A Hybrid Autoencoders and Self-Organizing Maps for IoT Malware Detection

Huu Noi Nguyen[1], Van Cuong Nguyen[1],
Van Loi Cao[1], and Nguyen Ngoc Tran[1]

Le Quy Don Technical University
https://www.lqdtu.edu.vn/
{noi.nguyen, cuongpd, loi.cao, ngoctn}@lqdtu.edu.vn

Abstract. In this paper, we study the combination of Autoencoder (AE) and Self-Organizing Maps (SOM) for feature extraction and detecting IoT attacks. The AE learns latent representation to reveal useful features for clustering algorithms, e.g SOM. The AE can find and extract the most valuable features from data while SOMs can self-organize and map input instances to a neuronal map. Neurons are categorized in a suitable group that has a specific label. This label-map is used to classify normal and anomalous later. We compared the AE+SOM with pure SOM and other dimensionality reduction (PCA+SOM) on the NBaIoT dataset. Our study focuses on the ability of the proposed algorithm on detecting unknown attacks and on transfer learning. In all cases, the final results showed that the AE+SOM is better than other algorithms. The exported model can be applied in the real system for anomaly detection.

Keywords: SOM · AE · PCA · IoT · anomaly detection · transfer learning

1 Introduction

IoT technology plays an important role in enhancing smart applications in real life, such as smart healthcare, smart home, smart transportation, and smart education [7,22]. Therefore, the diverse and large-scale nature of IoT systems with different components involved in the implementation of such systems presents new security challenges [1]. IoT attack methods are increasingly and diverse, such as on-board attack, security gateway attack, control server attack, eavesdropping attack on incoming traffic detection. Therefore, the risk of insecurity in IoT networks is higher than in other networks, and traditional solutions may not be effective for such systems [16]. One of the prominent attack examples is the attack using BotNet Mirai. Mirai is a special type of botnet that has recently caused large-scale DDoS attacks by exploiting vulnerabilities in IoT devices [16].

Machine learning (ML), particularly Deep learning (DL) are powerful methods and are currently widely used in many different problems such as classification and detection of malicious code, detection of network attacks, predicting

the anomalous behavior of network data to give early warnings [13, 25]. With the problem of anomaly detection, ML/DL can rely on training datasets to build models; then these models will be applied to classify network data streams as "normal" or "anomaly" [20,27,28]. In addition, ML/DL methods can be applied in predicting new attacks, often mutations of previous attacks. Currently, federated learning is also used for IoT networks, to accommodate the dispersion of IoT network devices [20].

One of the algorithms in IoT malware detection system is autoencoders (AE). It was introduced by Japkowicz et al. [12] in 1995. The AE was applied for novelty detection at that time. An autoencoder is a neural network that learns to reconstruct its input at the output layer. A bottleneck layer compresses redundancies in the input data while non-redundant information remains [12]. We used AEs as building blocks in deep neural networks [11], and after training, the output layer is discarded, and the hidden layer is used as a new feature representation.

Currently, almost all learning methods perform learning from one class, usually learning from normal data [4, 27], and then proceed to classify or forecast the data into normal or abnormal. Almost all algorithms in these researches are based on AE. But AE usually learns and represents well for a single distribution. It means AE extremely learns from one class in supervised learning manner. But in unsupervised learning manner, when the distributions are multiple, and the number of records is different, then AE will learn most features from the majority class (majority class occupies most samples in the dataset) and the latent is mostly features learned from this class.

In the other manner, PCA tries to project data into a new coordinate system so that the data are much separated as possible [30]. It means, PCA keeps all the features, which make the data are most separable. So it the data is more separated, so PCA is more accurately represented. On the contrary, if data are the same or mixed so the PCA is not good for representing hidden features.

In this research, we consider the influence of the AE on the performance of anomaly detection when both normal and abnormal data were used for training. We combine AE and SOM to leverage the power of the AE in useful features extraction and SOM in clustering of normal and abnormal data. We also tested different ratio data to assess the effectiveness of the algorithm. We assume that when the data ratio is skewed, for example, the normal data is much more than the abnormal or vice versa, then the AE will learn most features from the major class, and the minor class will not take effect in the common results. In experiments (which are presented in more detail in Section 5 and 6) we performed tests in different parts of the dataset to verify this assumption. Because the other researches [4, 27] is quite good for attack detection (or even unknown attack detection), so in our experiments, we focus on transferring our trained model from one device with specific attack to another device and other attacks.

In this paper, our main contributions are as follows:

- We use the latent representation of Autoencoders (AE) combined with Self-Organizing Maps (SOM) to build the unsupervised learning models for detecting IoT attacks.

- We perform a lot of tests on different datasets with different scenarios. The data ratio is used in a wide range to assess the effectiveness of malware detection of the model
- We apply our model in transfer learning. The model is trained on one device and can be used to detect attacks (same type or different type) on other devices. This saves time and resources for the training process.

The paper is organized as follows. In Section 1 we highlight the importance of anomaly detection and raised the problem of unsupervised learning methods. Section 2 presents some related works. The background of the research is described in Section 3. Our main proposed model for unsupervised learning detection system is presented in Section 4. All description of the dataset, experiments and evaluation is shown in Section 5. Section 6 is about results and discussion. We draw some conclusion in Section 7. Lastly, some additional testing results and figures are placed in Appendix A.

2 Related Works

In this section, we discuss recent works which are used for detecting anomalies in the IoT field. Most of the algorithms used are based on supervised learning methods, we discuss the use of autoencoders (AEs) in particular. Next, we also carried out some researches that tend to solve the classification problem using the unsupervised technique.

The AE-based methods have been widely used as a feature learner for latent representation [2, 3, 5, 8, 10, 18, 23, 26]. This latent is later used for anomaly detection models. The latent feature representation can be learned in different manners like supervised learning [18,26], semi-supervised learning [5], and unsupervised learning [8, 10]. The common way is as follow, all methods are trained on the training data (usually one-class learning), and when the training process is done, only the encoder is used for further stages. This encoder tries to learn the latent features from the input, and this latent is next used in the detection models constructed by traditional machine learning methods. Since the dimension of latent data is decreased and much less than that of the input data, then the classification models are mostly faster than previous.

Recently Cao et al. [5] introduced two regularized AEs, namely SAE and DVAE for capture the normal behaviors of network data. These regularizers AEs are attempted to put normal data towards a small region at the origin of the latent feature space, which can result in reserving the rest of the space for anomalies occurring in the future. These regularized AEs were designed to overcome the problem of identifying anomalies in high-dimensional network data. The latent representation of SAE and DVAE was then used for enhancing simple one-class classifiers. In a supervised manner, Vu et al. [26] proposed Multi-distribution VAE (MVAE) to represent normal data and anomalous data into two different regions in the latent feature space of VAE. Originally, variational autoencoders (VAEs) learn to map input data into a standard Gaussian distribution $\mathcal{N}(0, 1)$

in its middle hidden layer. The proposed model was evaluated on two publicly network security datasets, and it produces promising performance.

In unsupervised manner, Gustavo et. al. [6] explore the power of SOM for multi-label classification. Since the SOM has ability to map input instances to a map of neurons. After performing SOM, similar instances are grouped in the same class. Also using SOM, Andreas Rauber et. al. [21] presented the LabelSOM approach, which can automatically label and train the SOM with the features of the most relevant input data in a particular cluster. In [24] J. Tian et. al. proposed a method that can improve SOM for anomaly detection. For a given test data, all the neighbors are identified by using the k-nearest neighbor algorithm. The Euclidean distance was used to measure the distance between the test data observation and the centroid of the neighbors.

