Deep Learning for Unsupervised Anomaly Detection with Contaminated Data Comparison of Methods and Sensitivity Analysis

Author:
Pascal Bühler

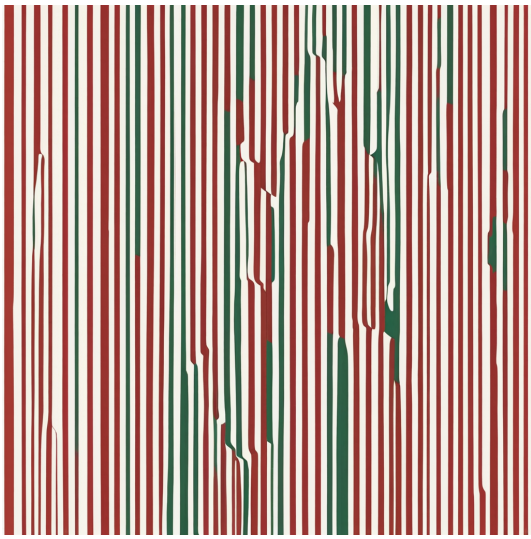

Figure 1: Stable Diffusion interpretation of Unsupervised Anomaly Detection AI-Art [Nvi23]

A project paper presented
in fulfillment of the requirements for the degree of
Master of Science in Engineering, Data Science

Advisors:
Dr. Lilach Goren Huber

Institute of Data Analysis and Process Design

Zurich University of Applied Sciences ZHAW

Switzerland

17th of September 2023

Abstract

The primary objective of anomaly detection is to recognize data points that demonstrate systematic deviations from the majority of data within an unlabeled dataset. A widely held assumption is that a pristine training dataset, uncontaminated by anomalies, is at our disposal; however, in practical applications, this assumption is frequently challenged.

This work explores state-of-the-art anomaly detection as well as a more sophisticated method of Neural Transformation Learning (NTL) by replicating and extending partly the results on the image dataset FMNIST provided by Bosch Research.

The findings show that not only the amount of contamination but their assumption as well as the size of the training data is crucial to the performance. Further, a simple refine strategy does outperform more sophisticated methods of latent outlier exposure (LOE).

In the objective of this work, it was also found that even the well-known benchmark dataset FMNIST contains not only mislabeled data but is contaminated with samples that do not belong in the dataset.

Overall this work motivates further research with the promising NTL model and a simple refine strategy in unsupervised anomaly detection.

Contents

1 Introduction **3**
- 1.1 Anomaly Detection . . . 4
- 1.2 Learning settings . . . 5
- 1.3 Unsupervised Learning in Anomaly Detection . . . 6

2 Data Exploration Fashion-MNIST **7**
- 2.1 Image data . . . 7
- 2.2 Features . . . 9
  - 2.2.1 Dimensionality Reduction . . . 11
  - 2.2.2 Representativeness of Testdata . . . 14
  - 2.2.3 One vs. All . . . 14
  - 2.2.4 Sensitivity and contamination . . . 15
- 2.3 Dataset Summary . . . 16

3 Method **17**
- 3.1 One Class Support Vector Machines . . . 17
- 3.2 Local Outlier Factor . . . 18
- 3.3 Isolation Forest . . . 19
- 3.4 Autoencoder . . . 20
- 3.5 Variational-Autoencoder . . . 21
- 3.6 Neural Transformation Learning . . . 22
  - 3.6.1 NTL Model . . . 22
- 3.7 Frameworks . . . 24
  - 3.7.1 Blind . . . 25
  - 3.7.2 LOE-hard . . . 25
  - 3.7.3 LOE-soft . . . 25
  - 3.7.4 Refine . . . 25

4 Experiment **26**
- 4.1 Metrics, Precision Recall . . . 26
- 4.2 Baseline models SOTA . . . 26
  - 4.2.1 Autoencoder . . . 27
  - 4.2.2 Isolation Forest . . . 28
  - 4.2.3 OCSVM, VAE, and LOF . . . 29
  - 4.2.4 Comparison to NTL blind . . . 29
- 4.3 NTL model in different frameworks . . . 30
  - 4.3.1 Sensitivity of assumed contamination α . . . 30
  - 4.3.2 Sensitivity of reduced training set α . . . 34

5 Conclusion **35**
- 5.1 Outlook . . . 35

6 References **36**

7 Appendix **39**

1 Introduction

Anomaly Detection (AD) is a crucial aspect of various industrial and technological applications. It involves automatically identifying anomalous data instances without being explicitly trained on how anomalies may appear. Anomaly Detection plays a vital role across various sectors, some key applications include:

- **Cybersecurity**: Detecting intrusions, network attacks, and malicious activities in computer systems and networks.
- **Industrial IoT (Internet of Things)**: Monitoring sensor data to identify faults and anomalies in industrial equipment and processes.
- **Fraud Detection**: Identifying fraudulent transactions or activities in financial systems.
- **Health and Medical**: Detecting anomalies in medical images, such as detecting tumors or abnormalities in X-rays or MRIs.
- **Quality Control**: Identifying defects or deviations in manufacturing processes and products.
- **Predictive Maintenance**: Monitoring and detecting anomalies in machinery or equipment to predict failures and optimize maintenance schedules.
- **Natural Language Processing**: Detecting unusual or suspicious language patterns in text data, such as identifying spam emails.
- **Environmental Monitoring**: Detecting anomalies in environmental data, like pollution levels or weather patterns.
- **Anomaly Detection in Time Series Data**: Identifying unusual patterns in time series data, such as sudden spikes or drops in stock prices or sensor readings.
- **Image and Video Analysis**: Detecting unusual objects or events in images or videos, such as surveillance footage.

Researchers and developers creating anomaly detection systems often face a challenge: the training datasets for these systems might not be pristine, containing flawed instances from the beginning. This can lead to the system inadvertently learning from these errors and potentially missing new erroneous instances.
A prevalent method in machine learning, when there are no labels and no distinction between anomalies and normal instances, is called "unsupervised anomaly detection." This project work examines the use of cutting-edge techniques, including autoencoders and isolation forests, for image data with the benchmark dataset, Fashion-MNIST. These methods are also compared to BOSCH's newer Neural Transformation Learning (NTL) [Qiu+22b] technique, replicating the results from their research.
The frequency of anomalies in datasets can be uncertain, [Qiu+22a] tackled this issue in their LOE framework, expanding amongst others, on their NTL method. These frameworks are evaluated for their sensitivity to anomalies, and the analysis is supplemented with new findings. The aim is to provide insights that support the advancement of such systems.

1.1 Anomaly Detection

Anomaly Detection (AD) involves the task of identifying outliers within a dataset. An outlier, in this context, refers to an instance that significantly deviates from the prevailing data patterns. This divergence is always measured in comparison to the remaining dataset, emphasizing that outliers are inherently defined within the context of the rest of the data. The emergence of outliers within a dataset can stem from various origins, as outlined in the comprehensive taxonomy provided by [KD19], categorized in Table 1.

Reason	**Description**
Data error	Mistake in the data. e.g. a human which is recorded 18 meters tall in census data.
Normal variance	Natural outliers, in a normal distribution 99.7% of the data lie within 3 standard deviations from the mean, but the 0.3 % is still valid data e.g. a 2.10 meter tall woman in the height distribution.
Data from other Distribution classes	E.g. Some user IP addresses on a customer-facing website exhibit a higher number of daily page views, which may be due to automated programs or bots accessing the site's content or utilities; this bot traffic is considered "normal" within a separate class called "traffic from programs" distinct from regular human user traffic.
Distributional Assumptions	Incorrect assumptions about the distribution of the normal data. e.g. frequency measure of visitors in a mall which does not account for a special feast.

Table 1: common reasons for outliers in a dataset

Understanding the reasons behind outliers is crucial in deciding the appropriate action after detecting them. In certain situations, the goal is to identify and address outliers directly, like in credit card fraud monitoring, where abnormal transactions are isolated, flagged, and verified for authenticity. On the other hand, in some cases, outliers need to be filtered out as they can distort the final results.
Outlier detection is often used as a preprocessing step for other data science tasks. For instance, removing ultra-high-income earners might be necessary to create a more representative model of a country's income distribution. These outliers are legitimate data points but are intentionally disregarded to draw generalized conclusions.
This work, however, does not deal with the treatment of outliers, instead, it only focuses on detecting them. Further, in this paper, the term **"normal"** is utilized to refer to the data that holds significance, while **"anomaly"** is employed to characterize outliers within the dataset.

1.2 Learning settings

In the realm of Machine Learning, training systems can be broadly categorized into four main approaches: supervised, unsupervised, self-supervised, and semi-supervised learning.

- **Supervised Learning**: In supervised learning, the algorithm is trained on labeled data, where input data is paired with corresponding output labels, enabling the algorithm to make predictions based on learned patterns from the labeled examples.
- **Unsupervised Learning**: Unsupervised learning involves training the algorithm on unlabeled data to discover inherent patterns, structures, or relationships within the data without explicit guidance or predefined output labels.
- **Self-Supervised Learning**: Self-supervised learning is an extension of unsupervised learning. Just like unsupervised learning, it also operates on unlabeled data to uncover hidden patterns and relationships without any external guidance or predefined labels. The key idea behind self-supervised learning is to design a pretext task or auxiliary task that indirectly forces the model to understand the underlying structure or relationships within the data.
- **Semi-Supervised Learning**: Semi-supervised learning combines labeled and unlabeled data during training to leverage the benefits of both approaches, using the labeled data for guidance and the unlabeled data to enhance the model's understanding of the data's underlying patterns.

Each of these approaches brings its own set of advantages and challenges, and the choice of which method to use heavily depends on the specific problem, available data, and desired outcomes.

In Anomaly Detection the following settings of training data are described by [Yoo+22] p.3. If all samples, normal as well as anomalous are labeled (Figure 2a)), the problem can be treated as a supervised classification task and difficulties can arise in the imbalanced label distribution. When only normal samples are available 2b), the problem is known as One-Class (OC) classification.
Various works deal also with the setting of additional unlabeled data in a semi-supervised setting: Figure 2 c),d), and e). where labeled data is only partly available for either normal and anomalous data or just for one of each. This work focuses on the last approach Unsupervised AD (2 f)), which deals with the complete absence of any information regarding the label distribution of normal and anomalous data in the dataset.

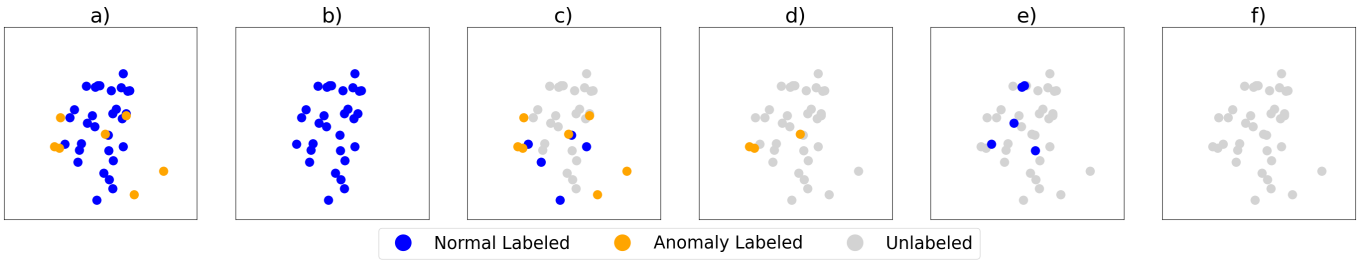

Figure 2: Anomaly Detection Training problem settings: **a)** Normal and Anomalous data is labeled N+A, **b)** only N is available and labeled, **c)** A is labeled, Normal data only partly labeled N+N, **d)** Anomalies only partly labeled A+A. Normal data not labeled N, **e)** Normal data partly labeled N+N+A, **f)** no labels available N+A. Own illustration inspired by [Yoo+22] p2. fig 1.

1.3 Unsupervised Learning in Anomaly Detection

In AD the term Unsupervised is often used when there are no labels to the training-set [Yoo+22]p.2, but it is still assumed that the data only consists of normal samples which makes it actually a supervised task.

Since this work focuses on the unsupervised setting (2 f)) the following formal description can be laid out: Let U be the unlabeled Dataset, A the anomalous dataset and the normal dataset N be subsets of U, with the contamination ratio α_0 where

$$\alpha_0 \in \{0, .., 1\}$$

Generally in Unsupervised Anomaly Detection, only U is observed.
We have the following relationships:

1. U consists of elements from sets N and A,
$$U = N \cup A$$
2. Sets N and A are distinct,
$$N \cap A = \emptyset$$
3. The size of N is given by
$$|N| = \lfloor (1 - \alpha_0) \times |U| \rfloor$$
4. The size of A is given by
$$|A| = \lceil \alpha_0 \times |U| \rceil$$

In other words, the unlabeled set U is the combination of all elements in sets N and A, and these sets are separate, with no overlap in elements. The size of set N is determined by the complement factor $(1 - \alpha_0)$ fraction of the size of U, and the size of set A is determined by the factor α_0 times the size of U.

Hence a "clean" dataset would only consist of normal data and therefore $\alpha_0 = 0$. In practice, however it is mostly not suitable to assume that there is no contamination since a clean dataset would have to be cost intensively reviewed by a human.

A fully unsupervised approach in this regard would then mean no information about the labels is available nor assumptions about the contamination rate in the data are made. Training a model under these conditions is commonly termed as: **blind** training.

Another approach involves the assumption of the real contamination ratio α_0 for training a system. In this work this assumption is denoted by α, which is in line with the description in the work of [Qiu+22b] and [Qiu+22a].

2 Data Exploration Fashion-MNIST

This chapter explores challenges in image data from the Fashion MNIST dataset, particularly in classifying visually similar apparel and footwear. Feature extraction plays a crucial role in transforming raw data into a representative form. ResNet-152 is employed for feature extraction, generating a 2048-dimensional feature vector for each image. Feature scaling is essential for improved model convergence and better performance with distance-based algorithms. T-SNE visualization helps identify distinct clusters corresponding to different classes. The chapter also addresses the anomaly detection task for the specific dataset, and contamination issues for enhanced model training. Overall, insights gained here inform the basis for the experiments in this work.

2.1 Image data

Fashion-MNIST, short FMNIST, is a widely used and well-known dataset from the Zalando research group. It serves as a benchmarking dataset among various machine learning tasks and is integrated into all major framework libraries such as TensorFlow [Mar+15] and PyTorch [Pas+19] and is often used for teaching purposes. As of August 2023, the google scholar [Goo23] search for "Fmnist" results in over 1800 results. Such Benchmark datasets are important for testing different approaches against each other and reporting results on a common dataset [You18]. The dataset contains 70000 28*28 grayscale images of 10 different clothing items classes which are evenly distributed see: Table 2.

