the researcher the painful process of copy and pasting parameters.

Basic and complex examples of configuration files are given to allow researchers to get started quickly.

Data configuration files

The dataset file datasets/configure_dataset contains a variable called datasets, which is a dictionary of different keys, where these keys match the strings in the run configuration file mentioned above. Each corresponding item is a dictionary which contains the paths to the training image folder, and testing image folder. The keys of in these dictionaries are self explanatory. To add a dataset a user needs to add a key and dictionary and fill the necessary paths.

Command line use

The wrapper is designed to be as easy as possible to run, with minimal command line arguments. The call *python3 run.py –config configs/my_experiment.py –device 'cuda:0'* is sufficient to load a config file and overwrite the gpu device. The config file contains everything else needed.

Ipython experimentation

Enabling easy experimentation is a goal of VisionAD. Early stage algorithm development is often done in an Ipython environment. The necessary boilerplate code can be loaded into a Ipython environment such as Jupyter notebook. The researcher can experiment with the algorithm class whilst the data loading and evaluation code is handled. A starter notebook is provided to demonstrate this using the trivial random classifier.

Other features

As mentioned above, the algorithm does not give algorithms access to the ground truths during training, to ensure data leakage is not possible. The wrapper also contains a number of other bug checking features such as ensuring the outputs of the algorithms are the right dimensions. The wrapper also measures total training time, training time per image, total inference time, and inference time per image. The wrapper allows all metrics to be logged to Weights and Biases (Wandb) if desired, and code is also provided to pull these results from Wandb and parse them into the tables shown in this publication.

Some algorithms require synthetic anomalies. To facilitate this, we allow a callback to be ran on the training data before it is sent through the dataloader. The training dataloader outputs three items, the training image, the result of this callback, and the image file name. The default of the callback result is to return 0, which would be passed to the algorithm and ignored. However in the case that synthetic anomalies are added to the training images, this callback could return the corresponding mask, which the algorithm can use.

All VisionAD algorithms work on Windows and Linux, CPU and GPU. Currently all algorithms fit on an RTX 3090 24Gb GPU.

Appendix B

Imagewise AUC

Table 2 we show the results using the Pixelwise AUC metric.

	efficientad	reverse_distillation	fastflow2d	msflow	ppdm	fastflow2d_altub	patchcore
bottle	0.999	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
cable	0.979	0.982	0.986	0.997	0.979	0.98	**0.998**
capsule	0.987	0.984	**0.995**	0.991	0.982	0.99	0.991
carpet	0.987	**1.0**	0.988	0.987	0.997	0.989	0.978
grid	0.998	0.994	**1.0**	0.99	0.927	**1.0**	0.994
hazelnut	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
leather	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
metal_nut	**1.0**	**1.0**	0.999	0.997	**1.0**	**1.0**	**1.0**
pill	**0.996**	0.974	0.979	0.962	0.968	0.965	0.975
screw	0.945	0.986	0.945	0.926	0.982	0.923	0.974
tile	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
toothbrush	**1.0**	**1.0**	0.981	**1.0**	**1.0**	0.975	0.914
transistor	**1.0**	0.977	0.998	0.999	**1.0**	**1.0**	0.995
wood	0.998	0.996	0.998	**1.0**	**1.0**	**1.0**	0.993
zipper	**0.997**	0.982	0.996	**0.997**	0.992	0.993	0.995
mean	**0.992**	**0.992**	0.991	0.99	0.988	0.988	0.987

	cfa	pfm	pefm	cflow	simplenet	memseg	cdo	ast
bottle	0.998	**1.0**	**1.0**	0.995	**1.0**	**1.0**	**1.0**	0.987
cable	**0.998**	0.996	0.996	0.978	0.993	0.957	0.908	0.957
capsule	0.958	0.958	0.945	0.963	0.99	0.97	0.83	0.895
carpet	0.984	0.99	0.996	0.999	0.976	0.927	0.999	0.994
grid	0.96	0.991	0.992	0.953	0.728	0.938	0.952	0.759
hazelnut	0.998	**1.0**	**1.0**	0.998	**1.0**	0.992	0.991	0.959
leather	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
metal_nut	0.999	**1.0**	**1.0**	0.997	0.999	**1.0**	0.992	0.965
pill	0.986	0.976	0.987	0.948	0.974	0.91	0.918	0.877
screw	0.868	0.962	0.915	0.934	0.885	0.997	**0.999**	0.735
tile	**1.0**	0.997	0.997	0.999	**1.0**	**1.0**	**1.0**	0.996
toothbrush	**1.0**	0.886	0.892	0.967	0.906	**1.0**	0.853	0.961
transistor	0.999	0.978	0.996	0.904	**1.0**	0.969	0.921	0.976
wood	0.988	0.996	0.997	0.992	0.995	0.999	0.992	0.971
zipper	0.985	0.969	0.97	0.975	0.996	0.778	0.971	0.98
mean	0.981	0.98	0.979	0.973	0.963	0.963	0.955	0.934

Table 2: VisionAD results on the MVTec dataset based on the Imagewise AUC metric.