In this work, we attempt to investigate the latent representation of AEs for IoT malware detection tasks in an unsupervised manner. This means that we use AEs to learn a latent representation for both normal and IoT malware without labels. The latent representation is later facilitated by SOM for discovering clusters.

3 Background

3.1 Autoencoder

An autoencoder is an unsupervised learning algorithm based on neural network architecture. An AE is a feed-forward neural network that attempts to reconstruct the original input data at the output layer. The traditional AE is used for dimensionality reduction and feature learning. The AE architecture shows in 3.1 consists of three parts: the encoder, the code (bottleneck) and the decoder.

The hidden layer h that described a code used to represent the input [9]. An encoder function f is used to learn the input and represented as **code**, the decoder g is used to reconstruct the data from the encoded representation.

Mathematically, given data x with no-labels and the function f for encoder and function g for decoder. Then we have the following equations:

$$z = f(x) = a_e(wx + b) \tag{1a}$$

$$\hat{x} = g(z) = g(f(x)) = a_d(w'.f(x) + b') \tag{1b}$$

where a_e and a_d are the activation functions of the encoder and decoder, $\hat{x}$ is x's reconstruction.

The reconstruction loss function (e.g. squared loss error) is to minimize the difference between the input x and the output $\hat{x}$.

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 \tag{2}$$

The AEs have many applications, such as image compression, image denoising, feature extraction, image generation, sequence to sequence prediction and recommendation systems.

3.2 Principle Component Analysis

Dimensionality reduction is one of the important techniques in Machine Learning. Dimensionality reduction, to simplifying, is finding a function, which takes the input of an initial data point $x \in R^D$ with D is high and creates a new data point $x \in R^K$, which has a dimension number $K < D$.

One of the simplest dimensionality reduction algorithms is based on a linear model. This method is called Principal Component Analysis (PCA) [30]. This method is based on the observation that the data are not normally distributed randomly in space, but are often distributed near-certain special lines/faces. PCA considers a special case when those special faces are linear in sub-spaces. Some modern PCA algorithms are Linear PCA, Kernel PCA, Sparse PCA, Non-linear PCA, Robust PCA.

3.3 Self-Organizing Maps

The Self-Organizing Maps (SOM) [14,15] or Kohonen maps are self-organizing neural networks that are able to map similar instances to a group in a map, then each neuron is placed next to each other. This map provides a mapping from high-dimensional input space to lower-dimensional output space (usually two-dimensional). SOMs apply competitive learning as opposed to error-correction learning (such as back-propagation with gradient descent), and they use a neighborhood function to preserve the topological properties of the input space. In other words, competitive learning is an unsupervised learning method, and it is most suitable to illustrate the appropriateness of learning from a single-layer neural network.

SOMs have been successfully applied in a number of fields, such as identification, data clustering and text prediction. The data types include sound, image and text. More details of SOMs will be described in the section 4.2.

3.4 Transfer Learning

Transfer learning is the application of skills/knowledge learned from one problem (source domain - D_S), with specific application (source task - T_S) to another problem (target domain - D_T) with another application (target task - T_T) which is relevant. Transfer learning aims to improve the learning of the function $f_T(\cdot)$ for the application T_T on the domain D_T [29].

4 The proposed hybrid AEs and SOMs

This section presents our proposed method for IoT anomaly detection. It is a hybrid between AEs and SOMs, consisting of two phases as shown in Fig. 1:

Phase 1: We employ AEs to construct a new feature space from unlabeled data. The new feature is in lower dimensionality and is expected to reveal more robust features. Other feature reduction methods like PCA are also involved for comparison.

Phase 2: SOMs are presented on the resulting feature of AEs. The trained data will help SOMs find and label clusters associated with normal data and IoT malware. These cluster labels are used for classifying.

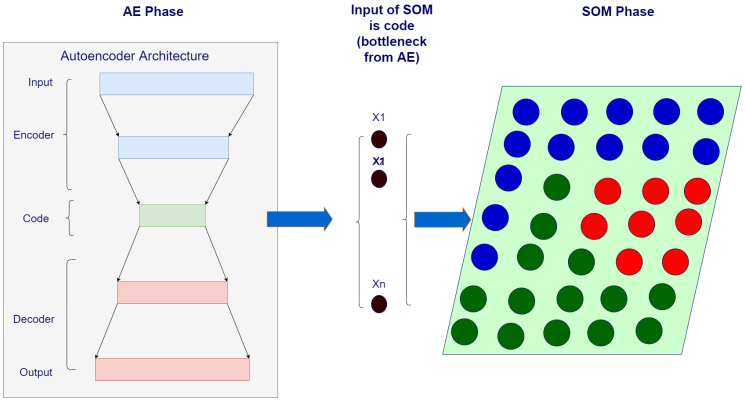


Fig. 1: The system architecture

4.1 Latent representation of AEs

We aim to find a representation for the original data, which is in a lower dimension but reserves most of the important information as the original data. As described in section 3, the AE is a neural network that attempts to reconstruct the input from the bottleneck. There are many versions developed from the original AE, in this work we use the simple AE with a 3-layer of an encoder and a 3-layer of a decoder. The AE architecture is shown in the left phase in Fig. 1.

The input data is passed in autoencoder model for training. The process is terminated when the output layer is constructed similar to the input. It means the reconstruction error was minimized. After training the AEs, the decoder part is discarded, while the encoder part is kept and used to extract features from the original.

4.2 Anomaly Detection using SOM

Mapping Process On the training SOM, the winning neuron is archived using Euclidean distance, which is represented in Equation 3.

$$d_j(x) = \sqrt{\sum_{i=1}^{A}(x_i - w_{ji})^2} \tag{3}$$

where x is the attribute vector of the instance and w_j is the weight vector of the j^{th} neuron, A is the number of attributes of an instance.

When the winning neuron is obtained, its weights will be adjusted to approximate it to the instance. Since then, a map of its neighborhood is defined. The process is continued as follow: the weights of each neighborhood also is updated, and approximate to the winning neuron; a good choice for finding for the neighborhood is using Gaussian function 4, where $h_{j,i}$ is the neighborhood of the winning neuron i, while j is the older winning neuron, the distance $d_{j,i}$ is a distance between neurons, the σ defines the spreading of neighborhoods. [6].

$$h_{j,i} = exp(-\frac{d_{j,i}^2}{2\sigma^2}) \tag{4}$$

The weight update process is given by Equation 5, where w_j is the weight vector, x is input instance, η is a learning rate.

$$\Delta w_j = \eta h_{j,i}(x - w_j) \tag{5}$$

Finally, the weight vector at iteration $(t+1)$ is updated by Equation 6.

$$\Delta w_j(t+1) = w_j(t) + \eta h_{j,i}(x - w_j(t)) \tag{6}$$

The training process of the algorithm is shown in Algorithm 1. Firstly, the weight matrix is initialized by generating randomly or using PCA method. Secondly, the winning neuron is selected from the neuron grid Ω by using the distance metric (e.g. Euclidean). Finally, the weights of all related neurons will be updated using Equation 6 [6]. The process is terminated when the network is converged, and the weights in the map might have the same distribution as the input vectors. It means, after the finite number of iterations, the inputs are placed in appropriate positions in the Kohonen network (another name of Self-organizing maps).