Statistic	**Value**
Number of Images	70000
Size of Trainset	60000
Size of Testset	10000
Image Size	$1 \times 28 \times 28$ pixels
Number of Classes	10
Nr of images per class	7000

Table 2: Statistics of FMNIST dataset [XRV17]

In Figure 3, five samples from each of the 10 classes in the FMNIST dataset are presented. Upon inspecting the samples, it becomes evident that certain classes, such as "t-shirt," "shirt," "pullover," and "coat," share visual similarities, making them difficult to distinguish solely based on their appearance. Additionally, some classes, like "sandal" and "sneaker," exhibit considerable resemblance, especially when comparing the third sample of "sandal" with the first sample of "sneaker."

While the class "ankle boot" appears somewhat distinct from other footwear classes, it still exhibits similarities to them. However, classes like "trousers" and "bags" seem to have more distinctive features, making them relatively easier to differentiate from the rest of the classes. These observations demonstrate the inherent challenges in classifying certain categories within the Fashion MNIST dataset, particularly when dealing with visually similar apparel items and footwear.

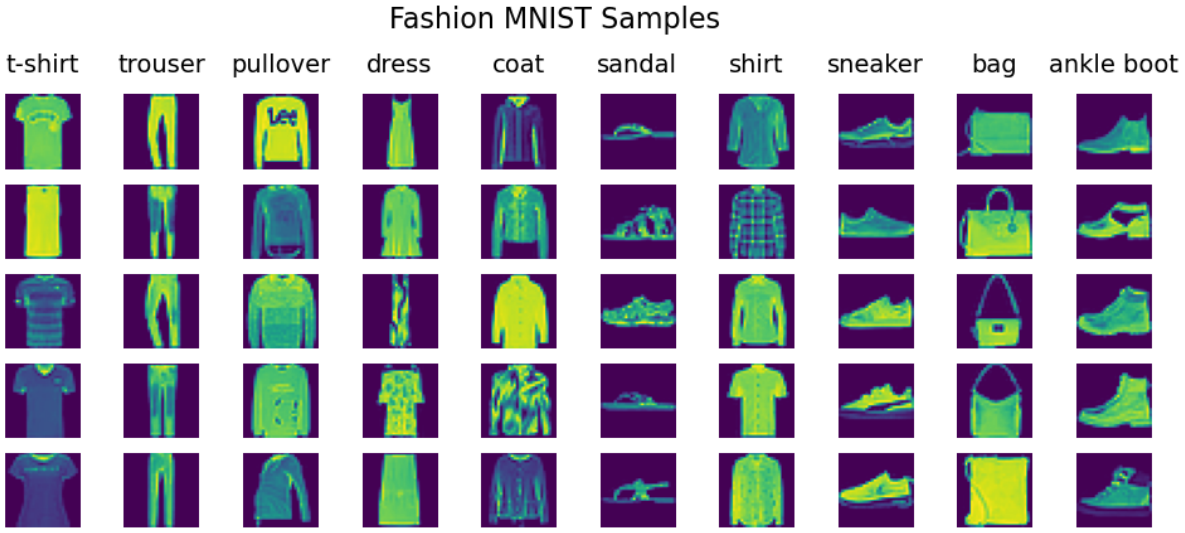

Figure 3: samples from FMNIST Dataset each column is a class with 5 samples. sandal , sneaker and ankle-boot similar. t-shirt, pullover, shirt , coat and dress shares visual similarities. Bag and trouser seem to be rather distinct. (Source: Author's own creation visualizing samples from the FMNIST dataset[XRV17].

2.2 Features

Feature extraction is a crucial step in machine learning and deep learning tasks, where the goal is to transform raw input data into a more compact and representative form, called features [KKN14]3b). These features should capture the essential information and patterns present in the data, making it easier for the learning algorithm to understand and make predictions. Put differently, feature extraction refers to the process of transforming data from its initial space into a new space. This new space is chosen so that the altered data is more appropriate for accomplishing the particular task in question.

The mapping from the data space $\mathcal{X}$ to the feature space $\mathcal{Y}$ can be represented as:

$$f : \mathcal{X} \to \mathcal{Y}$$

explicitly there is a function f which transforms each sample $x \in \mathcal{X}$ into a feature $y \in \mathcal{Y}$:

$$y = f(x)$$

The authors of the [Qiu+22a] [Qiu+22b] used the ResNet-152 pre-trained on Imagenet [Den+09], as a feature extraction function $f(\cdot)$. In this work, the results presented are also based on these features.

ResNet-152 [He+15] is a deep convolutional neural network (CNN)[DV16] architecture that belongs to the ResNet family. It consists of a series of convolutional layers with residual connections, which help alleviate the vanishing gradient [Bro21] problem during training. The "152" in ResNet-152 refers to the total number of layers in the network, making it a very deep model. It gradually learns to extract hierarchical features from the input images, enabling it to identify complex patterns and shapes. When using ResNet-152 for feature extraction in FMNIST, the final fully connected layers responsible for classifying the images into one of the 10 categories are removed. Instead, the global average pooling layer that comes just before the fully connected layers is retained. This pooling layer aggregates the spatial information across all the pixels and outputs a fixed-size feature vector for each image [DV16]. The global average pooling layer in ResNet-152 takes the output feature map from the last convolutional layer and computes the mean value for each feature map channel. This process reduces the spatial dimensions to 1x1 while retaining the channel dimensions. In the case of ResNet-152, this results in a 2048-dimensional feature vector for each image in the FMNIST dataset.

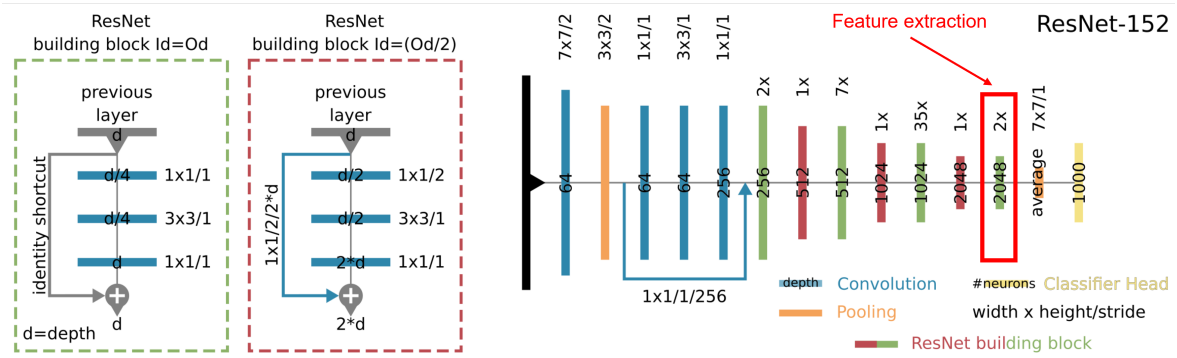

Figure 4: Schematic Description of a ResNet152 CNN, the Last Pooling layer is used to extract features. Source: [Hoe20] fig 7.

By analyzing all the extracted features from the FMNIST dataset a maximum value of 18.3, a minimum of 0, and a mean overall of the features of 0.42 is observed. Overall most of the features are right-skewed, as a visual inspection shows (Figure 5, where only 20 percent of the features are visualized).

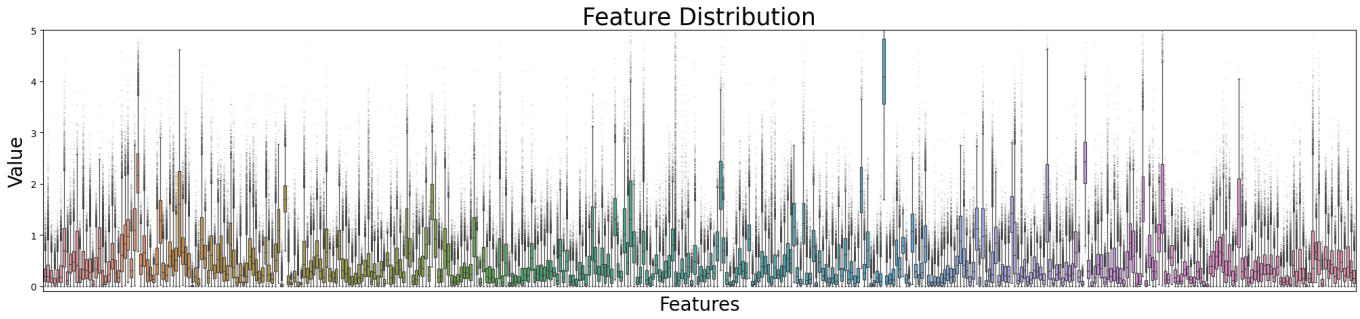

Figure 5: Boxplots of 20 percent of the features from the FMNIST dataset extracted by ResNet-152, Most of the features are right skewed, (Source: Author's own creation using Python)

The observation of the feature statistics leads to a technique called "Feature Scaling" which is a crucial preprocessing step in machine learning algorithms, because it brings the features to a common scale, ensuring that all dimensions have comparable importance during the learning process [Bha23][Ras14]. When features have significantly different scales, it can lead to several issues that adversely affect model performance.

One major concern is the effect on gradient-based optimization algorithms. Many machine learning models, such as neural networks, use optimization techniques that rely on gradient descent to minimize the loss function. If features have vastly different ranges, the gradients of the loss function with respect to each feature will also have different magnitudes. As a result, certain dimensions with larger scales may dominate the learning process, making it difficult for the model to converge to an optimal solution. This can result in slow convergence, or in some cases, the model may not converge at all.
Another problem that arises from unscaled features is the impact on distance-based algorithms. In clustering algorithms like k-means or in distance-based classifiers like k-nearest neighbors (KNN) and support vector machines (SVM) used in section 3.2,3.1, the distance between data points plays a critical role. When features are not scaled, dimensions with larger values can dominate the distance metric, leading to suboptimal clustering or classification results.

Additionally, unscaled features can adversely affect regularization techniques. Regularization is used to prevent overfitting by penalizing large weights in the model. When features have different scales, the regularization terms may have varying impacts on each feature's weights, leading to biased regularization. Proper scaling ensures that the regularization term works consistently across all features and helps the model generalize better to unseen data.
Interpreting the importance of features in a model can also be misleading when features are not scaled. Some algorithms, such as decision trees or random forests, provide feature importance scores. Without scaling, features with larger magnitudes may appear more important solely due to their scale, even if they don't carry significant information for the task. Scaling ensures that feature importance scores are based on the true contributions of each feature to the model's performance.
The Methods in [Qiu+22a] [Qiu+22b] do not deal with scaling features, they were processed as returned by ResNet-152. However for the baseline models in section 4.2 different scaling techniques are applied individually.

2.2.1 Dimensionality Reduction

Since the extracted features have very high dimensionality it is complicated to interpret them meaning-wise. As a solution to this challenge, dimensionality reduction algorithms aim to tackle the curse of dimensionality [KM17], with the goal of improving data quality by reducing data complexity. They are mainly categorized into feature selection and feature extraction. The former looks for the most informative features and eliminates less informative ones. The latter combines the features into fewer new features through algebraic transformations. Examples of such algorithms are PCA, MDS, LLE, or t-SNE. Each of these algorithms has different specification and their performance does also depend on the data characteristics, quality and size, this topic is thoroughly discussed in [ASS21].

In this work t-SNE [vH08] is applied to the embedded feature vectors from ResNet-152 and visualized in a 2D representation. t-SNE works by first creating probability distributions that represent pairwise similarities between data points in the high-dimensional space and in the lower-dimensional space. It then minimizes the difference between these distributions using gradient descent, which effectively preserves the local structure of the data while mapping it to a lower-dimensional space. This helps reveal clusters and patterns in the data when visualized.

To be consistent with the formulation of 2.2 The mapping from the featurespace $\mathcal{Y}$ to the lower dimensional featurespace $\mathcal{Z}$ can be represented as:

$$g : \mathcal{Y} \to \mathcal{Z}$$

explicitly there is a function g which transforms each featuresample $x \in \mathcal{Y}$ into a feature $y \in \mathcal{Z}$:

$$y = g(x)$$

$g(\cdot)$ in this case is the t-SNE algorithm.

When the t-SNE algorithm is applied to the features, the resulting data points and their according original image can be visualized in the 2-dimensional space as in Figure 6. This reduction has also been performed directly on the original images $\mathcal{X} \to \mathcal{Z}$, Appendix Figure 24. However, the results from the reduction on the features seem to contain more information about the underlying structure which is a pointer that the ResNet-152 gained meaningful features and therefore further analysis is held on the representation from the features directly.

By visual inspection of Figure 6, 4 rather distinct clusters become apparent: left: shoes, bottom-center: bags, right: upper-body-clothing, top-center: trousers. The interpretation is as follows: starting counterclockwise on the shoe cluster: from top to bottom: sandal, sneaker and ankle-boot. The sandals with high heels are to the right and sandals which are flat are more to the left, whereas sandals with more material to it are closer to the sneakers. In the sneaker cluster there is also a smooth transition from more flat shoes to rather higher shoes from top to bottom. The ankle-boot cluster shows the ankle boots with heels on the bottom and more closed ones on the top. In the bags cluster, bags with handles are forming a cluster on the lower side, whereas pures are forming the upper half of the cluster. The upper-body-clothing cluster seems to be a bit more difficult to interpret, it has the pullovers on the bottom with a transition to the center to the coats. Shirts are scattered almost over the whole cluster but are probably most dense to the center-right. T-shirts form the upper right part of the cluster with printed logos being at the top. From the center of the cluster to the top left the dresses are formed. In the transition to the top center cluster of trousers, there seem to be dresses that resemble similarity with some kind of overall. The cluster of trousers itself is formed distinct from the others but a structure within is not so clearly visible. Especially the overall groups of 4 clusters represent what was also observed by just inspecting a few samples in subsection 2.1 within Figure 3.

