Denoising Latent Representation with SOMs for Unsupervised IoT Malware Detection

Huu Noi Nguyen, Nguyen Ngoc Tran, Tuan Hao Hoang and Van Loi Cao

Le Quy Don Technical University, 236 Hoang Quoc Viet, Ha Noi, 10000, Viet Nam.

*Corresponding author(s). E-mail(s): loi.cao@lqdtu.edu.vn;
Contributing authors: noi.nguyen@lqdtu.edu.vn;
ngoctn@lqdtu.edu.vn; haoth@lqdtu.edu.vn;

Abstract

The latent feature space of AutoEncoder-based structure models has been widely developed for unsupervised learning in cyber-security domain, and has shown remarkable performance. Our previous work has introduced a hybrid AutoEncoders (AEs) and Self-Organizing Maps (SOMs) for unsupervised IoT Malware detection. However, the paper has only examined the characteristics of the latent representation of ordinary AEs in comparison to that of Principle Component Analysis (PCA) on different ratios of IoT malware data. In this paper, we extend our previous work by employing Denoising AEs (DAEs) to enhance to the generalization ability of latent representation as well as optimizing hyper-parameters of SOMs to improve the hybrid performance. This aims to further examine the characteristics of AE-based structure models (i.e. DAE) for the task of IoT malware detection, particularly identifying unknown/new IoT attacks and transfer learning. Our model is evaluated and analyzed extensively by a number of experiments on the NBaIoT dataset. Well-known feature representation methods such as PCA and AEs are included in the experiments in order to highlight the behavior of the DAE learner. The experimental results demonstrate that the latent representation of DAEs is often superior to that of AEs and PCAs in the task of identifying IoT malware.

Keywords: AutoEncoders, Denoising Autoencoders, Self-Organizing Maps, IoT Malware, IoT anomaly detection, transfer learning

1 Introduction

Internet of Things (IoTs) has played a vital role in developing smart services for Human's life, such as smart home, healthcare, transportation and education services [1, 2]. However, the heterogeneous nature and the diversity of IoT have make it vulnerable to many kinds of attacks [3]. Meanwhile, new generations of IoT malware are continuously launching to bypass IoT security systems and exploit valuable resources on IoT. Onboard attacks, security gateway attacks, control server attacks, and eavesdropping attacks are just a few typical examples of IoT attacks. Botnet Mirai which is one of the most popular IoT malware has recently been employed to launch large-scale DDoS attacks by exploiting vulnerabilities in IoT devices [4]. Therefore, the IoT security solutions should be intensively and extensively developed in order to keep up with the rapid evolution of IoT malware[4].

Machine learning (ML) has been widely used to develop solutions for many security problems like detecting malicious codes, identifying and classifying network attacks, and recognizing anomalous behaviors of network data [5–8]. When applying ML, IoT malware and malicious actions are often modelled as anomalies while normal activities of IoT can be treated as normal [9–12]. Depending on the availability of anomaly data and the objective of detection task, supervised, unsupervised or semi-supervised learning approach can be used for constructing detection models. In addition, advances in machine learning (learning strategies), such as federated learning, transfer learning, and few-shot learning have been introduced to strengthen ML in dealing with challenging task in cyber-security, particularly IoT security [13]. Amongst the above approaches, unsupervised learning is more popular due to the scarcity of cyber-attack data and their labels as well as the high demand in identifying unknown/new malicious activities.

Recently, AutoEncoder-based techniques, such as ordinary AutoEncoders (AEs), Denoising AEs, Variational AE, have been used to construct useful latent feature space from the input data for eliminating subsequent detection/-classification methods [10, 12, 14, 15]. For semi-supervised anomaly detection, Nguyen et al. [15] proposed a hybrid between AEs and K-means for automatically discovering sub-classes in normal data. The hybrid was trained with an iterative strategy, while AEs with a regularized loss function attempt to learning a "good" latent representation in its middle hidden layer, K-means try to explore sub-clusters in the latent representation. However, the subsequent clustering method, namely K-means, is quite simple makes the hybrid difficult to learning complex latent representation produced from AEs without any regularized terms in its loss function. Our previous work [16] has investigated the latent representation of AEs in supporting an unsupervised learning method, namely Self-Organizing Maps (SOMs), for identifying IoT malware. In the work, we have extensively examined the characteristics of AEs on a wide range of experiments (i.e. a number of IoT malware scenarios as well as the ratio of IoT malware in data). Moreover, SOMs is an advanced clustering method that can discover clusters from complex feature representations (i.e.

clusters with arbitrary shapes). The experiments from [16] demonstrated that the latent representation of AEs can help SOMs learn highly unbalanced data and IoT malware whose behaviors being similar to benign. However, AEs tend to learn identity from data that is more likely fall into overfitting. In addition, SOMs are employed with default the hyper-parameter setting that is difficult for SOMs to obtain the best performance.

In this paper, we extend our previous work [16] by introducing Denoising AEs to avoid learning identity and employing optimization methods to search for the best hyper-parameters of SOMs. This study aims to extensively investigate the latent representation of AE-based structure networks (i.e. Denoising AEs) in learning multiple classes simultaneously for unsupervised anomaly detection problems. In other words, we attempts to give answers for the questions that whatever the latent representation of DAEs can benefit unsupervised learners better than that of AEs or not? What kinds of IoT malware groups do DAEs prefer to learn? Similar to [16], we create a number of data scenarios with different ratios of IoT malware and benign to train DAEs. SOMs are chosen as the subsequent clustering methods working on the latent representation of DAEs. Instead of using default settings as in [16], we employ well-known optimization techniques to search for the best hyper-parameter settings of SOMs. Ordinary AEs and PCA are included in our study as baselines for comparison with DAEs. AEs can help SOMs learn highly unbalanced data and IoT malware whose behaviors being similar to Benign while PCA tends to support SOMs working well on quite balanced data with its classes being very separated from each others [16]. The NBaIoT dataset [17] is used for evaluating our proposed models on different aspects such as detecting unknown/new malware, transferring knowledge for detecting IoT malware on different IoT devices and detecting different IoT malware groups.

The contributions of this paper can be listed as follows:

- Investigate the latent representation of DAEs for unsupervised learning in detecting IoT malware.
- Apply optimization techniques for estimating hyper-parameters of the subsequent clustering methods, namely SOMs, in hybrid between latent representation models (DAEs, AEs, or PCA) and SOMs.
- Carry out a number of experiments to evaluate hybrid the latent representation models and SOMs, called AESOM, DAESOM and PCASOM on different aspects of IoT malware detection models such as the ability to detect unknown/new IoT attacks, and to transfer model knowledge.

The remainder of paper is organized as follows. Section 2 and 3 briefly present some related works and background, respectively. Our proposed models are described in details in Section 4. Following this, experiments, results and discussions are presented and illustrated in Sections 5 and 6. Section 7 draws some conclusion and future directions for the paper.

2 Related Works

In this section, we'll go through some recent research that has been used to detect IoT anomalies. Most of approaches used are based on unsupervised/semi-supervised learning methods. We discuss the use of AEs in particular. Next, we also carry out some studies that tend to solve the classification problem using the unsupervised technique.

