A Novel Transfer Learning Model for Intrusion Detection Systems in IoT Networks

Ly Vu[1], Quang Uy Nguyen[2], Dinh Thai Hoang*[1],
Diep N. Nguyen[1], and Eryk Dutkiewicz[1]

[1]University Technology of Sydney, NSW, Australia
[2]Le Quy Don Technical University, Hanoi, Vietnam
{Ly.T.Vu}@student.uts.edu.au,{uynq}@lqdtu.edu.vn
{hoang.dinh,diep.nguyen,eryk.dutkiewicz}@uts.edu.au

Abstract. Internet of Things (IoT) has been playing an important part in human's lives. Nonetheless, the rapid growth of IoT-based services may lead to an increase in IoT-based attacks that compromise sensitive user data. IoT-based attack detection and prevention is thus essential for the development of IoT-based network technologies. However, this is a difficult task because the data transmitted between IoT devices is often complex and heterogeneous while IoT-based threats/attacks change rapidly over time. Machine learning-based threat detection methods, that are trained with old/existing labelled data, may become less effective in the future. For that, it is critical to develop effective learning models that can leverage both labelled and unlabelled data to timely adapt with the fast-varying attack methods. To this end, this chapter proposes a Deep Transfer Learning (DTL) model to build an effective IoT attack detection model from both labelled and unlabelled data collected from multiple IoT devices. The proposed solution combines the Multi Maximum Mean Discrepancy (M2D) distance and two AutoEncoder (AE) networks named as Multi Maximum Mean Discrepancy-AutoEncoder (M2DA). The first AE is trained on labelled IoT data, while the second AE is trained on unlabelled IoT data. The purpose of the training process is to transfer the learned label information from the first AE to the second AE by the M2D distance. As a result, the second AE can efficiently classify and detect anomaly (attacks) even on unlabelled IoT network data. We further study the performance of the proposed M2DA framework using nine well-known commercial IoT data sets with different botnet families and attacks.

Keywords: Deep Transfer Learning, IoT, Attack Detection, Intrusion Detection, Machine Learning.

1 Introduction

The Internet of Things, or IoTs, describes objects and sensors that are integrated in electronic devices, buildings, and vehicles to perform transmission functions in a wireless environment. With the strong development of the Internet today, IoT devices bring more and more benefits to human lives. [1] (e.g., in healthcare, smart city, smart transportation, smart building). However, this fast growth has also led to a sharp increase in cyber attacks aimed at harming Internet and IoTs users. According to a study in [2], network attacks have caused damage valued at more than US$500,000, including lost revenue, customers, opportunities, and so on.

With expanding IoT network, attackers are exploiting undefended gaps in the security of network environments [2]. Un-patched and un-monitored IoT devices are easily attacked to destroy the operation of IoT networks. Meanwhile, IoT attacks are developing along with the IoT network becoming more automated. Moreover, the resources (power and computation) of IoT devices are often limited, making it difficult for them to deploy effective protection methods [3,4]. Consequently, development of IoT attack detection systems to protect IoT devices from IoT attacks is a crucial task to expand the applications of IoTs [5–7].

An IoT attack detection system performs network traffic data analysis to detect anomalous behavior in IoT network. Basically, three popular approaches are often used for analyzing network traffic to detect network attacks [8], i.e., knowledge-based methods, statistic-based methods, and machine learning-based methods. First, in order to detect IoT attacks, knowledge-based methods generate network attack rules or signatures to match network behaviors. The popular knowledge-based method is an expert system that extracts features from training data to build the rules to classify new traffic data. Knowledge-based methods can detect attacks robustly in a short time. However, they need high-quality prior knowledge about cyber attacks. Moreover, these methods are often unable to detect zero-day attacks.

Second, statistic-based methods consider network traffic activity as normal traffic. In the sequel, an anomaly score is calculated by some statistical methods on the currently observed network traffic data. If the score is more significant than a certain threshold, it will raise the alarm for this network traffic [8]. There are several statistical methods, such as information gain, information entropy and conditional entropy [9]. These methods explore the network traffic distribution by capturing the essential features of network traffic. Then, the distribution is compared with the predefined distribution of normal traffic to detect anomalous behaviors.

Third, machine learning-based methods to detect network attack are of paramount interest recently because they provide highly efficient attack detection models [3, 10, 11]. The main idea of machine learning methods is to build a model for detecting attacks automatically based on training datasets. As a result, the attack detection model can identify the attack traffic based on the general attributes of network traffic behaviours.

Although machine learning, especially deep learning, has achieved great success in network attack detection, there are still some issues that can affect the accuracy of IoT attack detection systems. One of these issues is that the machine learning based methods are usually based on the assumption that we can collect labelled data of both normal and attack classes. However, the labelling process is usually performed manually by humans, which is time-consuming and expensive. Thus, in some problem domains, e.g., IoT environment, due to the quick evolution of network attacks, it is often unable to label for all samples when they are collected from different IoT devices. In other words, it is desirable to develop detection models that can detect attacks to various IoT devices without labelled information.

To handle the above issue, this chapter introduces a novel deep transfer learning (DTL) model based on AutoEncoders (AEs). The new model is called Multi-Maximum Mean Discrepancy AE (M2DA) that includes two AEs. M2DA is trained using both labelled dataset (source domain) and unlabelled dataset (target domain). Here, source domain and target domain are network traffic data collected from two different IoT data. By this way, the knowledge (label information) is transferred from the source domain to the target domain, thereby improving accuracy of IoT attack detection on the target domain. Specifically, the first AE in M2DA (AE_1) inputs data and label from the source domain while the second AE (AE_2) inputs only data from the target domain. In the training phase, the latent representation of AE_2 is learnt to be close to the latent representation of AE_1. Consequently, the latent representation of AE_2 can classify data from the target domain more correctly.

The major contributions of this chapter include:

- We introduce a novel DTL model namely M2DA to use the knowledge of label information of a related data domain. This model aims to alleviate the problem of "lack label information" in IoT network traffic.
- We propose to use the Maximum Mean Discrepancy (M2D) metric to make the latent representation of AE_1 and those of AE_2 to be as close as possible. This metric is also very helpful to enhance the efficiency of the transferring knowledge between AE layers.
- We conduct a large number of experiments using nine IoT attack datasets to investigate the effectiveness of M2DA. The results are compared with the canonical AutoEncoder and the recently proposed DTL models [12, 13].

2 Background

2.1 Transfer Learning

A model that can use the knowledge learned from a source problem for a learning task on a target problem is a Transfer Learning (TL) technique. Usually, the target problem is different from the source problem but they are related data distributions. Here, we define two IoT datasets as the source problem and the target problem that present two related data distributions. More specifically, we can consider two IoT data distributions that are different due to being collected from different IoT devices but related data distributions due to presenting normal and IoT attacks.

