Highlights

Sparse Random Neural Networks for Online Anomaly Detection on Sensor Nodes

Sam Leroux, Pieter Simoens

- We propose a novel anomaly detection model based on sparse random neural networks.
- The sparsity allows for an efficient implementation on resource constrained devices.
- Our model achieves high anomaly detection accuracies on different datasets.
- Unlike most contemporary approaches, our model supports continuous online learning.

Sparse Random Neural Networks for Online Anomaly Detection on Sensor Nodes

Sam Leroux, Pieter Simoens

IDLab, Department of Information Technology at Ghent University - imec, Technologiepark 126, B-9052, Ghent, Belgium

Abstract

Whether it is used for predictive maintenance, intrusion detection or surveillance, on-device anomaly detection is a very valuable functionality in sensor and Internet-of-things (IoT) systems. In this paper, we introduce a novel anomaly detection technique based on sparse, random neural networks. The sparsity in the model allows for a very efficient implementation on embedded or resource constrained hardware. Our approach supports continuous online learning where the model is deployed to the sensor device without any prior training. As new data becomes available, the model is updated and becomes better at detecting anomalies. We experimentally validate our approach on several default benchmark data sets in the visual domain as well as on industrial quality inspection and predictive maintenance tasks. We show that our approach achieves a very favorable trade-off between computational cost and anomaly detection accuracy.

Keywords: Anomaly detection, Continuous learning, TinyML, Edge AI, Sparse neural networks, Random neural networks.

1. Introduction

On-device anomaly detection is a commonly requested feature in Internet-of-Things (IoT) and sensor applications [1], with use cases including predictive maintenance [2], industrial inspection [3], intrusion detection [4] and smart surveillance [5]. While there are many algorithms available that accurately detect anomalies in various types of input data, most of them are ill-suited for deployment on resource constrained edge devices. Despite their limited computational resources, edge devices are an attractive platform for anomaly detection algorithms thanks to their close proximity to the sensor producing the data. In many distributed sensing applications, it would not be scalable to stream all data to a central location for processing. In the case of anomaly detection where, by definition, the events of interest are extremely rare, this would result in a very large communication overhead, especially for rich input data (e.g. video or audio). In addition, privacy considerations might impose constraints on what can be shared with an algorithm deployed in a cloud backend. In those cases, edge deployment of anomaly detection models provides a privacy friendlier solution since the data is processed on a trusted, on-premise, device.

In this paper, we introduce a novel approach, based on sparse random neural networks for on-device anomaly detection. It addresses three main requirements:

Requirement 1: Efficiency: Common approaches for anomaly detection rely on deep (variational) autoencoders [6], Generative Adversarial Networks [7] or Transformers [8]. These achieve high detection accuracies but are computationally very expensive. More traditional techniques such as those based on nearest neighbour analysis require less computations but instead use a large amount of memory to store the entire training dataset in a lookup table[9]. Edge devices however have limited computational resources and memory, making it difficult to deploy these types of models.

Requirement 2: Continual updating: The real world is not static. Changes in operating conditions, wear of the machine, sensor drift, ... all cause the distribution of sensor data to change over time. Without continual learning, anomaly detection algorithms will start reporting increasing numbers of false positives. To deal with the domain shift of the input data distribution, an on-device anomaly detection solution should support continual learning, allowing it to be updated continuously over time as new data comes in. Ideally, the model is updated locally, on the edge device itself. That way no sensor data ever leaves the device. Note however that continual learning might also cause false negatives, for example when a machine slowly degrades over time. Updating the model can then compensate for the anomalies, causing it to miss them until it is too late. It will thus strongly depend on the application whether continuous learning is an advantage or a disadvantage.

Requirement 3: No prior training: We could even go one step further and deploy the model to the device without prior training. A benefit of this approach is that the model specializes on a specific sensor instance [10]. Two similar sensors, mounted on two similar machines, will produce slightly different data, depending on the configuration, location or settings of the machine. In addition, the events that the algorithm should flag as anomalous might also be specific to each machine. It is therefore difficult to develop one global anomaly detection model that is capable of handling all these different environ-

Email address: sam.leroux@ugent.be (Sam Leroux)

ments. Instead, the anomaly detection algorithm will need to be specialized to each machine. Ideally, we train a new dedicated model from scratch for each individual sensor instance. Most existing approaches require an offline training phase before the model can be used for inference. This is less appropriate for on-device anomaly detection as it would require us to record a large number of data points and then either train the model on the edge device (slow) or to transmit the data to the cloud for training (expensive and privacy sensitive). Instead, we would like to deploy the anomaly detector immediately to the device without prior training, the algorithm will then learn automatically over time to model the normal inputs and will become more accurate at detecting any deviating events.

In this paper, we present a novel method that fits these requirements. It is based on a sparse random neural network that is efficient to evaluate and to store. The model requires no prior training and is continuously updated. Despite its simplicity, it achieves competitive results on different visual anomaly detection tasks. We focus on visual anomaly detection because it is applicable in many different domains and because the high dimensionality of the input data is challenging for embedded execution. In addition to the anomaly detection accuracy, we evaluate three characteristics that are important for real-world deployment of the model: The robustness against noisy training data, the performance in the limited data setting and the computational cost.

The remainder of the paper is organized as follows. In section 2 we give an overview of existing literature and provide some insights into why random neural networks extract useful features, despite their random nature. We introduce our model in section 3 and then validate its performance on three benchmark datasets in section 4. We also investigate its capability for online learning, its robustness against noisy training data and its performance in the limited training data setting. We further analyze the computational cost of the model and provide visualizations of the learned features. We also show how our model can be used in combination with rich feature extraction models in section 5 and apply it on time series data in section 6. We conclude in section 7 and give pointers for future research.

2. Related work

2.1. The unreasonable effectiveness of random neural networks

In a random neural network, all or some of the connections are left untrained after their random initialization. This allows for an extremely efficient training process, compared to fully trained architectures [11]. Despite their simplicity, random neural networks are capable of extracting useful feature representations in a variety of different domains. In addition to their training efficiency, randomization allows to use non-differentiable operations in the architecture such as Heaviside step functions [12]. This makes it easier to build models that use binary activations or decision logic in the architecture.

There are different hypotheses on how random neural networks can extract useful information. One interpretation focuses on the connections with kernel machines and Gaussian Processes [13] while others explore the effect of random transformations in metric space. Assuming that points in the input data belonging to different classes form larger angles than points belonging to the same class, it has been shown that the random neural network performs an embedding which reduces the angles of the points from the same class while increasing those of different classes, in effect increasing the separation among classes [14].

In the following paragraphs, we briefly discuss the history of random neural networks. For a more in depth overview on this topic, we refer to some excellent surveys [11, 15].

Neural networks with a single hidden layer with fixed, random connections have been known for a long time. The most well studied implementations are Random vector functional-links (RVFLs) [16]. RVFLs can be used for classification and regression. They approximate a function using a weighted sum of randomly extracted features. The weights used in the weighted sum are then optimized by formulating the optimization problem as an ℓ_2-regularized least squares [11]. Extensions of RVFLs then build ensembles [17, 18] or optimize the distribution from which the random parameters are selected [19]. For an in depth overview of recent advances in RVFLs, we refer to [20].

In recent years, the study of Random neural networks has been focused on deeper architectures where multiple layers are stacked to obtain a more powerful function approximator. This research also provides insight into the behavior of a deep neural network in the initial stages of training (i.e. when the weights are still close to their initial, random values). It also allows us to understand how much of the performance of a neural network is due to the training algorithm and how much can be attributed to the biases encoded in the architecture (e.g. the use of convolutional filters or pooling to introduce invariance). It has been shown that a randomly initialized convolutional neural network (CNN) can extract enough structural information to perform image denoising [21], image classification [22, 23], style transfer [24] and audio classification [25].

In this work, we focused on feed forward models. Yet, there is also a substantial history of approaches that applied the concepts of random neural networks to recurrent models. These are especially useful if there is a temporal relationship between inputs. One noteworthy paradigm is Reservoir Computing [26]. The basic idea is to use random weights for the recurrent hidden layer and only train the last part of the network. The random connections then transform the input to a high-dimensional feature space where the learning problem can be performed using a linear model. Other similar techniques include Echo State Networks[27] which, like our approach, also use a high sparsity level, Liquid State Machines [28] and Fractal Prediction machines [29].

Several works have proposed the use of random projections for

anomaly detection tasks [30, 31, 32, 9, 33]. These works however mainly focus mostly on anomaly detection performance whereas we also demonstrate other key benefits such as a low computational cost (thanks to the sparsity), support for continuous online learning and increased robustness to noisy training data.

Our approach is most similar to the Arrays of (locality-sensitive) Count Estimators (ACE) [9] which uses Signed Random projections (SRP) to hash inputs to binary hashcodes. A single neuron in our neural network performs the same operation, i.e. a multiplication with a random vector, followed by a thresholding function. The ACE algorithm uses L of these hash functions, combined into a meta-hash function. This is similar to how we combine multiple neurons into a single layer. Our approach differs in that we can use multiple layers, each followed by a non-linear activation function (step function). This allows us to extract more complex features from the data. In addition, we force most of the weights to be zero, resulting in sparse matrix-vector multiplication. We also focus more on the online learning aspect and show that our method can automatically adapt to changes in the input data distribution.

2.2. Sparsity in neural networks

An important part of our approach is the high sparsity rate of the neural network. This drastically reduces the memory footprint and computational cost and enables efficient evaluation on resource constrained devices. For these reasons, sparse neural networks have gathered much attention in the past years. In addition, biological brains are also highly sparse and this sparsity is often quoted to be crucial in scaling biological brains [34].

It is well known that typical deep neural networks are over-parameterized [35, 36]. There is good empirical evidence that this is not strictly needed to perform the task itself but instead seems to make training the model with stochastic gradient descent easier [37, 38]. This however comes at the expense of an increased memory consumption and computational cost, making them less suited for resource constrained edge devices. In addition, the over-parameterization might also increase the risk of overfitting [39] or might even make the model more sensitive to adversarial attacks [40].

In the following paragraphs, we focus mainly on approaches that use sparsity to improve the efficiency of the models during inference. For a more in-depth overview, we instead refer to [41].

Sparsificiation can be applied at different stages during development. The most common approach is to use a standard dense training procedure, after which weights are pruned from the model [41]. The model is then typically finetuned to recover some of the lost accuracy. The disadvantage of this approach is that it does not reduce the training cost since a full dense model needs to be trained first. It is therefore not very well suited to our use case of continuous online, on-device learning. Other approaches increase the sparsity automatically during training [42], even before the model has been trained to convergence. This is usually cheaper than the previous approach but is also more complex as the pruning schedule might need to correct for errors caused by premature pruning. In the extreme case, we can start with a sparse model and train the entire model in the sparse regime by removing and adding elements during training. This provides the most benefits in terms of computational cost but often requires complex hyper parameters and training routines.

Regardless of the stage in which the pruning is applied, there are different techniques that can be used to select the weights to remove. Simple heuristics such as those based on weight magnitude [43, 44] often work well. Other more advanced approaches consider the sensitivity of the output of a neuron to the input sample [45], the similarity between the activations of two neurons [46], second order gradient information [47] or bayesian statistics [48].

It is crucial to note that a sparse model does not necessarily guarantee a speed-up in practice. Since hardware such as Graphical Processing Units (GPUs) are designed for dense computations, small dense models are often faster in practice than sparse models of a comparable size [49]. There is however an increased interest in hardware and software solutions that can exploit sparsity, especially for resource constrained IoT applications. For an in-depth overview of these approaches, we refer to [50].