As a feature learner for latent representation, AE-based approaches have been frequently used [9, 12, 14, 17–21]. Later on, this latent is used in anomaly detection models. The latent feature representation can be trained in a variety of ways, including supervised learning [9, 17], semi-supervised learning [12], and unsupervised learning [14, 20]. The common approach is as follow: all algorithms are trained on the training data (usually one-class learning), and only the encoder is utilized for further stages when the training process is completed. This encoder attempts to extract latent features from the input, which are then used in typical machine learning approaches to develop detection models. Because the dimension of latent data is smaller and less than that of input data, classification models are often faster.

Recently Cao et al. [12] introduced two regularized AEs, namely SAE and DVAE for capture the normal behaviors of network data. The purpose of these regularizers AEs is to direct normal data to a small region near the origin of the latent feature space, so saving the remainder of the space for future anomalies. The purpose of these regularized AEs was to solve the problem of detecting anomalies in high-dimensional network data. The latent representation of SAE and DVAE was then employed to improve simple one-class classifiers. In a supervised manner, Vu et al. [9] proposed Multi-distribution VAE (MVAE) to represent normal data and anomalous data into two separate regions in the latent feature space of VAE. Originally, Variational EutoEncoders (VAEs) learn to map input data into a standard Gaussian distribution $\mathcal{N}(0, 1)$ in its middle hidden layer. The suggested model was tested on two publicly available network security datasets and showed encouraging results.

In unsupervised manner, Gustavo et. al. [22] explore the power of SOM for multi-label classification. Since the SOM has ability to map input instances to a map of neurons. Similar instances are grouped into the same class after SOM. Also using SOM, Andreas Rauber et. al. [23] presented the LabelSOM model, 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 [25], Ferles et. al. presented the DASOM which incorporates the latter into a hierarchically organized hybrid model with a grid of topologically ordered neurons as the front-end component. The idea is to put a layer of hidden representations between the input space and the self-organizing map's neural lattice. DASOM's efficiency, performance, and projection capabilities

are demonstrated through a variety of studies that include optical recognition of text and images, as well as cancer type grouping and categorization. With the same approach, Wickramasinghe et. al. [26] also presented the DeepSOM for unsupervised image classification.

This work 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 capacity of Denoising AutoEncoder (DAE) to data denoise, enhancing model accuracy on IoT latent feature representation, was then examined. The latent representation then facilitate SOMs for discovering clusters.

3 Background

3.1 AutoEncoder

An AutoEncoder is a feed-forward neural network that attempts to reconstruct the original input data at the output layer [27, 28]. The traditional AE is used for dimensional reduction and feature learning. The AE architecture is shown in Fig. 1 consists of two parts, Encoder, Decoder, connecting by its bottleneck layer. The hidden layer h that described a code used to represent the input [29]. The encoder function f is used to learn the input and represented as *code*, the decoder function 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 $\hat{x}$.

$$\begin{aligned} L(x, \hat{x}) = \|x - \hat{x}\|^2 &= \|x - a_d(w'.f(x) + b')\|^2 \\ &= \|x - a_d(w'.a_e(wx + b) + b')\|^2 \end{aligned} \tag{2}$$

3.2 Denoising AutoEncoder

The Denoising AutoEncoder (DAE) [30] works like traditional AE, but instead of directly reconstructing from the latent representation of the input, it tries to reconstruct the input from the "noise" version. Firstly, the initial input x is used to build the corrupted version $\tilde{x}$. The "noise" x_{noise} is drawn using the additive isotropic Gaussian noise: $x_{noise} = \mathcal{N}(0, \sigma_{noise})$. In the experiments, the σ_{noise} is set to 1.0. Next, the corrupted version $\tilde{x}$ is calculated by adding noise to the original input: $\tilde{x} = x + x_{noise}$.

The original formula of AE is rewritten as follow:

$$z = f(\tilde{x}) = a_e(w\tilde{x} + b) \tag{3a}$$

$$\hat{x} = g(z) = g(f(\tilde{x})) = a_d(w'.f(\tilde{x}) + b') \tag{3b}$$

As the same to AE, the reconstruction loss function of DAE is also to minimize the difference between output $\hat{x}$ and original input x.

$$\begin{aligned} L(x, \hat{x}) = \|x - \hat{x}\|^2 &= \|x - a_d(w'.f(\tilde{x}) + b')\|^2 \\ &= \|x - a_d(w'.a_e(w\tilde{x} + b) + b')\|^2 \\ &= \|x - a_d(w'.a_e(w(x + x_{noise}) + b) + b')\|^2 \end{aligned} \tag{4}$$

where a_e and a_d are the activation functions of the encoder and decoder, respectively; the output $\hat{x}$ is $\tilde{x}$'s reconstruction. The reconstruction loss function (e.g. squared loss error) is to minimize the difference between the input x and the re $\hat{x}$. The difference from the traditional AE is $\hat{x}$ is obtained from the corrupted input instead original x.

3.3 Principle Component Analysis

One of the most significant techniques in Machine Learning is dimensionality reduction. It is to find a function that takes the input of a data point $x \in R^D$ (D is the number of dimensions) and create a new data point $z \in R^K$ with $K < D$.

One of the simplest dimensionality reduction algorithms is based on a linear model. This algorithm is called Principal Component Analysis (PCA) [31, 32]. 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 Kernel PCA [33], Sparse PCA [34], Nonlinear PCA [35], Robust PCA [36].

3.4 Self-Organizing Maps

The Self-Organizing Maps (SOM) [37, 38] 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 a high-dimensional input space to lower-dimensional output space (usually two dimension). SOMs employ 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.

The winning neuron is archived on the training SOM using Euclidean distance, which is represented in Equation 5.

$$d_j(x) = \sqrt{\sum_{i=1}^{A}(x_i - w_{ji})^2} \tag{5}$$

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 6, 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. [22].

$$h_{j,i} = exp(-\frac{d_{j,i}^2}{2\sigma^2}) \tag{6}$$

The weight update process is given by Equation 7, where w_j is the weight vector, x is input instance, η is a learning rate.

$$\Delta w_j = \eta h_{j,i}(x - w_j) \tag{7}$$

Finally, the weight vector at iteration $(t+1)$ is updated by Equation 8.

$$\Delta w_j(t+1) = w_j(t) + \eta h_{j,i}(x - w_j(t)) \tag{8}$$

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 8 [22]. 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).

In 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.

3.5 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 [39].

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

4 The Hybrid DAESOM

This section presents our proposed hybrid model for detecting IoT malware. Similar to our previous study [16], our approach also consists of two phases as shown in Fig. 1. The difference is that, a DAE is used instead an ordinary AE as in [16], and training SOM in phase 2 includes the task of optimizing hyper-parameters. The details of these phases are described as follows:

- ***Phase 1***: A DAE is trained on unlabeled data (normal data and IoT Malware) to construct its latent representation. The latent feature space which is in a lower dimension often discover more meaningful features. The feature space is then fed into the subsequent clustering method.
- ***Phase 2***: A SOM plays as a classification-based method working directly on the DAE's feature space. It will learn to discover clusters in the feature space in training stage, and classify data into proper clusters (normal data and IoT malware) in the querying stage. The hyper-parameter optimization is included in this phase.