We assume that X is the input data space and Y is its label space. D_S and D_T are the domain distributions of a source problem and a target problem, respectively. The source domain has data and label as $D_S = (X_S, Y_S) = (x_S^i, y_S^i)_{i=1}^{n_S}$ and the target domain does not have a label set, i.e., $D_T = (X_T) = (x_T^i)_{i=1}^{n_T}$. n_S and n_T present the numbers of data samples of the source domain and the target domain, respectively. The learning task is transferring label information from the source domain to the target domain. As a result, the TL model can classify data in the target domain. By minimizing the distance between the source domain and the target domain at the latent representation space, the latent representation of the target domain is easy to be classified as the source domain. In the TL field, researchers usually use Maximum mean discrepancy (M2D) [14] or Kullback Libler divergence (KL) [13] due to their effectiveness to minimize distance between the two data distributions. Moreover, TL that is based on deep neural networks is called Deep Transfer Learning (DTL). In this chapter, we present a DTL method based on a neural network, namely AutoEncoder (AE) to enhance the accuracy in IoT attack detection. The next subsection will present the fundamental background of the M2D metric and AE.

2.2 Maximum Mean Discrepancy (M2D)

Similar to Kullback-Libler divergence (KL) [13], M2D is used to quantify the distance between two distributions. Comparing to KL, M2D is more flexible due to the ability of estimating the nonparametric distance [14]. Moreover,

M2D can avoid computing the intermediate density of distributions. The calculation of M2D is presented as Eq. (1).

$$\text{M2D}(X_S, X_T) = |\frac{1}{n_S}\sum_{i=1}^{n_S} \Gamma(x_S^i) - \frac{1}{n_T}\sum_{i=1}^{n_T} \Gamma(x_T^i)|, \tag{1}$$

where n_S and n_T are the numbers of samples in the source and target problems, respectively, Γ present the representation of the original data x_S^i or x_T^i.

2.3 AutoEncoder

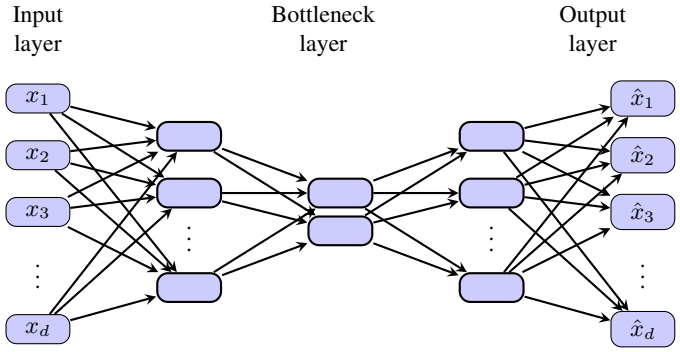


Fig. 1: Architecture of AutoEncoder (AE).

An AutoEncoder (AE) is a neural network that has two parts: an encoder and a decoder. The aim of this neural network is reconstructing its input at its output [15–17]. As shown in Fig. 1, the encoder inputs the data $x = x_1, x_2, \ldots, x_d$ and outputs the latent representation of x, i.e, z where d is the number of dimension of the input data x. The decoder receives z as input and its output is $\hat{x}$. The objective function of AE aims to minimize the distance between x and $\hat{x}$. Thus, x and $\hat{x}$ have the same dimension as d.

We denote $\psi = \{W, b\}$ and $\omega = \{W', b'\}$ to be the parameter sets of the encoder and decoder where W, W', b, and b' are the weights and the biases, respectively. Let $X = x^1, x^2, ..., x^n$ be the training dataset. The q_ψ projects the input x^i to the latent representation z^i. The decoder p_ω aims to map the latent representation z^i to output $\hat{x}_i$. The output $\hat{x}_i$ has a similar structure as the input x_i. The calculations of the encoder and decoder are described in Eq. (2) and Eq. (3), respectively.

$$z = q_\psi(x) = a_f(Wx + b), \tag{2}$$

$$\hat{x} = p_\omega(z) = a_g(W'z + b'), \tag{3}$$

where a_f and a_g represent the activation functions of the encoder and the decoder, respectively. Figure 1 illustrates an AE structure where the input dimension, the number of layers, and the bottleneck layer size are d, 5, and 2, respectively.

Specifically, the objective of AE is to reconstruct the input layer at the output layer. Thus, the training process of AE aims to minimize the Reconstruction error (RE). RE is the loss function of AE described in Eq. (4). This is the expected negative log-likelihood of the i-th data sample. The expectation is taken over the representation with respect to the encoder's distribution. Minimizing this loss function motivates the decoder to reconstruct the input data. If the decoder can not reconstruct the input data well, it will incur a cost in this loss function.

$$\delta_{AE}\left(x^i, \psi, \omega\right) = \mathbf{E}_{q_\psi(z|x^i)}\left[logp_\omega(x^i|z)\right]. \tag{4}$$

The training process is minimizing the summation of the loss function, $\epsilon\left(x^i, \hat{x}^i\right)$[1], over all training data samples. This loss function is the difference between the input x^i and the output $\hat{x}^i$. The mean squared error (MSE) and cross-entropy loss are commonly used to represent the difference between input and output data. Thus, the loss function of AE in Eq. (4) is rewritten as Eq. (5).

$$\delta_{AE}\left(x^i, \psi, \omega\right) = \frac{1}{n}\sum_{i=0}^{n} \epsilon\left(x^i, \hat{x}^i\right). \tag{5}$$

[1] $\epsilon\left(x, \hat{x}\right) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(x^i - \hat{x}^i\right)^2}$.

3 Related Work

Using machine learning to detect network attacks has been widely used in recent studies. [3, 17–19]. A real-time unsupervised network anomaly detection algorithm that uses a discrete-time sliding window to continuously update the feature space and clustering to detect anomalies has been proposed by Juliette et al. [18]. [19] proposed a TL technique that is used to estimate a temporal property of the stream. They used statistically robust control charts to recognize deviations. However, this technique highly depends on the selected threshold value. Another work [17] tried to represent network traffic data into a latent representation space to improve the classification tasks. However, general machine learning algorithms need to assume that the data to be predicted in the future has a similar distribution to the data used to train the model. However, there are many data analysis applications that do not guarantee this assumption [20, 21].

Therefore, TL technologies including Deep Transfer Learning (DTL) are being widely used in data analysis applications today. Based on using deep neural networks, DTL can be categorized into four groups, such as instance based DTL, mapping based DTL, network based DTL, and adversarial based DTL [21–23].

The instance based DTL method selects data samples from the source domain to complement the training set in the target domain. These data samples are assigned appropriate weights in the training set in the target domain. [24] proposed the bi-weighting domain adaptation that can map the feature spaces of both source and target domains to the common coordinate system. In the new representation space of features, samples in the source domain are assigned to appropriate weights. To learn sample weights, [25] introduced a DTL metric framework together with learning a distance between two domains. This framework makes knowledge transfer between domains more effective. An ensemble DTL introduced in [26] can leverage samples from the source domain to train on the target domain.