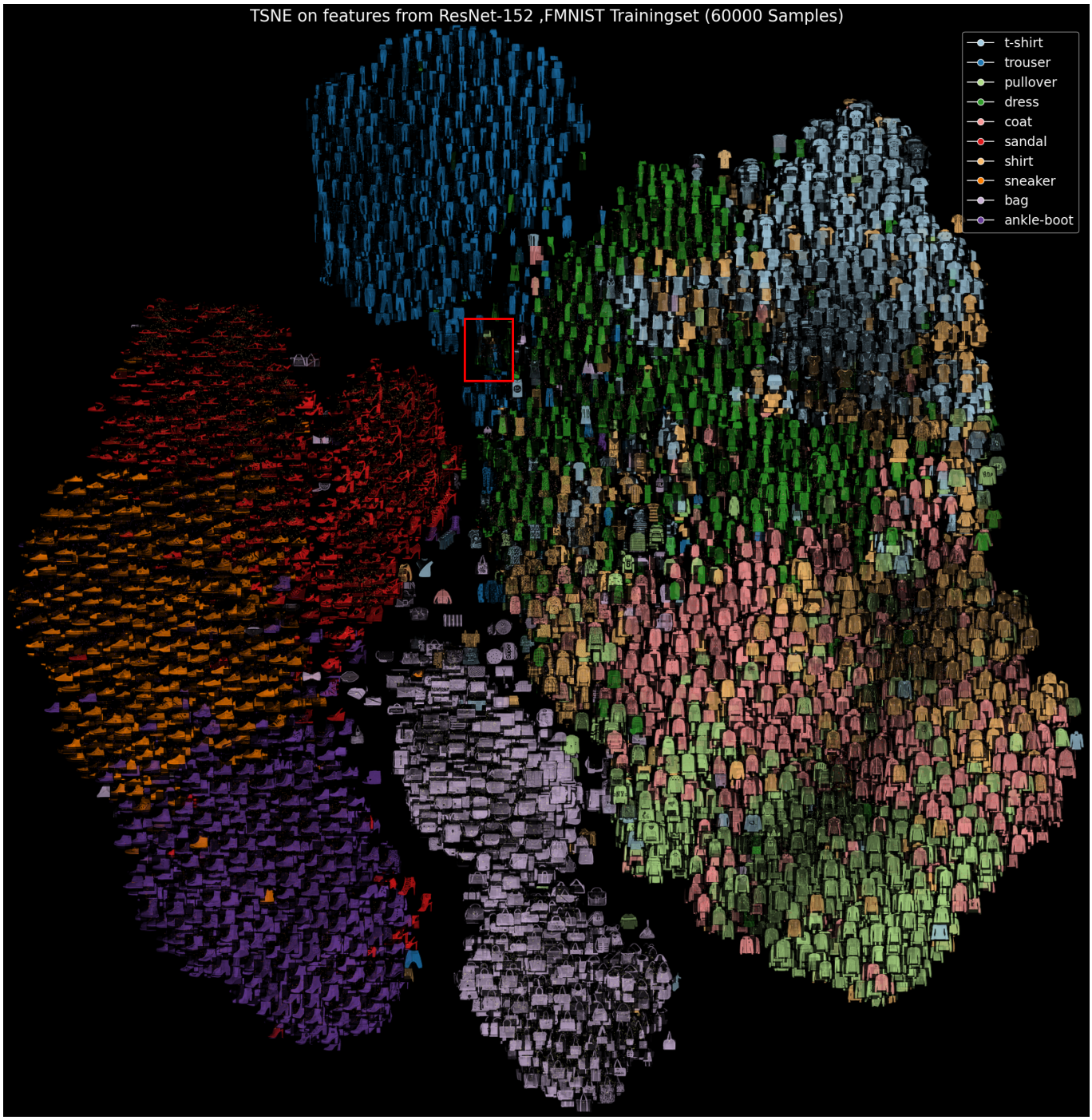

Figure 6: Visualization of T-SNE on feature vectors colorized by the labels, there are 4 main clusters: on top, there is trouser, on the bottom center bags. To the right, there is a bigger cluster with upper-body clothing: t-shirt, pullover, dress, coat and shirt. Left side: cluster with shoes: sandal, sneaker and ankle-boot. The red box marks the area where humans are in the images as well., (Source: Author's own creation using Python)

If carefully inspected, an interesting part in Figure 6 between the right cluster upper-body-clothing and the top cluster trouser is observed. When the images in the area marked by the red box are inspected, it becomes apparent that they all resemble similarities. The images in this region contain a person and not only an object. The person in these images wears mostly lower and upper body clothing as well as shoes, as far that can be seen in a 28by28 image. Interestingly the positions in the 2D plane of these images are between dress and trousers and seem to be also close to the shoe cluster which could be interpreted as the algorithm found several features in such images to place them there. In Figure 7, 52 of such images are visualized from this part of the representation. The labels are not clear on these images since they mostly contain several clothing items. So they should actually be excluded from the dataset.

Mislabeled data is a common source of error and can happen in the process of labeling. In the work of [MM19] they found that at least 64 instances are mislabeled in the FMNIST dataset, however, only one of these instances is also found in the inspection in Figure 7 meaning that it can be concluded that at least 115 samples are either mislabeled or contain inconclusive data. This is only for the training set, which means that at least 0.19 percent of the training dataset is contaminated with wrongly assigned labels or humans. This is further reasoning for the assumption of having contaminated data even if it appears to be clean.
On google scholar [Goo23] the "FMNIST" search results in ca. 1800 results (August 2023), However, it is not evaluated how many of these works deal also with the contamination of the dataset, and if it would make a difference to the performance of the results inferred with models trained on the set. Most probably the impact would be marginal since the deviations in the image statistic are very small and could be viewed as a sort of noise.

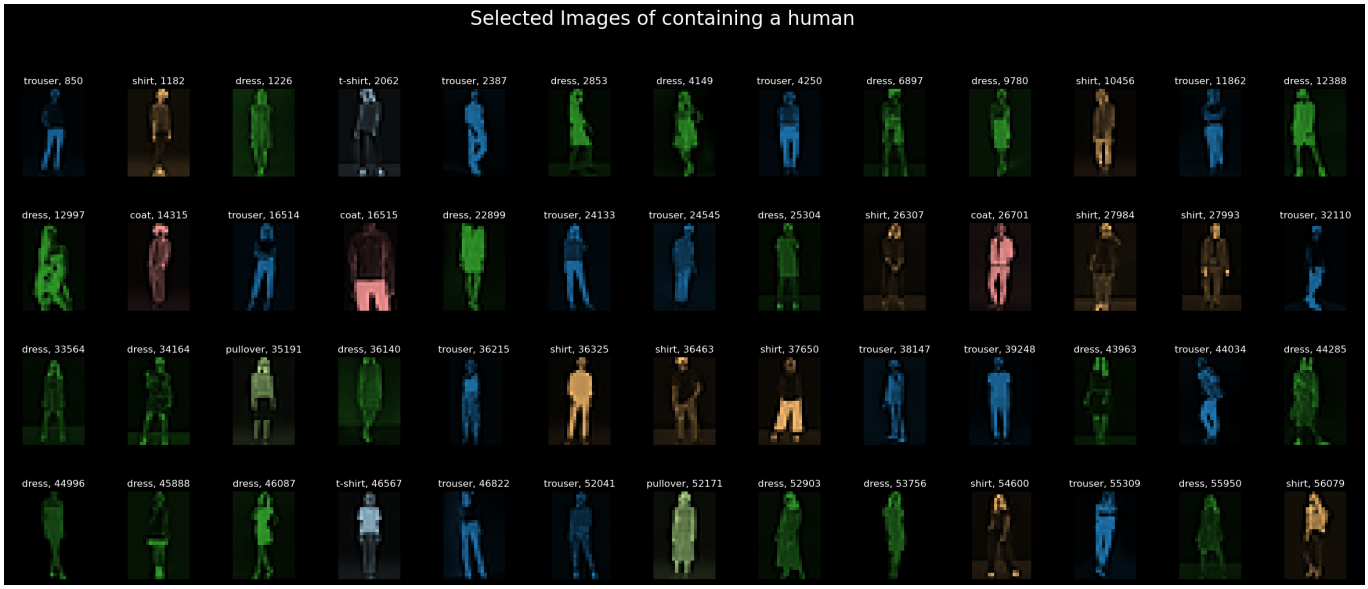

Figure 7: Samples from the location in the red box of Figure 6, these samples are mostly inconclusive about their content regarding the label, the images where a human is wearing a dress would probably be the most appropriate one. (Source: Author's own creation using Python)

2.2.2 Representativeness of Testdata

To visualize the split of the test and trainset, both of the feature-sets from FMNIST are transformed with t-SNE, and each sample is represented by a point colored according to their label in Figure 8. It is observed that the structures obtained in the training set remain similar in the testing set. The cluster where the humans are observed as seen in Figure 6 is also slightly better visible in this illustration.

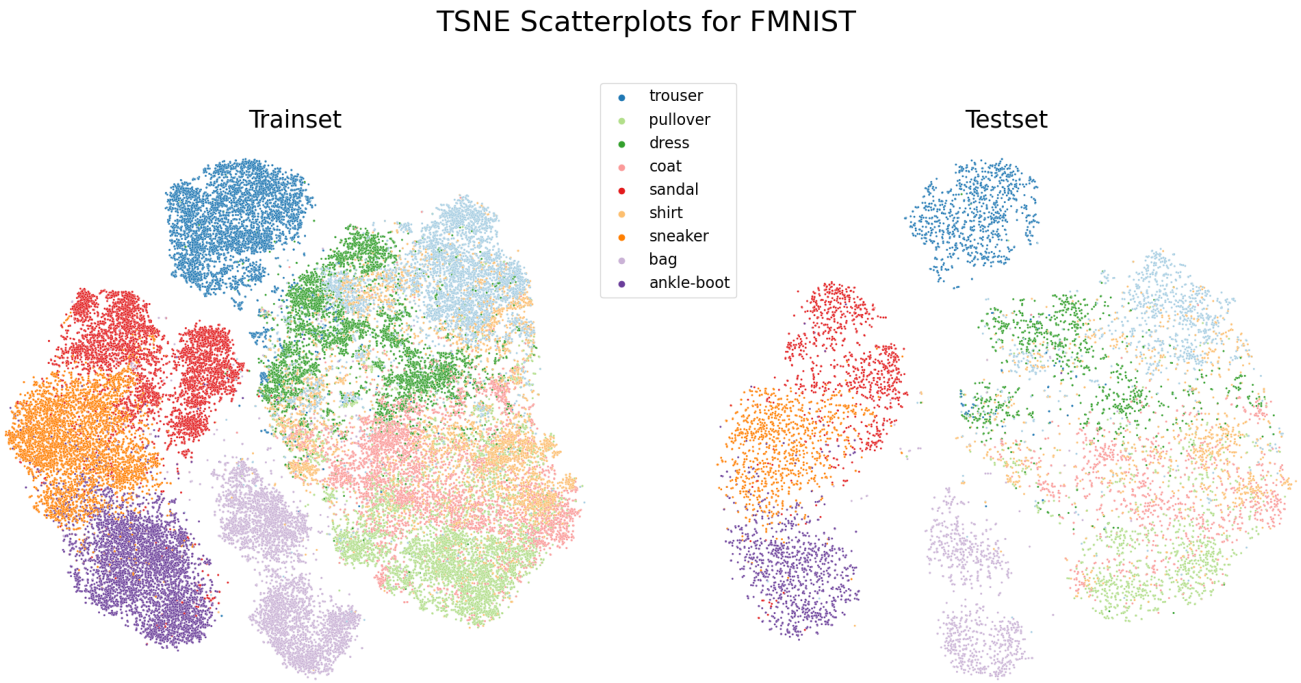

Figure 8: On the left there is the t-SNE plot on the training set and the right side shows the t-SNE on the Testset, the 10 different classes are distinguished by color. The structure on test set seems similar to the one on the training set. (Source: Author's own creation using Python)

2.2.3 One vs. All

The idea of the One versus All (OvA) setting or One versus Rest (OvR) which is used in this work is as follows: Each class is treated as an independent binary classification problem, with an individual classifier for each class against the rest of the classes combined.

Hence for the FMNIST dataset 10 problems for each $Class_i$ are obtained. Each of these classes has different aspects and poses an individual problem that is distinct from the others. In Figure 9 each of the individual problems is visualized with settings explained in the next subchapter 4.3.1.

2.2.4 Sensitivity and contamination

For the unsupervised anomaly detection task, explored in this work the One vs All setting is applied. Consider the FMNIST training set $\mathbb{Z}$ with 10 subsets Z_i for each $Class_i$ $i \in \{1...10\}$. For the Unsupervised anomaly detection, as described in section 1.2, for each of the classes a set U_i is defined which consists of N_i and A_i , where $N_i = \{x \in Z_i\}$ and $A_i = \{x \in Z_k \mid k \neq i\}$ whereas α_0 defines the fraction of U_i which is contaminated.
In other words for each class in the FMNIST dataset, there is a dataset that consists of samples from the class as well as samples from the 9 other classes which would be considered contamination. Therefore clean data, as described in 1.3 would only consider the data from each $Class_i$ for training. However, if the training data is contaminated it will contain samples that do not belong to the $Class_i$ itself. In the work of [Qiu+22a][Qiu+22b] the contamination is obtained by randomly sampling from the other classes. Effects of contamination with other data than just from the rest of the classes are discussed in 5.1. In the sensitivity analysis in section 4.3.1 the effects of the contamination ratio α on the performance is reviewed.

For the FMNIST dataset, the 10 different subsets Z_i are all equal in their cardinality. In Practice, however, datasets often contain imbalanced data. The size-sensitivity analysis 4.3.2 deals with the impact of reducing the size of the training data. For the analysis from each of the classes, only a fraction $\lambda \in \{0..1\}$ of the original training data per class is considered. The data points are randomly sampled from the class.
Figure 9 visualizes a 99 percent reduced dataset with a 1 percent contamination ratio. The illustration shows that with this approach if there is a small contamination rate α and a small λ the training set U_i might contain contaminated samples only from specific classes.

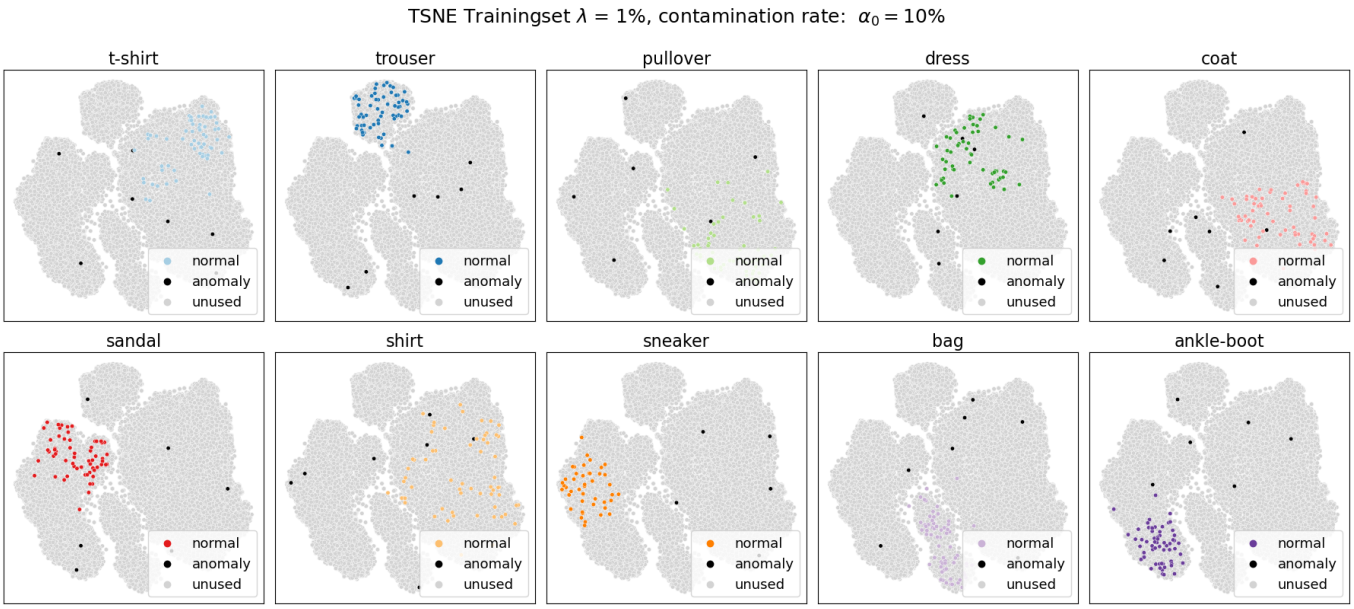

Figure 9: t-SNE visualization for each of the classes in FMNIST dataset, the visualization shows only 1 percent of the original training data with a contamination rate α_0 of 10 percent. The contaminated data points added from the other classes are marked in black. For small λ and small α_0 the training set contains contamination not from all other classes. e.g. shirt has no contamination data points from the trouser (top) cluster. (Source: Author's own creation using Python)

2.3 Dataset Summary

The observations in the previous section 2.2 provide the following insights which form the basis for the experiments.