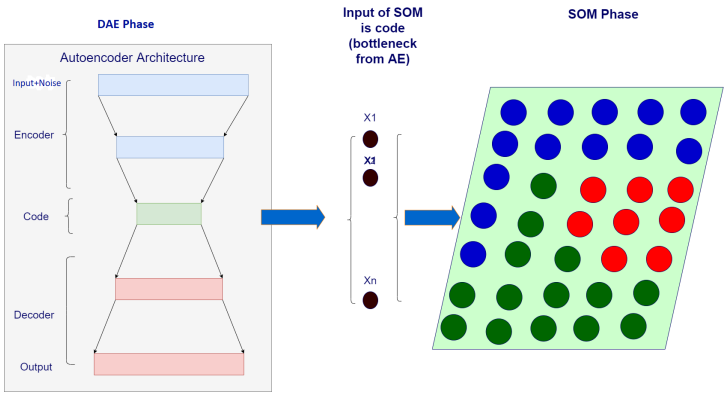


Fig. 1: The system architecture

Another feature representation such as AEs and PCA are also involved in phase 1 of the our proposed model to highlight the performance of DAEs.

4.1 Latent Representation of DAEs

In this work, we employ an ordinary DAE with a 3-layer in its encoder and a 3-layer in its decoder. The DAE architecture is shown in Fig. 1. The DAE is trained in the unsupervised learning manner. This means that the training dataset is a unlabelled dataset consisting of both normal data and IoT malware. To investigate the effectiveness of the DAE feature space in unsupervised learning task, we create four versions of the training datasets with four different level of unbalances (ratio r) between IoT malware and normal data on each IoT malware group. The values of r are manually chosen as 0.01, 0.1, 1.0, and 5.0, they are most the same as described in [16]. To examine the behavior of DAE on different IoT malware groups, the normal data is combined with each of IoT malware groups to generate training datasets for each IoT device. For training process, a training dataset is passed into an AE model with the objective of minimizing the difference between the original data and its reconstruction. The process is terminated if the training error is satisfied an early-stopping criteria. Once the training process is done, the decoder part is discarded while the encoder part is preserved as a feature learning component facilitating the subsequent clustering-based technique.

On visualizing data (more details are described in section Results and Analysis in [16]), the graph shows that Gafgyt tends to be close to Benign. This can lead to bad results when the model classifies because Gafgyt and Benign look like noise to each other. To overcome this situation, the DAE is used to denoise before passing the data into the model.

4.2 SOM-based Classification Algorithm

Given instance x_i, and the task is to classify it 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 [22].

The trained SOM maps test instance into labels-map. Once its closest neurons found, a vector (known as the prototype vector) is produced by averaging the class vectors of the training instances mapping to this neuron, denoted by $\bar{v}$. It is the classification probability of the test instance falling into each class. The formulation of the prototype vector is shown in Equation 9. 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.

$$\bar{v}_{n,j} = \frac{\mathsf{S}_{n,j}}{\mathsf{S}_n} \tag{9}$$

To map instance n to class c_j, a *predetermined threshold* is set, typically 0.5, on the $\bar{v}_{n,j}$. The position whose the $\bar{v}_{n,j}$ value is not smaller than the *threshold*

receives the value 1, and 0 otherwise. The more details about algorithm 2 are presented in the previous paper [16].

Algorithm 2 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

In the Algorithm 2 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.

4.3 Hyper-parameters Optimization for SOM

The machine learning model includes two types of parameters:

- Parameters of model: they are all the parameters (e.g. weights, bias) those values are changed (updated) through training process.
- Hyper-parameters: all parameters that are set independently before training and their values can not be changed through training process (e.g., an estimate of the number of people in a random forest).

In SOM, the hyper-parameters includes: n - the dimension of the SOM map, η - the learning rate and σ - the spreading of the neighborhood function (see Eq. 6).

To find the best hyper-parameters for the model, the best way is to optimize them by using different optimization algorithms [40]. Some algorithms that are used to optimize SOM model are listed below:

- Random search [41]: it is another version of grid search. It selects the values of parameters randomly from predefined set to find the best solution for the model. But the drawback of this method is missing some optimized values from search space.
- Tree-structured Parzen Estimators (TPE) [40, 41]: The main idea of this algorithm is to handle the hyper-parameters in a tree-structured form. The number of layers of neural nets and the number of neurons in each layer defines a tree structure. Given the configuration space χ, the TPE models

Table 1: The SOM classification results on hyper-parameters tuning

Device	Test	Algorithm				Algorithm			
		tpe	rand	atpe	anneal	tpe	rand	atpe	anneal
		Train on gafgyt				Train on Mirai			
D1	Gafgyt	0.998	0.994	0.997	0.998	0.996	0.978	0.994	0.881
	Mirai	0.662	0.777	0.796	0.709	0.998	0.995	0.998	0.994
D3	Gafgyt	0.995	0.995	0.994	0.994				
D5	Gafgyt	0.997	0.998	0.998	0.997	0.961	0.985	0.994	0.985
	Mirai	0.822	0.684	0.675	0.723	0.991	0.992	0.995	0.991
D6	Gafgyt	0.996	0.996	0.998	0.997	0.968	0.965	0.981	0.891
	Mirai	0.682	0.679	0.681	0.683	0.999	0.995	0.996	0.998
D8	Gafgyt	0.993	0.996	0.996	0.995	0.652	0.958	0.981	0.960
	Mirai	0.778	0.675	0.670	0.645	0.989	0.995	0.995	0.992

$p(x \mid y)$ by transforming the generative process, replacing the distributions of the configuration space is described using uniform, log-uniform, quantized log-uniform, and categorical variables.

- Adaptive TPE [40]: The Adaptive TPE (ATPE) is a variant of TPE, which yields a model over χ by placing density in the vicinity of K observations $\beta = x^{(1)}, ...x^{(K)} \subset H$ (H is a list of observed variables). Each continuous hyper-parameter was specified by a uniform prior over some interval (a, b) or a Gaussian, or a log-uniform distribution.
- Annealling [41]: This algorithm is a simple variation of random search which tries to select one of the previous testing points as the starting, and then sample each hyper-parameter from a distribution similar to the one specified in the prior.

In our experiments, we use the HyperOpt [1] (Bergstra et. al. [42]) to find the best algorithm for optimizing the SOM hyper-parameters. The steps to find the optimized values for hyper-parameters are following:

- Define a minimize function
- Define a search space
- Build a database in which all the current evaluation values of the search are stored
- Build the search algorithm

In experiments, devices (D1, D3, D5, D6, D8) are used for testing. SOM is trained and tested on the same/unknown attack type in the same device. As the result show in table 1, the best result is with ATPE algorithm. Since that, the ATPE is used inside the SOM to find optimized hyper-parameters.