The mapping based DTL approach aims to map samples from both the source domain and target domain into a new feature space. Thus, in the new representation space, the source domain representation and the target domain representation are similar and are considered as a training set of a deep neural network. The work in [27] introduced an adaptation layer and an additional domain confusion loss based on M2D to learn a new representation space. In this new representation space, the source domain and target domain are invariant. To measure the distance of the joint distributions, joint M2D was introduced in [28] to transfer the data distribution between different domains.

The adversarial based DTL approach refers to the use of adversarial networks inspired by generative adversarial nets (GAN) [29] to get a representation space. This new representation space is suitable to both the source domain and target domain. [30] introduced a new loss function of GAN combining a discriminative model with a DTL method. A randomized multi-linear adversarial network was introduced in [31] to find the multiple feature layers.

The network based DTL approach reuses the network structure and parameters trained in the source domain for training the target domain. [32] used some pretrain weights of front-layers of Convolutional Neural Network (CNN) trained on the ImageNet dataset to map images of other datasets to intermediate image representation. It helps to transfer knowledge learned from a big dataset, e.g., Imagenet, to other object recognition tasks with a small training set size. [33] introduced a method to learn adaptive classifiers with a residual function. Several DTL approaches based on AEs were introduced in [12, 13, 34]. They used different AE-based models such as AE [13], denoising AE [34], and sparse AE [12] for some specific TL applications. In previous work based on AE, the transferring process is executed only on the bottleneck layer. They may use the Kullback–Leibler divergence (KL) metric [13] or M2D metric [12] to minimize the representation data at the bottleneck layers of the source domain and the target domain.

To take the advantages of deep neural networks for learning network traffic representation, we develop a network based DTL method. In the IoT environment, we are unable to label data collected from all IoT devices. To handle this issue, we only label IoT traffic data from several IoT devices and transfer the label information to other devices to enhance accuracy in the IoT attack detection problem. More specifically, we can transfer knowledge at the higher level of features from the source domain with labels to the target domain without labels. This approach improves the accuracy of learning tasks on the target domain with limited samples and no labels.

In this chapter, we introduce a DTL model, i.e., M2DA that uses a non-linear mapping of a neural network (i.e., AE) to enhance the accuracy of IoT attack detection on some datasets collected IoT devices without label information. The key idea of the M2DA model (compared with previous AE-based DTL methods [12, 13]) is that the transferring task is executed on *every encoding layer* of AE while it is only executed in the bottleneck layer (latent representation) of AE in previous work. It helps to greatly encourage the latent representation of the target domain to be similar to the latent representation of the source domain. The results illustrate that using M2DA can enhance the accuracy of the classification task at the target domain for IoT attack detection problems.

4 Proposed Deep Transfer Learning Model

In this section, we first describe the system architecture of the IoT attack detection system. Then, we present the M2DA model in detail.

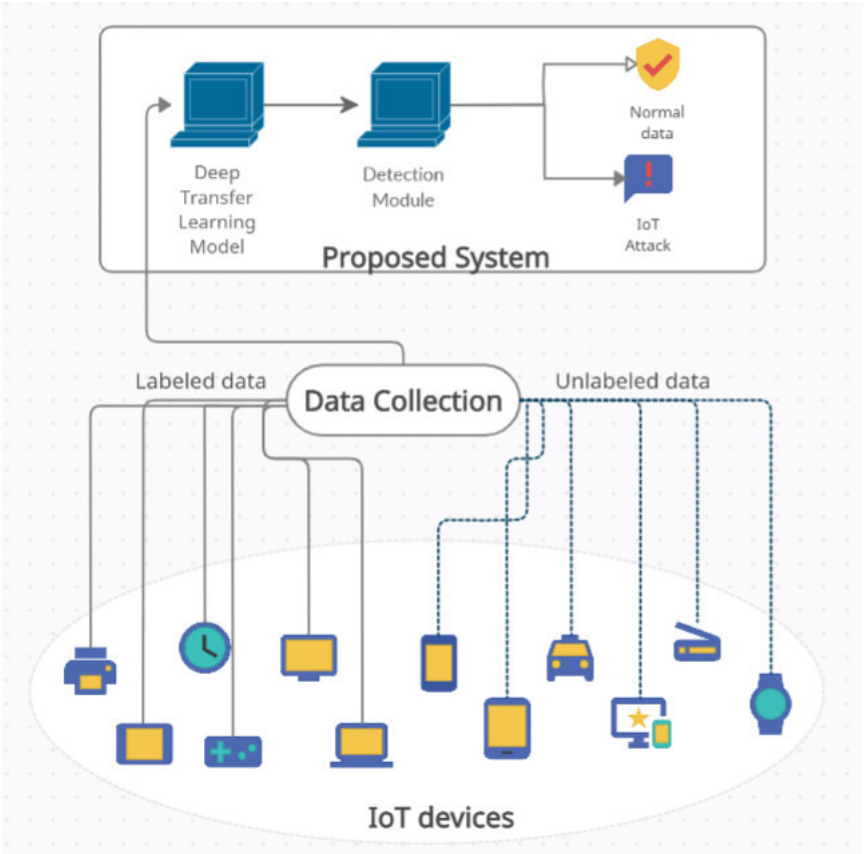


Fig. 2: Proposed system structure.

4.1 System Structure

Fig. 2 illustrates the architecture of IoT attack detection model that uses a DTL model. The system include two phases. In the first phase, we gather data from the IoT devices using a data collection function. This data can be labelled or not depending on the cost of labelling. We can select to label data from some IoT devices. The labelling process usually includes the following two steps [37]. First, the Tcptrace tool is used to extracted data samples from the captured network traffic [38]. Second, the data samples are labelled manually by analyzing the network flow by a tool, e.g., the Wireshark tool [39]. Due to the expensive cost of the second step and the large number of IoT devices in the system, the number of IoT devices that have labelled data are often smaller than those with unlabelled data.

In the second phase, the data samples in the form of feature vectors are used for training the M2DA model. During the training, the labelled information is used to adjust the latent representation of the first AE in M2DA. Moreover, the latent representation of the second AE is also trained to mimic that of the first AE. This is achieved by minimizing the distance between the latent representations of the first and the second AE. When the training is completed, the DTL model is used to classify the incoming traffic from the IoT devices into normal or attack. We will describe the DTL model in detail in the next subsection.

4.2 Multi-Maximum Mean Discrepancy AutoEncoder

This subsection introduces the Multi-Maximum Mean Discrepancy AutoEncoder (M2DA). M2DA (Fig. 3) consists of two AEs (i.e., AE_1 and AE_2) that are trained together. The first AE, i.e., AE_1 is trained on the data samples of the source problem x_S while the second AE, i.e., AE_2, is trained on the data samples of the target problem x_T. The loss function of M2DA has three components: The first component is the reconstruction error, (δ_{RE}), the second component is the the supervised error, (δ_{SE}), and the third component is the Multi-Maximum Mean Discrepancy (δ_{M2D}).