2.3. Online anomaly detection

Anomaly detection has been a very active research domain in the past decades since it has so many valuable applications in industry, healthcare, finance and security [51]. Many algorithms have been developed for different use cases and input data types. Neural network based approaches typically outperform other approaches, especially on rich input data (images, audio, video) [51]. The most common approaches are based on deep (variational) autoencoders [6], Generative Adversarial Networks [7] or Transformers [8]. Their performance however comes at a very high computational cost, making them less suited for resource constrained edge devices.

While most approaches require a separate training and evaluation stage, we are instead more interested in the continuous learning setting where the model needs to adapt to changes in the environment. This is a very challenging task since the updating procedure needs to be efficient while at the same time avoiding catastrophic forgetting [52, 53]. Interesting approaches combine the feature extraction power of neural network-based models with the continual learning capability of statistical detection methods [54] or allow models to be selectively updated in an efficient way [55]. Other approaches use human feedback to correct the model [56]. It is unlikely that there will only be a single instance of the sensor. In a factory for example, there will be multiple identical production lines. It would then be interesting the exchange information between the different models that monitor the different lines. Federated or collaborative learning

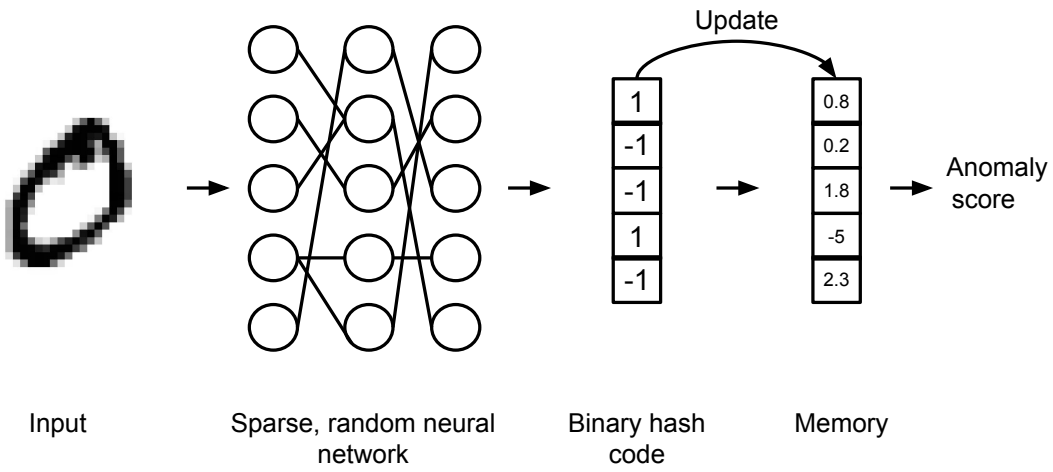


Figure 1: Our model uses a randomly initialized neural network with a high sparsity to transform the input to a binary hash code. We keep track of a running average of the hash codes observed so far in a memory. The anomaly score is calculated as the dot product of the hash code and the memory.

would be able to exploit this to improve the overall anomaly detection accuracy [57].

3. Approach

Algorithm 1 Initialization and forward pass of the model

L: Number of layers in the model
n_i: Number of neurons in layer i
s_i: Sparsity rate of layer i
λ: Learning rate
$\mathbf{W_i}$: Weight matrix of layer i
memory: Memory array

procedure INIT
  for $i = 1$ **to** L **do**
    for $j = 1$ **to** n_i **do**
      $\mathbf{W_{ij}} = Uniform(-1, 1)$
      $\mathbf{W_{ij}} = \mathbf{W_{ij}} \times Bernoulli(1 - s_i)$
    end for
  end for
end procedure

procedure FORWARD($\mathbf{x}$)
  for $j = 1$ **to** L **do**
    $x = sparse\_dot(\mathbf{x}, \mathbf{W_i})$
    $x = binarize(\mathbf{x})$
  end for
  $score = \mathbf{x} \cdot \mathbf{memory}$
  $memory = \lambda * \mathbf{x} + (1 - \lambda) * \mathbf{memory}$
  return score
end procedure

Our approach is based on Locality Sensitive Hashing (LSH) [58], an algorithmic technique which uses hash functions to map input data points to buckets. These functions are designed to map similar inputs to the same bucket. LSH is typically used for data clustering [59] and nearest neighbor search [60] but can also be used for anomaly detection [61]. Since anomalous data points differ substantially from normal inputs, they will be mapped to different buckets. An anomaly score is then obtained by counting how many normal training samples are allocated to the same bucket as the test sample. Different families of hash functions such as p-stable hashing [62] and randomized trees [63] have been proposed before while other, more advanced methods use properties of the training data to select suitable hash functions [64].

A major benefit of LSH-based approaches for anomaly detection is that they have a limited computational and memory footprint. The computational cost depends on the number and type of hash functions but is typically much lower than that of alternative approaches such as One-class SVMs . In addition, the memory consumption is constant, independent of the number of training examples. Alternative approaches such as those based on nearest-neighbor analysis store and process the entire dataset to evaluate a new sample. Their computational cost and memory consumption will increase as the model is trained on more training data. In addition, LSH-based approaches are better suited for continual learning since updating the model requires only updating the bucket counters.

We propose to replace the hash functions in the LSH model with a random sparse neural network with binary activations. This combines three main benefits. First, the weights of the model are chosen at random and are not updated afterwards. There is no need to collect a training dataset nor to update the weights before or during deployment. Our approach follows the well known paradigm that states that randomization is (computationally) cheaper than optimization [65]. In addition, the weights of the model are completely characterized by a random seed. It is therefore not needed to store or transfer the exact weights as they can always be recovered by sampling from the random

generator initialized with that specific seed. A second important benefit is the sparsity of the weights. In our experiments, only 0.1% to 10% of the connections have a non-zero weight. The multiplications with a zero weight can then be skipped, resulting in a drastic speed up. Sparse connectivity is often used to reduce the computational cost and memory footprint of deep neural networks [66]. Obtaining a sparse model can be challenging however. Most approaches rely on pruning after training to remove connections based on second order gradient information [47], weight magnitude [44] or bayesian statistics [48]. This is less suited for online, on-device training as it involves training a full, dense model and then slimming it down to a more efficient version. There are also approaches that train a sparse model from scratch but this typically results in less accurate models [67]. The sparsity of our model introduces no training difficulties as the model is initialized randomly and the weights are never updated. Finally, the neural network is designed by stacking one or more of these layers, each followed by a step activation function. This effectively binarizes the activation of each layer. Binary activations are commonly used to reduce the computational cost of deep neural networks [68, 69, 70].

In addition to the random sparse neural network, our anomaly detection model also requires an array of floating point counters as can be seen in Figure 1. This is the only part of the model that changes over time. To process a new datapoint, we first pass it through the neural network to obtain a fingerprint of the input consisting of a binary activation vector with -1 and +1 values. To model the data observed so far, we keep track of an exponential moving average of the activations (the memory). This allows us to easily update the model over time as new data has been observed. To predict the anomaly score for a data point, we simply calculate the dot product between its activation vector and the memory. Algorithm 1 describes the entire process. We also include a Python implementation in Appendix A.

Intuitively, the sparse random neural network projects the input datapoint to a high dimensional feature space. Even though the weights of the neural network are initialized randomly, this transformation improves the separability of the input data [14]. The memory can then be interpreted as the average feature representation observed so far and the dot product effectively calculates a similarity metric between the feature representation of the input and this average (prototype) representation.

To initialize the neural network, we first need to decide on the number of layers and neurons in each layer. The only constraint is the number of neurons in the last layer, which should match the size of the memory array. We sample the weights randomly from a uniform distribution between -1 and 1 and set most of the weights to zero by sampling from a Bernoulli distribution with probability $p = 1 - s$ where s is the desired sparsity rate e.g. 0.999, meaning that 99.9% of the connections will have a weight of zero.

4. Experiments

We first validate our approach on three benchmark datasets: MNIST [71], FashionMNIST [72] and CIFAR10 [73]. These are commonly used datasets to evaluate anomaly detection algorithms [7]. Their visual nature makes it easier to qualitatively understand the reported anomalies. Each dataset contains ten labeled classes. The MNIST and FashionMNIST datasets contain 28×28 grayscale images of respectively handwritten digits and clothing items. The more challenging CIFAR10 dataset contains 32×32 RGB images of daily objects such as cars, ships and various animals. We follow the common methodology in anomaly detection literature to treat the data from one class as normal and the other classes as abnormal [74]. The model is trained on the normal class and then has to flag inputs from other classes as anomalous during testing. We did not perform dataset augmentation. Other approaches such as [7] improve the generalization by training on an augmented version of the original dataset. They take random crops, flips and rotations of the images and they randomly adjust the hue, saturation, brightness and contrast. This increases the performance but makes it difficult to objectively compare the performance with other methods that use different augmentation techniques since it is unclear how much of the increase in anomaly detection performance can be attributed to the model and how much to the data augmentation. Unless otherwise noted, we use all available training data (of a single class) to train the models.

Table 1 describes the architectural details of the sparse networks used in our experiments. For the MNIST dataset, we used a single layer model with 1000 neurons and a sparsity rate of 99%. For the FashionMNIST dataset we used a single-layer with 5000 neurons and a higher sparsity of 99.9%. For the more challenging CIFAR10 dataset, we used three layers of 10000, 5000 and 1000 neurons respectively and a sparsity rate of 99.9%. We set the exponential moving average parameter λ to 0.1. These hyper parameters were determined by grid search, using a subset of the train data for validation. We will analyze the impact of these hyperparameters in section 4.2.

In the next sections, we experimentally analyze different aspects of our model. We start in section 4.1 with a quantitative analysis of the anomaly detection performance. We also show some qualitative examples to prove that the model indeed detects sensible anomalies. In the subsequent sections, we focus on different aspects such as the impact of the sparsity rate (section 4.2), the capability for online learning (section 4.3), the robustness against noisy training data (section 4.4) and the performance when training data is scarce (section 4.5). We then also analyze the computational cost (section 4.6) and show how we can visualize the features that the model has learned (section 4.7).

4.1. Anomaly detection performance

Table 3 presents the anomaly detection performance in terms of the Area Under the ROC-Curve (AUC-ROC) score for each

Table 1: Model architecture details for each of the three benchmark datasets.

Dataset	MNIST	FashionMNIST	CIFAR10
Input size (pixels)	$28 \times 28 = 784$	$28 \times 28 = 784$	$32 \times 32 \times 3 = 3072$
Train samples	60,000	60,000	50,000
Test samples	10,000	10,000	10,000
Number of layers	1	1	3
Number of neurons	1000	5000	10000, 5000, 1000
Memory array size	1000	5000	1000
Layer sparsity	99%	99.9%	99.9%
Non-zero weights	7,840	3,920	85,720

class of each dataset. As is common in anomaly detection literature, we use all available training images and then report the performance on a held-out test set with 1.000 normal images (belonging to the same class as the training data) and 1.000 abnormal images (randomly selected from the other nine classes). We also report the average score over all ten classes. During testing, we set the exponential moving average parameter λ to 0. That way, our model is never updated using test data.

We compare our approach against commonly used anomaly detection algorithms such as One-Class SVMs (OC-SVM) with an RBF-kernel, Isolation Forests (IF), Local Outlier factor (LOF) and a Principal Component Analysis based approach (PCA). For this last method, we train PCA to reduce the dimensionality of the input samples to five (ten for the more complex CIFAR10 dataset), after which we reconstruct the input using the inverse transform. We then use the reconstruction error (Mean squared error) as anomaly metric. All baseline methods use the scikit-learn [75] implementation. We also list the performance of some recent state-of-the-art approaches. These are all based on deep neural networks and obtain a very high accuracy. They however also use a much more complex architecture, often based on large networks such as ResNets or Generative Adversarial Networks, making them less suited for resource constrained edge devices. In addition, these also do not support continuous online learning. The results for these models were copied from the respective papers.