Algorithm 1 The SOM algorithm

Input: $X = [q, (a + l)]$, e: number of epochs
Output: $W = [n, a]$
Main loop
  for $i \leftarrow 1$ to e **do**
    Initialize weight matrix W
    for $j \leftarrow 1$ to q **do**
      $o(x_j) = argmin_k \|x_j - w_k\|$, $k \in \Omega$
      $w_k(i+1) = w_k(i) + \eta(i) h_{k,o(x_j)}(i)(x_j(i) - w_k(i))$
    end for
  end for
  return W

In the Algorithm 1, X is a dataset, q is a number of instances, a is a number of attributes, l is a number of labels and n is a number of neurons.

Classification Algorithm Given instance x_i, and the mission is to classify this instance into an appropriate class. Firstly, all classes are represented by binary vector v_i. The j^{th} position in this vector is corresponding to the j^{th} class (c_j). If the instance x_i is a member of class c_j, then the $v_{i,j}$ has value 1, and 0 vice versa [6].

The test instance is mapped to labels-map and after its closest neurons are found, one vector (called prototype vector) is obtained by averaging the class vectors of the training instances mapped to this neuron and notated as $\bar{v}$. It is the possibility of classifying the test instance into each class. The formulation of the prototype vector is shown in Equation 7. The S_n is the training set which mapped to neuron n, while the $S_{n,j}$ is the training instances which mapped to the neuron n and classified to class c_j.

$$\bar{v}_{n,j} = \frac{S_{n,j}}{S_n} \tag{7}$$

To map instance n to class c_j, the $\bar{v}_{n,j}$ is compared to the *threshold*. We set the value of 0.5 for the *threshold*. Then all the positions whose values are greater than or equal to 0.5 receives the value 1, and 0 otherwise.

The algorithm of classifying the instance using the SOM model is shown in Algorithm 2 [6]. The sample is classified using the labels-map which received after training SOM (Algorithm 1). A label c_j is assigned to the neuron is the majority of samples mapped in that neuron have label c_j. The algorithm function will assign the most common label in the dataset in case that no class is suited for this neuron. The steps are as follow:

Step 1: The winning neuron is selected from the neuron grid Ω;

Step 2: Get training instances mapped to winning neuron;

Step 3: Calculate prototype vector;

Step 4: Compare the prototype vector to *threshold*;

Step 5: Get the appropriate class.

Algorithm 2 The SOM-based classification algorithm

Input: $X^{train} = [q, (a + l)],\ W = [n, a]$
Output: P
Main loop
  for $j \leftarrow 1$ to m **do**
    $o(x_j^{test}) = argmin_k \|x_j^{test} - w_k\|,\ k \in \Omega$
    $T \leftarrow$ instances mapped to $o(x_j^{test})$;
    $\bar{v}_j \leftarrow$ average of the label vectors from T;
    $x_j^{test} \leftarrow x_j^{test} + \bar{v}_j$
    $p_j \leftarrow \bar{v}_j$
    p_j compares to *threshold*
  end for
  return P

Where q is the number of training instances, a is the number of attributes dataset, l is the number of labels, m is the number of instances in the testing set, W is the weight matrix and P is the prediction matrix.

5 Experiments

In this section, we present the experiment settings and evaluation of three models AE+SOM, SOM and PCA+SOM for classifying the IoT attacks. We use SOM as an unsupervised method for classifying, PCA and AE as dimensionality reduction methods for data preprocessing. The detailed description of the dataset, parameter settings for experiments, and metrics used to evaluate our proposed methods are presented in the rest of this section.

In experiments, we conducted assessing the ability of proposed model on detecting *known attack*, *unknown attack* and *transfer learning*. As described in Section 1, on the results analysis in Section 6, we focus on *unknown attack* and *transfer learning*.

Ability to detect unknown attack types. There are two types of attacks in the dataset (Mirai and Gafgyt) in each device (exclude device D3 and device D7, which contains only benign and Gafgyt). So our idea is to train the algorithm in one type of data and apply it later in other types.

Transfer learning. In our experiments, we used one device for learning, and test on the type of attack in the same device and on the other devices.

5.1 Datasets

The NBaIoT dataset [1] was introduced by Y. Meidan et al. [17]. It contains data samples collected from nine different IoT devices, as described in Table 1. For each device, the two most popular kinds of attacks are launched such as Mirai and BASHLITE (or Gafgyt) for generating Malware data together with benign data. These devices can be categorized into four main groups: doorbell, thermostat, monitor and camera/webcam. Each record in the dataset consists of 115 features extracted by using Kitsune [19].

In experiments, we employed two groups of devices: doorbell (D1 and D3 in the table 1) and camera (D5, D6 and D8 in the table 1). For the training phase, we used only one type of attack on each device. The resulting models were then utilized to evaluate other attack types on the same device and on different devices.

We carried out groups of scenarios, which are shown in Table 2.

Since both devices D3 and D7 series have only one device, we have not included them in our experiments at this time.

In this phase, we divided the train data into two parts: train and validation with a ratio 0.7 for training and 0.3 for validation. We also used Early-Stopping to early terminate the training process when the result reaches an acceptable value.

[1] https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT

Table 1: The dataset description

Device ID	Device Name	Type	Benign	Gafgyt	Mirai
D1	Danmini_Doorbell	Doorbell	49548	652100	316650
D2	Ecobee_Thermostat	Thermostat	13113	512133	310630
D3	Ennio_Doorbell	Doorbell	39100	316400	
D4	Philips_B120N10_Baby_Monitor	Monitor	175240	312273	610714
D5	Provision_PT_737E_Security_Camera	Camera	62154	330096	436010
D6	Provision_PT_838_Security_Camera	Camera	98514	309040	429337
D7	Samsung_SNH_1011_N_Webcam	Webcam	52150	323072	
D8	SimpleHome_XCS7_1002_WHT_ Security_Camera	Camera	46585	303223	513248
D9	SimpleHome_XCS7_1003_WHT_ Security_Camera	Camera	19528	316438	514860

Table 2: The training and testing scenarios

No	Scenarios	Training	Testing		
			Same device & same attack	Same device with different attacks	Different Devices
Group 1	1.1	D1, Gafgyt	D1, Gafgyt	D1, Mirai	D3, Gafgyt
	1.2	D1, Mirai	D1, Mirai	D1, Gafgyt	D3, Gafgyt
	1.3	D3, Gafgyt	D3, Gafgyt		D1, Mirai D1, Gafgyt
Group 2	2.1	D5, Gafgyt	D5, Gafgyt	D5, Mirai	D6, Gafgyt D8, Mirai
	2.2	D5, Mirai	D5, Mirai	D5, Gafgyt	D6, Mirai D8, Gafgyt
	2.3	D6, Gafgyt	D6, Gafgyt	D6, Mirai	D5, Gafgyt D8, Mirai
	2.4	D6, Mirai	D6, Mirai	D6, Gafgyt	D5, Mirai D8, Gafgyt

Since the benign data is less than attack data, so we included all benign data for training in all scenarios. The attack data for training was taken from the original dataset with ratios $0.01, 0.1, 1.0, 5.0$ compared to benign data respectively. The relative ratio between attack and benign data is signed as r (**Ratio A:B**).

5.2 Parameters Setting

Firstly, we set up the size of the encoded (bottleneck) for used algorithms (AE and PCA). Since the original size of the features is 115, so we chose the ratio is 0.33 and this produced the size 29 for encode respectively.

Next, we set up parameters for autoencoder, the input layer has the same dimension as the input data. The encoder in the model consists of four hidden layers which have the decreasing size of 75%, 50%, 33%, 25% of the input layer's size. The bottleneck dimension is 29 (compare to 115 in the original). The decoder has absolutely the same number of hidden layers as the encoder, but with increasing order. For training, we split training data into training and validation set with a ratio of $0.8 : 0.2$. The loss function used is MSE (mean squared error), the optimization method is Adam. Early stopping is also used to save the time of training when the result reached to the expected one. The number of epochs is 50 and the batch size is 200. Additionally, the *tanh* activation was used in all layers.