- The feature extraction with the pre-trained CNN ResNet-152 extracts useful features that can be used for further analysis.
- Contamination of a "clean" datasets seems to be a common occurrence, as shown by the example of FMNIST.
- Some of the FMNIST classes are not easily distinguishable. Potential to test different, more sophisticated methods. Each of the subclasses poses a different binary classification task and has to be handled separately. This should also be observed by adding contamination to the different classes. Each of the subproblems should be affected differently by contamination.

3 Method

In the methodology section, diverse strategies were employed to address the Unsupervised Anomaly Detection (AD) task within this study:
The subsequent methods were selected to set a foundational level for the OvR problem on the FMNIST dataset. Three conventional machine learning approaches, alongside two deep learning approaches, were opted for. Additionally, the Neural Transformation model is presented, and several frameworks designed to handle contaminated data are elucidated.
Importantly, these methods were utilized in their default setups and weren't extensively fine-tuned for the dataset.

3.1 One Class Support Vector Machines

As of 2018 One-class Support vector machines(OCSVM) are one of the most popular techniques in anomaly detection [NV18]. This estimator is best suited for anomaly detection when the training set is not contaminated by outliers. That said, outlier detection in high-dimension, or without any assumptions on the distribution of the inlying data is very challenging, and a One-class SVM might give useful results in these situations depending on the value of its hyperparameters. [sci23a] OCSVM is based on the idea of finding a hyperplane that best separates the normal data from the anomalies. The main concept can be summarized as follows:

Training

OCSVM is trained using only the normal data points (possibly contaminated). The objective of the algorithm is to find the optimal hyperplane that encloses the normal class while maximizing the margin from the origin. In a two-dimensional dataset, the hyperplane is simply a line, while in higher dimensions, it becomes a hyperplane. The hyperplane is positioned such that it captures as many normal data points as possible within its margin.

Margin and Outliers

The margin is the distance between the hyperplane and the closest normal data points. Data points outside the margin are considered outliers or anomalies since they are farther away from the normal class.

Support Vectors

During training, only a subset of the normal data points, known as support vectors, influences the positioning of the hyperplane. These support vectors are the data points that are closest to the margin and directly affect the decision boundary.

Anomaly Detection

To identify anomalies in new, unseen data, One-Class SVM calculates the signed distance of each data point to the hyperplane. Positive distances indicate that the point lies within the boundary (normal data), while negative distances indicate that the point is outside the boundary (anomalies).

One-Class SVM is advantageous for several reasons:
High-dimensional data is handled efficiently. α_0 has not to be known in advance. It's effective when the number of anomalies is significantly smaller than the normal data points.
However, there are some considerations when using OCSVM: The algorithm's performance is sensitive to the choice of the kernel function and its parameters. Choosing an appropriate kernel is essential to achieve good results. One-Class SVM is primarily suited for datasets with a single, well-defined normal class. If the normal class is not well-separated or if there are multiple normal classes, the performance may suffer [NV18].

3.2 Local Outlier Factor

Local Outlier Factor (LOF) is based on the idea that outliers are data points that have significantly different densities compared to their local neighborhood [Jay] [sci23b]. The main concept behind LOF can be summarized as follows:

Local Density

LOF measures the local density around each data point. It calculates the density by determining the distance between a data point and its k-nearest neighbors (where k is a user-defined parameter). A higher density implies that the data point has many nearby neighbors in its local region.

Local Reachability Distance

For each data point, LOF computes the reachability distance from the data point to each of its k-nearest neighbors. The reachability distance is a measure of how far away each neighbor is from the data point. It combines the distance between the data point and its neighbor with the density of the neighbor, providing an estimate of the data point's "reachability" within its local region.

Local Outlier Factor

The LOF score for each data point is calculated by comparing its local reachability distance to that of its k-nearest neighbors. If a data point has a significantly higher reachability distance compared to its neighbors, it indicates that the point is less dense and isolated, making it a potential outlier. The LOF score is defined as the average ratio of the reachability distances between the data point and its k-nearest neighbors.

Anomaly Detection

Data points with higher LOF scores are considered outliers or anomalies since they exhibit significantly different local densities compared to their neighbors. These points are located in regions of the data space that are sparser than the surrounding areas.

LOF is advantageous for several reasons: It captures the local structure of data, allowing it to handle datasets with varying density regions and complex geometric shapes. It doesn't assume a specific distribution of data, making it robust to different data types. It can be effective in detecting outliers in high-dimensional datasets. However, LOF also has some limitations:
The choice of the parameter k can impact the performance of the algorithm. Selecting an appropriate value for k is essential, and it often requires experimentation or domain knowledge.
In high-dimensional datasets, LOF's performance may degrade due to its sensitivity towards it [Smi20].

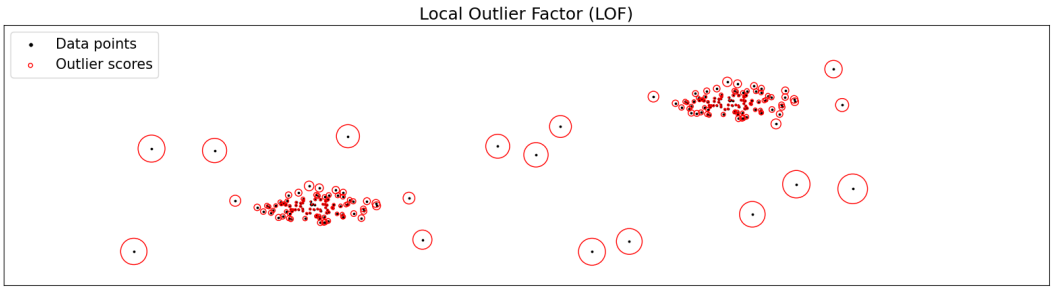

Figure 10: illustration of LOF, The red circles are the scores calculated for each datapoint, Own depiction, inspired by [sci23c]

3.3 Isolation Forest

The main idea behind the Isolation Forest (i-Forest) algorithm is to isolate anomalies by constructing binary trees and exploiting the fact that anomalies are likely to be fewer and more isolated than normal data points. [LTZ12][sci23d]
The algorithm works by randomly selecting a feature and a split value to create a partition between data points. It then continues to randomly select features and split values until all data points are isolated in their own partitions (terminal nodes). The number of splits required to isolate each data point is used as a measure of the data point's anomaly score (Figure 11 path to red points).
Step by step the algorithm works as follows:

Random Selection of Split Feature, Split Value

The algorithm randomly selects a feature from the input data. It then randomly selects a value within the range of the selected feature to create a split. All data points having values below the split value go to the left branch, and all data points with values above the split value go to the right branch.

Recursive Splitting

The process of randomly selecting a feature and split value is repeated for each resulting partition. This recursive splitting continues until each data point is isolated in its own partition (terminal node).

Anomaly Detection

The anomaly score for each data point is calculated based on the average number of splits required to isolate that data point across all the trees in the forest. The intuition behind this is that anomalies will require fewer splits to be isolated, as they are far from the normal data points and more easily separable. The data points with lower average isolation depths are considered to be anomalies, as they require fewer splits to be isolated.

Isolation Forest is advantageous for several reasons [LTZ12] 8.:
It adeptly manages high-dimensional data with efficiency. Pre-knowledge of anomaly count is unnecessary for its operation. Its robustness to noisy data remains unaffected by irrelevant features.

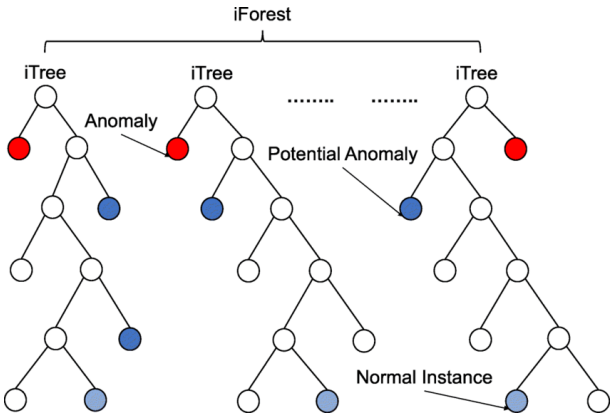

Figure 11: Graphical Representation of an isolation Forest Graph. The path length to the terminal node is used to determine if a sample is an anomaly. Source [RFA21] Fig.1

3.4 Autoencoder

An autoencoder is a type of artificial neural network used for unsupervised learning and dimensionality reduction [Kha21]. It is primarily used for feature learning and data compression, particularly in tasks involving data reconstruction and anomaly detection. The architecture of an autoencoder consists of two main components: the encoder and the decoder, with the goal of reconstructing the input as good as possible.
the mapping can be denoted by :

$$g : \mathcal{X} \to \mathcal{X}$$

explicitly there is a function $g(\cdot)$ which consist of the encoder $e()$ and decoder $d()$ which transforms each featuresample $x \in \mathcal{X}$ into a feature $y \in \mathcal{X}$:

$$y = d(e(x))$$

Encoder

The encoder is the first part of the autoencoder, responsible for compressing the input data into a lower-dimensional representation, also known as the "latent space" or "encoding." It takes the input data and processes it through a series of hidden layers, reducing the dimensions at each layer. The final layer of the encoder produces the compressed representation of the input data, typically with a lower dimensionality than the original data.

Decoder

The decoder is the second part of the autoencoder, responsible for reconstructing the compressed representation back to the original input data. It takes the encoding produced by the encoder and processes it through another set of hidden layers, gradually increasing the dimensions. The final layer of the decoder aims to reconstruct the original input data as closely as possible.

Reconstruction loss

The key objective of an autoencoder is to minimize the reconstruction error, which is the difference between the original input data and the data reconstructed by the decoder. This is usually achieved by using a loss function, such as mean squared error (MSE), to measure the difference between the input and the output.
During the training process, the autoencoder learns to capture the most important features or patterns in the data within the compressed latent space. This process is essentially a form of unsupervised learning, as the autoencoder does not rely on labeled data. Instead, it learns to represent the data distribution in the input space, with the goal of accurately reconstructing the original data points.

Anomaly Detection

Autoencoders can be leveraged for anomaly detection by measuring the reconstruction error. Data points with high reconstruction errors are likely to be outliers or anomalies.
Autoencoders are a powerful tool for unsupervised learning tasks, especially when dealing with high-dimensional and complex data. However, they may suffer from overfitting, and the architecture's choice and hyperparameter tuning are crucial for their successful application to a specific problem.

3.5 Variational-Autoencoder

A Variational Autoencoder (VAE) is a type of autoencoder that extends the traditional autoencoder, Figure 12, by introducing probabilistic components, enabling it to model the underlying probability distribution of the data. The main idea behind VAEs is to map the input data to a latent space where each point represents a potential data point, allowing the model to generate new data samples by sampling from the latent space. Unlike traditional autoencoders, VAEs explicitly model the distribution of the latent space, making them more effective in capturing the underlying structure of complex data distributions [Deh18].
Here's how a Variational Autoencoder works:

Encoder

The encoder part of the VAE maps the input data to the parameters of a probability distribution in the latent space. Instead of directly producing a fixed encoding like a traditional autoencoder, the encoder produces two vectors, representing the mean μ and the standard deviation σ of a multivariate Gaussian distribution in the latent space.

Reparameterization Trick & Latent Space

To ensure the model remains differentiable and enables the use of backpropagation during training, the VAE uses the reparameterization trick. Instead of sampling directly from the Gaussian distribution, it samples from a standard Gaussian distribution (e.g., N(0, 1)), and then scales and shifts the samples using the mean μ and standard deviation σ obtained from the encoder. The samples obtained from the reparameterization step form the latent space of the VAE. This latent space is a continuous, probabilistic space that captures the underlying structure of the data distribution.

Decoder

The decoder part of the VAE takes the samples from the latent space and maps them back to the data space. It reconstructs the original data by applying a series of transformations through hidden layers until it produces the output data. The output of the decoder is the generated data that should resemble the input data.

Anomaly Detection

The VAE is trained to minimize the reconstruction loss, which measures the difference between the generated data and the original data. Similar to the AE these the loss is interpreted as a score to identify anomalies.
By incorporating the latent space as a probability distribution, VAEs enable smooth interpolation between data points, allowing for more meaningful and controllable data generation. [ren]

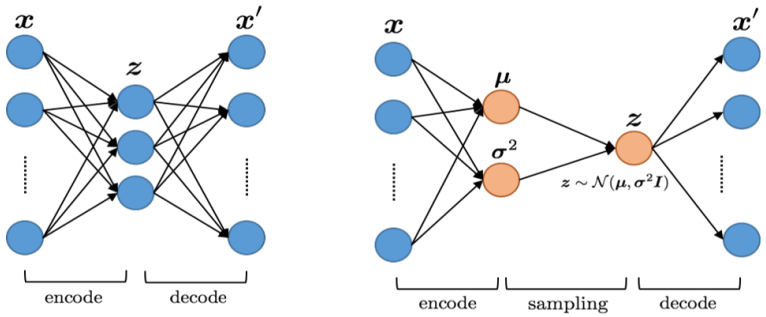

Figure 12: Schematic Autoencoder (Left) and Variational Autoencoder (right), The main difference is that in VAE parameters of a distribution are learned from which the latent variable is sampled. (Source: [ren])

3.6 Neural Transformation Learning

In [Qiu+22b] Neural Transformation Learning (NTL) is introduced, this subchapter aims to describe the model and provide an overview of its functionality.

3.6.1 NTL Model

(NTL) [Qiu+22b] 3.1 comprises two main components: a set of learnable transformations and a learnable encoder. Both components are jointly trained using a deterministic contrastive Loss (DCL)(3). The objective of this training is twofold: firstly, it aims to optimize the parameters for the transformations and the encoder; secondly, the score of the loss function can be utilized during inference to determine an anomaly score (4).
The core concept is to make each transformed sample as similar as possible to the original sample, while simultaneously ensuring maximum dissimilarity from all other transformed samples. This objective is accomplished through the use of the deterministic contrastive loss, DCL, which has demonstrated remarkable success in unsupervised tasks [WL21]. Figure 13 provides a schematic representation of this concept.