Table 3 shows that our method obtains anomaly detection performances that are comparable to the baseline methods and even to some of the much more complicated deep learning based approaches.

We show a qualitative analysis of the results in Table 2. For each dataset, we show examples of normal test images (belonging to the class the model was trained on). We show images that have the lowest anomaly score (correctly predicted to be normal) and images that have the highest anomaly score. The latter being false positives (incorrectly predicted to be anomalies). False positives have a large impact on anomaly detection systems since they typically require further analysis by a human expert. These examples show that despite its simplicity, our model is able to capture some typical attributes of the normal class and that the false positives indeed differ substantially from the normal images, even though they belong to the same class and are not labeled as anomalous. For the MNIST dataset for example, the model even detects a wrongly labeled sample in the dataset (first digit of class "three" should be labeled "five"). For the FashionMNIST dataset, the anomalies show multiple instances of the same clothing item in one picture or substantially different styles (e.g. class five: "Sandal"). The CIFAR10 dataset is harder to interpret but here the anomalies are either close-up images where only part of the object is visible, images with text in the background or unusually colored objects (e.g. a blue frog).

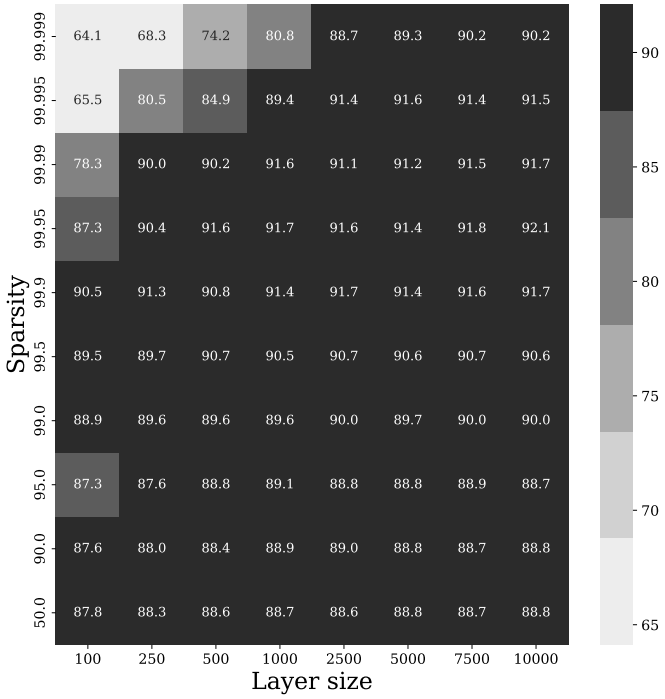


Figure 2: Average AUC over all classes on the FashionMnist dataset for a single layer model with a different number of neurons and sparsity rate.

4.2. Impact of sparsity and layer size

Each layer in the neural network is characterized by two hyper parameters: the number of neurons and the sparsity rate. To evaluate the impact of these hyperparameter values on the

Table 2: Examples of normal images with a low anomaly score (True negatives) and a high anomaly score (potential false positives) for the different classes of the MNIST, FashionMNIST and CIFAR10 dataset.

	MNIST		FashionMnist		CIFAR10	
Class	Low anomaly score	High anomaly score	Low anomaly score	High anomaly score	Low anomaly score	High anomaly score
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						

anomaly detection performance, we performed a grid search of the model for FashionMNIST. Figure 2 reports the average AUC over all classes for a one layer model with the number of neurons varying from 100 to 10000 and varying sparsity from 50% to 99.999%.

Although most of the combinations result in a similar performance, we can make two observations. First, for a given sparsity rate, a higher layer size tends to result in a higher AUC score. This is not surprisingly as a larger layer size increases the networks capacity to extract features from the data. The sparsity rate however has a more complex relation with the final AUC score. The highest scores are usually obtained at sparsity rates of 99 % to 99.99 %. As the layer size increases, we can see that the model can tolerate a higher sparsity rate.

For completeness, we performed the same experiment on the MNIST and CIFAR10 dataset. These results can be found in Appendix B.1 and support a similar conclusion. One remarkable observation is that the FashionMNIST dataset can tolerate a higher sparsity rate than the MNIST dataset. A possible explanation of this can be found in the fact that the MNIST data samples are already very sparse (most of the values in the input are zero). Where the MNIST samples show a narrow pen stroke, the FashionMNIST samples show larger objects that are typically filled in (many more elements with non-zero values). As all the connections to the input are random, a sparse input will require more connections to pick up on a useful feature.

4.3. Online learning

A key benefit of our approach is that it naturally supports online continuous learning. Every time a sample is processed, we return an anomaly score and update the memory array using a running average as explained previously. Note that only the memory array is updated and the neural network weights remain at their original, random values. In Figure 3 we show how our method supports online learning to deal with domain shift. We used a held-out test set of FashionMNIST samples belonging to class zero (t-shirt) and one (pants) and show the median predicted anomaly score on these two test sets with the blue and orange line respectively. We begin the experiment with a completely untrained model (a zero-valued memory array). The x-axis shows the number of samples that were presented to the model. We first present 100 images belonging to class zero and each time calculate the median anomaly score on both test sets. After the first few inputs, both anomaly scores drop rapidly. This is because both classes share some similarities, i.e. they both show a white object on a black background. As more images of class zero are processed by the neural network, we see a stronger decrease of the anomaly score of class zero and the model becomes better at recognizing the anomalies. For the next hundred samples, we only present samples of class one. We observe that the model starts predicting a lower average anomaly score for class one, and correspondingly, the average anomaly score of class zero increases as more samples of class one are observed. After just twenty samples, the model has recovered from the shift in input data. For the next hun-

Table 3: Anomaly detection ROC-AUC (%) on the three benchmark datasets where one class is considered normal and the others anomalous. For our method, we report the average AUC over 10 runs, together with the standard deviation.

Dataset	Method	0	1	2	3	4	5	6	7	8	9	Avg
MNIST	Ours	97.5	99.3	89.1	92.2	91.8	78.4	92.7	93.4	89.1	90.8	91.5 ± 0.06
	IF	96.7	99.4	75.8	83.3	87.2	75.0	87.7	91.3	75.2	87.5	85.9
	LOF	99.7	99.5	95.2	97.2	96.9	97.8	99.8	97.7	91.5	97.5	97.3
	OC-SVM	97.6	99.1	80.1	87.2	91.3	74.3	92.1	92.3	83.0	89.7	88.7
	PCA	96.1	99.8	83.1	89.8	92.1	91.0	96.3	94.9	82.9	93.6	91.9
	DSEBM [76, 6]	94.9	98.7	69.0	80.2	83.3	67.4	85.6	90.4	72.1	86.8	82.8
	CAE + OC-SVM [6]	95.4	97.4	77.6	88.6	83.6	71.3	90.1	87.2	86.5	87.3	86.5
	VAE [7]	92.1	**99.9**	81.5	81.4	87.9	81.1	94.3	88.6	78.0	92.0	87.7
	AE [7]	98.8	99.3	91.7	88.5	86.2	85.8	95.4	94.0	82.3	96.5	91.9
	GANomaly [77]	97.2	99.6	85.1	90.6	94.5	94.9	97.1	93.9	79.7	95.4	92.8
	AnoGAN [78]	99.0	99.8	88.8	91.3	94.4	91.2	92.5	96.4	88.3	95.8	93.7
	ADGAN [79]	99.5	**99.9**	93.6	92.1	94.9	93.6	96.7	96.8	85.4	95.7	94.7
	InceptionCAE [6]	98.7	99.7	96.7	95.2	95.0	95.2	98.3	97.0	96.2	97.0	96.9
	OCGAN [80]	99.8	**99.9**	94.2	96.3	97.5	98.0	99.1	98.1	93.9	98.1	97.5
	DASVDD [81]	99.7	**99.9**	95.4	96.2	98.1	97.2	99.6	98.1	94.2	98.3	97.7
	GeoTrans [82]	98.2	91.6	99.4	99.0	99.1	99.6	**99.9**	96.3	97.2	99.2	98.0
	ARNet [83]	98.6	**99.9**	99.0	99.1	98.1	98.1	99.7	99.0	93.6	97.8	98.3
	GAN-based [7]	**99.9**	**99.9**	**99.7**	**99.8**	**99.7**	**99.8**	99.7	**99.7**	**99.5**	**99.4**	**99.7**
Fashion-MNIST	Ours	92.2	97.4	89.4	93.9	89.8	88.9	80.4	98.6	88.3	98.7	91.8 ± 1.05
	IF	91.0	97.7	87.7	94.3	90.4	92.7	80.6	98.4	88.4	97.9	91.9
	LOF	83.3	95.4	87.4	90.5	91.7	89.0	80.8	96.9	81.7	97.6	89.4
	OC-SVM	88.5	97.2	85.9	90.4	87.4	87.15	80.1	98.2	82.1	97.6	89.5
	PCA	89.7	97.2	86.6	91.5	88.0	86.3	83.2	98.0	81.3	92.7	89.5
	GANomaly [77]	80.3	83.0	75.9	87.2	71.4	92.7	81.0	88.3	69.3	80.3	80.9
	DSVDD [84, 85]	79.1	94.0	83.0	82.9	87.0	80.3	74.9	94.2	79.1	93.2	84.8
	DSEBM [76, 6]	91.6	71.8	88.3	87.3	85.2	87.1	73.4	98.1	86.0	97.1	86.6
	OCGAN [80]	85.5	93.4	85.0	88.1	85.8	88.5	77.5	93.9	82.7	97.8	87.8
	DAE [85]	86.7	97.8	80.8	91.4	86.5	92.1	73.8	97.7	78.2	96.3	88.1
	VAE [85]	87.4	97.7	81.6	91.2	87.2	91.6	73.8	97.6	79.5	96.5	88.4
	ADGAN [79]	89.9	81.9	87.6	91.2	86.5	89.6	74.3	97.2	89.0	97.1	88.4
	CAE + OC-SVM [6]	88.0	97.3	85.5	90.0	88.5	87.2	78.8	97.7	85.8	98.0	89.7
	ICS [86]	88.3	98.9	88.2	92.1	90.2	89.4	78.3	98.3	88.6	98.5	91.1
	DASVDD [81]	91.2	99.0	89.3	93.7	90.7	93.8	82.8	98.6	89.4	97.9	92.6
	HRN [85]	92.7	98.5	88.5	93.1	92.1	91.3	79.8	99.0	94.6	98.8	92.8
	GeoTrans [82]	99.4	97.6	91.1	89.9	92.1	93.4	83.3	98.9	90.8	99.2	93.5
	ARNet [83]	92.7	99.3	89.1	93.6	90.8	93.1	85.0	98.4	**97.8**	98.4	93.9
	InceptionCAE [6]	92.4	98.8	90.0	95.0	92.0	93.4	85.5	98.6	95.1	97.7	93.9
	GAN-based [7]	**99.5**	**99.6**	**98.2**	**98.6**	**98.1**	**99.5**	**95.9**	**99.4**	97.6	**99.6**	**98.6**
CIFAR10	Ours	75.6	64.8	56.9	58.8	68.4	63.1	69.4	65.0	78.2	75.3	67.6 ± 1.03
	IF	65.2	45.1	63.8	51.0	74.1	50.9	70.6	52.5	68.6	53.8	59.6
	LOF	64.5	43.0	67.1	50.9	73.0	48.1	70.9	49.0	66.7	40.4	57.4
	OC-SVM	61.7	43.0	61.9	49.9	72.6	50.1	68.1	50.6	65.7	50.6	57.4
	PCA	64.0	39.2	64.7	52.4	72.1	47.7	66.8	47.9	66.4	39.7	56.1
	VAE [7]	62.0	66.4	38.2	58.6	38.6	58.6	56.5	62.2	66.3	73.7	58.1
	CAE + OC-SVM [6]	62.4	44.4	64.2	50.7	74.8	50.9	72.4	51.0	67.0	50.8	58.9
	AE [7]	57.1	54.9	59.9	62.3	63.9	57.0	68.1	53.8	64.4	48.6	59.0
	DSEBM[76, 6]	64.1	50.1	61.5	51.2	73.2	54.6	68.2	52.8	73.7	63.9	61.3
	ADGAN[79]	63.2	52.9	58.0	60.6	60.7	65.9	61.1	63.0	74.4	64.4	62.4
	GANomaly[77]	93.5	60.8	59.1	58.2	72.4	62.2	88.6	56.0	76.0	68.1	69.5
	InceptionCAE [6]	66.7	71.3	66.8	64.1	72.3	65.3	76.4	63.7	76.9	72.5	69.6
	DASVDD [81]	68.6	64.3	55.8	58.8	58.6	64.0	62.6	71.0	64.6	81.1	73.7
	GeoTrans[82]	74.7	95.7	78.1	72.4	87.8	87.8	83.4	95.5	93.3	91.3	86.0
	ARNet[83]	78.5	89.8	86.1	77.8	90.5	84.5	89.2	92.9	92.0	85.5	86.6
	GAN-based[7]	**92.6**	93.6	86.9	85.4	89.5	87.8	93.5	91.0	94.6	91.7	90.6
	OODformer [8]	92.3	**99.4**	**95.6**	**93.1**	**94.1**	**92.9**	**96.2**	**99.1**	**98.6**	**95.8**	**95.7**