Define ψ_S, ω_S and ψ_T, ω_T be the weight set of the encoder and decoder of AE_1, AE_2, respectively, respectively. The reconstruction errorδ_{RE} is the summation of RE of AE_1, i.e., RE_S and RE of AE_2, i.e., RE_T in Fig. 3. This term aims to force the output of AE_1 and AE_2 closely to their input. Specifically, the RE_S attempts to reconstruct x_S at $\hat{x}_S$ and RE_T tries to reconstruct x_T at $\hat{x}_S$. Since the output of the AEs is constructed from the latent vector, the RE term helps the AEs to encode the useful information of the input data at the latent vector.

Thus, the latent representation can be used as a new representation for the input data. In other words, we can do the classification task on the latent vector instead of the original data. Based on above analysis, the δ_{RE} term is computed as follows:

$$\delta_{\text{RE}}(x_S^i, \psi_S, \omega_S, x_T^i, \psi_T, \omega_T) = \epsilon(x_S^i, \hat{x}_S^i) + \epsilon(x_T^i, \hat{x}_T^i), \tag{6}$$

where $\epsilon()$ is the mean square error function [17], x_S^i, $\hat{x}_S^i$ are input and output of the source domain, respectively. x_T^i and $\hat{x}_T^i$ are the input and the output of the target domain, respectively.

The second component in the M2DA loss δ_{SE} aims to classify the input sample AE_1 into normal or attack using the labelled information. Specifically, this term tries to map the latent representation of AE_1, i.e., z_S, to the label y_S of the input sample. We use a softmax function [16] to learn to map z_S to y_S. To calculate the softmax function, z_S and y_S must have the same dimension. Therefore, the size of the latent vector must equal to the number of the class label. This loss term helps to distinguish the latent representation space into separated class, e.g., normal and attack. Thus, it can be defined as the following equation:

$$\delta_{\text{SE}}(x_S^i, y_S^i, \psi_S.\omega_S) = -\sum_{j=1}^{C} y_S^{i,j} \log(z_S^{i,j}), \tag{7}$$

where z_S^i and y_S^i are the latent vector and label of the $i-th$ data sample in the source problem x_S^i. $y_S^{i,j}$ and $z_S^{i,j}$ represent the $j-th$ element of the vector y_S^i and z_S^i, respectively.

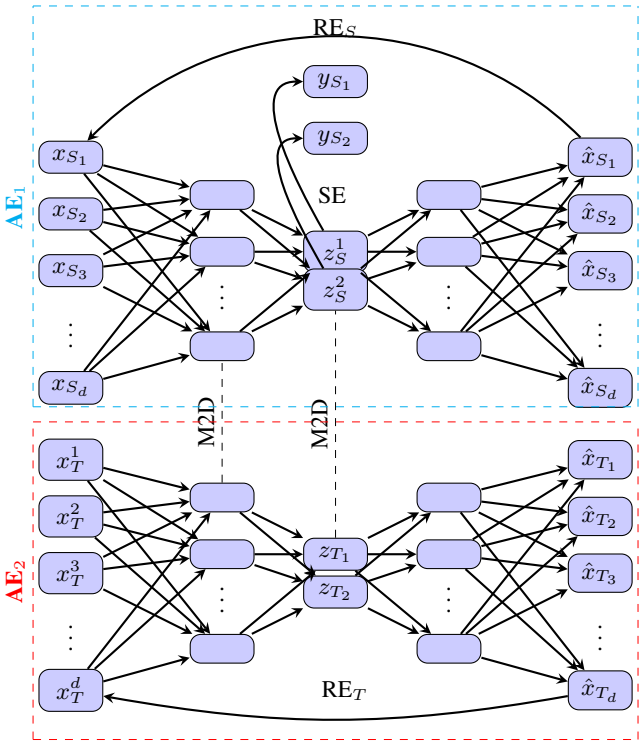


Fig. 3: Architecture of M2DA.

The third component of the M2DA loss, δ_{M2D}, aims to transfer the knowledge from the first AE to the second AE. This is achieved by minimizing the M2D distance between every layer in the encoder of AE_1 and the corresponding layer in the encoder of AE_2. Thus, this loss forces the latent vector of the AE_2 (the target problem) closely to the latent vector of the AE_1 (the target problem). The δ_{M2D} loss is calculated in Eq. (8):

$$\delta_{\text{M2D}}(x_S^i, \psi_S, \omega_S, x_T^i, \psi_T, \omega_T) = \sum_{k=1}^{K} \text{M2D}(\Gamma_S^k(x_S^i), \Gamma_T^k(x_T^i)), \tag{8}$$

where K is the number of layers of the encoders in the AEs. $\Gamma_S^k(x_S^i)$ and $\Gamma_T^k(x_T^i)$ are the k-th layer of the encoders of AE_1 and AE_2, respectively, $\text{M2D}(,)$ is the Multi-Maximum Mean Discrepancy distance in Eq. (8). Finally, the loss of M2DA consists of the loss components in Eq. (6), Eq. (7), and Eq. (8) are rewritten in Eq. (9).

$$\delta = \delta_{\text{SE}} + \delta_{\text{RE}} + \delta_{\text{M2D}}. \tag{9}$$

The key idea of M2DA compared with the previous transfer learning model [12, 13] is that the transferring task is executed in multiple encoding layers. The previous models [12, 13] transfer only the bottleneck layer from the first to the second AE. Therefore, M2DA allows more knowledge to be transferred from the source problem (AE1) to the target problem (AE2). One possible drawback of M2DA is that its training process is more time consumming due to the multiple calculations of the distance between the layers in the encoders of AE_1 and AE_2. However, in the inferring process, only AE_2 is used to detect IoT attacks in the target problem. Thus, the M2DA model does not increase the detection time compared to the other models [12, 13].

4.3 Training and Predicting Process using M2DA

Training Process Algorithm 1 presents the steps for training the M2DA model. The first step is to normalize data samples from the source domain and the target domain. The second step is fitting the data x_S and label y_S to input of AE_1 and fitting the data x_T to input of AE_2. The final step is the training process that is executed by minimizing the loss term Eq. (9).

After training, we have the trained M2DA model that includes two AEs.

Algorithm 1 Training process of M2DA.

INPUT:
Data sample set x_S and corresponding label set y_S of the source domain
Data sample set x_T of the target domain
OUTPUT: Trained M2DA model.
BEGIN:
Step 1. Normalizing x_S and x_T
Step 2. Fitting x_S and y_S to AE_1, fitting x_T is to AE_2.
Step 3. Training M2DE by minimizing the loss function Eq. (9).
return Trained M2DE model including two parts: AE_1 and AE_2.
END.