For the PCA, we also passed the size of the bottleneck (coded size) as the AE. All other parameters of the PCA are set to default.

For training SOM, firstly we used tuning techniques for searching the best parameters. The parameters are σ, *learning rate* η. After these parameters were found, we used them for training SOM, building the label-maps, determining the outliers percentage. All these parameters are used later for detecting and classifying anomaly data.

5.3 Evaluation Metrics

We utilize Area Under the Curve (AUC) for evaluating the performance of our proposed methods on different scenarios.

The true positive rate (TPR) and false positive rate (FPR) were calculated by the following formulas.

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

By plotting TPR against FPR, we received the Receiver Operating Characteristic curve (ROC curve) at different thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The AUC was calculated as the entire area underneath the ROC curve. AUC provides an aggregate measure of performance across all possible classification thresholds.

6 Results and Discussion

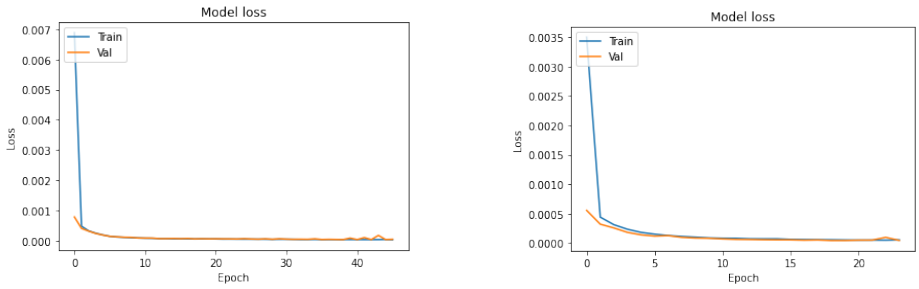
In this section, we describe the results of the experiment process in detail. All the results were collected from scenarios (described in section 5). We also carried out an explanation of the received results.

Because the classification of attacks is inconsistent, our experiments focus on classifying benign and attack types. As seen above in 1, we have two main different groups of attack types, Mirai and Gafgyt respectively, so in the final stage, we combined all types of sub-class in a single class (must be Mirai or Gafgyt). We have two classes at the final stage are **benign** and **attack**.

All experiments were implemented in Python using Keras [2], Scikit-learn [3] and Minisom [4] frameworks. The computer used has the following specifications: Ubuntu 18.04 LTS, Intel(R) Core i7 930 CPU, and 18 GB memory.

Experiments process: All experiments were performed as described in Table 2. The more details about results are shown in Appendix A.

First, we train the AE model for extracting valuable features. The AE model sweeps all devices (D1, D3, D5, D6, D8) for each attack type (Mirai, Gafgyt). The reconstruction losses for training processes of D1-Gafgyt and D3-Gafgyt are shown in Fig. 2. The Fig. 2a shows the reconstruction loss on training D1-Gafgyt and the Fig. 2b shows the result on training D3-Gafgyt.



(a) AE training loss on device 1

(b) AE training loss on device 3

Fig. 2: The AE losses

6.1 Data Analysis

Mirai [5] and Gafgyt (also known as BASHLITE) [6] are all botnets that developed to inject to IoT devices and generate the DDoS traffic from that bots.

[2] https://keras.io/
[3] https://scikit-learn.org/
[4] https://github.com/JustGlowing/minisom
[5] https://en.wikipedia.org/wiki/Mirai_(malware)
[6] https://en.wikipedia.org/wiki/BASHLITE

Gafgyt is a branch of Mirai which is developed upon the existing source code. But the difference is the Gafgyt focuses on generating DDoS traffic, while Mirai focuses on making the victim become a bot.

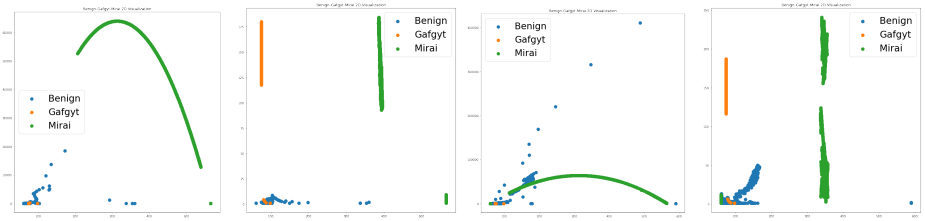
Table 3: The statistics of datasets

Device	Dataset	Mean	Median	Std
D1	Benign	0.14	0.02	0.17
	Gafgyt	16.40	0.05	123.90
	Mirai	45.74	0.14	223.91
D3	Benign	0.08	0.03	0.13
	Gafgyt	4.32	0.01	32.65
	Mirai			
D5	Benign	0.09	0.02	0.18
	Gafgyt	2.22	0.01	17.11
	Mirai	4.05	0.04	21.70
D6	Benign	0.10	0.03	0.18
	Gafgyt	1.49	0.00	10.36
	Mirai	2.71	0.03	13.45
D8	Benign	0.12	0.00	0.21
	Gafgyt	0.47	0.00	3.15
	Mirai	1.72	0.00	7.53

In NBaIot, both Mirai and Gafgyt consist of five different types of attacks. Mirai contains ACK, Scan, SYN, UDP, UDPPLAIN, and Gafgyt contains Combo, Junk, Scan, TCP, and UDP attacks. After normalizing (we use the normalized model from benign, and use this model for normalizing both Mirai and Gafgyt), the statistical table (Table 3) shows that the Mirai is far from both benign and Gafgyt, while Gafgyt and benign are closer to each other and in some cases Gafgyt tends to mix with benign data (see Fig. 3). The metrics used to compare are *mean*, *median* and *standard deviation*. Consider dataset in all devices (D1, D3, D5, D6, D8), the values of Mirai are much higher than benign and Gafgyt.

To confirm this assumption, we visualize the normalized data in a coordinate system. In Figure 3, we use two pairs of features for visualizing data. The Mirai is scattered using green, while benign is in blue and Gafgyt is in orange color. It is clear that Mirai is differed from benign and Gafgyt and all values are far compared to the origin of axes.

From the above consideration, we draw a conclusion: Mirai differs from both benign and Gafgyt; Gafgyt and benign share some common features. The traditional ML methods can be used and can bring good results on Mirai prediction, while it is difficult to detect the Gafgyt malware. In the detection system, if ML models are used, Mirai can be detected with a higher probability while the accuracy of Gafgyt detection is decreased.



(a) D1-First Pair of Features (b) D1-Second Pair of Features (c) D5-First Pair of Features (d) D5-Second Pair of Features

Fig. 3: The Data Visualization of D1 and D5

6.2 Unknown attack detection

As described in Table 2, we performed experiments to determine how methods can detect unknown attacks. So we train the model on Mirai and tested it on Gafgyt and vice versa.

In our opinion, the attacks are sharing some common properties, e.g. the source IP address, the target IP address, the protocol used... Since then, we did training on one type of attack and testing on the other one. Devices D1, D5, D6, D8 are included in these tests due to these devices contain both Gafgyt and Mirai together with benign in the dataset. We sequentially did training on Gafgyt, testing on Mirai, and later training on Mirai, and testing on Gafgyt. We also calculate the *mean*, *median* and *standard deviation* to compare the effectiveness of models used (AE+SOM, PCA+SOM, pure SOM). The results for testing on D1 and D5 are shown in Table 4 (G stands for Gafgyt and M stands for Mirai respectively); the rest results for D6 and D8 are shown in Appendix A. The AE+SOM is good when data is unbalanced, and PCA+SOM is good for most cases when the ratio between benign and attack is around 1.0.