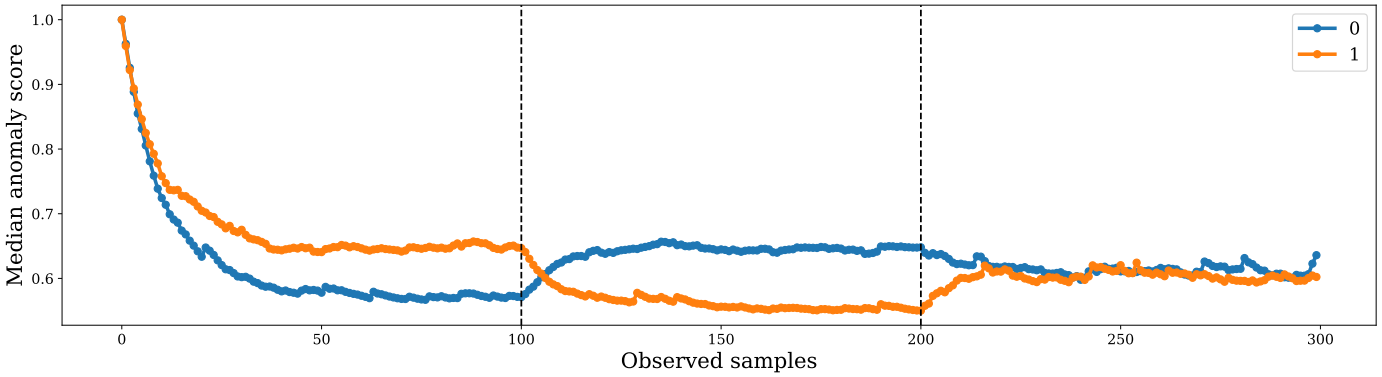


Figure 3: Online learning capability of our method. We start at time step 0 with a completely untrained model. We then present 100 randomly selected samples from class 0 of the FashionMNIST dataset to the model and measure the median anomaly score for the held out test samples of class zero (blue line) and class one (orange line). As more examples of class 0 have been observed, the anomaly score of class 0 decreases rapidly. At step 100, we introduce a domain shift and present samples from class 1. The model quickly adjusts and the predicted anomaly score of class 1 decreases while the samples from class 0 are predicted to be anomalous. Starting from step 200, we provide samples from both classes, the predicted anomaly score converges to a value in between both extremes.

dred samples, we present images randomly sampled from both classes. We can see that both median anomaly scores converge to a similar value.

4.4. Robustness against noisy training data

As is common in literature, we trained the models of the previous sections on a dataset void of anomalies. In practical applications however, the input data might be contaminated with anomalous data points. As this will reduce the anomaly detection performance, it is interesting to investigate the robustness of the methods to anomalies in the training data. Therefore, we introduced anomalies in the training data and measured the AUC score for different models, trained on different levels of contaminated training data. While in the previous sections, the training data belonged to a single class (the normal class), we now include a small fraction of samples randomly sampled from the other classes as distractions. Figure 4 shows these results for the MNIST dataset. For completeness, we performed the same experiment on the other two datasets and include the results in Appendix B.2. These support similar conclusions.

As expected the anomaly detection performance decreases as the frequency of anomalous samples in the training data increases. Not all methods are equally sensitive however. The Local Outlier Factor seems to be the most affected by the training data noise. One possible explanation is that Local Outlier Factor relies on nearest neighbor analysis. As we add anomalous training data points from different classes, clusters will occur that correspond to each of these classes. An anomalous input will then lie close to one of these clusters, resulting in a lower anomaly score or false negative. A benefit of our method and other methods such as the PCA-based approach or the One-class SVM is that they do not store individual training data points. Instead, they use aggregated information to learn a decision boundary that is more robust to noisy training data. Compared to the other methods, our approach is significantly more robust against noisy training data. This can be explained by the use of the running average to update the model. A method such as Local Outlier Factor stores all training data. Anomalous data points observed during training will keep influencing the predictions in the future. Our approach however will "forget" these anomalies and instead will focus on the features that are observed regularly.

4.5. Performance in the limited data setting

In this section, we evaluate the anomaly detection performance of our model when limited training data is available. We envision a use case where the anomaly detector is deployed in a new environment without prior training. After only a few examples, the model should start returning useful predictions. Figure 5 shows the AUC score on the MNIST dataset, averaged over all classes for the different models as a function of the number of train samples. The leftmost point corresponds to just 5 samples while the rightmost point corresponds to using the entire training data set. The local outlier factor method benefits most from the increase in training data. This is not completely unexpected as this method relies on a k nearest neighbors query to estimate a local density around the test point. As more training data is available, the density estimation becomes more representative of the real data distribution. Our method outperforms most other methods and already achieves a score close to its maximum score after only 100 samples have been observed. For completeness, we repeated the same experiment for the FashionMNIST and CIFAR10 experiments. The results can be found in Appendix B.3 and support a similar conclusion.

4.6. Computational cost

The computational cost of an anomaly detection algorithm is crucial when it needs to be deployed on resource constrained edge hardware. As these devices are often battery powered, lower computational costs directly translate in longer operation times and lower maintenance costs. In table 4, we show the execution time of the different models from Table 1, measured on a Raspberry Pi 2B embedded PC with a 900 MHz quad-core ARM Cortex-A7 CPU. We use the execution time on this resource constrained platform as a proxy for the computational

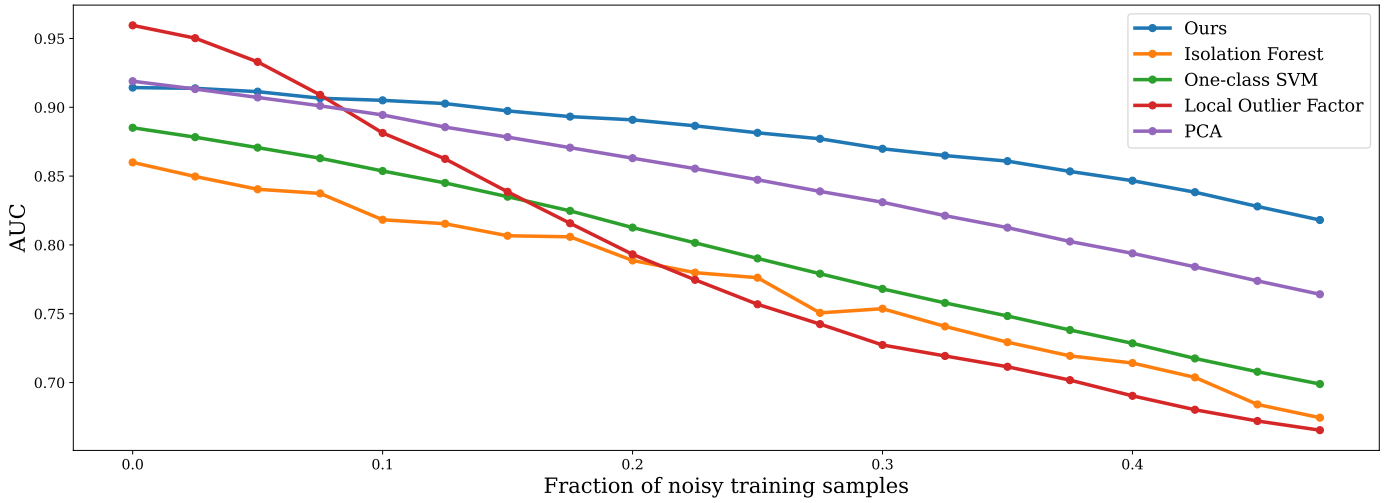


Figure 4: Average AUC over all classes for the MNIST dataset with an increasing ratio of anomalies in the training data (0.1 = 10% anomalies). As the number of noise train samples increases, the AUC of all methods decreases. The Local Outlier Factor and isolation forest seem to be the most affected by this. Our method is the most robust.

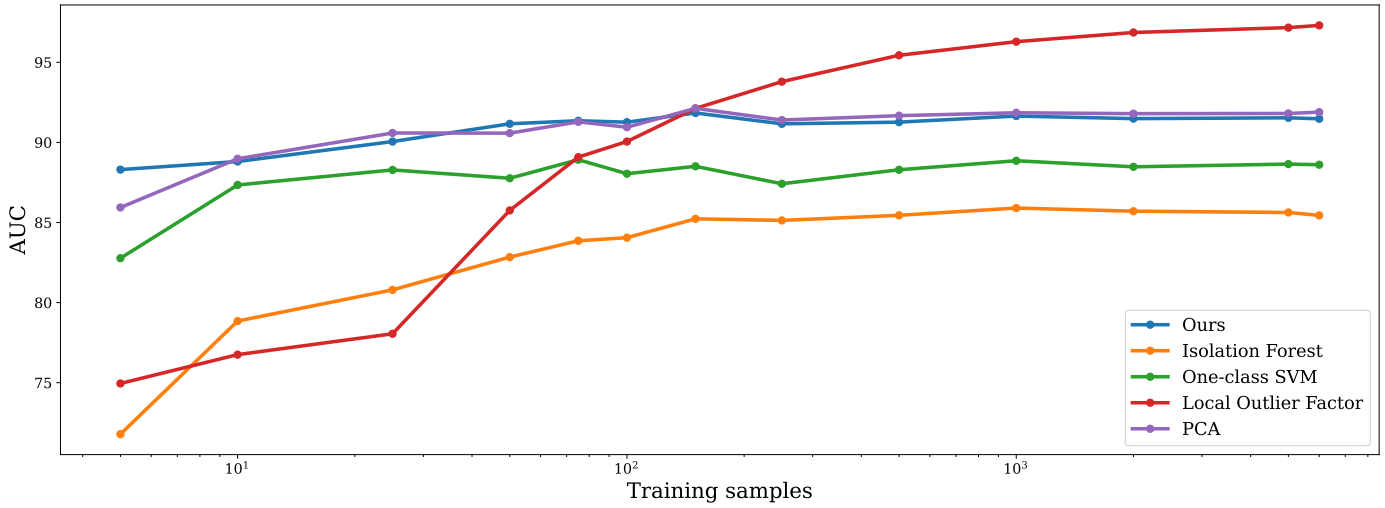


Figure 5: Average AUC over all classes for the MNIST dataset with an increasing number of training samples. The AUC score increases for all methods with increased training data. The Local Outlier Factor method benefits most. Our method and the PCA based approach perform best in the limited training data setting.

cost and corresponding energy consumption. We show the average time for running inference on a single sample. Since our method supports continuous online learning, this time also corresponds to the time needed to update the model with a new data point. The other models do not support continuous learning. For these, we first trained the model on the entire training dataset and then measured the average inference cost for a test sample. Table 4 shows that our method outperforms most of the alternative approaches, only the PCA-based anomaly detection algorithm has a comparable execution time. For completeness, we also evaluated a simple autoencoder model. For the MNIST and FashionMNIST this model has six fully connected layers with 256, 64, 32, 64, 256, 784 neurons respectively. For the CIFAR10 dataset we use six convolutional layers with a 3x3 filter size, 2x2 maxpooling in between the layers of the encoder and 2x2 upsampling in the decoder. The layers have 8, 16, 32, 16, 8 and 3 convolutional filters respectively. These architectures were choosen because they obtained a similar AUC score as our approach. Although these models are still much smaller than the other deep neural network based approaches from Table 3, they are already 10 to 15 times slower than our model.