Predicting Process After training, the weights of AE_2 are used to predict a new data sample of the target domain in four steps as in Algorithm 2. First, a network traffic data sample $x_{i,T}$ is extracted as the feature vector and normalized. Second, we put it to the input of AE_2. Third, we take $z_{i,T}$ as the latent representation of $x_{i,T}$ by AE_2. Finally, we calculate the label $y_{i,T}$ by the softmax function. The predicting process is calculated based on only one AE, i.e., AE_2. Consequently, the predicting process time is similar to an AE-based model.

Algorithm 2 Predicting process of the M2DA model.

INPUT:
$x_{i,T}$: A testing sample of the target domain.
INPUT:Trained weights of AE_2 model, a data sample $x_{i,T}$
OUTPUT: Label $y_{i,T}$
BEGIN:
Step 1. Normalizing $x_{i,T}$
Step 2: Putting normalized $x_{i,T}$ vector to the input of the trained AE_2
Step 3: Calculating $z_{i,T}$ is the latent representation of $x_{i,T}$ at the bottleneck layer of AE_2
Step 4. Calculating $y_{i,T}$ = softmax ($z_{i,T}$)
return $y_{i,T}$
END.

5 Experiment Description

5.1 Bot-IoT Datasets (IoT Datasets)

In the simulations, we consider 9 IoT attack datasets [3] that were collected from 9 IoT devices under 2 common IoT botnet types, These datasets were collected from nine commercial IoT devices with two most popular IoT botnet families, Mirai and BASHLITE (Gafgyt). Each botnet type includes 5 dissimilar IoT attacks. Here, it is noted that 3 datasets IoT3, IoT6 and IoT7 contain only one type of botnet type. Thus, each of these datasets includes five types of IoT attacks generated from one type of botnet family. All other datasets contain both Mirai and BASHLITE attacks. Thus, one of these datasets includes ten types of IoT attacks. The features are designed for over a sliding window of 100 connections as in Table 1:

We encode the categorical feature by one-hot encoding and remove some identify features, such as source IP address (saddr), destination IP address (daddr). Then, each data sample has 115 features. These features can be categorized into three groups: stream aggregation, time-frame, and the statistics attributes. We divide each IoT dataset into a training set and a testing set. The testing set contains the different IoT attacks compared with attacks in the training set. The details of nine IoT datasets are presented in Table 2.

Table 1: Feature description of IoT datasets.

Feature name	**Feature description**
pkSeqID	Identifier of row
Stime	Start time to collect sample
flgs	State of flow flags in transactions
flgs number	Representation of flags in number
Proto	Representation of transaction protocols present in network flow
proto number	Representation of protols in number
saddr	Source IP address
sport	Source port number
daddr	Destination IP address
dport	Destination port number
pkts	Total packets in transaction
bytes	Number of bytes in transaction
state	State of transaction
state number	Representation of feature state in number
ltime	Last time of collected data sample
seq	Sequence number
dur	Total duration of collected data sample
mean	Average duration of aggregated data samples
stddev	Standard deviation of aggregated data samples
sum	Total duration of aggregated data samples
min	Minimum duration of aggregated data samples
max	Maximum duration of aggregated data samples
spkts	Number of Source-to-destination packets
dpkts	Number of Destination-to-source packets
sbytes	Number of Source-to-destination bytes
dbytes	Number of Destination-to-source bytes
rate	Total packets per second in transaction
srate	Number of Source-to-destination packets per second
drate	Number of Destination-to-source packets per second
TnBPSrcIP	Total bytes per source IP
TnBPDstIP	Total bytes per destination IP
TnP_PSrcIP	Total packets per source IP
TnP_PDstIP	Total packets per destination IP
TnP_PerProto	Total packets per protocol
TnP_Per_Dport	Total packets per destination port
AR_P_Proto_P_SrcIP	Mean of rate per protocol per source IP (calculated by packets per duration)
AR_P_Proto_P_DstIP	Mean of rate per protocol per destination IP
N_IN_Conn_P_SrcIP	Number of inbound connections per source IP
N_IN_Conn_P_DstIP	Number of inbound connections per destination IP
AR_P_Proto_P_Srcport	Mean of rate per protocol per source port
AR_P_Proto_P_Dstport	Mean of rate per protocol per destination port
Pkts_P_State_P_Protocol_P_DestIP	Number of packets grouped by state of flows and protocols per destination IP
Pkts_P_State_P_Protocol_P_SrcIP	Number of packets grouped by state of flows and protocols per source IP

5.2 Evaluation Metric

We use the AUC score to evaluate the effectiveness of our proposed model. AUC stands for Area Under Receiver Operating Characteristics (ROC) Curve [40] that is calculated by plotting the True Positive Rate (*TPR*) against the False Positive Rate (*FPR*) at various threshold settings of a classifier. We assume that the classifier need to classify positive data samples and negative data samples. The *TPR* score evaluates the number of actual positive data samples that predicted correctly (Eq. (10)). The *FPR* score is the ratio of the real negative data samples that are incorrectly predicted (Eq. (11)).

$$TPR = \frac{TP}{TP + FN}, \tag{10}$$

$$FPR = \frac{FP}{TN + FP}, \tag{11}$$

where *TP* and *FP* are the number of positive samples predicted correctly and incorrectly, respectively. *TN* and *FN* are the number of negative samples predicted correctly and incorrectly, respectively.

Table 2: 9 IoT datasets.

Dataset	Device Name	Training Attacks	Number of Training samples	Number of Testing samples
IoT1	Danmini Doorbell	combo, ack	239488	778810
IoT2	Ecobee Thermostat	combo, ack	59568	245406
IoT3	Ennio Doorbell	combo, tcp	174100	181400
IoT4	Philips B120N10 Baby Monitor	tcp, syn	298329	800348
IoT5	Provision PT 737E Security Camera	combo, ack	153011	675249
IoT6	Provision PT 838 Security Camera	ack, udp	265862	261989
IoT7	Samsung SNH 1011 N Webcam	combo, tcp	182527	192695
IoT8	SimpleHome XCS7 1002 WHT Security Camera	combo, ack	189055	674001
IoT9	SimpleHome XCS7 1003 WHT Security Camera	combo, ack	176349	674477

The space under the ROC curve represents the AUC score. It measures the average quality of the classification model at different thresholds. With a perfect classifier, the ROC curve will plot in the top left corner (*FPR* = 0, *TPR* = 100%). Whereas the ROC curve of a worst classifier will score in the bottom right corner (*FPR* = 100%, *TPR* = 0). The AUC score of a random classifier is 0.5, and the AUC score for a perfect classifier is 1.0.