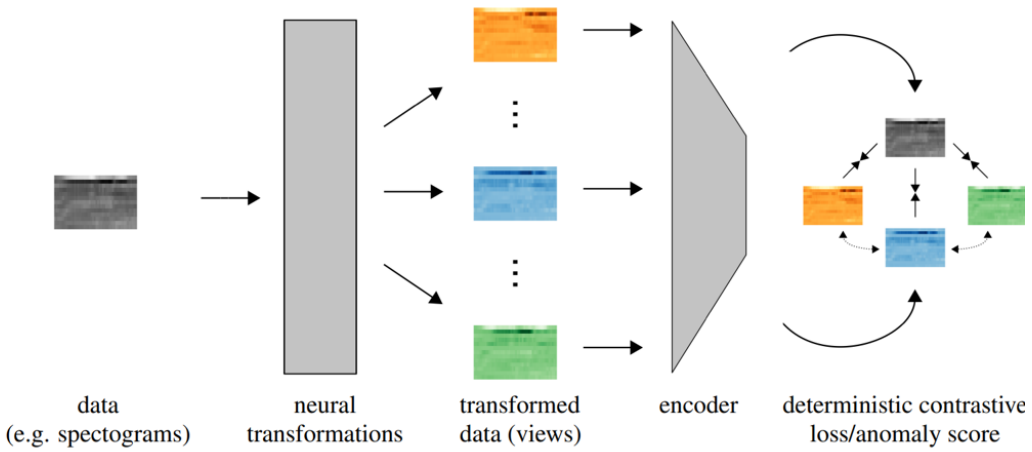

Figure 13: Neural Transformation Learning, for a sample a given set of Transformations are learned and encoded, these transformations should be very similar to the original image but very different from each other transformation. For an anomaly, the transformations should be not similar to the original, but similar to the other transformations. Hence a score can be derived from the DCL which allows to distinguish between anomaly and normal sample, Source: [**qiu2022neural**] Fig.1

To be consistent with the notation of this work the "data" space $\mathcal{Y}$ (In this case already feature vectors from ResNet152 , section: 2) with samples $D = \{x^{(i)} \sim \mathcal{Y}\}_{i=1}^{N}$ is considered. Also K transformations $T := \{T_1, \ldots, T_K \mid T_k : \mathcal{Y} \to \mathcal{Y}\}$ are considered , these transformations are learnable, i.e., they can be modeled by any parameterized function whose parameters are accessible to gradient-based optimization, the parameters of transformation T_k are denoted by θ_k. In this work, the transformations for T_k are fully connected feed-forward Multi-Layer Perceptrons (MLP). The feature encoder $f_\phi(\cdot)$ aswell is a neural network with decreasing neurons per layer.

Mechanism

The DCL (3) encourages each transformed sample $x_k = T_k(x)$ to resemble the original sample x while encouraging it to be dissimilar, from other transformed versions of the same sample, hence a scoring function (1) for 2 transformed samples is defined as follows:

$$h(x_k, x_l) = \exp\left(\frac{\text{sim}(f_\phi(T_k(x)), f_\phi(T_l(x)))}{\tau}\right), \tag{1}$$

where τ denotes a temperature parameter, and the similarity is defined as the cosine similarity

$$\text{sim}(a, a_0) = \frac{a^T a_0}{\|a\| \cdot \|a_0\|} \tag{2}$$

in an embedding space $\mathcal{A}$. The encoder $f_\phi(\cdot) : \mathcal{Y} \to \mathcal{A}$ serves as a features extractor.
The Lossfunction is given by:

$$L = E_{x\sim D}\left[-\sum_{k=1}^{K} \log \frac{h(x_k, x)}{h(x_k, x) + \sum_{l\neq k} h(x_k, x_l)}\right]. \tag{3}$$

The term in the nominator of DCL (3) pulls the embedding of each transformed sample close to the embedding of the original sample. This encourages the transformations to preserve relevant semantic information. The denominator pushes all the embeddings of the transformed samples away from each other, thereby encouraging diverse transformations. The parameters of NeuTraL AD $\theta = [\phi, \theta_{1:K}]$ consist of the parameters ϕ of the encoder, and the parameters $\theta_{1:K}$ of the learnable transformations. All parameters θ are optimized jointly by minimizing the DCL objective function. For inference the Loss (3) can be directly used as anomaly score (4):

$$S(x) = -\sum_{k=1}^{K} \log \frac{h(x_k, x)}{h(x_k, x) + \sum_{l=1}^{K} h(x_k, x_l)} \tag{4}$$

Anomaly Detection

If NTL is trained on normal data, it is assumed that an anomaly does not provide a good reconstruction with the transformation and as well does not provide diverse transformations. Therefore the following example can be derived:

Assume that arbitrarily the temperature parameter τ is set to 0.2, so for a normal sample, the $sim(x, x_k)$ (2) should be close to 1 and $sim(x_l, x_k)$ should be closer to 0, hence for anomaly $sim(x, x_k)$ close to 0 , $sim(x_l, x_k)$ should be closer to 1.
Therefore for an anomaly $h(x_k, x)$ goes to 1 and $h(x_k, x_l)$ to 148 and vice versa for a normal sample.
So for a normal sample the term in the $log()$ in (4) is close to $1 \sim \frac{148}{148+k*1}$ and for an anomaly close to $0 \sim \frac{1}{1+k*148}$.
Thus, a small value of $S(x)$ corresponds to a normal sample, while a high value of $S(x)$ indicates an anomalous sample, making it suitable for anomaly scoring.

3.7 Frameworks

The NTL model does not include an assumption parameter about the contamination, the aim is to construct an anomaly detection classifier using (deep) learning methodologies, designed for contaminated data. The central challenge revolves around simultaneously deducing the binary labels yi of whether a sample is anomalous or not, during training, all while optimizing the utilization of this information to effectively train an anomaly detection model. In [Qiu+22a] therefore different frameworks were introduced to deal with this problem.
The assumption here is that there are two loss functions L_n^θ for normal data and L_a^θ for anomalous data. A new joint loss function is introduced which considers L_n^θ and L_a^θ:

$$L(\theta, y) = \sum_{i=1}^{N} (1 - y_i) L_\theta^n(x_i) + y_i L_\theta^a(x_i). \tag{5}$$

Since this work only uses NTL with the described frameworks, L_n^θ and L_a^θ are provided, where $L_n^\theta = (3)$ with the following:

$$L_n^\theta(x) := -\sum_{k=1}^{K} \log \frac{h(x_k, x)}{h(x_k, x) + \sum_{l=1}^{K} h(x_k, x_l)}, L_a^\theta(x) := -\sum_{k=1}^{K} \log(1 - \frac{h(x_k, x)}{h(x_k, x) + \sum_{l=1}^{K} h(x_k, x_l)}) \tag{6}$$

To infer an anomaly score while training the following expression is used:

$$S_{\text{train}_i} = L_\theta^n(x_i) - L_\theta^a(x_i). \tag{7}$$

For a normal sample $L_\theta^n(x_i)$ is small and $L_\theta^a(x_i)$ is high, therefore S_{train_i} will be negative for normal samples and positive for anomalous samples.
With assumed contamination ratio α a cutoff at the α quantile is determined and the labels y_i can be inferred, Every of the following frameworks in 3.7 infers the labels differently. Further equation (5) can now be minimized. In Practice the (5) is subject to Gradient Descent and the anomaly scores are calculated per minibatch. Figure 14 depicts the pseudocode for a batch-wise GD of the frameworks.

Algorithm 1 Training process of Frameworks

```
Input: Contaminated training set D (α_0 anomaly rate), hyperparameter α
Model: NTL with θ
for each Epoch do
    for each Mini-batch M do
        Calculate the anomaly score S_{train_i} for x_i ∈ M
        Estimate the label y_i given S_{train_i} and α
        Update the parameters θ by minimizing L(θ, y)
    end for
end for
```

Figure 14: Pseudocode for the LOE / Refinement frameworks

The fundamental concept of LOE [Qiu+22a] involves leveraging parameter sharing. Through this approach, the labeled anomalies provide instructive information to the model, indicating areas in feature space where normal data is not anticipated. This concept forms the foundation of Outlier Exposure, which generates synthetic labeled anomalies to improve the effectiveness of anomaly detection. [Qiu+22a] introduce two frameworks for Outlier exposure: LOE-hard, LOE-Soft as well as a simpler refinement strategy and blind training which all handle the labeling of y_i during the training process differently.

3.7.1 Blind

Blind training implies that no assumptions have been made about the contamination ratio (α) . As such, every label in the training set is presumed to be 0. Note that Y consists of 1 vector element where each entry is the label y_i for the data x_i.

$$Y = \{0\}^N \tag{8}$$

Therefore the training of the model only consideres $L_n^\theta(x)$.

3.7.2 LOE-hard

Loe hard consideres for each sample either the $L_\theta^n(x_i)$ or $L_\theta^a(x_i)$ based the quantile α and the labels assigned by:

$$Y = \{y \in \{0,1\}^N : \sum_{i=1}^{N} y_i = \alpha N\} \tag{9}$$

3.7.3 LOE-soft

Based on :

$$Y = \{y \in \{0,0.5\}^N : \sum_{i=1}^{N} y_i = 0.5\alpha N\} \tag{10}$$

LOE soft considers at each iteration either only $L_\theta^n(x_i)$ or $0.5 * L_\theta^n(x_i)$ and $0.5 * L_\theta^a(x_i)$

3.7.4 Refine

The Refine strategy has a small alteration in the loss since, per batch only the predicted normal samples are subjected to the loss the inference of y_i is the same as in LOE hard (9) however the loss changes to:

$$L(\theta, y) = \sum_{i=1}^{N} (1 - y_i) L_\theta^n(x_i)). \tag{11}$$

4 Experiment

The arrangement of the experiment section is as follows: initially, the performance of the baseline models is assessed to provide an understanding of the capabilities of state-of-the-art methods in addressing the task. Subsequently, a comparison is drawn between this performance and that of the NTL model. The NTL model is then subjected to testing using various frameworks. The resulting outcomes are then compared to the outcomes documented in the paper. Moreover, an examination is conducted to explore the impact of assumed contamination and reduced training data on the sensitivity of these approaches.

4.1 Metrics, Precision Recall

In the works of [Qiu+22b] and [Qiu+22a] AREA under the ROC (Receiver Operator Curve) is reported, however for imbalanced datasets AUROC is not always suitable [Tra21]. For the reason to compare the results, the same metrics are used as well as the more suitable metrics of Precision-Recall are calculated to compare the baseline models to the approach with NTL, etc. Also, it seems to be important to analyze each class individually since each class poses its own classification task, the paper however only reports an average over all the classes which provides an overall metric.
Since the task is a One versus All approach, a binary classification result can be visualized per class with Precision vs Recall curves. This representation allows for the comparison of imbalanced datasets. The interpretation of such a curve is as follows: for each sample of the test set there is an anomaly score obtained, and the classification of whether the sample is an anomaly or normal data is solely based on the score. When the threshold for classifying is altered, for each such threshold precision and recall can be obtained. In the imbalanced AD setting with 10 percent normal data and 90 percent anomalies, the minimum precision obtained at the maximum threshold has to be 90 percent ,since if all samples in the dataset are classified as anomaly, it would be correct in 90 percent of the cases. For recall there must be a limit of 1 since when all samples are classified as anomalies all the samples are True Positives hence the recall = 1.0. The Precision-Recall curves illustrate the tradeoff between recall and precision and allow the provision of an operation threshold to calibrate a system for a given precision as a function of recall. A perfect system with only TP and TN at every threshold would yield always Precision = 1 for every Recall. Hence if the AREA under the Precison-Recall Curve (AUPR) is calculated, a perfect system would yield 1. Average Precision (AP) is also a measurement that can be used to measure how well the model performed, it's almost equivalent to AUPR. In this work, the AUPR is approximated by the trapezoid rule. The bands around the mean line are the maximum respective minimum values which are obtained by 5 computation runs.

4.2 Baseline models SOTA

The models introduced in section 3 do not all perform very well with the settings that have been chosen. Therefore only the results from the Autoencoder, i-Forest and NTL are described thoroughly. The results on LOF and Variational Autoencoder can be found in Appendix 7.

4.2.1 Autoencoder

Figure 15 illustrates a PR curve for each of the classes with different contamination ratios for the training set with the Autoencoder model. Overall it can be observed that training with contaminated data decreases performance. It becomes apparent that there are differences between the classes. In the context of the "shoes" category, a relatively consistent performance trend emerges in the absence of data contamination. However, when contamination is introduced, the performance of sandals exhibits a slightly less favorable outcome compared to sneakers and ankle boots. This observation emphasizes the impact of contamination on the nuanced distinctions among different types of footwear. Moving on to the "upper-body-clothing" group, the anticipated decrease in performance is evident, as discussed in Chapter 2.3. The complexity of differentiating between various classes within this group contributes to the expected decline. Intriguingly, amidst the contamination, pullovers and coats exhibit a somewhat improved performance compared to the other categories such as t-shirts, shirts, and dresses. This nuanced behavior highlights the sensitivity of different clothing subtypes to contamination and how certain items remain relatively robust. The "bag" class stands out with its performance in the absence of contamination, underpinned by the distinctiveness of this cluster, as elaborated in 2.2.1. However, a noticeable drop in performance occurs when contamination infiltrates the training set. Interestingly, a similar drop isn't as pronounced in the "trousers" class. This disparity in response to contamination is interesting. A plausible hypothesis could be that the feature extraction for trousers differs significantly from those of other items, leading to a certain level of immunity against contamination-induced performance deterioration. In contrast, features relevant to handbags seem more intertwined with those of other classes, rendering them more susceptible to performance degradation when contaminated with disparate features. It's crucial to note that the precise composition of these distinguishing features remains unexplored, leaving an avenue for further investigation. Analyzing the specific attributes contributing to these observed behaviors could potentially yield valuable insights for understanding the robustness of various classes under contamination scenarios.