The efficiency of our model results from the sparsity of the neural network. Where usually, a neural network has a dense weight matrix, in a sparse neural network, most of the connections have a weight of zero. These zero values can then be skipped during evaluation. This however requires us to store the weight matrix in a different structure that allows algorithms to exploit the sparsity. In our experiments, we use the Compressed Sparse Column (CSC) format [87] implemented in SciPy [88]. There is however also an overhead involved in performing operations on sparse matrices as compared to dense matrices and the

Table 4: Execution time in milliseconds, measured on a Raspberry Pi 2B 900Mhz CPU

Model	MNIST	FashionMNIST	CIFAR10
Ours	2	1	7
IF	490	490	580
LOF	80	80	38
OC-SVM	15	12	11
PCA	2	2	5
AE	15	15	103

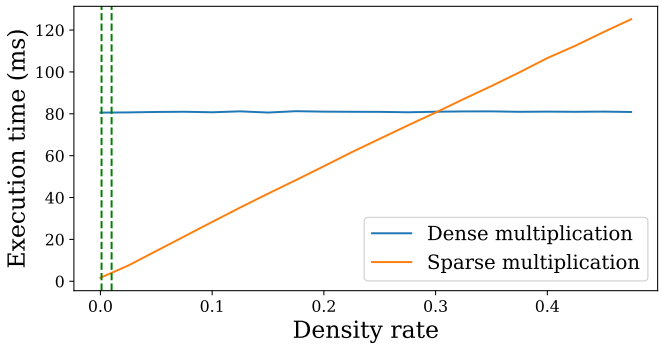


Figure 6: Execution time in milliseconds of multiplying a dense 784-dimensional vector with a sparse 784 by 5000 matrix with varying sparsity, measured on a Raspberry Pi 2B 900Mhz CPU. The green dashed lines represent a density rate of 0.001 and 0.01 (sparsity of 99.9% and 99% respectively) which are used in our models. At a density rate of 30%, the overhead outweighs the speed improvements of the sparse operations.

theoretical reduction in operations does not necessarily result in a reduced execution time on real-world hardware. In Figure 6, we analyze the relationship between sparsity and execution time. We multiply a 784-dimensional vector with a 784×5000 matrix. This is the operation that is performed by the single layer MNIST/FashionMNIST models. We measure the execution time on the Raspberry Pi where either the multiplication is performed using a dense implementation (where most of the values happen to be zero) or using a sparse implementation. As we increase the density rate of the matrix (number of non-zero values = 1-sparsity) we see that the execution time of the sparse multiplication increases accordingly. At a density rate of around 0.3 (70% zero values), the overhead of the sparse implementation already outweighs the benefits . In our models, we however use a much higher sparsity of 99% to 99.9%, as indicated by the green dashed lines. At this sparsity rate, the sparse implementation has a speedup of around 40X compared to the dense implementation. This trade-off however strongly depends on the hardware, a Graphical Processing Unit (GPU) for example, is designed for parallel computations and will not be able to exploit the sparsity at a comparable rate.

4.7. Visualization of learned features

It is often interesting to visualize the information that a machine learning model has learned as this improves the interpretability. If we use a one layer neural network, we can easily reverse the information flow by calculating the dot product between the memory array and the transposed weight matrix of the neural network. The resulting vector has the same number of elements as an input sample. By reshaping this vector into the original shape of the data sample, we obtain a prototype input that would maximize the similarity with the memory array, i.e. the most typical sample. Table 5 shows examples of these images for each class of all three datasets. Although these images are noisy (due to the sparsity and randomness in the neural network), we can clearly see that the model has learned to represent very typical samples, especially for the MNIST and FashionMNIST dataset. The CIFAR10 dataset is more complex but still, the model has learned to encode the typical background color (e.g. blue for planes and ships, green for deer and dog, ...) and the location of the foreground object. We could still calculate the visualization if we use multiple layers but due to the step activation function, the noise level would increase.

5. Evaluation on higher resolution data

While the datasets used in the previous sections are among the most commonly used ones to benchmark anomaly detection algorithms for visual data [7], they are not necessarily representative of real-world applications. In this section, we apply our approach to anomaly detection in food production. We use the Industry Biscuit (Cookie) dataset [89], containing pictures of Tarallini biscuits, taken during production. There are three types of anomalies where the cookie is either incomplete, burnt or where a foreign object such as a screw is visible. Figure 6 shows some example input images. While this is still a relatively constrained setting, it is more challenging than the previous datasets because the images are much larger (640×480 RGB color images) and anomalous cookies can sometimes be detected only from a small region in the image.

In the previous sections, we used the raw pixel values as input to our model. As the images become larger and more complex, this is not feasible anymore and we should perform a feature extraction step first. In this section, we use pretrained deep convolutional neural networks (CNNs) to extract rich features from the input data. This is a commonly used approach that typically outperforms manually designed feature descriptors [90]. Despite being trained on a completely different dataset, these models still extract features that are useful in a variety of tasks. As we use pretrained CNNs that are not updated anymore, the feature extraction does not defeat our goal of online learning.

Table 7 compares the performance of our method and the baseline methods when different feature extraction models are used. These are all commonly used CNNs, trained on the ImageNet dataset [91]. The second column in the table shows the feature size that is extracted by the network, ranging from 512 to 4096 floating point values. The results for our model are obtained using a two-layer neural network with 500 neurons in each layer. As the different feature extraction models return differently sized feature representations, this might skew the results since different models would have a different number of

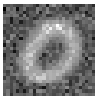
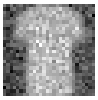
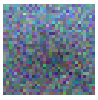
Dataset	0	1	2	3	4	5	6	7	8	9
MNIST										
FashionMNIST										
CIFAR10										

Table 5: Visualization of the learned prototype inputs.

parameters. To allow for a fair comparison, we set the sparsity rate to 99.9% for the Densenet model with a feature size of 1024 and scaled the sparsity rate accordingly for the other models to result in the same number of total calculations.

There is remarkable difference in detection performance between feature extraction models, despite being trained on the same dataset. Overall, we find that the largest models (DenseNet and Resnet152) result in the highest performance. The small EfficientNet models typically result in a lower performance. For the different anomaly detection techniques, we draw similar conclusions as in the experiments above. The PCA approach obtains the highest AUC score while our method performs on-par or slightly below the other baselines. Our method however supports continuous online learning and is computationally less expensive. Other state-of-the-art approaches obtain AUC scores between 83% and 93% on this dataset [92].

We also show some qualitative results in Table 6. The first row shows images that were correctly predicted to have a low anomaly score, we can see that the model has correctly captured this, despite variations in lighting and orientation. The second row shows correctly identified anomalies while the third row shows anomalies that were not detected. These are all of the type where a small part of the cookie is missing. The last row shows images that were incorrectly predicted to be anomalous. These are either slightly distorted or lie close to the border of the image.

6. Evaluation on time series data

In the previous sections, we focused on image data. Visual anomaly detection is an important task with many real-world applications in quality control and product safety. In addition, the visual nature of the data made it easier to intuitively understand the behavior of our model. There are however also many use cases where the input data is in the form of a time series. In this section, we show how our method can also be applied to this domain.

We experimented with the NASA Intelligent Maintenance Systems (IMS) bearing dataset [93]. It was collected on a test rig which consists of an electric motor that drives a shaft mounted with 4 double row Rexnord ZA-2115 bearings. For each bearing, accelerometer data was collected at a rate of 20kHz. The entire lifetime of the bearing is recorded and the task is to detect when the bearing starts to degrade and ultimately fails. The dataset consists of three experiments with four failed bearings in total.

We applied our model in an online fashion to this dataset. Every sample is passed through the model which returns an anomaly score and updates its internal memory. We first transformed the data from the time domain to the frequency domain by applying a Fast Fourier Transform (FFT) to every window of 1 second (20.000 samples). The resulting 10.000 components are then passed through our random sparse anomaly detection model with two layers of 1000 neurons each and a sparsity rate of 90%. The model uses a learning rate of 0.05. To decide when to flag an input window as anomalous, we need to compare the returned anomaly score with a predefined threshold. For this, we calculated the mean (μ) and standard deviation (σ) of the predicted anomaly score of the first 10% of the dataset and set the threshold to ($\mu + 5\sigma$). The intuition behind this is that in the first part of the experiment, the bearing is still in a healthy state. We discard the first 100 predictions in this threshold calculation since the model returns a very high anomaly score for these samples as it is still learning to model the data. Table 8 shows the earliest detected anomalous samples for the four failed bearings, for different state-of-the-art methods. Our approach with sparse random neural networks performs similar or even sometimes slightly better than the existing approaches. A benefit of our approach however is that it does not require a separate training stage as it is trained in an online fashion.

7. Conclusion and future work

In this paper, we presented an anomaly detection technique based on sparse, random neural networks. The sparsity allows for a very efficient implementation on resource constrained devices while the random aspect avoids the need for a separate training stage, allowing the model to be updated continuously on device as new data comes in. We experimentally validated our approach and showed that it achieves competitive anomaly

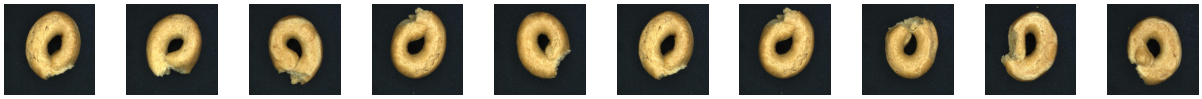
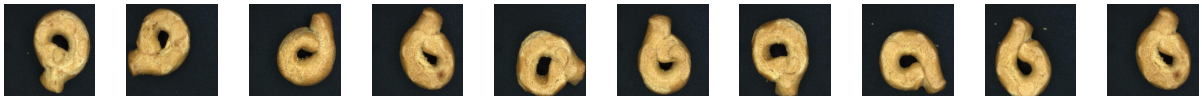
Type	Examples
True Negatives	
True Positives	
False Negatives	
False Positives	

Table 6: Examples of true negatives, true positives, false negatives and false positives, predicted by our approach on the Industry Biscuit dataset.

Table 7: Anomaly detection performance of our method and baseline anomaly detection techniques on the Industry Biscuit dataset where different pretrained deep neural networks are used a feature extractors.