The AUC value has the minimum value is 0.504 and the maximum value is 0.996. It means in all test cases, models achieved more than 50% of anomaly detection in the worse case, and mostly 100% in the best case.

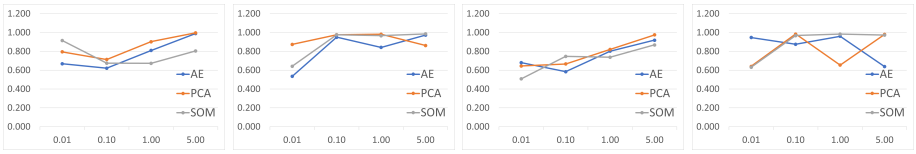
The influence of data ratio r: We also visualized the AUC values for all tests in the Fig. 4. The AUC is plotted against the data ratio r. All the models bring lower AUC when the data ratio r is 0.01, then increased when the r is 0.1 and 1.0. When the r is 5.0, some models are still good, even reached to 1.0 value (mostly when Gafgyt is included in training data), but some models are worse (when Mirai is included in training data). This confirms again for our assumption, that the model is worse when Mirai is used for training. When r is too small (0.01, 0.1) or high (5.0) the AUC is smaller than the balanced value (1.0).

6.3 Transfer Learning

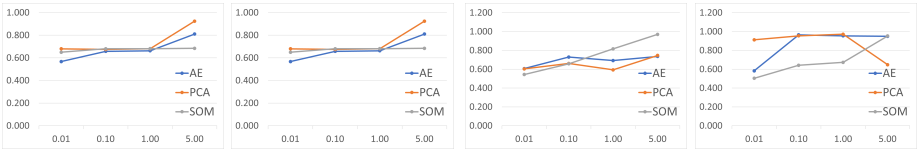
We also performed experiments with transfer learning. The model trained from one device with a specific attack type can be transferred to another with

Table 4: Unknown attack detection results (D1-D3)

Train-Test	**Model**	**Data ratio r**				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D1G-D1M	AE+SOM	0.668	0.620	0.809	0.988	0.771	0.738	0.143
	PCA+SOM	0.795	0.713	0.903	0.996	0.852	0.849	0.107
	SOM	0.914	0.674	0.673	0.804	0.767	0.739	0.101
D1M-D1G	AE+SOM	0.535	0.950	0.841	0.972	0.825	0.896	0.175
	PCA+SOM	0.874	0.976	0.982	0.861	0.923	0.925	0.056
	SOM	0.642	0.976	0.966	0.985	0.892	0.971	0.145
D5G-D5M	AE+SOM	0.679	0.584	0.802	0.919	0.746	0.741	0.126
	PCA+SOM	0.644	0.666	0.821	0.975	0.776	0.743	0.133
	SOM	0.508	0.746	0.737	0.869	0.715	0.742	0.131
D5M-D5G	AE+SOM	0.946	0.874	0.959	0.637	0.854	0.910	0.129
	PCA+SOM	0.641	0.984	0.654	0.979	0.815	0.817	0.167
	SOM	0.631	0.968	0.983	0.972	0.889	0.970	0.149



(a) D1G-D1M (b) D1M-D1G (c) D5G-D5M (d) D5M-D5G



(e) D6G-D6M (f) D6G-D6M (g) D8G-D8M (h) D8M-D8G

Fig. 4: Unknown attack detection results on the same device

other attacks. We divided testing devices into two groups for performing tests. The first group is doorbell (D1 and D3), and the second group is camera (D5, D6, D8). The detailed results were presented in the following subsections.

Transferring model testing on doorbell devices The doorbell devices consist of two devices D1 and D3. First, we did the train on Gafgyt and Mirai on device D1 and tested on Gafgyt on device D3. Next, we did the train on Gafgyt on device D3 and tested on all data attack types on device D1 (both Mirai and Gafgyt were used for testing). The results are shown in Table 5.

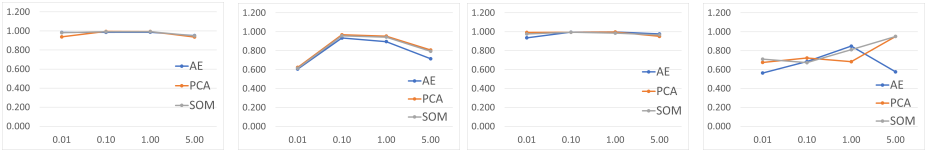
The AE+SOM model is more consistent on an unbalanced dataset (the data ratio is 0.01 and 5.0), while the PCA+SOM is better on the balanced dataset (the data ratio is 0.1 and 1.0). The AE+SOM model which was trained on Gafgyt also shows better results than the model trained on Mirai data. As data analyzed in section 6.1, Gafgyt data and benign are close to each other, so the AE learns and explores features better on benign+Gafgyt. The AE is not good on benign+Mirai, because Mirai is lying further from origin coordinate than Gafgyt and benign. The AUC score of model AE+SOM got a minimum value of 0.563 while PCA+SOM got 0.624 correspondings to the data ratio of 0.01.

On contrary, PCA+SOM is trying to project data from one coordinate system to another one, it keeps all features that have a close correlation to others. When the data is balanced, the PCA+SOM is given better results than the unbalanced data. The maximum AUC score is 0.997 when the data ratio r is balanced (1.0) on both PCA+SOM and AE+SOM models.

Table 5: The transfer learning testing results on D1-D3

Train-Test	**Models**	**Data ratio** r				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D1G-D3G	AE+SOM	0.984	0.985	0.987	0.951	0.977	0.985	0.015
	PCA+SOM	0.939	0.995	0.994	0.936	0.966	0.966	0.028
	SOM	0.981	0.991	0.993	0.948	0.978	0.986	0.018
D1M-D3G	AE+SOM	0.605	0.934	0.896	0.715	0.787	0.805	0.134
	PCA+SOM	0.624	0.968	0.954	0.807	0.838	0.881	0.139
	SOM	0.615	0.957	0.944	0.794	0.827	0.869	0.138
D3G-D1G	AE+SOM	0.935	0.995	0.997	0.976	0.976	0.986	0.025
	PCA+SOM	0.993	0.994	0.997	0.951	0.984	0.993	0.019
	SOM	0.977	0.995	0.986	0.967	0.981	0.981	0.010
D3G-D1M	AE+SOM	0.563	0.685	0.848	0.577	0.668	0.631	0.114
	PCA+SOM	0.676	0.722	0.684	0.950	0.758	0.703	0.112
	SOM	0.710	0.674	0.811	0.950	0.786	0.761	0.107

The AUC scores are plotted against data ratio (see Fig. 5). The direction of the the line tends to permanently on training and testing on Gafgyt and fluctuating on training or testing on Mirai.



