Practical Work in AI
Pushing Baselines for Anomaly Detection on MVTec

Christian Michael Krickl
Johannes Kepler University Linz
k11942694@students.jku.at

Abstract

We attempt to solve the MVTec Anomaly Detection task with traditional models (One Class SVM, Isolation Forest, Kernel Density Estimation, Local Outlier Factor, and Elliptic Envelope).
Various ResNet and ViT configurations are used as preprocessing on the images to overcome the scaling problem for high dimensional input. In addition to that, we perform experiments with respect to data augmentation and do a simple hyperparameter search.
We achieve an average AUROC of 0.970 and gain insights into why some augmentation techniques are counterproductive for specific objects.

1. Introduction

1.1. Project Description

The MVTec AD dataset [1] consists of images from 15 different objects, whereas, each image shows either a good or faulty object (scratched, dislocated, etc).
The task is to train a model for each object only on its good images with the prospect to separate good and faulty images in the test set.
We will attempt to solve this anomaly detection with a pre-trained neural net as image feature extractor and classic anomaly detection methods.

1.2. Motivation

Detecting outliers in the MVTec AD dataset is a rather solved task with the leading models having an AUROC performance of over 0.996 [8].

Most state of the art models employ various neural net architectures and ensemble techniques. We would like to find out what performance we can reach with traditional approaches and gather further data during the process, such as augmentation effectiveness and training/inference time.

2. Pipeline

2.1. Dataset

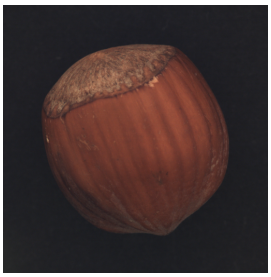
We perform the experiments on each of the 15 objects separately. The dataset contains 60-391 training samples.

Object	Original	Augmented
bottle	209	1254
cable	224	1344
capsule	219	1314
carpet	280	1680
grid	264	1584
hazelnut	391	2346
leather	245	1470
metal_nut	220	1320
pill	267	1602
screw	320	1920
tile	230	1380
toothbrush	60	360
transistor	213	1278
wood	247	1482
zipper	240	1440

Table 1. Training set size

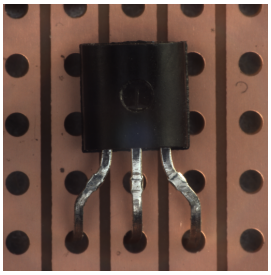
2.2. Augmentation

We augment the training set with the image transformations *rotate 90/180/270* and *flip horizontally/vertical*. Early experiments with other image transformations (such as *resize* or *rotate+crop*) have shown to be counterproductive. The reason for this is that the images are taken in a very controlled environment and are expected to always be in the same lighting, angle and size.
Furthermore, we will see in the results section 4 that some objects benefit from our applied *rotate* and *flip* augmentation because the orientation does not account for being a faulty object (eg. *hazelnut*), while other objects depend on being oriented in a specific way (eg. *transistor*)



(a) *hazelnut* good 1

(b) *hazelnut* good 2



(a) *transistor* good

(b) *transistor* faulty

All experiments are performed on either the augmented or the original training set.

2.3. Feature Extraction

Because our classical models do not scale well with high dimensional input data, we use various pretrained vision neural nets to extract relevant features. We should also benefit from the extraction by having more location independent features.
We use various configurations of *ResNet* [6] and *Vision Transformer (ViT)* [5].

Architecture	Initialization	Extracted Layers
ResNet18	random imagenet1k_v1	layer1 layer2 layer3 layer4
ResNet50	random imagenet1k_v2	layer1 layer2 layer3 layer4
ViT-B/16	random imagenet1k_v1	encoder_layer_0 encoder_layer_1 encoder_layer_2 encoder_layer_3
ViT-B/32	random imagenet1k_v1	encoder_layer_0 encoder_layer_1 encoder_layer_2 encoder_layer_3

Table 2. Image feature extraction models

The initialization of the model weights are sourced from the PyTorch Vision project [4].