[1] https://github.com/hyperopt/hyperopt

5 Experiments

In this section, we describe a set of experiments to evaluate the feature space of AEs for IoT Malware detection using unsupervised learning. To do this, we employ a SOM method as the subsequent classification-based method. In order to highlight the characteristics of AEs, the SOM working with other feature reduction method like PCA and the stand-alone SOM are assessed in comparison to the hybrid AEs and SOMs. We use the terms AESOM, DAE-SOM and PCASOM to refer to the hybrid an AEs and a SOM, the hybrid PCA and a SOM respectively. The rest of this section contains a description of the IoT dataset, hyper-parameter settings, and assessment metrics.

We set out to examine the properties of the AE feature space for unsupervised learning IoT Malware detection in three separate experiments. The experiments consists of *(1) IoT data analysis; (2) the ability in identifying unknown/new IoT attacks; and (3) the ability in transferring learning.* The details are as follows:

- The goal of IoT data analysis is to learn about the characteristics of three groups of data: benign, Gafgyt, and Mirai. This consideration is presented in this study [16],
- Unknown/new IoT attacks is to evaluate our proposed model on unknown/new IoT attacks from the same IoT device,
- Transfer learning is to evaluate our proposed model on the data from other devices.

All experiments are implemented in Python using Keras[2], Scikit-learn[3] and Minisom[4] frameworks.

5.1 Datasets

The NBaIoT dataset [5] was introduced by Y. Meidan et al. [17]. Data samples from NBaIoT are collected from nine different IoT devices, as described in Table 2. For each device, the two most popular kinds of IoT Malware 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 [43]. In addition, the NBaIoT dataset contains two main different groups of IoT attack types, namely Mirai and Gafgyt. Each group of IoT attacks consists of many sub-classes of attack types. However, our experiments focus on classifying benign and IoT attacks, thus Mirai and Gafgyt are examined instead of sub-classes of IoT attacks.

In our experiments, we use two groups of IoT devices: doorbell (D1 and D3) and camera (D5, D6 and D8) with different scenarios as presented in our

[2] https://keras.io/
[3] https://scikit-learn.org/
[4] https://github.com/JustGlowing/minisom
[5] https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT

Table 2: The nine IoT datasets in NBaoIoT

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

previous research [16]. The dataset is divided into two parts: train (70%) and test (30%). To avoid over-fitting, the early-stopping technique is used. For each IoT device, all benign data is used for training because it is much less than the amount of IoT Malware. The IoT Malware data for training is randomly selected from the original IoT Malware data with ratios of $0.01, 0.1, 1.0, 5.0$ in comparison to the amount of the benign data, respectively. The relative ratio between IoT attacks and benign data is signed as r.

5.2 Parameters Settings

Firstly, we set up the size of the encoded layer (bottleneck) for the AE and PCA. We choose the ratio between the size of the encoded layer and the original feature space as 0.25. Thus, the size of the bottleneck (encoded layer) is 29 ($0.25 * 115$). The sizes of other hidden layers are set as 0.75, 0.5, 0.33, 0.25 (the encoded layer) of the input layer's size, respectively. In training phase, we split the training data into training set (80%) and validation set (20%). The Adam method is used for parameter optimization with the loss function of Mean Squared Error (MSE). The maximum number of epochs is 50 (with the early-stopping method) and the batch size is 200. The *tanh* activation function is used in all layers.

The size of the bottleneck (encoded size) in PCA is set as the AE. All other PCA hyperparameters are set to default values. For training SOM, we use tuning techniques to search for the best hyperparameters. The hyper-parameters are n (dimension of SOM map), σ (the spreading of the neighborhood function) and *learning rate* η. Once these parameters found, SOM is trained to build the label-maps and determine the outlier percentage.

5.3 Evaluation Metrics

We utilize Area Under the Curve (AUC) for evaluating the performance of our proposed methods. To estimate AUC, the true positive rate (TPR) and

false positive rate (FPR) need to be calculated first by the following formulas.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN} \tag{10}$$

The Receiver Operating Characteristic curve (ROC curve) is received by plotting TPR against FPR at a number of thresholds. AUC can be 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

The experimental results are described and discussed in this section. The following tables and figures in this section provide more information about the results. In the next subsections, we present the reports and associated analysis for IoT data analysis, unknown IoT attack detection and transfer learning, respectively.

The analysis of the data was carried in previous research [16]. Based on the visualization and data investigation, we made conclusion that Mirai deviates significantly from both Benign and Gafgyt whereas Gafgyt and Benign may share some common features. Therefore, typical machine learning-based methods can achieve good performance on Mirai, but they may struggle to distinguish Gafgyt from Benign. Furthermore, the results of this analysis suggest that a training data comprised of Benign and Gafgyt may benefit AE learners while PCA may prefer the combination of Benign and Mirai. Due to this issue, we used the DAE to separate Gafgyt and Benign better. They can be considered as "noise" of each other. The DAE is used to deal with this "noise". In the following subsections, we will conduct a number of analyses on the results of the unknown/new IoT attack ability and the transfer learning ability to validate this suggestion.

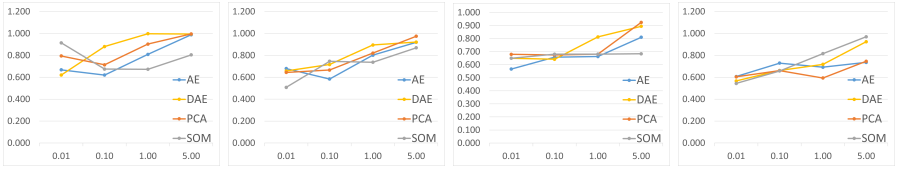
6.1 Unknown/new IoT Attack Detection

In this subsection, we present some analyses on the results of detecting unknown/new IoT attacks of our proposed models. The results of experiments are shown in Table 3 and AUC values are plotted in Fig. 2. For all the experiments, the models are trained on training data which consists of Mirai and Benign, and evaluated on Gafgyt, and vice versa. The device D3 is not included in these experiments because it misses Mirai. For each device, we report the AUC values produced by the three models on each setting of the training data, and also the mean and median over the four different settings of training data. The highest values amongst three methods on each setting of training data are highlighted in the gray color.