Pixelwise AUC

Table 3 we show the results using the Pixelwise AUC metric.

	cfa	fastflow2d	msflow	patchcore	pefm	reverse_distillation	cflow	cdo
bottle	0.99	0.989	0.985	0.988	0.985	0.987	0.987	**0.991**
cable	**0.989**	0.977	0.986	0.986	0.98	0.979	0.975	0.97
capsule	0.99	**0.991**	0.989	**0.991**	0.983	0.987	0.99	0.98
carpet	0.985	0.989	0.99	0.99	0.99	**0.992**	0.991	**0.992**
grid	0.978	**0.993**	0.981	0.986	0.988	0.991	0.974	0.986
hazelnut	0.987	0.983	0.982	0.989	0.991	0.991	0.989	**0.992**
leather	0.995	**0.996**	**0.996**	0.995	0.994	0.995	**0.996**	**0.996**
metal_nut	0.985	0.988	0.987	**0.99**	0.974	0.976	0.985	0.985
pill	0.985	0.978	0.988	0.981	0.979	0.984	**0.989**	0.983
screw	0.986	0.989	0.983	0.994	0.991	**0.997**	0.988	0.985
tile	**0.982**	0.968	0.981	0.972	0.962	0.964	0.967	0.974
toothbrush	**0.992**	0.983	0.982	0.989	0.988	0.991	0.984	0.988
transistor	0.984	0.98	0.983	0.937	0.973	0.936	0.948	0.911
wood	0.962	0.969	0.952	0.959	0.96	0.959	0.953	**0.973**
zipper	**0.99**	0.986	0.985	**0.99**	0.984	0.987	0.979	0.982
mean	**0.985**	0.984	0.983	0.982	0.981	0.981	0.98	0.979

	ppdm	fastflow2d_altub	efficientad	pfm	simplenet	ast	memseg
bottle	0.989	0.98	0.957	0.984	0.981	0.928	0.958
cable	0.981	0.964	0.984	0.969	0.975	0.961	0.794
capsule	0.982	0.987	0.985	0.985	0.99	0.97	0.926
carpet	**0.992**	0.985	0.964	0.989	0.984	0.984	0.93
grid	0.916	0.99	0.976	0.988	0.902	0.97	0.895
hazelnut	0.991	0.974	0.978	0.991	0.982	0.981	0.826
leather	0.993	0.995	0.991	0.994	0.994	0.985	0.988
metal_nut	0.98	0.978	0.967	0.975	0.988	0.959	0.771
pill	0.983	0.974	0.982	0.974	0.986	0.953	0.887
screw	0.995	0.979	0.987	0.99	0.99	0.983	0.885
tile	0.962	0.95	0.979	0.963	0.965	0.937	0.968
toothbrush	**0.992**	0.98	0.98	0.987	0.985	0.984	0.965
transistor	0.965	0.974	**0.985**	0.891	0.976	0.927	0.706
wood	0.956	0.948	0.953	0.961	0.942	0.923	0.909
zipper	0.977	0.979	0.961	0.983	0.984	0.967	0.891
mean	0.977	0.976	0.975	0.975	0.975	0.961	0.887

Table 3: VisionAD results on the MVTec dataset based on the Pixelwise AUC metric.

AUPRO

Table 4 we show the results using the Pixelwise AUC metric.

	reverse_distillation	pefm	cfa	cdo	fastflow2d	patchcore	pfm	ppdm
bottle	0.949	0.941	0.941	0.951	0.924	0.938	0.934	**0.952**
cable	0.913	0.91	0.909	0.89	0.84	**0.914**	0.893	0.903
capsule	0.923	0.909	0.936	0.909	**0.938**	0.937	0.922	0.889
carpet	0.957	0.957	0.939	**0.962**	0.946	0.943	0.95	0.952
grid	0.955	0.953	0.897	0.95	**0.956**	0.92	0.951	0.7
hazelnut	0.95	**0.955**	0.926	0.949	0.941	0.943	0.952	0.945
leather	0.979	0.983	0.972	0.984	**0.988**	0.962	0.985	0.972
metal_nut	0.941	0.937	0.923	**0.942**	0.912	0.94	0.933	**0.942**
pill	**0.967**	0.961	0.961	0.964	0.94	0.956	0.96	0.953
screw	**0.978**	0.94	0.922	0.92	0.932	0.966	0.942	0.968
tile	0.846	0.841	**0.901**	0.893	0.875	0.861	0.845	0.829
toothbrush	0.912	0.878	0.905	0.873	0.879	0.875	0.878	**0.913**
transistor	0.805	0.843	0.884	0.768	0.869	0.808	0.714	0.822
wood	0.909	0.924	0.892	0.932	**0.933**	0.883	0.906	0.873
zipper	0.946	0.937	0.951	0.931	0.933	**0.954**	0.937	0.925
mean	**0.929**	0.925	0.924	0.921	0.92	0.92	0.913	0.902