(a) D1G-D3G (b) D1M-D3G (c) D3G-D1G (d) D3G-D1M

Fig. 5: The AUC visualization for D1-D3

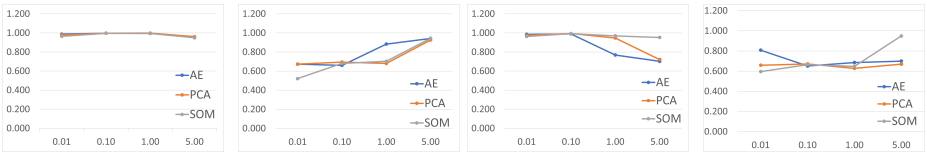
Transferring model testing on camera devices The camera group consists of devices D5, D6, and D8. While D5 and D6 are devices with a different version of one brand, D8 is a device of another one. Each device contains benign, Gafgyt and Mirai, so we train sequentially on benign+Gafgyt and benign+Mirai on one device and test on two rest devices. In this section, the testing results are shown for training dataset benign+Gafgyt of device D5 (Table 6), the other results for device D6 and D8 are shown in Appendix A.

Table 6: The transfer learning results (Train: D5-Gafgyt, Test: D6, D8)

Train-Test	**Models**	**Data ratio** r				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D5G-D6G	AE+SOM	0.988	0.997	0.996	0.958	0.985	0.992	0.016
	PCA+SOM	0.979	0.996	0.998	0.961	0.983	0.988	0.015
	SOM	0.964	0.995	0.994	0.948	0.975	0.979	0.020
D5G-D6M	AE+SOM	0.672	0.660	0.883	0.941	0.789	0.777	0.125
	PCA+SOM	0.673	0.694	0.680	0.925	0.743	0.687	0.105
	SOM	0.521	0.680	0.702	0.943	0.712	0.691	0.151
D5G-D8G	AE+SOM	0.985	0.989	0.768	0.702	0.861	0.877	0.128
	PCA+SOM	0.974	0.993	0.947	0.720	0.909	0.960	0.110
	SOM	0.963	0.993	0.969	0.953	0.969	0.966	0.015
D5G-D8M	AE+SOM	0.807	0.650	0.685	0.699	0.710	0.692	0.059
	PCA+SOM	0.658	0.671	0.627	0.669	0.656	0.664	0.017
	SOM	0.595	0.664	0.647	0.949	0.714	0.656	0.138

The AUC scores are also plotted in Fig. 6. The AUC scores are the results of training on Gafgyt of device D5, and testing on D6 and D8. AE+SOM, PCA+SOM and SOM give the better results when testing on Gafgyt (Fig. 6a and Fig. 6c), and worse when testing on Mirai (Fig. 6b, Fig. 6d). The plot for D5-Mirai in shown in Fig. 7 in Appendix A.

The influence of data ratio r: To assess the influence of r, we also performed tests in a range of values 0.01, 0.1, 1.0, and 5.0. The AE+SOM seems to be better than PCA+SOM and SOM on training on Gafgyt and testing on Gafgyt with r small. When r is 1.0, the AE+SOM is better at training on Gafgyt and testing on Mirai. In the last case, if the r is equal to 5.0 then pure SOM is



(a) D5G-D6G (b) D5G-D6M (c) D5G-D8G (d) D5G-D8M

Fig. 6: The AUC visualization for D5 (Train on Gafgyt)

the best. The value 5.0 means the amount of attack data is 5 times more than benign. But in that case, the dataset is more "balanced" compare to other values of 0.01 and 0.1. This proves that, when the amount of benign is much more or at least equal to attack, then AE+SOM can handle better. When the attack dominates benign, then the effectiveness of AE+SOM is increased. This point is already mentioned in the section 1 and 2 that AE fit with one class classification or in cases when amount data of majority class is much more than the other classes.

To compare the effectiveness of all algorithms on devices, we total up the minimum and maximum values of AUC scores in Table 7. As seen in this table, the AE+SOM model is shown better than other models. It means that the attacks are different, but they are sharing common characteristics, such as packet format, generation sources... AE can learn and extract most features from the data. The latent contains most characteristics of the original input.

Table 7: Algorithm Effectiveness Comparison

Device	**Metric**	**AE+SOM**	**PCA+SOM**	**SOM**
D1	Min	0.563	0.624	0.615
	Max	0.997	0.997	0.995
D5	Min	0.521	0.549	0.521
	Max	0.997	0.996	0.991
D6	Min	0.777	0.564	0.628
	Max	0.998	0.998	0.998
D8	Min	0.524	0.506	0.508
	Max	0.993	0.989	0.989

In IoT network, the amount of devices is huge, the protocols used are also diverse, and most of data is unlabeled. Therefore, the applying of a trained model from one known device to other devices is really meaningful. It can save time and resources for training. Especially, the new device does not have enough data for training. The transferring model can help to detect unknown malware attacks on such devices. It can detect unknown attacks (zero-day attacks) not only in the device types which are used for training but in new coming devices in the future.

7 Conclusions and Future work

In this work, we proposed the use of a combination of autoencoder with self-organizing maps (AE+SOM) for detecting attacks on IoT dataset NBaIoT. Our main contribution is to apply the unsupervised learning methods to build one end-to-end system for detecting anomalies. The results are acceptable and reliable because we trained and tested on different datasets and devices. Experiments show that the AE+SOM classifier is better than other methods (PCA+SOM, SOM) at the last stages for transfer learning. Almost all methods are able to detect not only unknown attacks in the same device but on another one. Our method can be applied to the real system for real-time classification, since the time for loading model and classifying is not too much.

In the future, our method can be extended in the following manners. First, we can use the joint-learning method to improve the classification process; it means we can use both AE and SOM in one training stage, instead of separation. Second, the data can be better clustered if the regularizers are used. We did not use any regularizer in our experiments, but this method can be good for separating data.

References

1. Abomhara, M., Køien, G.M.: Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks. Journal of Cyber Security and Mobility **4**(1), 65–88 (1 2015). https://doi.org/10.13052/jcsm2245-1439.414, https://riverpublishers.com/journal_read_html_article.php?j=JCSM/4/1/4
2. Bui, T.C., Cao, V.L., Hoang, M., Nguyen, Q.U.: A clustering-based shrink autoencoder for detecting anomalies in intrusion detection systems. In: Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019. Institute of Electrical and Electronics Engineers Inc. (10 2019). https://doi.org/10.1109/KSE.2019.8919446
3. Cao, V.L., Nicolau, M., Mcdermott, J.: A Hybrid Autoencoder and Density Estimation Model for Anomaly Detection (2016). https://doi.org/10.1007/978-3-319-45823-6_67, http://ncra.ucd.ie
4. Cao, V.L., Nicolau, M., McDermott, J.: One-class classification for anomaly detection with kernel density estimation and genetic programming. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 9594, pp. 3–18. Springer Verlag (2016). https://doi.org/10.1007/978-3-319-30668-1_1, https://link.springer.com/chapter/10.1007/978-3-319-30668-1_1
5. Cao, V.L., Nicolau, M., McDermott, J.: Learning Neural Representations for Network Anomaly Detection. IEEE Transactions on Cybernetics **49**(8), 3074–3087 (8 2019). https://doi.org/10.1109/TCYB.2018.2838668, https://ieeexplore.ieee.org/document/8386786/
6. Colombini, G.G., De Abreu, I.B.M., Cerri, R.: A self-organizing map-based method for multi-label classification. In: Proceedings of the International Joint Conference on Neural Networks. vol. 2017-May, pp. 4291–4298. Institute of Electrical and Electronics Engineers Inc. (6 2017). https://doi.org/10.1109/IJCNN.2017.7966399