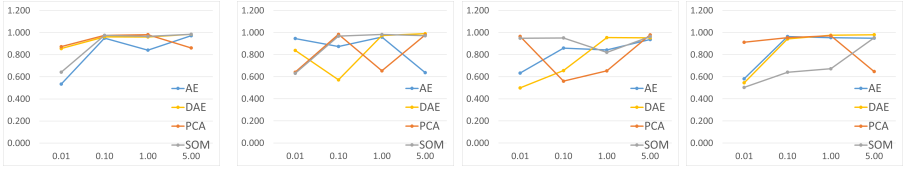
It can be seen from Table 3 that the DAESOM dominates other methods for attack detection. Because the DAESOM has filtered out noise from the input data and learn the best features of the data, it brought the better results than

Table 3: Unknown/new attack detection results on D1, D5, D6 and D8

Devices	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	0.10	1.00	5.00	mean	median	0.01	0.10	1.00	5.00	mean	median
		Train on Gafgyt - test on Mirai						Train on Mirai - test on Gafgyt					
D1	AESOM	0.668	0.620	0.809	0.988	0.771	0.738	0.535	0.950	0.841	0.972	0.825	0.896
	DAESOM	0.620	0.880	0.997	0.994	0.873	0.937	0.856	0.962	0.961	0.985	0.941	0.961
	PCASOM	0.795	0.713	0.903	0.996	0.852	0.849	0.874	0.976	0.982	0.861	0.923	0.925
	SOM	0.914	0.674	0.673	0.804	0.767	0.739	0.642	0.976	0.966	0.985	0.892	0.971
D5	AESOM	0.679	0.584	0.802	0.919	0.746	0.741	0.946	0.874	0.959	0.637	0.854	0.910
	DAESOM	0.657	0.716	0.895	0.921	0.797	0.806	0.839	0.572	0.973	0.990	0.844	0.906
	PCASOM	0.644	0.666	0.821	0.975	0.776	0.743	0.641	0.984	0.654	0.979	0.815	0.817
	SOM	0.508	0.746	0.737	0.869	0.715	0.742	0.631	0.968	0.983	0.972	0.889	0.970
D6	AESOM	0.567	0.657	0.663	0.810	0.674	0.660	0.634	0.859	0.843	0.937	0.818	0.851
	DAESOM	0.650	0.642	0.813	0.894	0.750	0.731	0.500	0.656	0.955	0.952	0.766	0.804
	PCASOM	0.680	0.674	0.680	0.924	0.740	0.680	0.965	0.560	0.654	0.979	0.790	0.810
	SOM	0.650	0.680	0.681	0.684	0.674	0.681	0.949	0.952	0.821	0.964	0.922	0.950
D8	AESOM	0.606	0.729	0.692	0.736	0.691	0.710	0.583	0.965	0.954	0.949	0.863	0.952
	DAESOM	0.566	0.661	0.718	0.924	0.717	0.689	0.546	0.943	0.976	0.981	0.861	0.959
	PCASOM	0.603	0.661	0.593	0.746	0.651	0.632	0.912	0.954	0.971	0.648	0.871	0.933
	SOM	0.543	0.657	0.816	0.970	0.747	0.737	0.504	0.641	0.673	0.953	0.693	0.657



(a) D1G - D1M (b) D5G - D5M (c) D6G - D6M (d) D8G - D8M



(e) D1M - D1G (f) D5M - D5G (g) D6M - D6G (h) D8M - D8G

Fig. 2: Unknown/new attack detection results on the same device

AESOM and remaining methods (PCASOM and pure SOM). The AUC values of DAESOM are increasing stable as the data rate increases. With traditional AE, the AESOM seems to perform better than PCASOM and pure SOM on some highly unbalanced settings, such as $r = 0.01$ and 0.1 while PCASOM prefers balanced training data ($r = 1.0$ and 5.0). When $r = 0.01$ and 0.1, benign dominates in training data. Thus, training data can be considered as a single class which will benefit the AE learners. On the other hand, PCA can yield good AUC values on $r = 1.0$. In the balance case, training data will have two equal classes, PCA can find a coordinate system in which the

training data can be highest separated. Moreover, the figure 2 illustrates some interesting characteristics of training data. All models trained on the training data of Benign and Gafgyt yield AUC values very similar to each others on the four ratios. While the AUC values of the models trained on the combination of Benign and Mirai tend to be deviated from each others.

6.2 Transfer Learning

The purpose of this subsection is to assess the ability to transfer models to various IoT devices. The models trained on a device using one kind of IoT attacks can be used to detect IoT attacks on other devices. We investigate two groups of transferring model knowledge such as doorbell (D1 and D3) and camera (D5, D6, D8). The detailed results are presented in the subsections that follow.

6.2.1 Transfer learning on the doorbell devices

There are two devices D1 and D3 in the doorbell. Firstly, the models are trained on Gafgyt and Mirai of the device D1, and tested on Gafgyt of the device D3 respectively. We then train the models on Gafgyt of the device D3, and test on Mirai and Gafgyt of the device D1, each. The experimental results are reported in Tables 4 and 5. The highest AUC values over three models are indicated in the gray color. Note that Mirai does not include in the data of the device D3.

Table 4: Transfer learning: train on D1 and test on D3

Test	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	0.10	1.00	5.00	mean	median	0.01	0.10	1.00	5.00	mean	median
		Train on D1 Gafgyt						Train on D1 Mirai					
D3G	AESOM	0.984	0.985	0.987	0.951	0.977	0.985	0.605	0.934	0.896	0.715	0.787	0.805
	DAESOM	0.970	0.995	0.995	0.964	0.981	0.982	0.627	0.968	0.968	0.938	0.875	0.953
	PCASOM	0.939	0.995	0.994	0.936	0.966	0.966	0.624	0.968	0.954	0.807	0.838	0.881
	SOM	0.981	0.991	0.993	0.948	0.978	0.986	0.615	0.957	0.944	0.794	0.827	0.869

When training on Gafgyt, DAESOM often outperforms the other methods in identifying both Gafgyt and Mirai from other devices. This can be seen from the first row on each devices in Tables 4 and 5. The reason can be that the AE learners may prefer to learn from data points that are not too separated from each other. In particular, the DAE has excluded noise from the input data and learned the nature of the data. Regard to the aspect, based on the data analysis [16], the combination of Benign and Gafgyt can be considered better than the training data consisting of Benign and Mirai.

On the other hand, we can see that PCASOM often yields good AUC values the balanced case ($r = 1.0$). The PCASOM has given the better results than AESOM and pure SOM. This suggests that PCA can create a better feature

Table 5: Transfer learning: train on D3 and test on D1

Test	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	0.10	1.00	5.00	mean	median	0.01	0.10	1.00	5.00	mean	median
		Train on D3 Gafgyt						No Mirai on D3					
D1G	AESOM	0.935	0.995	0.997	0.976	0.976	0.986						
	DAESOM	0.934	0.997	0.997	0.995	0.981	0.996						
	PCASOM	0.993	0.994	0.997	0.951	0.984	0.993						
	SOM	0.977	0.995	0.986	0.967	0.981	0.981						
D1M	AESOM	0.563	0.685	0.848	0.577	0.668	0.631						
	DAESOM	0.560	0.754	0.834	0.975	0.781	0.794						
	PCASOM	0.676	0.722	0.684	0.950	0.758	0.703						
	SOM	0.710	0.674	0.811	0.950	0.786	0.761						

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

Fig. 3: The AUC visualization for D1-D3

space for the case when training data consists of two highly separated classes. SOM working on the original data also produces the highest values in few cases.

The AUC scores are plotted against data ratio (see Fig. 3). The direction of the the line tends to permanently on training and testing on Gafgyt and fluctuating on training or testing on Mirai.