5.3 Experimental Setting

Hyper-parameter Setting We compares the performance of the M2DA model with previous AE-based models. We set the same hyper-parameter set for all the AE-based models. These hyper-parameters are common for AE-based models training with network traffic dataset. The value of these hyper-parameters are presented in Table 3. This experiment setting is based on the AE-based models for detecting network attacks in the literature [3, 4, 17]. Because we have the δ_{SE} loss term of M2DA, the number of neurons in the bottleneck layer is the same as the number of classes in the IoT dataset. Thus, the bottleneck layer has two neurons to represent two classes, i.e., normal and attack. The reason is that we need to classify into two classes in this bottleneck layer. The number of layers of each AE is 5. This follows the previous work for network attack detection [17]. Here, we note that the ADAM algorithm [35] has been widely adopted in the literature to optimize weights of trained models in training processes. For our proposed model, the Sigmoid function is used at the last layers (for the encoder and decoder), while the active function ReLu is used at all the AE's layers.

Table 3: Hyper-parameter setting for the DTL models.

Hyper-parameter	Number of layers	Neuron number of bottleneck layer	Optimization	Activation function
Value	5	2	Adam	Relu

Experimental Set We have carried out three sets of experiments in this chapter. The first set is to evaluate the effectiveness of M2DA in the transferring task. Here, we show the differences of the M2D distances between the bottleneck layer of AE_1 trained on the source domain and that of AE_2 trained on the target domain after training the three scenarios. These transfer learning processes are performed in one, two and three encoding layers. It can be observed that the transfer learning process will be more effective if the M2D distance is low [36].

The second experiment set is to compare the AUC scores of M2DA with that of (1) the conventional AE learning model, (2) the proposed transfer learning model using KL metric (i.e., KLA) [13] and (3) the proposed transfer learning model using the M2D metric (i.e., MDA) [12]. We choose two these DTL models for comparison due to two reasons. First, these models are based on the AE's architecture and the AE-based models are the most effective with network traffic analysis as proved in previous work [3, 4, 17] and (2) these DTL models are of the similar type of transfer learning as the M2DA model. Specifically, we focus on the DTL models where the source domain has data and label information and the target domain has data without label information. All models are trained on the training sets of both source domain and target domain and evaluated on the testing sets of the target domain.

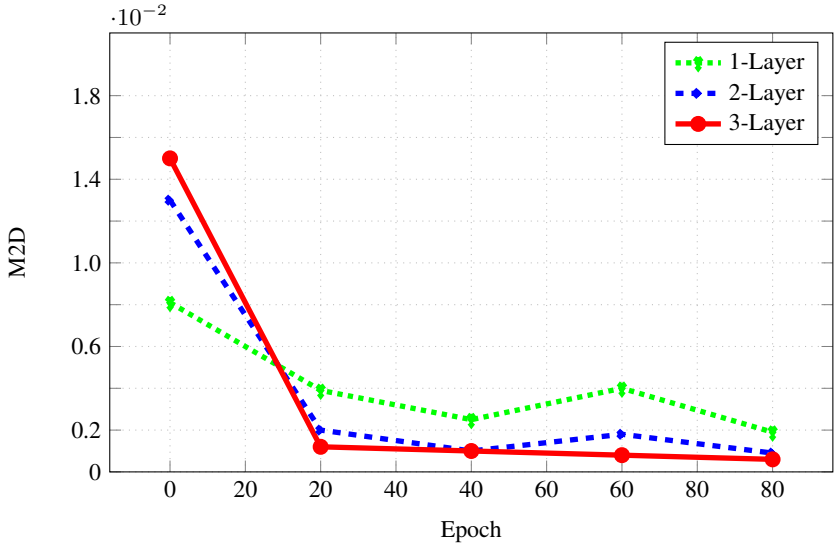


Fig. 4: M2D value that is transferred from the bottleneck layer of AE_1 to that of AE_2 when training on the source domain (IoT1) and the target domain (IoT2).

The third set is to measure the complexity and processing time of the experimental models. To measure the complexity of a neural network, we report the model size to measure the complexity of evaluated models. The model size is calculated by the number of the trainable parameters. The detailed results of the three experimental sets are presented in the next section.

6 Result and Discussion

6.1 Effectiveness of Transferring Task

M2DA implements the transferring task[2] between every layer in the encoder of AE_1 and AE_2. This transferring task aims to push the latent vector of AE_1 closer to the latent vector of AE_2. To evaluate if this transferring task is effective, we have conducted an experiment in which IoT1 and IoT2 are used as source and target datasets, respectively. We measure the M2D distance between the latent vectors of AE_1 and AE_2 in the three scenarios. We train the M2DA model where the M2D loss as in Eq. (8) is calculated on one, two and three layers of the encoder of M2DA. The lower distance value represents more knowledge transferred from the source to the target dataset.

The value of the M2D distance between the latent vectors of AE_1 and AE_2 is shown in Fig. 4. This figure shows that if more layers are deployed in the transfer learning tasks, the value of the the M2D distance is smaller. It means that more knowledge is transferred from the source to the target problem. This proves that the more layers of the encoders in M2DA are deployed transferring tasks, the more effectively the learning model can perform.

6.2 Accuracy Comparison

Fig. 5 presents the AUC scores of the tested models on the testing sets of the target datasets, i.e., (a) IoT1, (b) IoT2, (c) IoT3, (d) IoT4, (e) IoT5, (f) IoT6, (g) IoT7, (h) IoT8, and (i) IoT9. In each subfigure, the horizontal axis shows source datasets and the vertical axis shows the AUC score of a target dataset. At each source dataset in the horizontal axis, we present the results of the tested methods such as AE, KLA, MDA, M2DA. Moreover, the results of M2DA in Fig. 5 are shown in the yellow bars. It can be seen that AE provides the lowest accuracy

[2] The transferring task is executed by minimizing the M2D distance between on 1-Layer, 2-Layer, and 3-Layer corresponding encoding layers of two AEs.

among the four methods. This is not surprising since the AE model is trained and then tested on different datasets (e.g., trained on IoT1 and then tested on IoT2 datasets). Moreover, transfer learning is not implemented in the AE model, thus the performance of the AE is not satisfactory.

As a result, it can be observed that the performance (in term of the prediction accuracy) obtained by the AE model is much lower than those of the proposed learning mode1s. For example, if we take IoT3 dataset as the source domain and IoT2 dataset as the target domain, then the accuracies (in terms of AUC) obtained by the KLA and MDA, respectively, will be increased to 0.921 and 0.848, (vs. 0.658 obtained by the AE). These results clearly show that in the cases when IoT devices have data without label information, our proposed learning model can significantly improve the performance of the conventional AE model, especially in detecting IoT attacks.