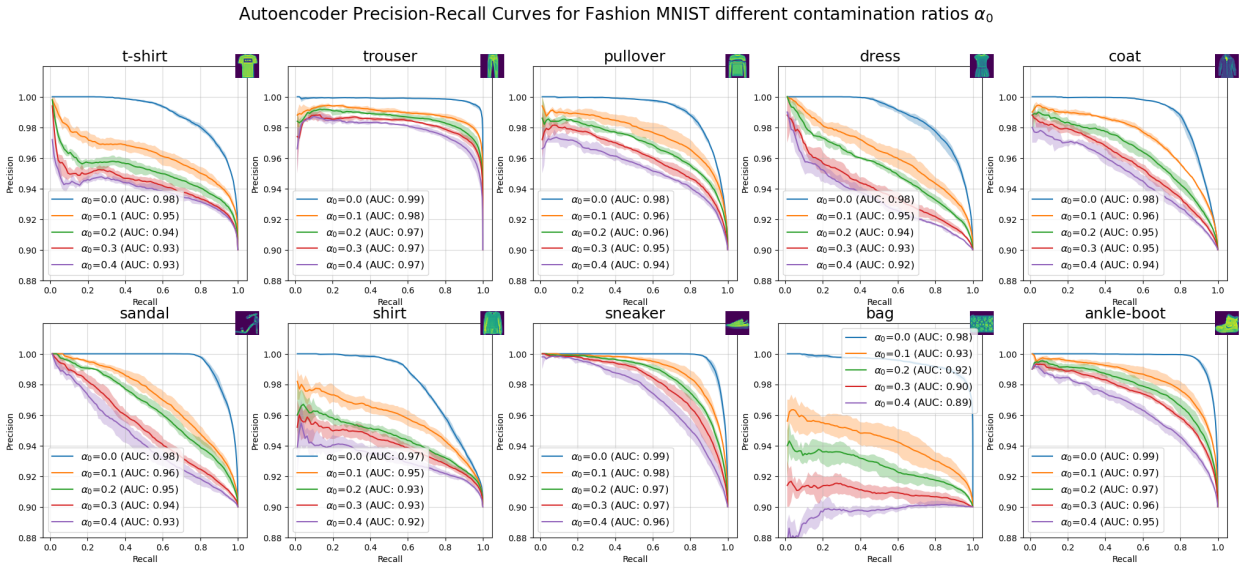

Figure 15: Precision-Recall curves per class for Autoencoder different contamination ratio in the training set. Each class has a different performance, bag experiences a significant performance decrease when contamination is added. (Source: Author's own creation using Python)

4.2.2 Isolation Forest

The Isolation Forest algorithm yields intriguing outcomes when dealing with contaminated data. The presence of contamination in the dataset does impact its performance, although not as significantly as observed in the case of methods like autoencoders or the Neural Transformation Learning (NTL) 4.3 approach. While the algorithm's performance experiences a slight deterioration compared to the Autoencoder across most classes, it's noteworthy that even with a contamination rate of 10 percent, the Isolation Forest's results are either better than, or at the very least, fall within the confidence bounds of the experiment conducted on clean data.
Remarkably, this behavior remains consistent across several classes, including sneakers, ankle boots, coats, trousers, and pullovers. These categories display minimal performance degradation even in the presence of contamination. This consistent and invariant behavior has the potential to be of significant value for subsequent evaluations, especially when the only assumption made is the presence of normal and anomalous data.
By relying solely on the fundamental understanding that there exist both normal and anomalous data points, this invariant response of Isolation Forest offers a valuable advantage in scenarios where assumptions about the specific data distribution or contamination levels might not hold.

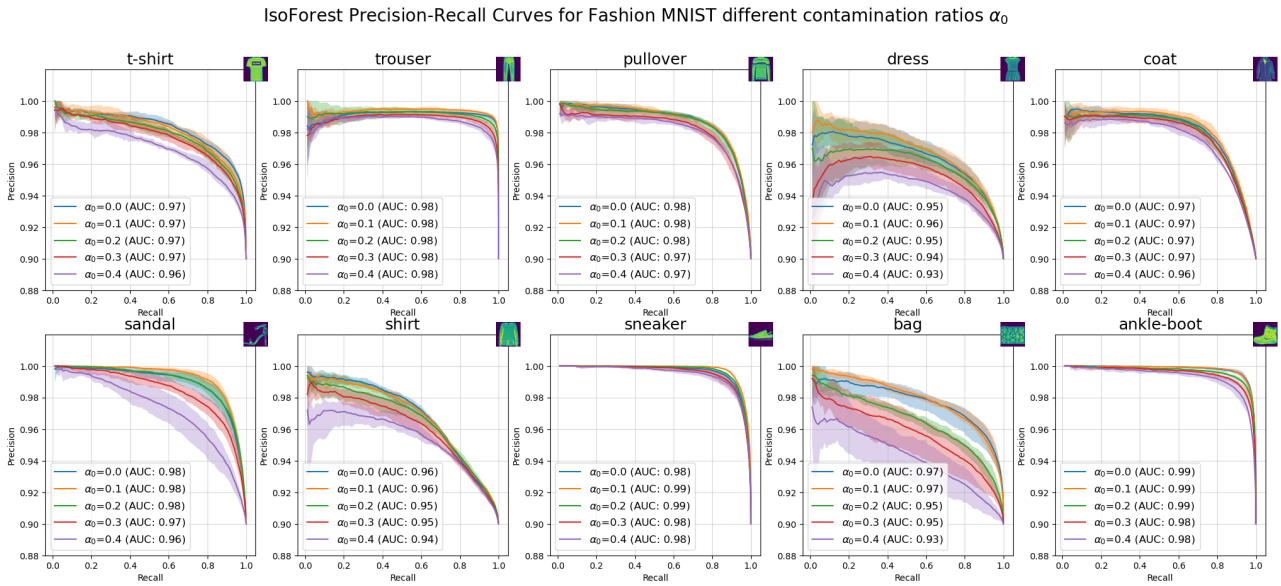

Figure 16: Precision-Recall curves per class, Isolation Forest, Contamination does not affect the performance gravely. (Source: Author's own creation using Python)

4.2.3 OCSVM, VAE, and LOF

The results of the OCSVM, VAE, and LOF can be found in the Appendix 7
The OneClass Support Vector Machine exhibits consistent performance across various contamination levels. With a 10 percent contamination rate, the model's performance remains largely comparable to situations without any contamination visualized in Figure 25.
The Variational Autoencoders' performance without contamination aligns with the Autoencoders and NTL approaches. However, the introduction of contamination has a profound impact, causing a complete disruption in their performance, to be observed in Figure 16.
On the other hand, the Local Outlier Factor algorithm proves to be unsuitable for the task at hand. Its performance, even without contamination, falls behind that of Autoencoders. Furthermore, when contamination is considered, the algorithm's performance is significantly affected, demonstrating a susceptibility that surpasses that of Autoencoders or NTL in Figure 23.

4.2.4 Comparison to NTL blind

The results from the Autoencoder are compared to the NTL with no assumption α about the contamination rate α_0, blind training, in Figure 17. Further, the results from the Autoencoder with contamination rates [0.1 0.2 0.3 0.4] are the same as in Figure 15. It is observed that NTL performs better on each task, The contamination does not affect the performance as strongly as in the case of the Autoencoder. This might be also due to the fact that the hyperparameters of NTL are optimized for this specific task. It shows at least that NTL seems to be a method that brings better or at least comparable results to SOTA. Which motivates further experiments with the frameworks from [Qiu+22a].

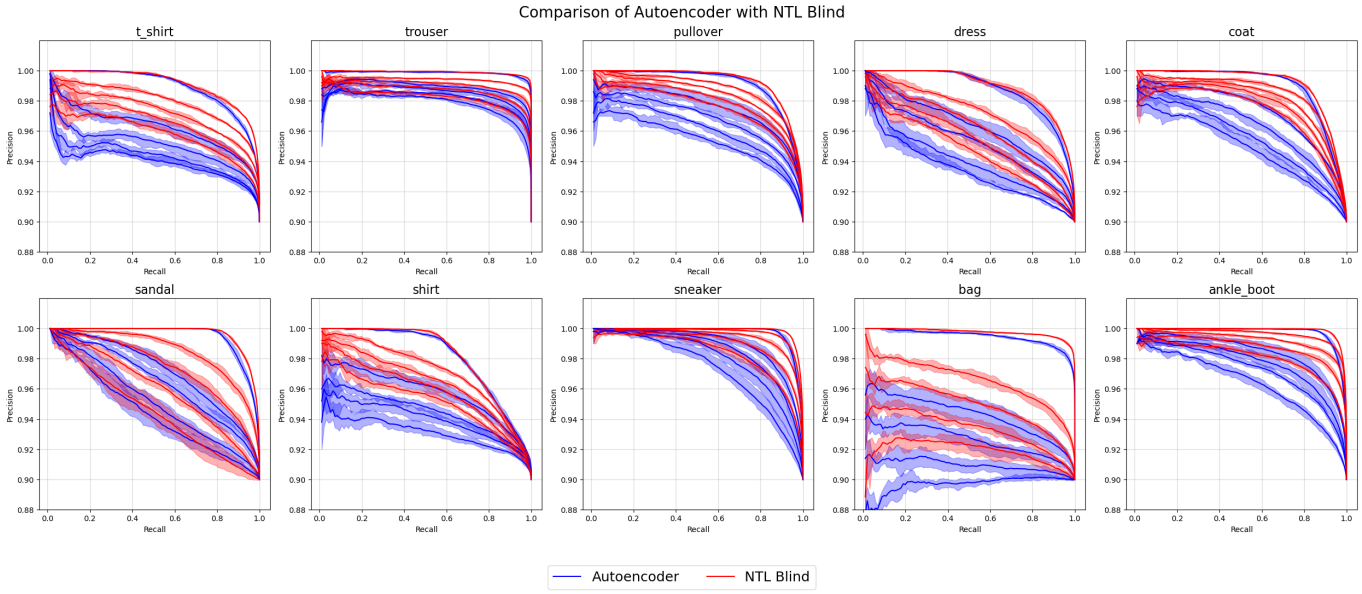

Figure 17: Precision-Recall curves per class Comparison between Autoencoder and NTL blind training, AE results are the same as in Figure contamination rates are decreasing [0.1 0.2 0.3 0.4], NTL shows outperformance on every task. (Source: Author's own creation using Python)

4.3 NTL model in different frameworks

The results from the frameworks described in 3.7 are reported in the paper of [Qiu+22a] Table 1, they report their performance with AUC with $\alpha_0 = 0.1$. Since in the implementation on GitHub [bos22] also Average Precision is reported as well as AUROC, it is assumed that the reported AUC is in fact AUROC.
Results are also reported for different assumed contamination ratios α on FMNIST dataset in [Qiu+22a] Figure 2, however, these experiments are conducted with $\alpha_0 = \alpha$. In Figure 18 the author tried to reproduce the results reported in [Qiu+22a] Figure 2, but was only partly successful. LOE hard was worse than the results from the paper. Strangely the results from the refine and blind approach were even better than reported in the paper. It is not clear where these disparities originate from. Experiments with either AP or different α and α_0 to exclude reportings of another metric, were also unsuccessful in reproducing the results.

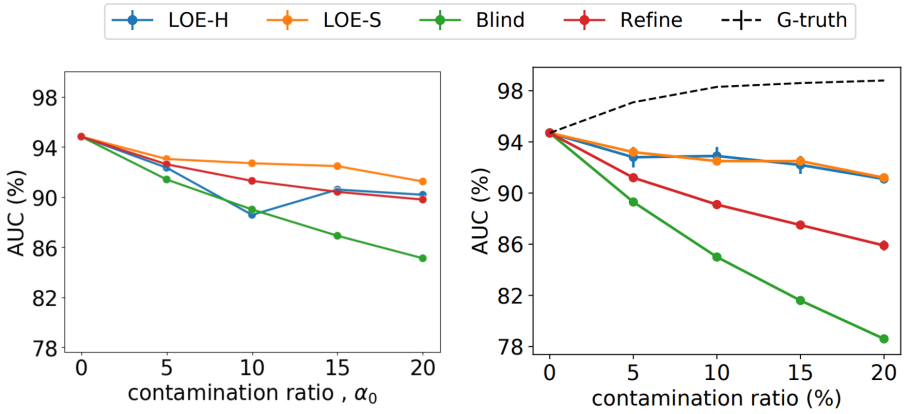

Figure 18: Results compared to the results of the paper, the real contamination ratio was also used in training. Own results are not consistent with results from the paper, (Source: Left: Author's own creation using Python) Right: [Qiu+22a] Figure 2.

4.3.1 Sensitivity of assumed contamination α

The LOE frameworks LOE hard and soft, as well as the Refine framework, inherently involve the assumption of the contamination ratio α. This allows the creation of the sensitivity analysis for each real ratio α_0 and each assumed contamination ratio α. In the work presented by [Qiu+22a], a prime example of this sensitivity analysis is showcased in Figure 3 of their research. Although this analysis is dissected in the context of the LOE framework, it centers exclusively on the CIFAR-10 dataset [Kri09]. Given the distinct focus of the present study on the FMNIST dataset, the attention naturally shifts to exploring the sensitivity of the assumed contamination ratio within the FMNIST domain.
Contrary to the paper's exclusive focus on reporting the mean values across all classes, this study presenting not only the mean outcomes but also delves into the extremities of each class's performance. This entails a comprehensive portrayal of the worst-case scenario (Minimum), the central tendency (Average), and the best value (Maximum) for every individual class within each experimental iteration. This approach inherently seeks to provide a more holistic evaluation of the results. Furthermore, the scope of sensitivity extends to encompass the parameters α and α_0, by 0, 30 and 40 percent. This calculated expansion ensures a comprehensive understanding of how variations in these parameters interact with the contamination ratio.

LOE Hard

Figure 19 shows the sensitivity analysis for the LOE Hard framework. On the left side, there is the maximum AUC per class, in the middle the average and on the right the minimum. The color scale is deliberately kept from AUC 0.6 to 1.0, the more yellowish the higher AUC, consistently for all other sensitivity analysis plots for direct comparison.
When inspecting the sensitivity performance for the average, it is observed the framework seems to perform best when $\alpha \sim \alpha_0$. For some of the combinations e.g. α=α_0=.15 a small increase of 0.05 in α does not affect the performance, however, if α is increased further the performance drops. This behavior can be observed as a more general pattern in the average performance. If the Maximum is inspected a similar behavior can be seen. With minimal performance, the pattern is only partly visible. One very interesting observation is that for the real contamination of 0, the performance is affected stronger in comparison to the performance experienced when the data is contaminated

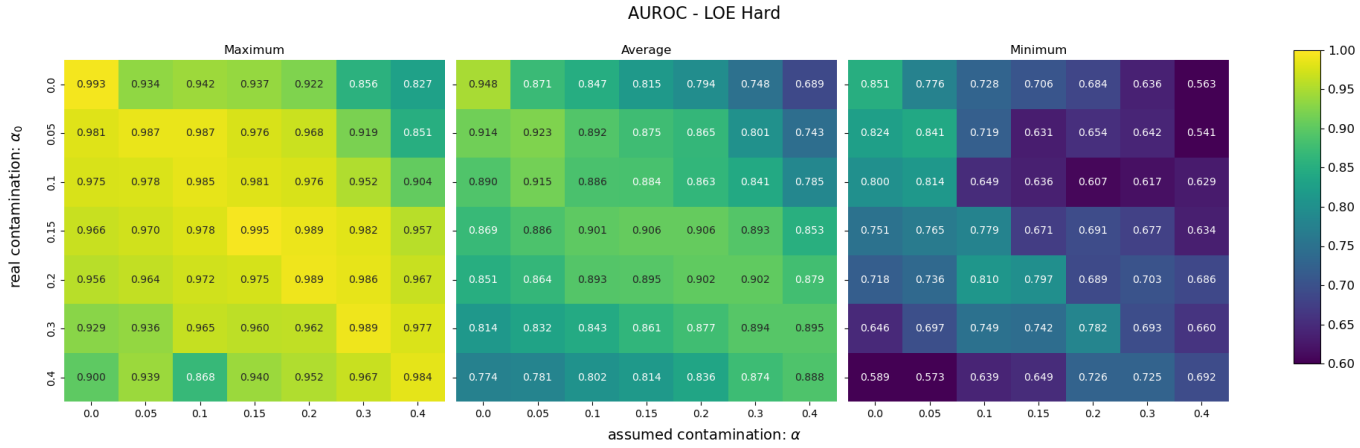

Figure 19: Sensitivity Analysis of the Loe Hard framework with each the Maximum value from the 10 classes, the average over the 10 classes, and the minimum value of the 10 classes over 5 runs. It seems that the performance on average is best when the assumed contamination ratio equals the real contamination ratio (Source: Author's own creation using Python)

LOE Soft

When the sensitivity analysis is inspected for the LOE soft framework Figure 20, the performance seems a bit better than LOE hard but only slightly. Moreover, the performance seems to be best when $\alpha = \alpha_0$, even stronger than LOE soft. Also an increase in α does increase the performance negatively for most of the cases. If the first row with α_0 is inspected, it is observed that the performance does not decrease as quickly as it does with LOE hard with an increase in α.