2.4. Models

Our anomaly detection models are based on the scikit-learn [7] implementations.
We also perform a basic hyperparameter search, which is based on common values used for the respective model.

2.4.1 PCA for Elliptic Envelope

Most scikit-learn implementations worked as expected, however, for *Elliptic Envelope* the memory complexity was too high for our machine with 32GB RAM.
As a workaround we use PCA to reduce the dimensionality of the input and introduce it as hyperparameter. PCA is in theory not ideal for this job because we eliminate dimensions with low variance, which might be highly relevant for outlier detection.
It was quite surprising that the best performing experiment for *toothbrush* is archived with this approach.

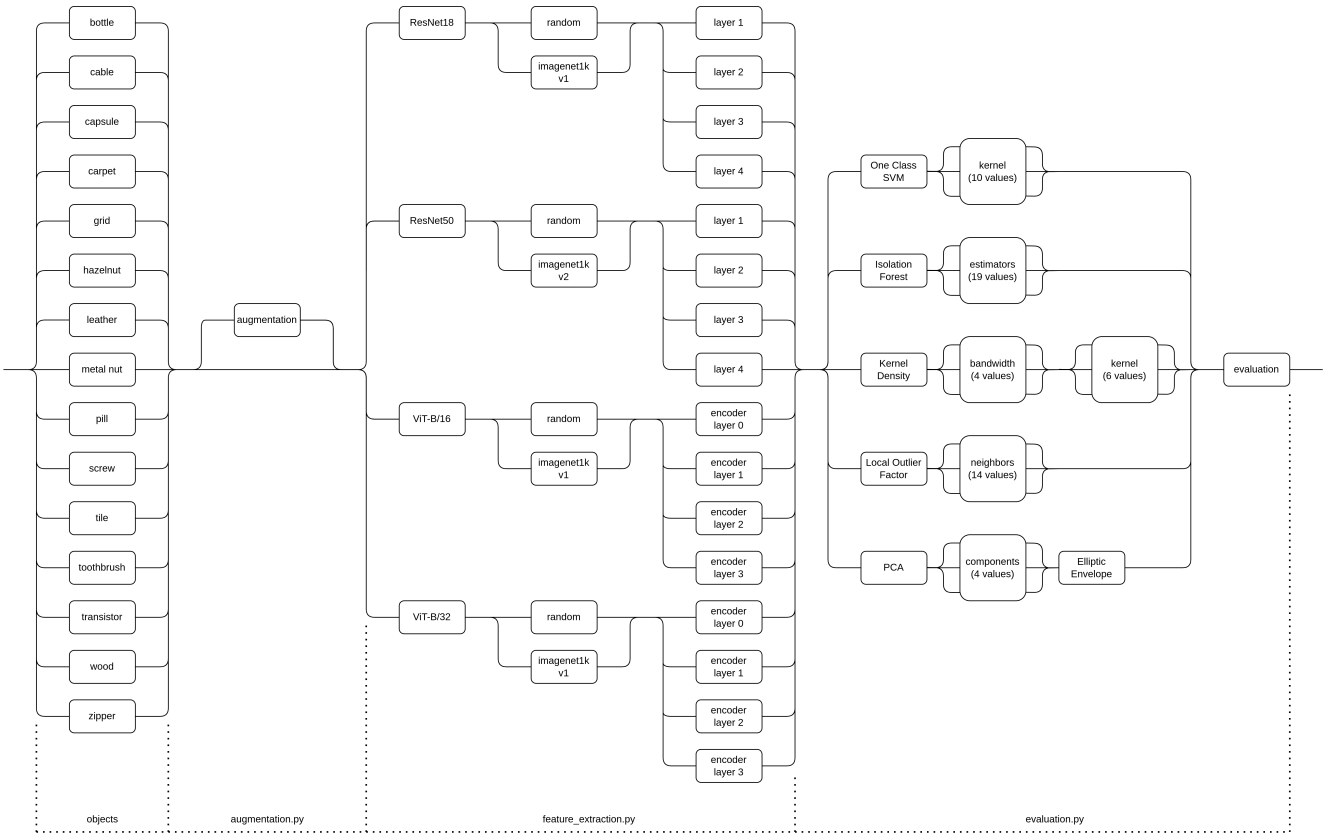


Figure 3. Possible experiment combinations

Model	Hyper-parameter	Values
OneClassSVM	kernel	linear rbf sigmoid poly3, poly5, poly8, poly13, poly21, poly34, poly55
IsolationForest	n_estimators	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
KernelDensity	bandwidth	0.5 1.0 scott silverman
	kernel	gaussian tophat epanechnikov exponential linear cosine
LocalOutlierFactor	n_neighbors	2, 4, 6, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64
EllipticEnvelope	pca components	32, 64, 128, 256

Table 3. Hyperparameters

3. Execution

The code is structured in a way that the augmentation, image feature extraction and training parts can be run separately. In addition to that, the training is made fault tolerant and will resume after a crash in an efficient manner. The augmented images, extracted features from the vision models and evaluation results are cached on disk to make crash recovery backups and improve computation speed.
Python was used for programming this task, including common libraries such as NumPy [2], Pillow [3], scikit-learn [7] and torchvision [4].