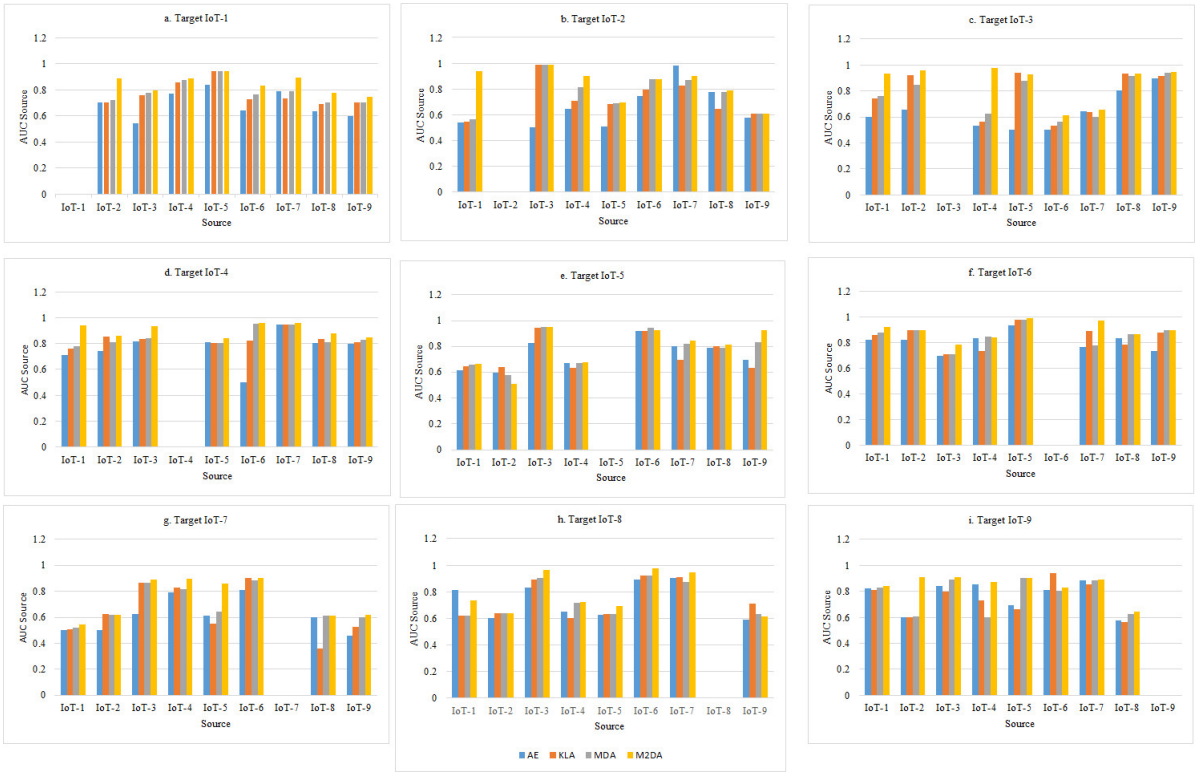


Fig. 5: AUC score on testing data of Target datasets.

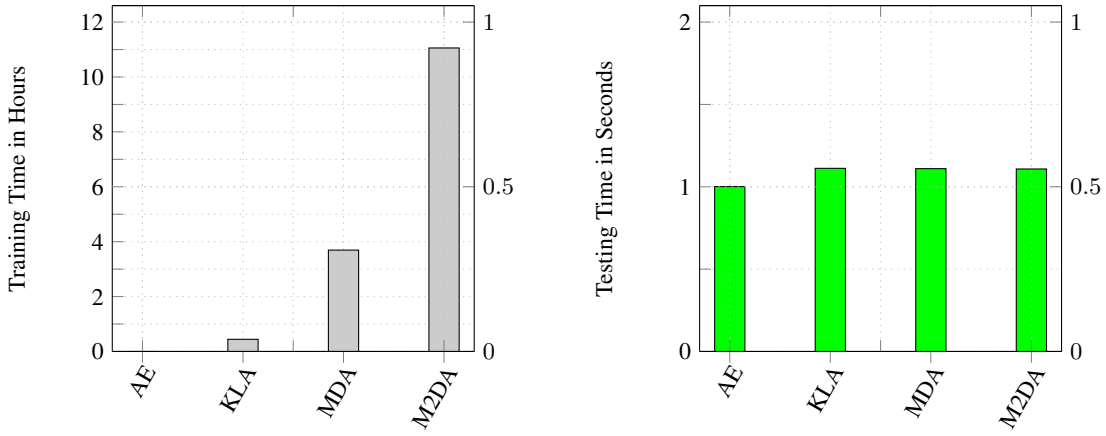
Among the transfer learning models, M2DA often achieves the best AUC score in almost all datasets. To give an example, when the source and target datasets are IoT2 and IoT3, respectively, the AUC score of M2DA is 0.956 compared to $0.659, 0.922, 0.849$ of AE, KLA, and MDA, respectively. We also observe similar results on the other datasets. These results shows that the M2DA model transfers more effectively the knowledge from the source to the target dataset. Consequently, M2DA usually enhances accuracy compared to AE, KLA, and MDA for IoT attack detection in the target problem without label information.

6.3 Complexity

In this section, we mainly focus on discussing the complexity of the proposed framework in terms of computing time. In particular, we first select the IoT2 data as the source domain for training and IoT1 data as the target domain for testing. Then, we preform the experiments and record time for training and testing data. The results are shown in Fig. 6 (a,b) in which the training time is represented by the grey bar with the unit of hours (Fig. 6 (a)) and the testing time is represented by the green bar with the unit of seconds (Fig. 6 (b)). Here, it can be observed that the training time for the proposed learning models is much higher than that of the conventional AE model. The main reason is that, unlike conventional AE learning models, the DTL models (including KLA, M2DA and MDA) require to calculate distances between encoding layers of the encoders of the first and second AE in every training iteration to execute the transferring task. Moreover, the time for training M2DA is also much longer than those of KLA and MDA. This is because M2DA executes the transfer learning in every encoding layer of the encoders whereas this task is only done in one layer (the bottleneck layer) of KLA and MDA.

Although, the training times of the transfer learning models are much higher compared to that of the AE, the inferring time of all the methods are mostly equal. The reason is that in the testing process, the data sample is only forwarded to one AE without the need for the distance calculation. For instance, the inferring times of AE, KLA, MDA, and M2DA on the IoT1 dataset are 1.001, 1.112, 1.110, and 1.108 second, respectively.

Table. 4 shows the complexity (the number of parameters) of the evaluated models. This table shows that while the number of the parameters of the AE model is much smaller, the number of the parameters of the three transfer learning models are mostly equal. This is due to the fact that the three models use a similar structure of neural networks. Moreover, in the inferring phase, the data sample is forwarded to only one AE (the second AE) of KLA, MDA, and M2DA . Thus, the times for detecting IoT attacks of all the four tested models are roughly equal as in Figure 6.



(a) Training time (Hours).

(b) Testing time (Seconds).

Fig. 6: Training time (a) and Average Testing time for one data sample (b) of evaluated models (AE, KLA, MDA, and M2DA) on the source domain as IoT1 dataset and the target domain as IoT2 dataset.

Table 4: Complexity of DTL models.