6.2.2 Transfer learning on camera devices

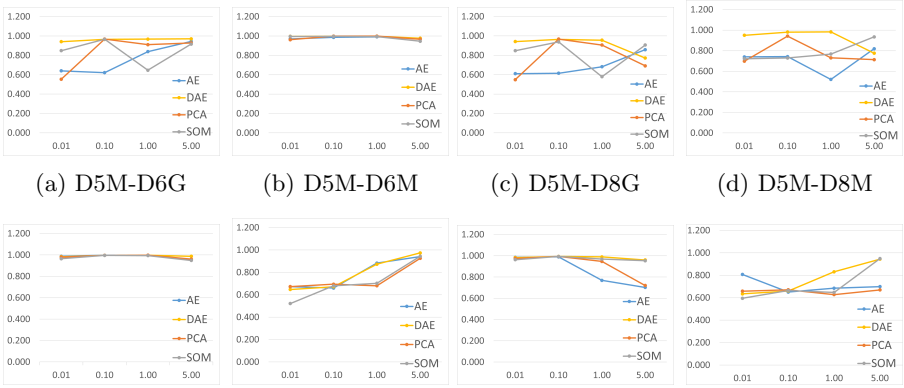
The camera group consists of the D5, D6, and D8 devices. D5 and D6 are the two device versions of the same brand, whereas D8 is from a different brand. In this experiment, we use the data from one device for training and data from the other two devices for testing. The experimental results are shown in Tables 6, 7 and 8.

The AUC scores are also plotted in Fig. 4. The AUC scores are the results of training on Gafgyt of device D5, and testing on D6 and D8. All algorithms DAESOM, AESOM, PCASOM and SOM give the better results when testing on Gafgyt (Fig. 4e and Fig. 4g), and worse when testing on Mirai (Fig. 4f, Fig. 4h).

The plot for D5-Mirai in shown in Fig. 4. This figure shows AUC scores for training on Mirai of device D5 and testing on D6 and D8. As described above, the Gafgyt data tends to be close to the benign, so it is more difficult when distinguishing between the benign and attack data. But as we see from the graph, the DAESOM is stable in detecting attacks with high AUC values.

Table 6: Transfer learning: train on D5 and test on D6 and D8

Test	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**0.01**	**0.10**	**1.00**	**5.00**	**mean**	**median**
		Train on D5 Gafgyt						**Train on D5 Mirai**					
	AESOM	0.988	0.997	0.996	0.958	0.985	0.992	0.640	0.622	0.839	0.944	0.761	0.739
	DAESOM	0.983	0.996	0.997	0.987	0.991	0.992	0.942	0.965	0.968	0.971	0.961	0.966
D6G	**PCASOM**	0.979	0.996	0.998	0.961	0.983	0.988	0.554	0.969	0.910	0.928	0.840	0.919
	SOM	0.964	0.995	0.994	0.948	0.975	0.979	0.849	0.966	0.646	0.916	0.844	0.882
	AESOM	0.672	0.660	0.883	0.941	0.789	0.777	0.969	0.986	0.991	0.974	0.980	0.980
	DAESOM	0.647	0.671	0.873	0.974	0.791	0.772	0.996	0.998	0.998	0.977	0.992	0.997
D6M	**PCASOM**	0.673	0.694	0.680	0.925	0.743	0.687	0.961	0.999	0.999	0.966	0.981	0.983
	SOM	0.521	0.680	0.702	0.943	0.712	0.691	0.995	0.996	0.994	0.946	0.983	0.995
	AESOM	0.985	0.989	0.768	0.702	0.861	0.877	0.610	0.615	0.681	0.858	0.691	0.648
	DAESOM	0.982	0.993	0.989	0.960	0.981	0.986	0.942	0.963	0.956	0.771	0.908	0.949
D8G	**PCASOM**	0.974	0.993	0.947	0.720	0.909	0.960	0.549	0.967	0.906	0.691	0.778	0.798
	SOM	0.963	0.993	0.969	0.953	0.969	0.966	0.848	0.938	0.579	0.906	0.818	0.877
	AESOM	0.807	0.650	0.685	0.699	0.710	0.692	0.740	0.743	0.521	0.819	0.705	0.741
	DAESOM	0.634	0.660	0.831	0.944	0.767	0.746	0.950	0.980	0.983	0.775	0.922	0.965
D8M	**PCASOM**	0.658	0.671	0.627	0.669	0.656	0.664	0.697	0.942	0.730	0.713	0.771	0.722
	SOM	0.595	0.664	0.647	0.949	0.714	0.656	0.722	0.729	0.767	0.934	0.788	0.748



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

(e) D5G-D6G (f) D5G-D6M (g) D5G-D8G (h) D5G-D8M

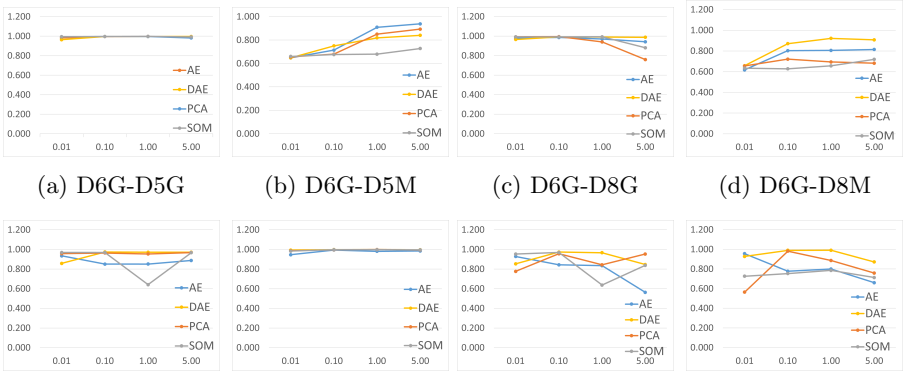
Fig. 4: AUC when train on D5, test on D6 and D8

In the remaining devices (D6, D8) we used the D6 and D8 as training devices, respectively. The trained models are applied later to the other devices (D5, D8 for the trained model from D6 and D5, D6 for the trained model from D8). The results are shown in the Table 7 and 8. The DAESOM continues giving better results than other methods. The AUC scores from test scripts are also plotted. Fig. 5 is a representation of testing results from device D6, while Fig. 6 is from device D8.