7. Dastjerdi, A.V., Buyya, R.: Fog Computing: Helping the Internet of Things Realize Its Potential. Computer **49**(8), 112–116 (8 2016). https://doi.org/10.1109/MC.2016.245
8. Erfani, S.M., Rajasegarar, S., Karunasekera, S., Leckie, C.: High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. Pattern Recognition **58**, 121–134 (10 2016). https://doi.org/10.1016/j.patcog.2016.03.028
9. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
10. Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier Detection Using Replicator Neural Networks. Tech. rep.
11. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (7 2006). https://doi.org/10.1126/science.1127647, https://science.sciencemag.org/content/313/5786/504https://science.sciencemag.org/content/313/5786/504.abstract
12. Japkowicz, N., Myers, C., Gluck, M.: A Novelty Detection Approach to Classiication. Tech. rep.
13. Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects (7 2015). https://doi.org/10.1126/science.aaa8415, https://science.sciencemag.org/content/349/6245/255https://science.sciencemag.org/content/349/6245/255.abstract
14. Kohonen, T.: The self-organizing map. Proceedings of the IEEE **78**(9), 1464–1480 (1990)
15. Kohonen, T.: Essentials of the self-organizing map. Neural Networks **37**, 52–65 (2013). https://doi.org/10.1016/j.neunet.2012.09.018, http://dx.doi.org/10.1016/j.neunet.2012.09.018
16. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**(7), 80–84 (2017). https://doi.org/10.1109/MC.2017.201
17. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Breitenbacher, D., Shabtai, A., Elovici, Y.: N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders. Tech. Rep. 9, http://archive.ics.uci.edu/ml/datasets/detection
18. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-BaIoT-Network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing **17**(3), 12–22 (5 2018). https://doi.org/10.1109/MPRV.2018.03367731, http://arxiv.org/abs/1805.03409http://dx.doi.org/10.1109/MPRV.2018.03367731
19. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. arXiv (2 2018), http://arxiv.org/abs/1802.09089
20. Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., Sadeghi, A.R.: DÏoT: A federated self-learning anomaly detection system for IoT. In: Proceedings - International Conference on Distributed Computing Systems. vol. 2019-July, pp. 756–767. Institute of Electrical and Electronics Engineers Inc. (7 2019). https://doi.org/10.1109/ICDCS.2019.00080
21. Rauber, A.: LabelSOM: On the labeling of self-organizing maps. In: Proceedings of the International Joint Conference on Neural Networks. vol. 5, pp. 3527–3532. IEEE (1999). https://doi.org/10.1109/ijcnn.1999.836235
22. Ray, S., Jin, Y., Raychowdhury, A.: The Changing Computing Paradigm with Internet of Things: A Tutorial Introduction. IEEE Design and Test **33**(2), 76–96 (2016). https://doi.org/10.1109/MDAT.2016.2526612

23. Song, C., Liu, F., Huang, Y., Wang, L., Tan, T.: LNCS 8258 - Auto-encoder Based Data Clustering. Tech. rep.

24. Tian, J., Azarian, M., Pecht, M.: Anomaly Detection Using Self-Organizing Maps-Based K-Nearest Neighbor Algorithm (2014)

25. Tsai, C.W., Lai, C.F., Chiang, M.C., Yang, L.T.: Data mining for internet of things: A survey. IEEE Communications Surveys and Tutorials **16**(1), 77–97 (3 2014). https://doi.org/10.1109/SURV.2013.103013.00206

26. Vu, L., Cao, V.L., Nguyen, Q.U., Nguyen, D.N., Hoang, D.T., Dutkiewicz, E.: Learning Latent Distribution for Distinguishing Network Traffic in Intrusion Detection System. In: IEEE International Conference on Communications. vol. 2019-May. Institute of Electrical and Electronics Engineers Inc. (5 2019). https://doi.org/10.1109/ICC.2019.8762015

27. Vu, L., Nguyen, Q.U.: An ensemble of activation functions in autoencoder applied to IoT anomaly detection. In: Proceedings - 2019 6th NAFOSTED Conference on Information and Computer Science, NICS 2019. pp. 534–539. Institute of Electrical and Electronics Engineers Inc. (12 2019). https://doi.org/10.1109/NICS48868.2019.9023860

28. Vu, L., Nguyen, Q.U., Nguyen, D.N., Hoang, D.T., Dutkiewicz, E.: Deep Transfer Learning for IoT Attack Detection. IEEE Access **8**, 107335–107344 (2020). https://doi.org/10.1109/ACCESS.2020.3000476

29. Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. Journal of Big Data 2016 3:1 **3**(1), 1–40 (5 2016). https://doi.org/10.1186/S40537-016-0043-6, https://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6

30. Wold, S., Esbensen, K., Geladi, P.: Principal Component Analysis. Tech. rep.

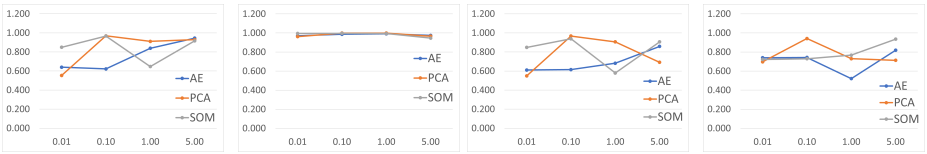
A Appendix

Table 8: Unknown attack detection results

Train-Test	**Model**	**Data ratio r**				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D6G-D6M	AE+SOM	0.567	0.657	0.663	0.810	0.674	0.660	0.087
	PCA+SOM	0.680	0.674	0.680	0.924	0.740	0.680	0.106
	SOM	0.650	0.680	0.681	0.684	0.674	0.681	0.014
D6M-D6G	AE+SOM	0.634	0.859	0.843	0.937	0.818	0.851	0.112
	PCA+SOM	0.965	0.560	0.654	0.979	0.790	0.810	0.186
	SOM	0.949	0.952	0.821	0.964	0.922	0.950	0.058
D8G-D8M	AE+SOM	0.606	0.729	0.692	0.736	0.691	0.710	0.051
	PCA+SOM	0.603	0.661	0.593	0.746	0.651	0.632	0.061
	SOM	0.543	0.657	0.816	0.970	0.747	0.737	0.161
D8M-D8G	AE+SOM	0.583	0.965	0.954	0.949	0.863	0.952	0.162
	PCA+SOM	0.912	0.954	0.971	0.648	0.871	0.933	0.131
	SOM	0.504	0.641	0.673	0.953	0.693	0.657	0.163

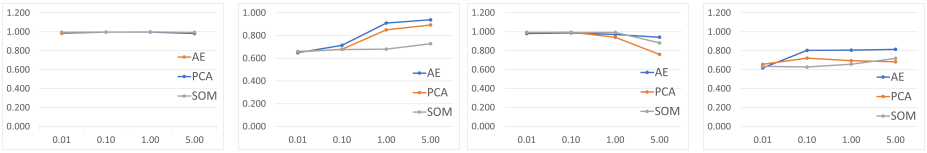
Table 9: The transfer learning results (Train: D5-Mirai, Test: D6, D8)

Train-Test	**Models**	**Data ratio r**				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D5M-D6G	AE+SOM	0.640	0.622	0.839	0.944	0.761	0.739	0.136
	PCA+SOM	0.554	0.969	0.910	0.928	0.840	0.919	0.167
	SOM	0.849	0.966	0.646	0.916	0.844	0.882	0.122
D5M-D6M	AE+SOM	0.969	0.986	0.991	0.974	0.980	0.980	0.009
	PCA+SOM	0.961	0.999	0.999	0.966	0.981	0.983	0.018
	SOM	0.995	0.996	0.994	0.946	0.983	0.995	0.021
D5M-D8G	AE+SOM	0.610	0.615	0.681	0.858	0.691	0.648	0.100
	PCA+SOM	0.549	0.967	0.906	0.691	0.778	0.798	0.167
	SOM	0.848	0.938	0.579	0.906	0.818	0.877	0.142
D5M-D8M	AE+SOM	0.740	0.743	0.521	0.819	0.705	0.741	0.111
	PCA+SOM	0.697	0.942	0.730	0.713	0.771	0.722	0.099
	SOM	0.722	0.729	0.767	0.934	0.788	0.748	0.086



