Kenneth Cheng

Professor Pantelis Monogioudis

CS 370-101

3 November 2024

Anomaly Detection Using Anomalib

Introduction

Anomaly detection has various real world applications such as identifying defects in the manufacturing or pharmaceutical domains. Choosing the correct model to perform this is crucial for ensuring reliable detection of anomalies at a sufficient speed. It is the standard for models to be benchmarked by an AUROC curve which is the area under the curve of true positive rate vs false positive rate. False negatives should almost never occur while true negatives are uninteresting and ignored. However, a model must not only have a high AUROC score, but also run quick enough to keep up with incoming information to prevent anomaly detection from becoming a bottleneck. This can be measured by latency, the time to process a single image. Different models have different AUROC scores and latencies. In this paper, we will be explaining how the PatchCore and EfficientAD anomaly detection models work and compare their AUROC scores by evaluating their performance on the MVTec-AD dataset using the Anomalib library.

Dataset

The first step is to download the MVTec-AD dataset using Anomalib, which is a massive dataset consisting of 15 object or texture categories of both defect-free and defective samples used to benchmark anomaly detection models. The anomalies can be structural or textural and have various difficulty levels for detection. Due to normal images being difficult to identify due

to large variations in how they look, subtle anomalies, various types of anomalies, and class imbalance due to many more normal samples than defective samples, the MVTec-AD dataset has proven to be a difficult dataset for anomaly detection models to perform well on. To simplify things, this paper will only focus on the flat surface categories of the dataset: tile, leather, and grid.

Models

After importing the MVTec-AD dataset, the Anomalib library is used to import models for PatchCore and EfficientAD, so that they do not have to be manually coded. This section will explain how these models work behind the scenes.

PatchCore

The PatchCore model is trained using only normal samples with the aim to achieve a high AUROC score and near perfect recall. It makes use of a convolutional neural network where different regions of the image or patches are connected to a set of neurons which apply filters to the patch. Applying the filters to different patches will extract different features. Shallow layers of the CNN will extract very simple features, while deeper layers extract features too specific to the input image, so to extract features that can be generalized to other images outside the dataset, PatchCore takes features from the middle layers. Essentially, it relies on the fact that new objects it tries to predict will contain similar features to other objects it has been pre-trained on, requiring the features to be general. Each feature is represented by the output of a neuron which in turn is a vector. There will typically be an enormous amount of input images each producing vectors for the multiple patches in the image all stored in a memory bank which can become extremely large. Thus, using various methods such as minimax facility location coreset selection, the memory bank can be drastically reduced to form a coreset, saving time and storage when

running the model. This means the coreset is a subsample of the full memory bank, selecting only the parts that best represent the image to allow the model to be fast enough for industrial use. With the coreset formed, the model can finally be run to detect anomalies by extracting patch-features from the image and comparing it its nearest neighbor in the coreset. It then takes the comparison with the maximal distance to be used in calculating the anomaly score which determines whether or not there is an anomaly. Essentially, what this does is try to associate each feature of the new image to a feature the model recognizes in its coreset, and if the two are drastically different, there is likely an irregularity in that location.

EfficientAD

The EfficientAD model is a student-teacher model that is mean to be faster than other models, while maintaining similar AUROC scores, making it ideal for real-world applications where speed matters. The way this is done is by using a CNN/Patch Description Network(PDN) with very few layers: 4 convolutional layers and 2 pooling layers. The teacher is trained on both normal and anomalous data while the student is trained only on normal data. As a result, the teacher model will learn to extract features representing anomalies while the student will not. Thus, by checking to see if the teacher detects a feature that the student does not, anomalies can be detected and localized. In order to make up for the simple CNN/PDN, the model achieves better AUROC scores through using a hard feature loss which avoids outliers/false positives along with an additional loss penalty to prevent the student model from imitating the teacher on anomalous data. Hence, the difference between the student and teacher can be used to create an anomaly map.

Training

After loading the dataset and defining the models, Anomalib can be used to train the models. Upon training, PatchCore created a coreset after searching through around 20k items in the memory bank, using a large amount of RAM. On the other hand, Anomalib provided a pre-trained student and teacher model for EfficientAd to help the model converge faster. PatchCore ended up taking longer to train since it had 24.9M trainable parameters and a huge memory bank as opposed to the 8.1M trainable parameters for EfficientAd. Once the models are trained, they are saved as files so the trained model can be loaded later without retraining. Instead of using Hugging Face, I opted to push the models using Git LFS.

Predictions

Predictions are made by passing in the saved model into Anomalib. After extracting the prediction and labels, Anomalib can also be used to compute the AUROC score and plot a ROC curve. The results can be seen in the chart below.

	PatchCore AUROC	PatchCore latency(items/s)	EfficientAd AUROC	EfficientAd latency(items/s)
Tile	0.9870	3.90	0.9871	3.86
Leather	1.0	3.66	0.8148	3.78
Grid	0.9791	4.14	0.9824	4.09
Average	0.9887	3.90	0.9281	3.91

Note: See github for ROC curve visualization

From these results, we can see that both models had similar AUROC scores and speed despite EfficientAd claiming to be much faster. However, EfficientAd performed poorly when detecting defects in leather since during training, the loss spiked up at the very end. In reflection, I could have introduced an early stopping mechanism or opted to retrain the model. Additionally, EfficientAd came nowhere near the claimed sub millisecond image processing speed(1000+ items/s) on a modern GPU claimed in the paper. This can be due to various factors such as the

lack of input image transformations, Anomalib overhead, etc. In the end, both models ended up with very high AUROC scores but bad latency since my device has only a single GPU. In order to achieve better latency for industrial use, multiple GPUs would be required.