In experiments, DAESOM seems to perform better than AESOM, PCA-SOM, and SOM on almost devices when training data including Gafgyt. This

Table 7: Transfer learning: train on D6 and test on D5 and D8

Test	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**0.01**	**0.10**	**1.00**	**5.00**	**mean**	**median**
		Train on D6 Gafgyt						**Train on D6 Mirai**					
	AESOM	0.982	0.996	0.998	0.993	0.992	0.994	0.934	0.850	0.851	0.887	0.881	0.869
	DAESOM	0.964	0.996	0.997	0.997	0.989	0.997	0.857	0.973	0.972	0.971	0.943	0.971
D5G	**PCASOM**	0.994	0.997	0.997	0.982	0.992	0.995	0.957	0.963	0.953	0.968	0.960	0.960
	SOM	0.995	0.996	0.998	0.992	0.995	0.996	0.968	0.968	0.641	0.966	0.886	0.967
	AESOM	0.648	0.714	0.909	0.938	0.802	0.812	0.946	0.993	0.981	0.984	0.976	0.982
	DAESOM	0.648	0.750	0.819	0.841	0.764	0.784	0.993	0.997	0.998	0.996	0.996	0.997
D5M	**PCASOM**	0.656	0.679	0.850	0.893	0.770	0.765	0.986	0.996	0.998	0.994	0.994	0.995
	SOM	0.660	0.676	0.680	0.728	0.686	0.678	0.983	0.997	0.997	0.995	0.993	0.996
	AESOM	0.981	0.986	0.970	0.942	0.970	0.975	0.928	0.844	0.835	0.563	0.792	0.839
	DAESOM	0.965	0.992	0.991	0.988	0.984	0.989	0.853	0.973	0.966	0.846	0.909	0.910
D8G	**PCASOM**	0.993	0.994	0.940	0.760	0.922	0.967	0.776	0.956	0.843	0.952	0.882	0.898
	SOM	0.993	0.991	0.993	0.881	0.964	0.992	0.953	0.969	0.637	0.838	0.849	0.895
	AESOM	0.618	0.803	0.806	0.814	0.760	0.804	0.956	0.777	0.799	0.660	0.798	0.788
	DAESOM	0.658	0.871	0.922	0.907	0.839	0.889	0.928	0.990	0.990	0.872	0.945	0.959
D8M	**PCASOM**	0.655	0.721	0.694	0.682	0.688	0.688	0.564	0.981	0.886	0.758	0.797	0.822
	SOM	0.634	0.628	0.657	0.719	0.660	0.646	0.726	0.753	0.786	0.713	0.745	0.740



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

(e) D6M-D5G (f) D6M-D5M (g) D6M-D8G (h) D6M-D8M

Fig. 5: AUC when train on D6, test on D5 and D8

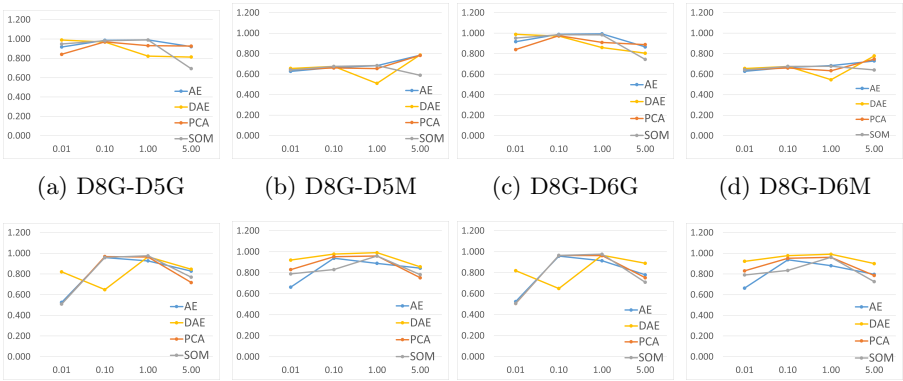
can be seen from the second row of the result table on each device, particularly on mean values. However, the table 8 does not demonstrate this point clearly on training on Gafgyt. This is because the device D8 was produced from a different brand to the devices D5 and D6. But in that case, the dataset is more "balanced" compare to other values of 0.01 and 0.1. Similar to the analysis of transferring model between D1 and D3, PCASOM and SOM stand-alone tend to produce high AUC values when training data is not too highly unbalanced.

The number of IoT devices in IoT networks is enormous, the IoT protocols are various, and the ratio between Benign and IoT malware is varied. It is

Table 8: Transfer learning: train on D8 and test on D5 and D6

Test	Models	Data Ratio				Metrics		Data Ratio				Metrics	
		0.01	**0.10**	**1.00**	**5.00**	**mean**	**median**	**0.01**	**0.10**	**1.00**	**5.00**	**mean**	**median**
		Train on D8 Gafgyt						Train on D8 Mirai					
	AESOM	0.917	0.986	0.990	0.921	0.954	0.954	0.525	0.958	0.926	0.828	0.809	0.877
	DAESOM	0.990	0.968	0.822	0.813	0.898	0.895	0.821	0.649	0.967	0.846	0.821	0.833
D5G	**PCASOM**	0.841	0.970	0.931	0.927	0.917	0.929	0.509	0.968	0.964	0.717	0.790	0.841
	SOM	0.949	0.982	0.990	0.693	0.903	0.965	0.509	0.960	0.975	0.769	0.803	0.864
	AESOM	0.627	0.665	0.683	0.785	0.690	0.674	0.662	0.938	0.889	0.843	0.833	0.866
	DAESOM	0.656	0.675	0.510	0.787	0.657	0.666	0.920	0.977	0.991	0.856	0.936	0.949
D5M	**PCASOM**	0.643	0.660	0.654	0.784	0.685	0.657	0.830	0.953	0.959	0.752	0.873	0.891
	SOM	0.641	0.675	0.684	0.589	0.647	0.658	0.789	0.830	0.959	0.781	0.840	0.809
	AESOM	0.917	0.989	0.993	0.866	0.941	0.953	0.524	0.957	0.914	0.779	0.794	0.847
	DAESOM	0.989	0.972	0.860	0.805	0.907	0.916	0.819	0.649	0.966	0.890	0.831	0.855
D6G	**PCASOM**	0.840	0.975	0.910	0.889	0.904	0.899	0.506	0.964	0.963	0.752	0.796	0.858
	SOM	0.952	0.985	0.984	0.745	0.916	0.968	0.508	0.965	0.976	0.709	0.790	0.837
	AESOM	0.628	0.665	0.683	0.729	0.677	0.674	0.663	0.937	0.880	0.796	0.819	0.838
	DAESOM	0.656	0.676	0.546	0.778	0.664	0.666	0.922	0.977	0.991	0.900	0.948	0.950
D6M	**PCASOM**	0.643	0.660	0.633	0.747	0.671	0.652	0.830	0.951	0.961	0.785	0.882	0.891
	SOM	0.642	0.676	0.677	0.642	0.659	0.659	0.790	0.835	0.961	0.726	0.828	0.813

beneficial to apply the trained model on an existing device to new devices. Based on the discussion in this section, it is highly recommended that the feature representation of AEs be utilized to detect both unknown attacks and transfer learning. The representation based on AEs learners combined with SOM appears to be best suited for transfer learning when training data is very imbalanced and contains IoT attacks similar to benign.



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

(e) D8M-D5G (f) D8M-D5M (g) D8M-D6G (h) D8M-D6M

Fig. 6: The AUC visualization for D8 (Train of Gafgyt)

7 Conclusions and Future work

This work investigates intensively and extensively the feature representation of DAEs for the unsupervised IoT Malware detection task. The study is operated to show different aspects of the proposed model such as identifying unknown/new attacks, transferring model knowledge (to build a detection model for different IoT devices, and to identify different IoT malware groups), and performing on highly unbalanced data. To extend our previous work, Denoising AEs is employed instead of AEs for enhancing the generalization ability of its latent representation as well as the hyper-parameter optimization task is involved in constructing hybrid DAEs and SOMs. Our models are evaluated and analyzed extensively by a number of experiments on the NBaIoT dataset. Well-known feature representation methods such as AEs and PCA are included in the experiments in order to highlight the behavior of DAE learners.