Models	AE [16]	KLA [13]	MDA [12]	M2DA
No. Parameters	25117	150702	150702	150702

7 Conclusion

In this chapter, we resolve "the lack of label information" in the IoT attack detection problem. In some situations, we are unable to collect network traffic data with its label information. For example, we can not label all collected IoT traffic data due to the cost of labelling tasks. Moreover, data distributions of data samples collected from different IoT devices are not the same. Thus, we develop a DTL technique namely M2DA that can transfer the knowledge of label information from a domain (i.e., data collected from one IoT device) to a related domain (i.e., data collected from a different IoT device) without label information. The results show that the M2DA model can help to identify IoT attacks more accurately.

However, the technique is subject to some limitations. M2DA is more complex due to the transferring process executed in multiple layers. It leads to time-consuming tasks for training M2DA. However, the predicting process of M2DA is mostly similar to that of the other AE-based models because it only uses one AE architecture for the predicting task.

In future work, to enhance the privacy of IoT data, we plan to use federated learning to train the M2DA model. In that case, we do not need to send IoT traffic data to the processing center. Besides that, we will test the effectiveness of the DTL model with network traffic from several different network environments. Last but not least, we will attempt to expand this model to other neural networks such as Convolutional Neural Network, Deep Belief Network.

References

1. Luong, N.C., Hoang, D.T., Wang, P., Niyato, D., Kim, D.I., Han, Z.: Data collection and wireless communication in internet of things (IoT) using economic analysis and pricing models: A survey. IEEE Communications Surveys Tutorials 18(4), 2546–2590 (Fourthquarter 2016)
2. 2018 annual cybersecurity report: the evolution of malware and rise of artificial intelligence. (2018), `https://www.cisco.com/c/en_in/products/security/security-reports.html#~about-the-series`, , Accessed: 2019-5-10.
3. 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 (Jul 2018)

4. Meidan, Y., Bohadana, M., Shabtai, A., Ochoa, M., Tippenhauer, N.O., Guarnizo, J.D., Elovici, Y.: Detection of unauthorized IoT devices using machine learning techniques. arXiv preprint arXiv:1709.04647 (2017)
5. N. Vlajic and D. Zhou, "Iot as a land of opportunity for ddos hackers," *Computer*, vol. 51, no. 7, pp. 26–34, 2018.
6. C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
7. R. Gow, F. A. Rabhi, and S. Venugopal, "Anomaly detection in complex real world application systems," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 83–96, 2018.
8. Jing, X., Yan, Z., Pedrycz, W.: Security data collection and data analytics in the internet: A survey. IEEE Communications Surveys & Tutorials 21(1), 586–618 (2018)
9. Lee, W., Xiang, D.: Information-theoretic measures for anomaly detection. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001. pp. 130–143. IEEE (2001)
10. Khattak, S., Ramay, N.R., Khan, K.R., Syed, A.A., Khayam, S.A.: A taxonomy of botnet behavior, detection, and defense. IEEE Communications Surveys Tutorials 16(2), 898–924 (Second 2014)
11. Nomm, S., Bahsi, H.: Unsupervised anomaly based botnet detection in IoT networks. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) pp. 1048–1053 (2018)
12. Wen, L., Gao, L., Li, X.: A new deep transfer learning based on sparse auto-encoder for fault diagnosis. IEEE Transactions on Systems, Man, and Cybernetics: Systems 49(1), 136–144 (2017)
13. Zhuang, F., Cheng, X., Luo, P., Pan, S.J., He, Q.: Supervised representation learning: Transfer learning with deep autoencoders. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
14. Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., Smola, A.J.: A kernel method for the two-sample-problem. In: Advances in neural information processing systems. pp. 513–520 (2007)
15. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in neural information processing systems. pp. 153–160 (2007)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
17. Cao, V.L., Nicolau, M., McDermott, J.: Learning neural representations for network anomaly detection. IEEE Transactions on Cybernetics 49(8), 3074–3087 (Aug 2019)
18. Dromard, J., Roudière, G., Owezarski, P.: Online and scalable unsupervised network anomaly detection method. IEEE Transactions on Network and Service Management 14(1), 34–47 (March 2017)
19. Ibidunmoye, O., Rezaie, A., Elmroth, E.: Adaptive anomaly detection in performance metric streams. IEEE Transactions on Network and Service Management 15(1), 217–231 (March 2018)
20. Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., Zhang, G.: Transfer learning using computational intelligence: a survey. Knowledge-Based Systems 80, 14–23 (2015)
21. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22(10), 1345–1359 (2009)
22. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: International Conference on Artificial Neural Networks. pp. 270–279. Springer (2018)
23. Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. Journal of Big data 3(1), 9 (2016)
24. Wan, C., Pan, R., Li, J.: Bi-weighting domain adaptation for cross-language text classification. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
25. Xu, Y., Pan, S.J., Xiong, H., Wu, Q., Luo, R., Min, H., Song, H.: A unified framework for metric transfer learning. IEEE Transactions on Knowledge and Data Engineering 29(6), 1158–1171 (2017)
26. Liu, X., Liu, Z., Wang, G., Cai, Z., Zhang, H.: Ensemble transfer learning algorithm. IEEE Access 6, 2389–2396 (2018)
27. Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., Darrell, T.: Deep domain confusion: Maximizing for domain invariance. arXiv preprint arXiv:1412.3474 (2014)
28. Long, M., Zhu, H., Wang, J., Jordan, M.I.: Deep transfer learning with joint adaptation networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2208–2217. JMLR. org (2017)
29. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
30. Tzeng, E., Hoffman, J., Saenko, K., Darrell, T.: Adversarial discriminative domain adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7167–7176 (2017)
31. Long, M., Cao, Z., Wang, J., Jordan, M.I.: Domain adaptation with randomized multilinear adversarial networks. arXiv preprint arXiv:1705.10667 (2017)
32. Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1717–1724 (2014)
33. Long, M., Zhu, H., Wang, J., Jordan, M.I.: Unsupervised domain adaptation with residual transfer networks. In: Advances in Neural Information Processing Systems. pp. 136–144 (2016)
34. Kandaswamy, C., Silva, L.M., Alexandre, L.A., Sousa, R., Santos, J.M., de Sá, J.M.: Improving transfer learning accuracy by reusing stacked denoising autoencoders. In: 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 1380–1387. IEEE (2014)
35. Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
36. Yang, J., Yan, R., Hauptmann, A.G.: Cross-domain video concept detection using adaptive svms. In: Proceedings of the 15th ACM international conference on Multimedia. pp. 188–197 (2007)

37. García, S., Zunino, A., Campo, M.: Botnet behavior detection using network synchronism. In: Privacy, Intrusion Detection and Response: Technologies for Protecting Networks, pp. 122–144. IGI Global (2012)
38. TCPtrace tool, http://www.tcptrace.org/, Accessed: 2019-11-30.
39. Wireshark tool, https://www.wireshark.org/, , Accessed: 2019-11-30.
40. Powers, D.: Evaluation: From precision, recall and fmeasure to roc, informedness, markedness and correlation. Journal of Machine Learning Technologies 2, 37–63 (01 2007).