Model	Size	Ours	IF	LOF	OC-SVM	PCA
Alexnet	4096	91.9	91.3	94.0	90.9	**94.9**
Densenet	1024	93.0	94.8	95.9	92.9	**96.9**
Resnet18	512	89.5	90.8	94.2	91.2	**95.4**
Resnet34	512	90.8	90.7	94.8	90.0	**95.9**
Resnet50	2048	91.5	91.0	93.3	90.1	**93.8**
Resnet101	2048	93.5	93.1	96.3	93.3	**97.1**
Resnet152	2048	92.8	91.0	94.0	90.4	**95.4**
VGG16	4096	90.0	88.5	91.3	88.1	**92.1**
Efficientnet b0	1280	86.3	85.6	**88.1**	83.1	86.9
Efficientnet b1	1280	80.7	81.5	85.9	77.3	**89.6**
Efficientnet b2	1408	86.6	86.1	87.6	83.9	**92.2**
Efficientnet b3	1536	87.8	87.1	91.2	87.3	**92.9**
Efficientnet b4	1792	**88.1**	77.5	85.2	81.4	86.8
Efficientnet b5	2048	85.3	83.1	85.8	79.6	**88.0**
Efficientnet b6	2304	84.5	84.7	85.2	80.7	**86.8**
Efficientnet b7	2560	84.3	87.8	90.2	82.1	**92.7**

Table 8: Detected failure points on different trials of the IMS dataset for different methods (lower is better). A single dash indicates that no result on this dataset was reported in the corresponding paper.

Model	S1B3	S1B4	S2B1	S3B3
Ours	2105	**1490**	**533**	6000
DS-PCA-HMM [94]	2120	1508	**533**	-
AEC [95]	2027	1641	547	**2367**
HMM-DPCA [96]	2120	1760	539	-
HMM-PCA [96]	-	1780	538	-
RMS [94]	2094	1730	539	No detection
MAS-Kurtosis [97]	1910	1650	710	No detection
VRCA [96]	-	1727	-	No detection
SVDD [98]	**1820**	-	534	-

detection accuracy on several benchmark data sets. Although our model is unable to achieve the accuracy rates of current state of the art deep learning models, we believe it provides a very good trade-off between computational cost and anomaly detection performance. In addition, we evaluated the capability for online learning and the robustness of the model against noisy training data. In all these experiments, our model compared favorably to other approaches.

There are several aspects that were beyond the scope of this work but that are interesting to investigate further. A first limitation is the MNIST, FashionMNIST and CIFAR10 datasets used here, while they are arguably among the most common datasets used to evaluate anomaly detection systems, making it easy to compare to other approaches, they are not necessarily representative of real world data. In addition, the setting where one class is seen as normal and all other classes as abnormal is somewhat artificial. The data sets from section 5 and 6 are already much more realistic but they are still relatively small and most anomalies are clearly visible. In future work, we will apply our approach to more challenging problems. It is unlikely that the random sparse neural networks will be able to extract complex enough features to model this data. A possible approach would be to first extract high level features like we did in section 5. A more thorough investigation of trade off between feature extraction complexity and model performance would be very interesting.

A second limitation of our work is that we restricted ourselves to a single sensor node. It is unlikely that there will only be a single instance of the sensor. In a factory for example, there will be multiple identical production lines. It would then be interesting the exchange information between the different models that monitor the different lines. Federated or collaborative learning would be able to exploit this to improve the overall anomaly detection accuracy.

A third limitation of our approach concerns requirement two. We argued that continuous adaptation is needed to deal with sensor drift but this also introduces a risk where the model is unable to notice that a machine is failing slowly over time because the model can update itself. It will depend strongly on

the application whether continuous learning is useful or not.

Finally, there is also a limitation concerning the types of input data that are suited for this approach. Because of the sparsity, the model will only use a part of the input. Data types such as images that have some redundancy (the anomaly is never just a single pixel) are a better match than tabular data for example, where a single feature might be anomalous.

We believe the field of TinyML/ edge computing will only gain importance in the next years. Random, sparse neural networks with their excellent computational cost/ accuracy trade-off seem to be a valuable tool in the arsenal of a TinyML practitioner. Further research is however required to theoretically analyze the performance of these models and to investigate whether they can benefit from increased neural network depth, skip connections, convolutional layers or recurrent connections.

Acknowledgments

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

[1] A. Diro, N. Chilamkurti, V.-D. Nguyen, W. Heyne, A comprehensive study of anomaly detection schemes in iot networks using machine learning algorithms, Sensors 21 (24) (2021) 8320.

[2] A. Kanawaday, A. Sane, Machine learning for predictive maintenance of industrial machines using iot sensor data, in: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), IEEE, 2017, pp. 87–90.

[3] M. Benmoussat, M. Guillaume, Y. Caulier, K. Spinnler, Automatic metal parts inspection: Use of thermographic images and anomaly detection algorithms, Infrared Physics & Technology 61 (2013) 68–80.

[4] W. Song, M. Beshley, K. Przystupa, H. Beshley, O. Kochan, A. Pryslupskyi, D. Pieniak, J. Su, A software deep packet inspection system for network traffic analysis and anomaly detection, Sensors 20 (6) (2020) 1637.

[5] B. Li, S. Leroux, P. Simoens, Decoupled appearance and motion learning for efficient anomaly detection in surveillance video, Computer Vision and Image Understanding 210 (2021) 103249.

[6] N. Sarafijanovic-Djukic, J. Davis, Fast distance-based anomaly detection in images using an inception-like autoencoder, in: International Conference on Discovery Science, Springer, 2019, pp. 493–508.

[7] J. Kim, K. Jeong, H. Choi, K. Seo, Gan-based anomaly detection in imbalance problems, in: European Conference on Computer Vision, Springer, 2020, pp. 128–145.

[8] R. Koner, P. Sinhamahapatra, K. Roscher, S. Günnemann, V. Tresp, Oodformer: Out-of-distribution detection transformer, arXiv preprint arXiv:2107.08976 (2021).

[9] C. Luo, A. Shrivastava, Arrays of (locality-sensitive) count estimators (ace) anomaly detection on the edge, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 1439–1448.

[10] S. Leroux, B. Li, P. Simoens, Automated training of location-specific edge models for traffic counting, Computers and Electrical Engineering 99 (2022) 107763.

[11] C. Gallicchio, S. Scardapane, Deep randomized neural networks, Recent Trends in Learning From Data (2020) 43–68.

[12] K. Kawaguchi, B. Xie, L. Song, Deep semi-random features for nonlinear function approximation, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018.

[13] A. Daniely, R. Frostig, Y. Singer, Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity, Advances In Neural Information Processing Systems 29 (2016).

[14] R. Giryes, G. Sapiro, A. M. Bronstein, Deep neural networks with random gaussian weights: A universal classification strategy?, IEEE Transactions on Signal Processing 64 (13) (2016) 3444–3457.

[15] S. Scardapane, D. Wang, Randomness in neural networks: an overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7 (2) (2017) e1200.

[16] B. Igelnik, Y.-H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE transactions on Neural Networks 6 (6) (1995) 1320–1329.

[17] R. Katuwal, P. N. Suganthan, M. Tanveer, Random vector functional link neural network based ensemble deep learning, arXiv preprint arXiv:1907.00350 (2019).

[18] M. Alhamdoosh, D. Wang, Fast decorrelated neural network ensembles with random weights, Information Sciences 264 (2014) 104–117.

[19] D. Wang, M. Li, Stochastic configuration networks: Fundamentals and algorithms, IEEE transactions on cybernetics 47 (10) (2017) 3466–3479.

[20] A. Malik, R. Gao, M. Ganaie, M. Tanveer, P. Suganthan, Random vector functional link network: recent developments, applications, and future directions, arXiv preprint arXiv:2203.11316 (2022).

[21] D. Ulyanov, A. Vedaldi, V. Lempitsky, Deep image prior, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 9446–9454.

[22] A. Rosenfeld, J. K. Tsotsos, Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing, in: 2019 16th Conference on Computer and Robot Vision (CRV), IEEE, 2019, pp. 9–16.

[23] S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, R. Wang, On exact computation with an infinitely wide neural net, Advances in Neural Information Processing Systems 32 (2019).

[24] L. Du, How much deep learning does neural style transfer really need? an ablation study, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 3150–3159.

[25] J. Pons, X. Serra, Randomly weighted cnns for (music) audio classification, in: ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2019, pp. 336–340.

[26] B. Schrauwen, D. Verstraeten, J. Van Campenhout, An overview of reservoir computing: theory, applications and implementations, in: Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007, 2007, pp. 471–482.

[27] H. Jaeger, H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, science (2004).

[28] W. Maass, H. Markram, On the computational power of recurrent circuits of spiking neurons. submitted for publication (2002).

[29] P. Tino, G. Dorffner, Predicting the future of discrete sequences from fractal representations of the past, Machine Learning 45 (2) (2001) 187–217.

[30] M. Bauw, S. Velasco-Forero, J. Angulo, C. Adnet, O. Airiau, Deep random projection outlyingness for unsupervised anomaly detection, arXiv preprint arXiv:2106.15307 (2021).

[31] P. Navarro-Esteban, J. A. Cuesta-Albertos, High-dimensional outlier detection using random projections, TEST 30 (4) (2021) 908–934.

[32] S. M. Erfani, M. Baktashmotlagh, S. Rajasegarar, S. Karunasekera, C. Leckie, R1svm: a randomised nonlinear approach to large-scale anomaly detection, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 29, 2015.

[33] T. de Vries, S. Chawla, M. E. Houle, Density-preserving projections for large-scale local anomaly detection, Knowledge and information systems 32 (1) (2012) 25–52.

[34] K. Friston, Hierarchical models in the brain, PLoS computational biology 4 (11) (2008) e1000211.

[35] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning (still) requires rethinking generalization, Communications of the ACM 64 (3) (2021) 107–115.

[36] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. De Freitas, Predicting parameters in deep learning, Advances in neural information processing systems 26 (2013).