The experimental results strongly recommend that DAESOM can have the ability to transfer model knowledge: detecting IoT malware on different IoT devices, and identifying different IoT malware groups (also known as detecting new/unknown IoT malware). In particular, it can identify IoT attacks in some difficult situations such as highly unbalanced data and IoT malware whose behaviors being similar to benign. Developing better feature representation models for unsupervised IoT malware detection is a promising direction, and it is postponed to the near future.

Declarations

- Conflict of interest: The authors declare that they have no conflict of interest.
- Code availability: https://github.com/ladin157/AE-SOM-IoT-AD

References

[1] Dastjerdi, A.V., Buyya, R.: Fog computing: Helping the internet of things realize its potential. Computer **49**(8), 112–116 (2016)

[2] Ray, S., Jin, Y., Raychowdhury, A.: The changing computing paradigm with internet of things: A tutorial introduction. IEEE Design & Test **33**(2), 76–96 (2016)

[3] Abomhara, M., Køien, G.M.: Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. Journal of Cyber Security and Mobility, 65–88 (2015)

[4] Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: Ddos in the iot: Mirai and other botnets. Computer **50**(7), 80–84 (2017)

[5] Tsai, C.-W., Lai, C.-F., Chiang, M.-C., Yang, L.T.: Data mining for internet of things: A survey. IEEE Communications Surveys & Tutorials **16**(1),

77–97 (2013)

[6] Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015)

[7] Dinh, P.V., Shone, N., Dung, P.H., Shi, Q., Hung, N.V., Ngoc, T.N.: Behaviour-aware malware classification: Dynamic feature selection. In: 2019 11th International Conference on Knowledge and Systems Engineering (KSE), pp. 1–5 (2019). IEEE

[8] Hung, N.V., Dung, P.N., Ngoc, T.N., Phai, V.D., Shi, Q.: Malware detection based on directed multi-edge dataflow graph representation and convolutional neural network. In: the 11th KSE, pp. 1–5 (2019). IEEE

[9] 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: ICC 2019-2019 IEEE International Conference on Communications (ICC), pp. 1–6 (2019). IEEE

[10] 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)

[11] 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: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 756–767 (2019). IEEE

[12] Cao, V.L., Nicolau, M., McDermott, J.: Learning neural representations for network anomaly detection. IEEE transactions on cybernetics **49**(8), 3074–3087 (2018)

[13] Pang, G., Shen, C., Cao, L., Hengel, A.V.D.: Deep learning for anomaly detection: A review. ACM Computing Surveys (CSUR) **54**(2), 1–38 (2021)

[14] 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 (2016)

[15] Nguyen, V.Q., Nguyen, V.H., Le-Khac, N.-A., Cao, V.L.: Clustering-based deep autoencoders for network anomaly detection. In: International Conference on Future Data and Security Engineering, pp. 290–303 (2020). Springer

[16] Nguyen, H.N., Nguyen, V.C., Tran, N.N., Cao, V.L.: Feature representation of autoencoders for unsupervised iot malware detection. In: Future

Data and Security Engineering, pp. 272–290. Springer, Cham (2021)

[17] 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 (2018)

[18] Cao, V.L., Nicolau, M., McDermott, J.: A hybrid autoencoder and density estimation model for anomaly detection. In: International Conference on Parallel Problem Solving from Nature, pp. 717–726 (2016). Springer

[19] Bui, T.C., Cao, V.L., Hoang, M., Nguyen, Q.U.: A clustering-based shrink autoencoder for detecting anomalies in intrusion detection systems. In: 2019 11th International Conference on Knowledge and Systems Engineering (KSE), pp. 1–5 (2019). IEEE

[20] Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier detection using replicator neural networks. In: International Conference on Data Warehousing and Knowledge Discovery, pp. 170–180 (2002). Springer

[21] Song, C., Liu, F., Huang, Y., Wang, L., Tan, T.: Auto-encoder based data clustering. In: Iberoamerican Congress on Pattern Recognition, pp. 117–124 (2013). Springer

[22] Colombini, G.G., de Abreu, I.B.M., Cerri, R.: A self-organizing map-based method for multi-label classification. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 4291–4298 (2017). IEEE

[23] Rauber, A.: Labelsom: On the labeling of self-organizing maps. In: IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339), vol. 5, pp. 3527–3532 (1999). IEEE

[24] Tian, J., Azarian, M.H., Pecht, M.: Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. In: PHM Society European Conference, vol. 2 (2014)

[25] Ferles, C., Papanikolaou, Y., Naidoo, K.J.: Denoising autoencoder self-organizing map (dasom). Neural Networks **105**, 112–131 (2018)

[26] Wickramasinghe, C.S., Amarasinghe, K., Manic, M.: Deep self-organizing maps for unsupervised image classification. IEEE Transactions on Industrial Informatics **15**(11), 5837–5845 (2019)

[27] Bourlard, H., Kamp, Y.: Auto-association by multilayer perceptrons and singular value decomposition. Biological cybernetics **59**(4), 291–294 (1988)

[28] Hinton, G.E., Zemel, R.S.: Autoencoders, minimum description length, and helmholtz free energy. Advances in neural information processing systems **6**, 3–10 (1994)

[29] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT press, ??? (2016)

[30] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., Bottou, L.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research **11**(12) (2010)

[31] Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1-3), 37–52 (1987)

[32] Jolliffe, I.: Principal Component Analysis, Encyclopedia of Statistics in Behavioral Science, 30 (3), 487 (2002)

[33] Schölkopf, B., Smola, A., Müller, K.-R.: Kernel principal component analysis. In: International Conference on Artificial Neural Networks, pp. 583–588 (1997). Springer

[34] Zou, H., Hastie, T., Tibshirani, R.: Sparse principal component analysis. Journal of computational and graphical statistics **15**(2), 265–286 (2006)

[35] Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE journal **37**(2), 233–243 (1991)

[36] Candès, E.J., Li, X., Ma, Y., Wright, J.: Robust principal component analysis? Journal of the ACM (JACM) **58**(3), 1–37 (2011)

[37] Kohonen, T.: The self-organizing map. Proceedings of the IEEE **78**(9), 1464–1480 (1990)

[38] Kohonen, T.: Essentials of the self-organizing map. Neural networks **37**, 52–65 (2013)

[39] Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. Journal of Big data **3**(1), 1–40 (2016)

[40] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. Advances in neural information processing systems **24** (2011)

[41] Yu, T., Zhu, H.: Hyper-parameter optimization: A review of algorithms and applications. arXiv preprint arXiv:2003.05689 (2020)

[42] Bergstra, J., Yamins, D., Cox, D.: Making a science of model search:

Hyperparameter optimization in hundreds of dimensions for vision architectures. In: International Conference on Machine Learning, pp. 115–123 (2013). PMLR

[43] Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089 (2018)