	cflow	msflow	fastflow2d_altub	simplenet	efficientad	ast	memseg
bottle	0.903	0.899	0.881	0.871	0.782	0.814	0.786
cable	0.877	0.866	0.813	0.872	0.841	0.822	0.425
capsule	0.918	0.906	0.923	0.916	0.916	0.841	0.594
carpet	0.95	0.932	0.936	0.901	0.836	0.935	0.704
grid	0.895	0.897	0.937	0.709	0.901	0.848	0.43
hazelnut	0.94	0.917	0.931	0.91	0.905	0.876	0.652
leather	0.984	0.977	0.985	0.955	0.94	0.954	0.786
metal_nut	0.883	0.863	0.89	0.894	0.778	0.857	0.569
pill	0.946	0.935	0.907	0.933	0.889	0.802	0.197
screw	0.938	0.894	0.84	0.941	0.934	0.902	0.197
tile	0.837	0.88	0.831	0.822	0.879	0.79	0.734
toothbrush	0.856	0.812	0.795	0.826	0.831	0.824	0.524
transistor	0.775	0.864	0.784	0.824	**0.899**	0.748	0.401
wood	0.898	0.872	0.917	0.814	0.803	0.847	0.686
zipper	0.889	0.928	0.901	0.922	0.813	0.865	0.574
mean	0.899	0.896	0.885	0.874	0.863	0.848	0.551

Table 4: VisionAD results on the MVTec dataset based on the AUPRO metric.

PL

Table 5 we show the results using the introduced PL metric.

	reverse_distillation	patchcore	pefm	fastflow2d	cfa	pfm	msflow	ppdm
bottle	**0.985**	0.971	**0.985**	0.971	0.941	**0.985**	0.956	0.971
cable	0.925	**0.946**	0.939	0.782	0.932	0.905	0.884	0.898
capsule	0.75	**0.842**	0.6	0.783	0.742	0.683	0.692	0.642
carpet	0.918	0.876	0.918	0.887	0.876	0.876	0.897	0.866
grid	0.888	0.822	0.846	**0.899**	0.787	0.834	0.834	0.485
hazelnut	0.947	0.924	**0.954**	0.893	0.885	**0.954**	0.847	0.939
leather	0.949	0.859	0.98	0.99	0.899	0.99	0.99	0.909
metal_nut	**0.985**	0.97	**0.985**	0.955	0.947	0.97	0.962	**0.985**
pill	0.884	0.892	0.901	0.819	**0.918**	0.897	0.806	0.901
screw	**0.752**	0.745	0.555	0.635	0.504	0.584	0.445	0.701
tile	**0.957**	0.935	0.946	0.946	0.946	0.946	0.935	0.946
toothbrush	**0.929**	0.814	0.857	0.786	0.914	0.871	0.729	**0.929**
transistor	0.795	0.75	0.841	0.909	**0.932**	0.727	**0.932**	0.818
wood	0.905	0.875	0.917	**0.952**	0.887	0.881	0.917	0.786
zipper	0.92	**0.938**	0.915	0.892	0.909	0.898	0.835	0.875
mean	**0.899**	0.877	0.876	0.873	0.868	0.867	0.844	0.843

	cflow	cdo	simplenet	fastflow2d_altub	efficientad	ast	memseg
bottle	0.941	0.971	0.941	0.941	0.824	0.75	0.779
cable	0.864	0.837	0.884	0.721	0.823	0.759	0.374
capsule	0.717	0.525	0.792	0.708	0.75	0.305	0.225
carpet	0.907	**0.948**	0.784	0.856	0.773	0.885	0.227
grid	0.757	0.817	0.58	0.876	0.846	0.68	0.154
hazelnut	0.878	0.947	0.87	0.855	0.855	0.904	0.328
leather	**1.0**	0.99	0.899	**1.0**	0.838	0.899	0.505
metal_nut	0.932	0.962	0.924	0.932	0.833	0.931	0.386
pill	0.853	0.879	0.849	0.703	0.802	0.444	0.168
screw	0.555	0.474	0.642	0.431	0.453	0.321	0.007
tile	0.903	**0.957**	0.935	0.892	0.849	0.93	0.57
toothbrush	0.871	0.829	0.8	0.657	0.814	0.712	0.429
transistor	0.705	0.659	0.864	0.705	0.886	0.682	0.273
wood	0.881	0.935	0.774	0.946	0.815	0.832	0.226
zipper	0.841	0.869	0.83	0.79	0.631	0.574	0.165
mean	0.84	0.84	0.825	0.801	0.786	0.707	0.321

Table 5: VisionAD results on the MVTec dataset based on the introduced PL metric.

Tables 2-5 are what we believe to be the most comprehensive and fair set of results currently available for the MVTec dataset. We trialled the top 15 anomaly detection algorithms which were selected by the criteria described in Section 2.1. EfficientAD and CFA are the best scoring algorithms for I-AUC and P-AUC respectively, and Reverse Distillation is the best scoring algorithm for both AUPRO and PL.