[37] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[38] Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, J. E. Gonzalez, Train large, then compress: Rethinking model size for efficient training

and inference of transformers, arXiv preprint arXiv:2002.11794 (2020).
[39] B. Bartoldson, A. Morcos, A. Barbu, G. Erlebacher, The generalization-stability tradeoff in neural network pruning, Advances in Neural Information Processing Systems 33 (2020) 20852–20864.
[40] J. Cosentino, F. Zaiter, D. Pei, J. Zhu, The search for sparse, robust neural networks, arXiv preprint arXiv:1912.02386 (2019).
[41] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, Journal of Machine Learning Research 22 (241) (2021) 1–124.
[42] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, M. Jaggi, Dynamic model pruning with feedback, arXiv preprint arXiv:2006.07253 (2020).
[43] M. Hagiwara, Removal of hidden units and weights for back propagation networks, in: Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), Vol. 1, IEEE, 1993, pp. 351–354.
[44] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, Advances in neural information processing systems 28 (2015).
[45] M. DSTO, Neural net pruning-why and how.
[46] Y. Sun, X. Wang, X. Tang, Sparsifying neural network connections for face recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4856–4864.
[47] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, Advances in neural information processing systems 2 (1989).
[48] B. Dai, C. Zhu, B. Guo, D. Wipf, Compressing neural networks using the variational information bottleneck, in: International Conference on Machine Learning, PMLR, 2018, pp. 1135–1144.
[49] V. Sanh, T. Wolf, A. Rush, Movement pruning: Adaptive sparsity by fine-tuning, Advances in Neural Information Processing Systems 33 (2020) 20378–20389.
[50] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, B. Li, Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights, Proceedings of the IEEE 109 (10) (2021) 1706–1752.
[51] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, ACM Computing Surveys (CSUR) 54 (2) (2021) 1–38.
[52] K. Doshi, Y. Yilmaz, Rethinking video anomaly detection-a continual learning approach, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2022, pp. 3961–3970.
[53] A. Stocco, P. Tonella, Towards anomaly detectors that learn continuously, in: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2020, pp. 201–208.
[54] K. Doshi, Y. Yilmaz, Continual learning for anomaly detection in surveillance videos, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, 2020, pp. 254–255.
[55] S. Leroux, B. Li, P. Simoens, Multi-branch neural networks for video anomaly detection in adverse lighting and weather conditions, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2022, pp. 2358–2366.
[56] M. Du, Z. Chen, C. Liu, R. Oak, D. Song, Lifelong anomaly detection through unlearning, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1283–1297.
[57] Y. Meidan, D. Avraham, H. Libhaber, A. Shabtai, Cadesh: Collaborative anomaly detection for smart homes, IEEE Internet of Things Journal (2022) 1Publisher Copyright: Author. doi:10.1109/JIOT.2022.3194813.
[58] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 604–613.
[59] T. N. Mau, V.-N. Huynh, An lsh-based k-representatives clustering method for large categorical data, Neurocomputing 463 (2021) 29–44.
[60] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, C. Crushev, A survey on locality sensitive hashing algorithms and their applications, arXiv preprint arXiv:2102.08942 (2021).
[61] Y. Wang, S. Parthasarathy, S. Tatikonda, Locality sensitive outlier detection: A ranking driven approach, in: 2011 IEEE 27th International Conference on Data Engineering, IEEE, 2011, pp. 410–421.
[62] Y. Zhang, H. Lu, L. Zhang, X. Ruan, S. Sakai, Video anomaly detection based on locality sensitive hashing filters, Pattern Recognition 59 (2016) 302–311.
[63] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Transactions on Knowledge Discovery from Data (TKDD) 6 (1) (2012) 1–39.
[64] H. Hachiya, M. Matsugu, Nsh: normality sensitive hashing for anomaly detection, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2013, pp. 795–802.
[65] A. Rahimi, B. Recht, Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning, Advances in neural information processing systems 21 (2008).
[66] S. Srinivas, A. Subramanya, R. Venkatesh Babu, Training sparse neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2017.
[67] U. Evci, F. Pedregosa, A. Gomez, E. Elsen, The difficulty of training sparse neural networks, arXiv preprint arXiv:1906.10732 (2019).
[68] T. Simons, D.-J. Lee, A review of binarized neural networks, Electronics 8 (6) (2019) 661.
[69] S. Leroux, B. Vankeirsbilck, T. Verbelen, P. Simoens, B. Dhoedt, Training binary neural networks with knowledge transfer, Neurocomputing 396 (2020) 534–541.
[70] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, N. Sebe, Binary neural networks: A survey, Pattern Recognition 105 (2020) 107281.
[71] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[72] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).
[73] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).
[74] Z. Zhang, S. Chen, L. Sun, P-kdgan: Progressive knowledge distillation with gans for one-class novelty detection, arXiv preprint arXiv:2007.06963 (2020).
[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[76] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, H. Chen, Deep autoencoding gaussian mixture model for unsupervised anomaly detection, in: International conference on learning representations, 2018.
[77] S. Akcay, A. Atapour-Abarghouei, T. P. Breckon, Ganomaly: Semi-supervised anomaly detection via adversarial training, in: Asian conference on computer vision, Springer, 2018, pp. 622–637.
[78] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, G. Langs, Unsupervised anomaly detection with generative adversarial networks to guide marker discovery, in: International conference on information processing in medical imaging, Springer, 2017, pp. 146–157.
[79] L. Deecke, R. Vandermeulen, L. Ruff, S. Mandt, M. Kloft, Image anomaly detection with generative adversarial networks, in: Joint european conference on machine learning and knowledge discovery in databases, Springer, 2018, pp. 3–17.
[80] P. Perera, R. Nallapati, B. Xiang, Ocgan: One-class novelty detection using gans with constrained latent representations, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2898–2906.
[81] H. Hojjati, N. Armanfard, Dasvdd: Deep autoencoding support vector data descriptor for anomaly detection, arXiv preprint arXiv:2106.05410 (2021).
[82] I. Golan, R. El-Yaniv, Deep anomaly detection using geometric transformations, Advances in neural information processing systems 31 (2018).
[83] C. Huang, J. Cao, F. Ye, M. Li, Y. Zhang, C. Lu, Inverse-transform autoencoder for anomaly detection (2019).
[84] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, M. Kloft, Deep one-class classification, in: International conference on machine learning, PMLR, 2018, pp. 4393–4402.
[85] W. Hu, M. Wang, Q. Qin, J. Ma, B. Liu, Hrn: A holistic approach to one class learning, Advances in Neural Information Processing Systems 33 (2020) 19111–19124.
[86] P. Schlachter, Y. Liao, B. Yang, Deep one-class classification using intra-class splitting, in: 2019 IEEE Data Science Workshop (DSW), IEEE, 2019, pp. 100–104.
[87] M. H. Mofrad, R. Melhem, Y. Ahmad, M. Hammoud, Multithreaded layer-wise training of sparse deep neural networks using compressed

sparse column, in: 2019 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2019, pp. 1–6.

[88] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[89] Industry Biscuit (Cookie) dataset, howpublished = `https://www.kaggle.com/datasets/imonbilk/industry-biscuit-cookie-dataset`, note = Accessed: 2022-05-14.

[90] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, O. Wang, The unreasonable effectiveness of deep features as a perceptual metric, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 586–595.

[91] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[92] S. Bilik, K. Horak, Sift and surf based feature extraction for the anomaly detection, arXiv preprint arXiv:2203.13068 (2022).

[93] This data come from National Aeronautics and Space Administration Website. [link].
URL `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/#bearing`

[94] A. Rehab, I. Ali, W. Gomaa, M. N. Fors, Bearings fault detection using hidden markov models and principal component analysis enhanced features, arXiv preprint arXiv:2104.10519 (2021).

[95] R. M. Hasani, G. Wang, R. Grosu, An automated auto-encoder correlation-based health-monitoring and prognostic method for machine bearings, arXiv preprint arXiv:1703.06272 (2017).

[96] J. Yu, Health condition monitoring of machines based on hidden markov model and contribution analysis, IEEE Transactions on Instrumentation and Measurement 61 (8) (2012) 2200–2211.

[97] S. Kim, S. Park, J.-W. Kim, J. Han, D. An, N. H. Kim, J.-H. Choi, A new prognostics approach for bearing based on entropy decrease and comparison with existing methods, in: Annual Conference of the PHM Society, Vol. 8, 2016.

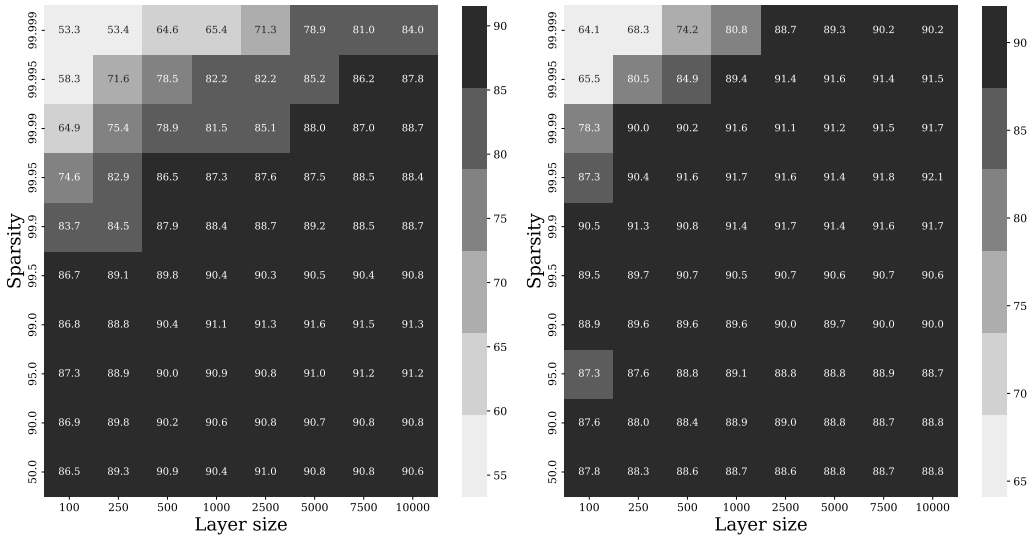
[98] C. Liu, K. Gryllias, A semi-supervised support vector data description-based fault detection method for rolling element bearings based on cyclic spectral analysis, Mechanical Systems and Signal Processing 140 (2020) 106682.

Appendix A. Source code

```python
import numpy as np
from scipy import sparse

class RandomSparseAnomalyDetector:
    def __init__(self, input_size, layer_sizes, layer_sparsities):
        self.weights = []

        # Initialize the weights randomly
        previous_size = input_size
        for (size, sparsity) in zip(layer_sizes, layer_sparsities):
            w = np.random.uniform(low=-1, high=1, size=(size, previous_size))
            w *= np.random.choice([0, 1], size=w.shape, p=[sparsity, 1-sparsity])

            # Convert to sparse representation
            w = sparse.csc_matrix(w)
            self.weights.append(w)

            previous_size = size

        # Initialize the memory with zero values
        self.memory = np.zeros(layer_sizes[-1])


    # Evaluates a single layer of the neural network
    def layer(self, x, w):
        # Perform sparse vector-matrix multiplication
        out = w.dot(x)

        # Binarize the output vector
        out = (out > 0).astype(np.int_)*2-1
        return out

    # Predicts the anomaly score and updates the model
    def forward(self, x, lr=0.0):
        # Reshape the input to one dimension
        x = x.flatten()

        # Evaluate all layers
        for w in self.weights:
            x = self.layer(x, w)

        # Calculate the anomaly score
        score = (x*self.memory).sum()

        # Update the memory
        self.memory = lr * x + (1.0 - lr) * self.memory

        return score
```