The most engineering effort went into making the training to be fault tolerant (so it can run unattended) and the correct extraction of the vision model layers.

3.1. Reproducibility

We ran our experiments on a PC with an AMD Ryzen 5600X and 32GB of memory. Due to bugfixes overall execution took about 2 weeks, however, raw compute time is for augmentation about 1 hour, feature extraction around 2 days, and training/evaluation 9.85 days.
To reproduce the results, 380GB of persistent storage is required.

Code and results as JSON/CSV/SQLite can be found here:
https://github.com/m1ckey/jku-practical-work
Results as Google Sheet:
https://docs.google.com/spreadsheets/d/16narg8QGXiT-fdJuZVnhQ9EA33XIcGu66Z39z9bWwdk/edit

4. Results

From the total 68160 experiments 378 failed due to crashes (all caused by *Local Outlier Factor*).
Nevertheless, we reach an average AUROC of **0.970** and perfectly solve *bottle*, *screw* and *tile*.

4.1. Runtime

Except for *Elliptic Envelope* we observe a linear increase in training time with respect to data dimensionality. The best performing model in this regard is *Isolation Forest* with an average training time of 0.255 seconds.

Object	AUROC
bottle	1.000
cable	0.942
capsule	0.917
carpet	0.945
grid	0.962
hazelnut	0.993
leather	0.992
metal_nut	0.935
pill	0.945
screw	1.000
tile	1.000
toothbrush	0.994
transistor	0.947
wood	0.998
zipper	0.975
average	**0.970**

Table 4. AUROC performance

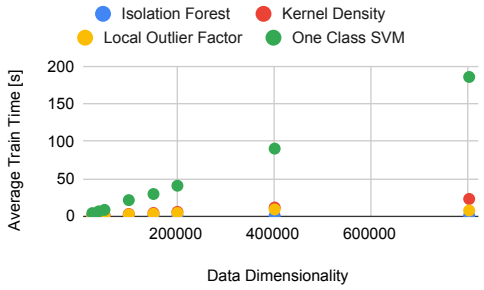


Figure 4. Average train time by data dimensionality

Model	Train time [s]	Test time [s]
Isolation Forest	0.255	1.885
Local Outlier Factor	2.761	1.126
Kernel Density	4.638	9.111
Elliptic Envelope	4.720	0.001
One Class SVM	34.769	2.452

Table 5. Average compute time

Elliptic Envelope has the best inference performance, however, a worse than linear training time complexity.