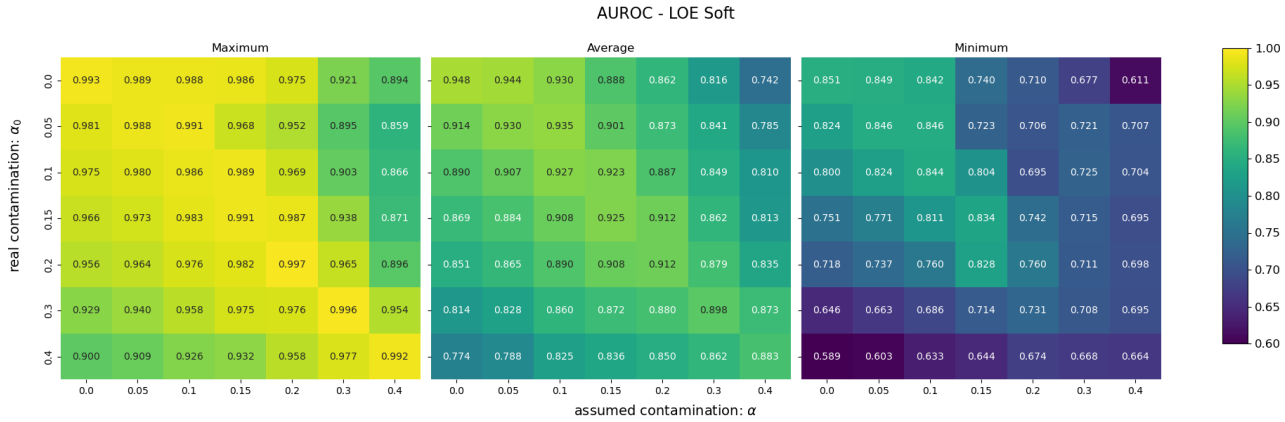

Figure 20: Sensitivity Analysis of the Loe Soft Framework Method with each the Maximum value from the 10 classes, the average over the 10 classes, and the minimum value of the 10 classes over 5 runs. On average the performance seems to be better than Loe Soft, and seems to work best when the contamination ratio is close or equal to the assumed contamination ratio (Source: Author's own creation using Python)

Refine

In the Refine framework in Figure 21 the interesting behavior of $\alpha_0 \leq \alpha$ yielding a better, or at least not worse performance can be observed. This behavior can also be observed for the Maximum and Minimum and not only for the Average.

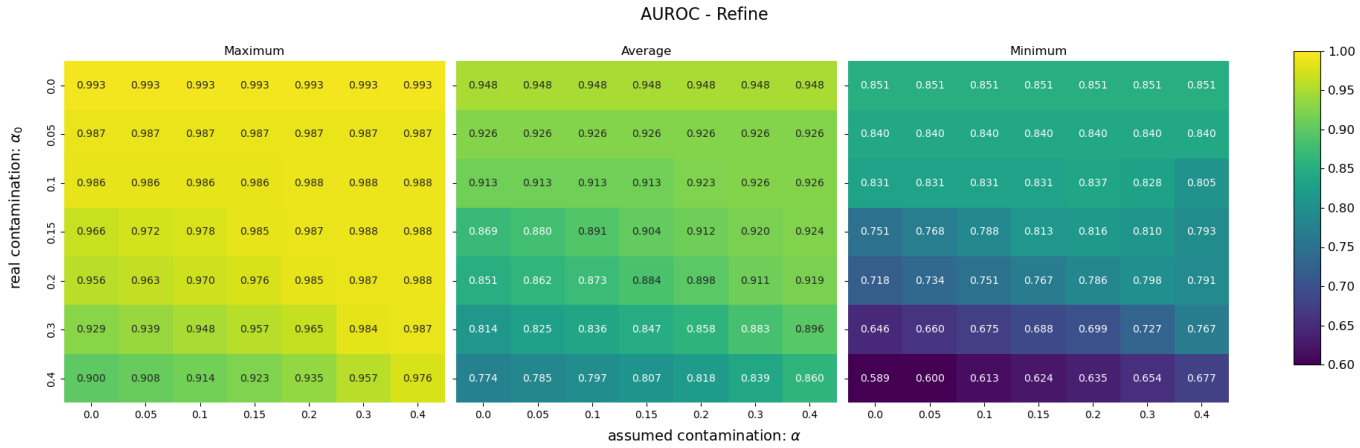

Figure 21: Sensitivity Analysis of the Refine Framework Method with each the Maximum value from the 10 classes, the average over the 10 classes, and the minimum value of the 10 classes over 5 runs. In comparison to LOE -Soft and -Hard, the refine method seems to be performing best in the sensitivity analysis, it can be observed it performs well when the assumed alpha is larger than the real alpha (Source: Author's own creation using Python)

Comparison to LOE

In [Qiu+22a]Figure 3. s. they state the following:

> "LOEH and LOES yield robust results and outperform "Refine" in the most cases."

This observation does not apply to the results observed in this work. The refine framework performance is at least as good as LOE, mostly even better, there are only a few combinations of α and α_0 which provide worse results than refine. One has to keep in mind that this is the average of only 5 computational runs and these numbers could also lie in the statistical fluctuation. Furthermore, the refine framework seems to be more robust regarding the assumption about contamination. Robust methods are very important for the unsupervised anomaly detection task since α_0 is not observed. Having a framework that delivers reliable results even with higher assumptions of contamination is a real benefit in creating a real-world system.

4.3.2 Sensitivity of reduced training set α

The sensitivity analysis of reducing training data holds an important role in thoroughly evaluating the performance of a system. This approach provides valuable insights into how well a system can adapt and perform when faced with limitations in available training data. In real-world scenarios, it's not uncommon for systems to encounter situations where the training data is scarce or reduced. This could happen due to various reasons such as budget constraints, data collection challenges, or privacy concerns. By subjecting a system to sensitivity analysis involving reduced training data, a realistic understanding of how it would perform under such constraints can be gained. A system that maintains acceptable performance levels with limited training data can be considered more resource-efficient and scalable. This is particularly important in scenarios where computational resources are limited.

In essence, sensitivity analysis of reducing training data provides a more complete picture of a system's capabilities and limitations. Therefore, the promising framework of refine is subjected to a reduction in the training set. In Figure 22 the results are presented with 100, 10, and 1 percent of the available training data with the subsampling method discussed in 4.3.1 For the refine Method with the average over all classes and 5 runs. If only 10 percent of the data is used, the overall performance gets a bit worse for almost all of the classes. The observation that refine still does not affect performance significantly if $\alpha_0 \leq \alpha$ can only be observed partly. If the training data is further reduced the observation does not hold anymore. The performance drops as expected and there is no clear pattern visible anymore. Since for the 1 percent training set with 0 contamination the performance does not drop that gravely, it can be softly concluded that each class of the dataset must consist of very representative features which can be learned efficiently.

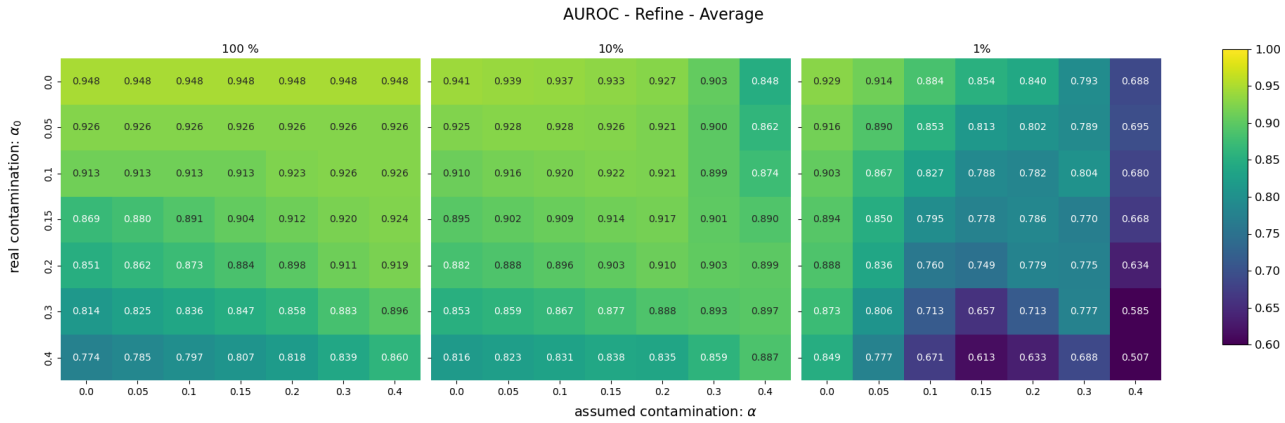

Figure 22: Sensitivity Analysis of the Refine Method for 3 different datasets, Full dataset 100 percent of training data and 10 and 1 percent, evaluated on the same test set. It is observed that for the full 10 percent, the refine method has the property that it works almost equally or sometimes better if a higher alpha is assumed as the real alpha. that does not seem to be the case if the model is only trained on 1 percent of the data (Source: Author's own creation using Python)

5 Conclusion

The Fashion MNIST dataset, presents a unique set of challenges. Each one-class classification problem in this dataset has its own unique subtleties. During dimensionality reduction, an intriguing discovery was made: even in a dataset as widely recognized and used as FMNIST, impurities are still evident. This finding underscores the importance of rigorous data checking, regardless of the dataset's reputation.
In terms of performance, neural transform learning (NTL) has shown notable effectiveness, eclipsing other methods such as autoencoders, SVM, and isolation forests. This showing of superiority highlights the potential of NTL as an anomaly detection tool and its relevance to current research.
Although the results of the Bosch study provided a basis for the experiments in this work, they could only be partially replicated. The reasons for this discrepancy remain unclear and require further investigation.
Compared to the simpler refinement strategy, the LOE soft and LOE hard frameworks appear to underperform. In particular, the refinement strategy approach of using only an estimated fraction 1-α per training step, not only simplifies the process but also shows greater robustness particularly evident when faced with the increase of α.
The implication is twofold: Simplicity can sometimes trump complexity, and adaptability to varying levels of contamination is critical to the robustness and effectiveness of a method.

5.1 Outlook

Since the NTL model and further refinement framework can not only be used for image data it's interesting to see how the finding of this work, especially the behavior of the refinement would also apply to different data domains such as tabular data, video, and time-series. Further, the comparison to an even more naive refinement strategy where a fraction assumed as contaminated training data is omitted could be subject to new experiments.
NTL was only used in this work with fully connected MLPs, further work could involve e.g. autoencoders as transformations or for the encoder a different encoding architecture.
The effect of specific types of contamination on performance is not addressed in this study. In practical applications, it's essential to consider the types of contamination that are likely to occur and determine their potential impact on performance.

6 References

[vH08] L.J.P. van der Maaten and G.E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". English. In: *Journal of Machine Learning Research* 9.nov (2008). Pagination: 27, pp. 2579–2605. ISSN: 1532-4435.

[Den+09] Jia Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[Kri09] Alex Krizhevsky. *CIFAR-10 - Canadian Institute For Advanced Research (CIFAR)*. https://www.cs.toronto.edu/~kriz/cifar.html. 2009.

[LTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-Based Anomaly Detection". In: *ACM Trans. Knowl. Discov. Data* 6 (2012), 3:1–3:39. URL: https://api.semanticscholar.org/CorpusID:207193045.

[KKN14] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. "A survey of feature selection and feature extraction techniques in machine learning". In: *2014 Science and Information Conference*. 2014, pp. 372–378. DOI: 10.1109/SAI.2014.6918213.

[Ras14] sebastian Raschka. *About Feature Scaling and Normalization*. July 2014. URL: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html.

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[DV16] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. DOI: 10.48550/ARXIV.1603.07285. URL: https://arxiv.org/abs/1603.07285.

[KM17] Eamonn Keogh and Abdullah Mueen. "Curse of Dimensionality". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 314–315. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_192. URL: https://doi.org/10.1007/978-1-4899-7687-1_192.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[Deh18] Thomas Dehaene. *Variational AutoEncoders for new fruits with Keras and Pytorch*. Nov. 2018. URL: https://becominghuman.ai/variational-autoencoders-for-new-fruits-with-keras-and-pytorch-6d0cfc4eeabd.

[NV18] Minh-Nghia Nguyen and Ngo Anh Vien. *Scalable and Interpretable One-class SVMs with Deep Learning and Random Fourier features*. 2018. arXiv: 1804.04888 [cs.LG].

[You18] Cliff Young. *Why Machine Learning Needs Benchmarks*. May 2018. URL: https://www.sigarch.org/why-machine-learning-needs-benchmarks/.

[KD19] Vijay Kotu and Bala Deshpande. "Chapter 13 - Anomaly Detection". In: *Data Science (Second Edition)*. Ed. by Vijay Kotu and Bala Deshpande. Second Edition. Morgan Kaufmann, 2019, pp. 447–465. ISBN: 978-0-12-814761-0. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00013-7. URL: https://www.sciencedirect.com/science/article/pii/B9780128147610000137.

[MM19] Nicolas M. Muller and Karla Markert. "Identifying Mislabeled Instances in Classification Datasets". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2019. DOI: 10.1109/ijcnn.2019.8851920. URL: https://doi.org/10.1109%2Fijcnn.2019.8851920.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Hoe20] Kuenzer C. Hoeser T. *Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends*. 2020. DOI: https://doi.org/10.3390/rs12101667. URL: https://www.researchgate.net/publication/341576780_Object_Detection_and_Image_Segmentation_with_Deep_Learning_on_Earth_Observation_Data_A_Review-Part_I_Evolution_and_Recent_Trends.