Appendix B. Additional results

Appendix B.1. Impact of sparsity



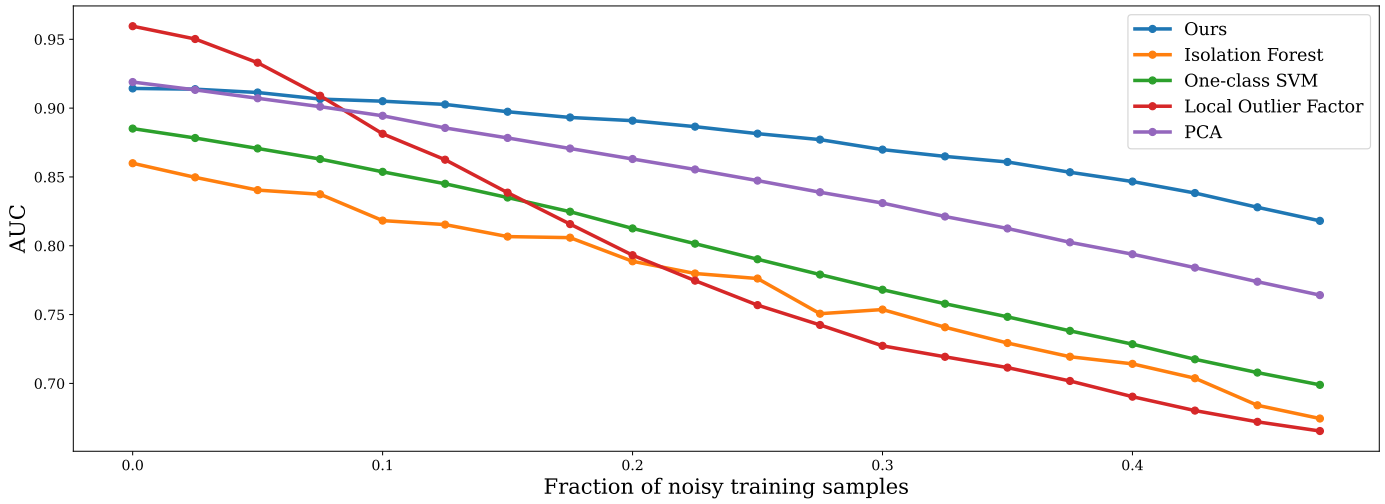
(a) MNIST

(b) FashionMNIST

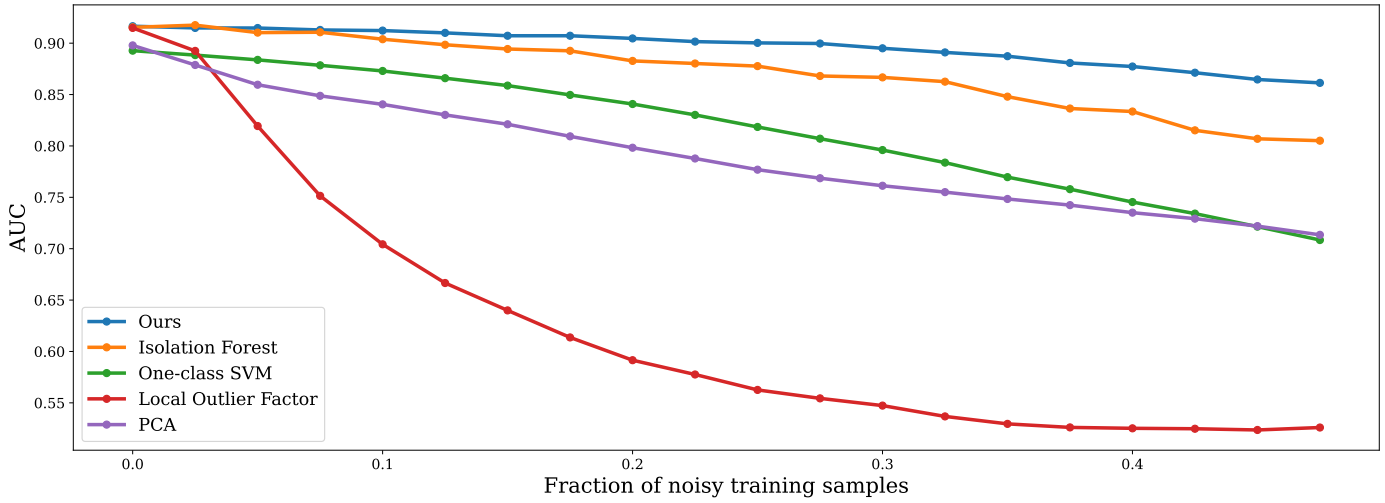
(c) CIFAR10

Figure B.7: Average AUC score over all classes for the MNIST (a), FashionMNIST (b) and CIFAR10 (c) dataset, using a one layer model with a varying number of neurons and corresponding sparsity rate. See section 4.2 for more details.

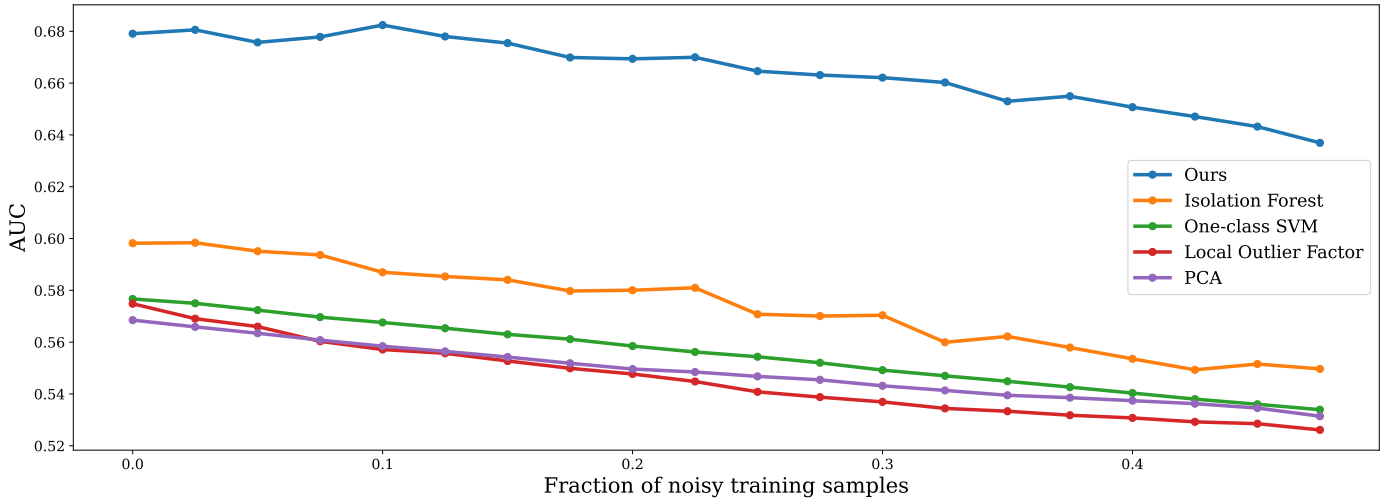
Appendix B.2. Robustness against noisy training data



(a) MNIST

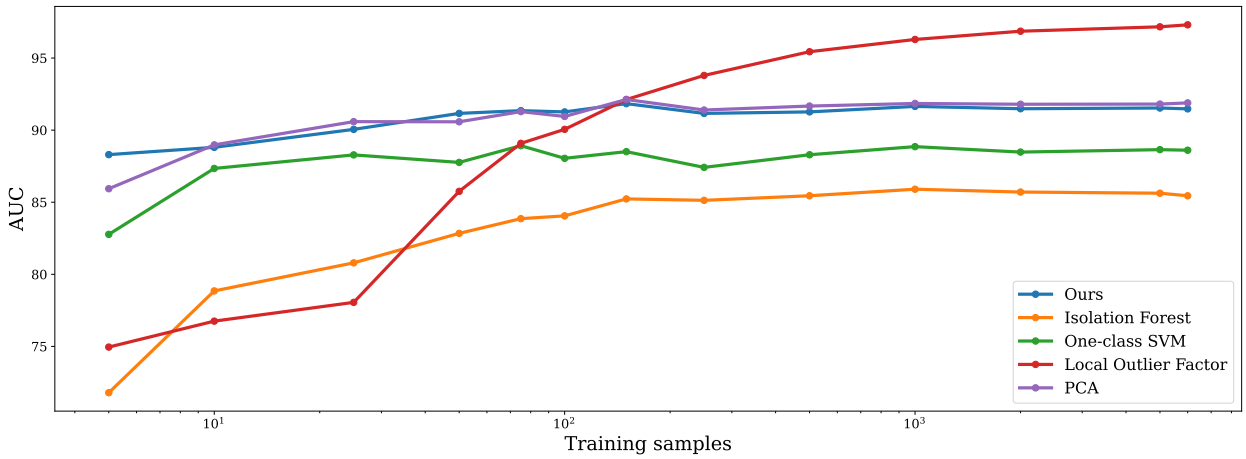


(b) FashionMNIST

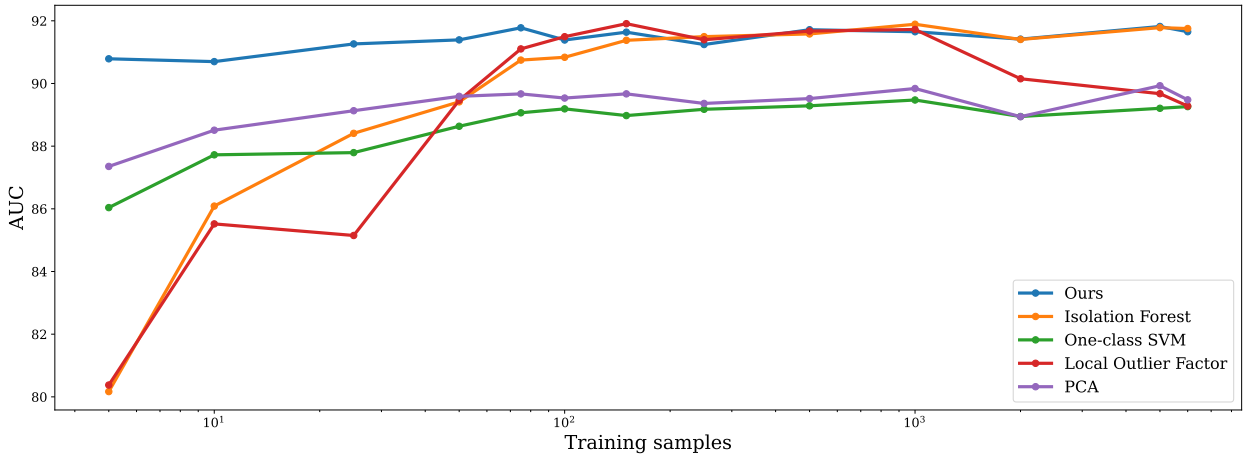


(c) CIFAR10

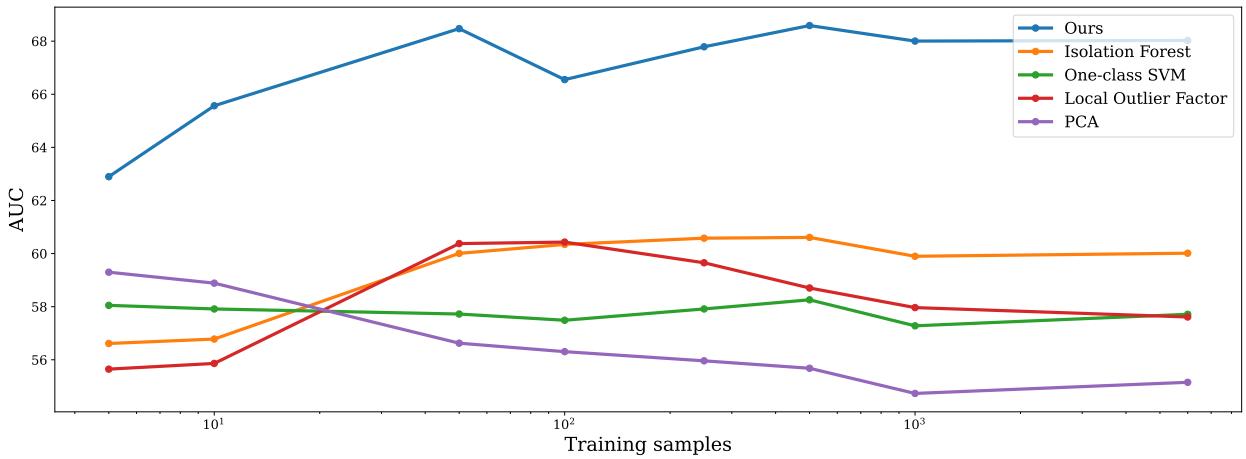
Figure B.8: Average AUC score over all classes for the MNIST (a), FashionMNIST (b) and CIFAR10 (c) dataset, using the models from table 1, trained on training data with an increasing number of anomalies. See section 4.4 for more details.



(a) MNIST



(b) FashionMNIST



(c) CIFAR10

Figure B.9: Average AUC score over all classes for the MNIST (a), FashionMNIST (b) and CIFAR10 (c) dataset, using the models from table 1, trained on an increasing number of training samples. See section 4.5 for more details.