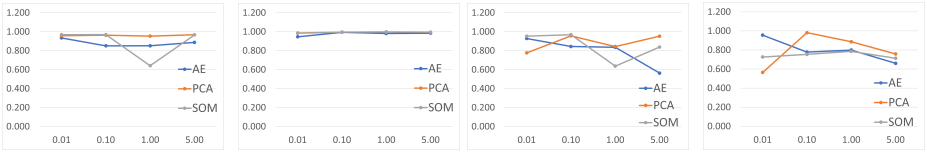
(a) D5M-D6G (b) D5M-D6M (c) D5M-D8G (d) D5M-D8M

Fig. 7: The AUC visualization for D5 (Train on Mirai)



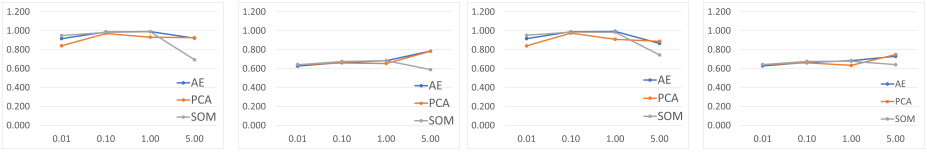
(a) D6G-D5G (b) D6G-D5M (c) D6G-D8G (d) D6G-D8M

Fig. 8: The AUC visualization for D6 (Train of Gafgyt)



(a) D6M-D5G (b) D6M-D5M (c) D6M-D8G (d) D6M-D8M

Fig. 9: The AUC visualization for D6 (Train on Mirai)

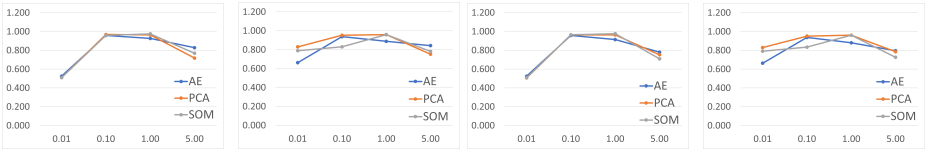


(a) D8G-D5G (b) D8G-D5M (c) D8G-D6G (d) D8G-D6M

Fig. 10: The AUC visualization for D8 (Train of Gafgyt)

Table 10: The testing results on devices D6-D5-D8

Train-Test	Models	Data ratio r				Metrics		
		0.01	0.10	1.00	5.00	mean	median	std
D6G-D5G	AE+SOM	0.982	0.996	0.998	0.993	0.992	0.994	0.006
	PCA+SOM	0.994	0.997	0.997	0.982	0.992	0.995	0.006
	SOM	0.995	0.996	0.998	0.992	0.995	0.996	0.002
D6G-D5M	AE+SOM	0.648	0.714	0.909	0.938	0.802	0.812	0.124
	PCA+SOM	0.656	0.679	0.850	0.893	0.770	0.765	0.103
	SOM	0.660	0.676	0.680	0.728	0.686	0.678	0.025
D6G-D8G	AE+SOM	0.981	0.986	0.970	0.942	0.970	0.975	0.017
	PCA+SOM	0.993	0.994	0.940	0.760	0.922	0.967	0.096
	SOM	0.993	0.991	0.993	0.881	0.964	0.992	0.048
D6G-D8M	AE+SOM	0.618	0.803	0.806	0.814	0.760	0.804	0.082
	PCA+SOM	0.655	0.721	0.694	0.682	0.688	0.688	0.024
	SOM	0.634	0.628	0.657	0.719	0.660	0.646	0.036
D6M-D5G	AE+SOM	0.934	0.850	0.851	0.887	0.881	0.869	0.034
	PCA+SOM	0.957	0.963	0.953	0.968	0.960	0.960	0.006
	SOM	0.968	0.968	0.641	0.966	0.886	0.967	0.141
D6M-D5M	AE+SOM	0.946	0.993	0.981	0.984	0.976	0.982	0.018
	PCA+SOM	0.986	0.996	0.998	0.994	0.994	0.995	0.005
	SOM	0.983	0.997	0.997	0.995	0.993	0.996	0.006
D6M-D8G	AE+SOM	0.928	0.844	0.835	0.563	0.792	0.839	0.137
	PCA+SOM	0.776	0.956	0.843	0.952	0.882	0.898	0.076
	SOM	0.953	0.969	0.637	0.838	0.849	0.895	0.132
D6M-D8M	AE+SOM	0.956	0.777	0.799	0.660	0.798	0.788	0.105
	PCA+SOM	0.564	0.981	0.886	0.758	0.797	0.822	0.156
	SOM	0.726	0.753	0.786	0.713	0.745	0.740	0.028



(a) D8M-D5G (b) D8M-D5M (c) D8M-D6G (d) D8M-D6M

Fig. 11: The AUC visualization for D8 (Train on Mirai)

Table 11: The testing results on devices D8-D5-D6

Train-Test	**Models**	**Data ratio r**				**Metrics**		
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**std**
D8G-D5G	AE+SOM	0.917	0.986	0.990	0.921	0.954	0.954	0.035
	PCA+SOM	0.841	0.970	0.931	0.927	0.917	0.929	0.047
	SOM	0.949	0.982	0.990	0.693	0.903	0.965	0.122
D8G-D5M	AE+SOM	0.627	0.665	0.683	0.785	0.690	0.674	0.059
	PCA+SOM	0.643	0.660	0.654	0.784	0.685	0.657	0.057
	SOM	0.641	0.675	0.684	0.589	0.647	0.658	0.037
D8G-D6G	AE+SOM	0.917	0.989	0.993	0.866	0.941	0.953	0.053
	PCA+SOM	0.840	0.975	0.910	0.889	0.904	0.899	0.049
	SOM	0.952	0.985	0.984	0.745	0.916	0.968	0.100
D8G-D6M	AE+SOM	0.628	0.665	0.683	0.729	0.677	0.674	0.036
	PCA+SOM	0.643	0.660	0.633	0.747	0.671	0.652	0.045
	SOM	0.642	0.676	0.677	0.642	0.659	0.659	0.018
D8M-D5G	AE+SOM	0.525	0.958	0.926	0.828	0.809	0.877	0.171
	PCA+SOM	0.509	0.968	0.964	0.717	0.790	0.841	0.191
	SOM	0.509	0.960	0.975	0.769	0.803	0.864	0.188
D8M-D5M	AE+SOM	0.662	0.938	0.889	0.843	0.833	0.866	0.104
	PCA+SOM	0.830	0.953	0.959	0.752	0.873	0.891	0.087
	SOM	0.789	0.830	0.959	0.781	0.840	0.809	0.071
D8M-D6G	AE+SOM	0.524	0.957	0.914	0.779	0.794	0.847	0.169
	PCA+SOM	0.506	0.964	0.963	0.752	0.796	0.858	0.189
	SOM	0.508	0.965	0.976	0.709	0.790	0.837	0.194
D8M-D6M	AE+SOM	0.663	0.937	0.880	0.796	0.819	0.838	0.103
	PCA+SOM	0.830	0.951	0.961	0.785	0.882	0.891	0.076
	SOM	0.790	0.835	0.961	0.726	0.828	0.813	0.086