[Smi20] Abir Smiti. "A critical overview of outlier detection methods". In: *Computer Science Review* 38 (2020), p. 100306. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100306. URL: https://www.sciencedirect.com/science/article/pii/S1574013720304068.

[ASS21] Farzana Anowar, Samira Sadaoui, and Bassant Selim. "Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE)". In: *Computer Science Review* 40 (2021), p. 100378. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2021.100378. URL: https://www.sciencedirect.com/science/article/pii/S1574013721000186.

[Bro21] Jason Brownlee. *Visualizing the vanishing gradient problem*. Nov. 2021. URL: https://machinelearningmastery.com/visualizing-the-vanishing-gradient-problem/.

[Kha21] Renu Khandelwal. *Anomaly Detection using Autoencoders*. Jan. 2021. URL: https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea.

[RFA21] Yousra Regaya, Fodil Fadli, and Abbes Amira. "Point-Denoise: Unsupervised outlier detection for 3D point clouds enhancement". In: *Multimedia Tools and Applications* 80 (July 2021), pp. 1–17. DOI: 10.1007/s11042-021-10924-x.

[Tra21] Tam D Tran-The. *Precision-Recall Curve is More Informative than ROC in Imbalanced Data: Napkin Math More*. Sept. 2021. URL: https://towardsdatascience.com/precision-recall-curve-is-more-informative-than-roc-in-imbalanced-data-4c95250242f6.

[WL21] Feng Wang and Huaping Liu. *Understanding the Behaviour of Contrastive Loss*. 2021. arXiv: 2012.09740 [cs.LG].

[bos22] bosch. *LOE*. https://github.com/boschresearch/LatentOE-AD. 2022.

[Qiu+22a] Chen Qiu et al. *Latent Outlier Exposure for Anomaly Detection with Contaminated Data*. 2022. arXiv: 2202.08088 [cs.LG].

[Qiu+22b] Chen Qiu et al. *Neural Transformation Learning for Deep Anomaly Detection Beyond Images*. 2022. arXiv: 2103.16440 [cs.LG].

[Yoo+22] Jinsung Yoon et al. *Self-supervise, Refine, Repeat: Improving Unsupervised Anomaly Detection*. 2022. arXiv: 2106.06115 [cs.LG].

[Bha23] Aniruddha Bhandari. *Feature Engineering: Scaling, Normalization, and Standardization*. July 2023. URL: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[Goo23] Google. Aug. 2023. URL: https://scholar.google.com/scholar?start=0&q=fmnist&hl=de&as_sdt=0,5.

[Nvi23] Nvidia. *Stable Diffusion*. Aug. 2023. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/playground/models/sdxl.

[sci23a] scikit-learn. Aug. 2023. URL: https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_anomaly_comparison.html.

[sci23b] scikit-learn. Aug. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html.

[sci23c] scikit-learn. Aug. 2023. URL: https://scikit-learn.org/0.20/auto_examples/neighbors/plot_lof_outlier_detection.html.

[sci23d] scikit-learn. Aug. 2023. URL: https://scikit-learn.org/stable/modules/outlier_detection.html#isolation-forest.

[Jay] Vaibhav Jayaswal. *Local Outlier Factor (LOF) — Algorithm for outlier identification*. URL: https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843.

[ren] renom. *Variational AutoEncoders for new fruits with Keras and Pytorch*. URL: https://www.renom.jp/notebooks/tutorial/generative-model/VAE/notebook.html.

7 Appendix

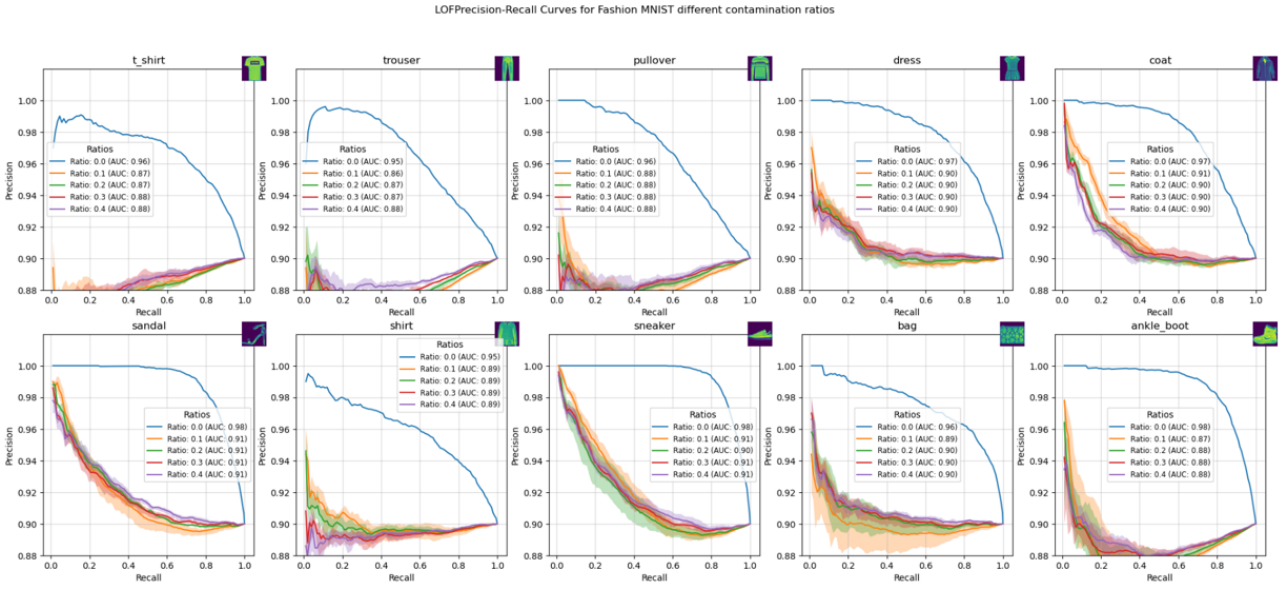

Figure 23: Local outlier factor, Not suitable for this task, performance without contamination is worse than Autoencoder and contamination does affect the performance significantly (Source: Author's own creation using Python)

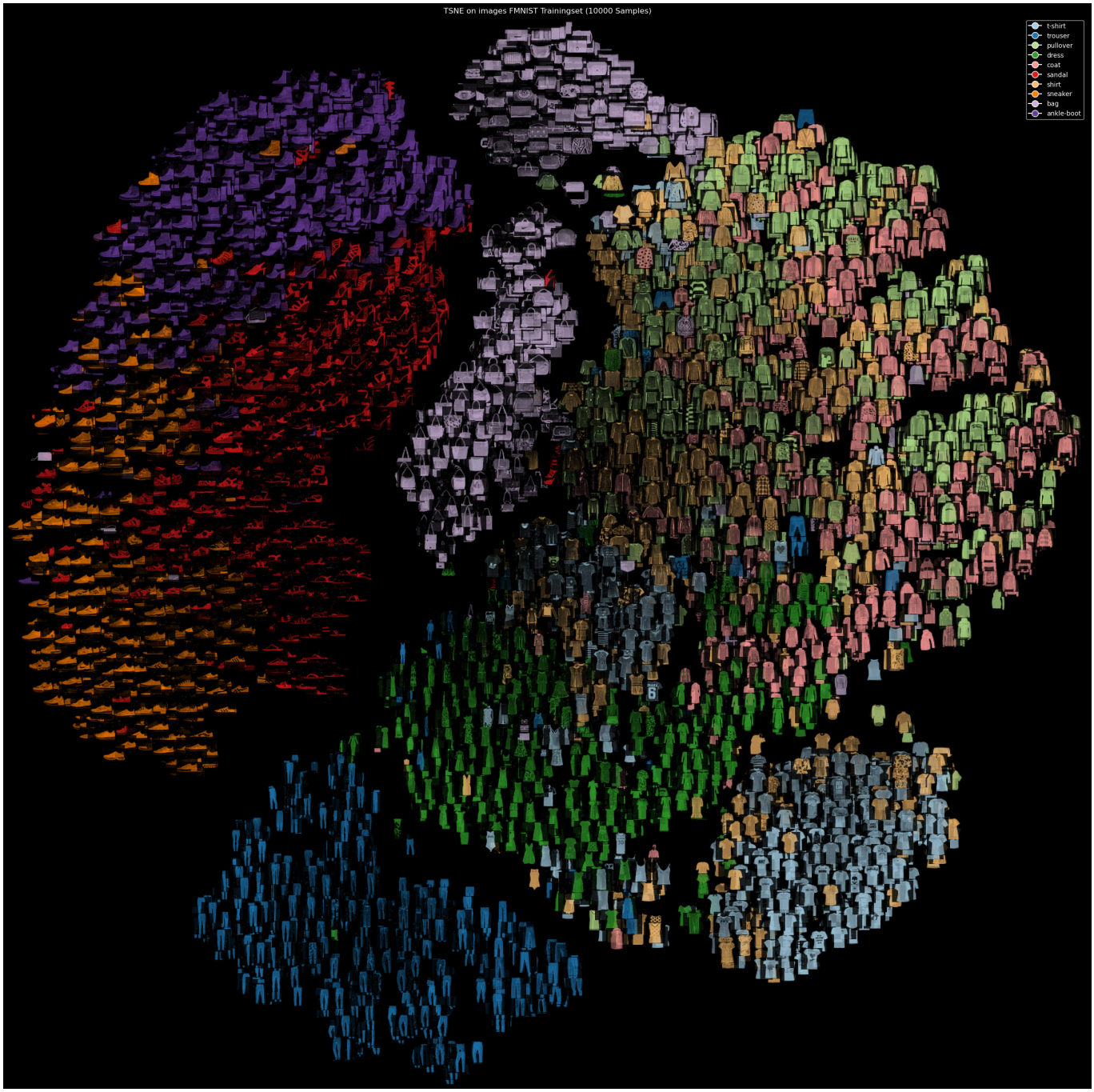

Figure 24: TSNE on the original images applied directly on the 784 pixels of each image, clusters in t-SNE on features are more compact and better separable. (Source: Author's own creation using Python)

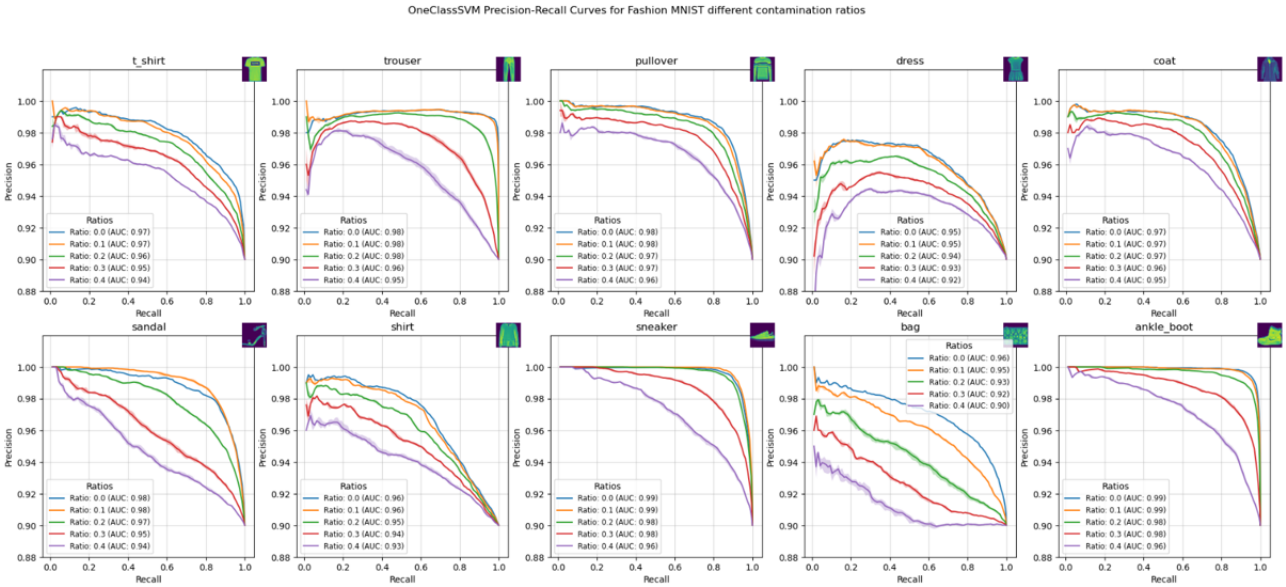

Figure 25: OneClass Support Vector machine. performance of the model is stable, 10 percent contamination mostly comparable to no contamination. (Source: Author's own creation using Python)

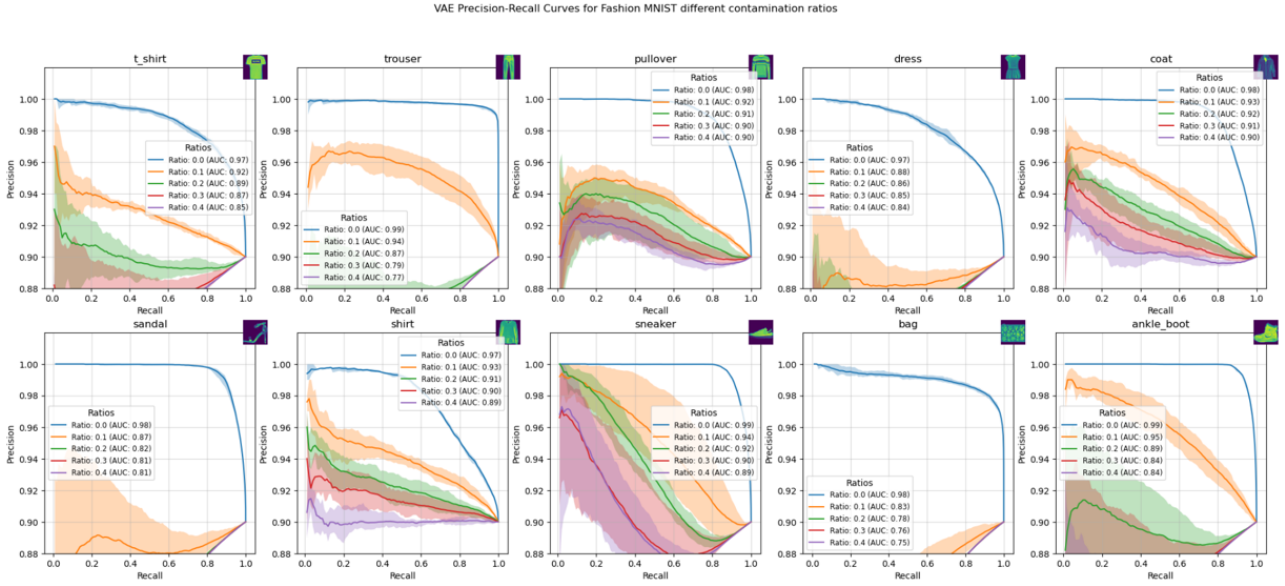

Figure 26: Variational Autoencoder, Performace without contamination comparable to other Autencoder and NTL, however contamination totally disrupts performance. (Source: Author's own creation using Python)