object	augmented	feature extraction architecture	feature extraction initialization	feature extraction layer	model	hyperparameter	auroc	train time	test time
bottle	TRUE	resnet18	imagenet1k_v1	layer4	LocalOutlierFactor	n_neighbors: 4	1.000	0.619	0.218
cable	TRUE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 4	0.942	1.442	0.435
capsule	TRUE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 12	0.917	1.335	0.418
carpet	TRUE	resnet18	imagenet1k_v1	layer3	IsolationForest	n_estimators: 600	0.945	0.397	1.071
grid	TRUE	vit_base_16	random	layer2	OneClassSVM	kernel: poly5	0.962	72.939	3.376
hazelnut	TRUE	resnet50	imagenet1k_v2	layer3	LocalOutlierFactor	n_neighbors: 6	0.993	15.418	2.274
leather	TRUE	resnet50	imagenet1k_v2	layer4	LocalOutlierFactor	n_neighbors: 20	0.992	3.294	0.995
metal nut	FALSE	resnet50	imagenet1k_v2	layer3	OneClassSVM	kernel: poly8	0.935	1.829	0.773
pill	FALSE	resnet50	imagenet1k_v2	layer2	LocalOutlierFactor	n_neighbors: 2	0.945	3.604	3.214
screw	FALSE	vit_base_32	random	layer0	OneClassSVM	kernel: rbf	1.000	0.441	0.615
tile	FALSE	resnet50	imagenet1k_v2	layer3	IsolationForest	n_estimators: 800	1.000	0.586	5.878
toothbrush	TRUE	resnet50	random	layer2	EllipticEnvelope	pca_n: 32	0.994	0.124	0.000
transistor	FALSE	resnet50	imagenet1k_v2	layer3	LocalOutlierFactor	n_neighbors: 4	0.947	1.578	1.449
wood	TRUE	resnet50	imagenet1k_v2	layer4	OneClassSVM	kernel: rbf	0.998	54.235	3.461
zipper	FALSE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 8	0.975	0.374	0.395

Table 6. Best performing experiments

4.2. Interpretation

From table 4 we can see which objects benefit from augmentation, which is dependent on whether or not the *rotate* and *flip* transformations interfere with the controlled environment, in which the images are taken.

ResNet18 and *ResNet50* seem to be most suitable as feature extraction models, and almost always use the non-random initialization.
When looking at the top-10 models (see Google Sheet / CSV) we observe that the *ViT* models seem to be more effective with a random initialization.
Furthermore, *layer 3* provides the best abstraction on average.

The most successful model across all objects is *Local Outlier Factor*, being involved in 10 out of 15 best performing experiments (including the tie in *screw* and *tile*).

AUROC performance varies from 0.917 to 1.0. No definite link to what causes the worse performance for some objects could be established.
A subjective guess is that our setup does not work well for defects which cover only a small image area. This would align with the worse performance observed for *capsule* (scratch), *carpet* (hole) and *pill* (crack).
A major drawback of this solution is the inability to do anomaly segmentation on the original image.

5. Conclusion

We achieved a surprisingly good detection AUROC of **0.970** with our approach.
The best performing models have a fast training and inference time, however, we needed an extensive search with 68160 experiments to find those, which makes it in total not very computationally efficient.
Designing everything in a fault tolerant way so that it can run unattended required the most engineering effort.

We can confidently conclude that augmentation helps to mitigate the small dataset size, as long as it does not interfere with the object specific controlled environment assumptions.
Furthermore, *Local Outlier Factor* is best suited for this specific approach and yields best average AUROC performance and good training/inference time.

References

[1] MVTec AD. Mvtec ad, 2023. [Online; accessed 12-June-2023]. 1

[2] NumPy Contributors. Numpy, 2023. [Online; accessed 12-June-2023]. 4

[3] Pillow Contributors. Pillow, 2023. [Online; accessed 12-June-2023]. 4

[4] PyTorch Vision Contributors. Pytorch vision, 2023. [Online; accessed 12-June-2023]. 2, 4

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[7] scikit-learn Contributors. scikit-learn, 2023. [Online; accessed 12-June-2023]. 2, 4

[8] Papers with Code Contributors. Papers with code ranking for mvtech ad, 2023. [Online; accessed 12-June-2023]. 1