MINISTRY OF EDUCATION AND TRAINING MINISTRY OF NATIONAL DEFENCE

MILITARY TECHNICAL ACADEMY

VU THI LY

DEVELOPING DEEP NEURAL NETWORKS FOR NETWORK ATTACK DETECTION

DOCTORAL THESIS

HA NOI - 2021

MINISTRY OF EDUCATION AND TRAINING MINISTRY OF NATIONAL DEFENCE

MILITARY TECHNICAL ACADEMY

VU THI LY

DEVELOPING DEEP NEURAL NETWORKS FOR NETWORK ATTACK DETECTION

DOCTORAL THESIS

Major: Mathematical Foundations for Informatics

Code: 946 0110

RESEARCH SUPERVISORS:

1. **Assoc. Prof. Dr. Nguyen Quang Uy**
2. **Prof. Dr. Eryk Duzkite**

HA NOI - 2021

ASSURANCE

I certify that this thesis is a research work done by the author under the guidance of the research supervisors. The thesis has used citation information from many different references, and the citation information is clearly stated. Experimental results presented in the thesis are completely honest and not published by any other author or work.

Author

Vu Thi Ly

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Nguyen Quang Uy for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I wish to thank my co-supervisor, Prof. Dr. Eryk Duzkite, Dr. Diep N. Nguyen, and Dr. Dinh Thai Hoang at University Technology of Sydney, Australia. Working with them, I have learned how to do research and write an academic paper systematically. I would also like to acknowledge to Dr. Cao Van Loi, the lecturer of the Faculty of Information Technology, Military Technical Academy, for his thorough comments and suggestions on my thesis.

Second, I also would like to thank the leaders and lecturers of the Faculty of Information Technology, Military Technical Academy, for encouraging me with beneficial conditions and readily helping me in the study and research process.

Finally, I must express my very profound gratitude to my parents, to my husband, Dao Duc Bien, for providing me with unfailing support and continuous encouragement, to my son, Dao Gia Khanh, and my daughter Dao Vu Khanh Chi for trying to grow up by themselves. This accomplishment would not have been possible without them.

Author

Vu Thi Ly

CONTENTS

ABBREVIATIONS

No.	Abbreviation	Meaning
1	AAE	Adversarial AutoEncoder
2	ACGAN	Auxiliary Classifier Generative Adversarial Network
3	ACK	Acknowledgment
4	AE	AutoEncoder
5	AUC	Area Under the Receiver Operating Characteristics Curve
6	CDAAE	Conditional Denosing Adversarial
7	CNN	Convolutional Neural Network
8	CTU	Czech Technical University
9	CVAE	Conditional Variational AutoEncoder
10	DAAE	Denosing Adversarial AutoEncoder
11	DAE	Denoising AutoEncoder
12	DBN	Deep Beleif Network
13	DDoS	Distributed Deny of Service
14	De	Decoder
15	Di	Discriminator
16	DT	Decision Tree
17	DTL	Deep Transfer Learning
18	En	Encoder
19	FN	False Negative
20	FP	False Positive
21	FTP	File Transfer Protocol
22	GAN	Generative Adversarial Network

No.	Abbreviation	Meaning
23	Ge	Generator
24	IoT	Internet of Things
25	IP	Internet Protocol
26	KL	Kullback-Leibler
27	KNN	K-nearest Neighbor
28	LR	Linear Regression
29	MAE	Multi-Distribution AutoEncoder
30	MDAE	Multi-Distribution Denoising AutoEncoder
31	MMD	Maximum Mean Discrepancy
32	MVAE	Multi-Distribution Variational AutoEncoder
33	NAD	Network Attack Detection
34	NCT	Nearest CenTroid
35	PCT	PerCepTron
36	R2L	Remote to Login
37	RE	Reconstruction Error
38	RF	Random Forest
39	RG	Regularization Phase
40	RP	Reconstruction Phase
41	ReLU	Rectified Linear Unit
42	SAAE	Supervised Adversarial AutoEncoder
43	SKL-AE	DTL method using the KL metric and transferring task is executed on the AE's bottleneck layer
44	SMD-AE	DTL method using the MMD metric and transferring task is executed on the AE's bottleneck layer
45	SMD-AE	DTL method using the MMD metric and transferring task is executed on the encoding layers of AE
46	SMOTE	Synthetic Minority Over-sampling Technique

No.	Abbreviation	Meaning
47	SVM	Support Vector Machine
48	SYN	Synchronize
49	TCP	Transmission Control Protocol
50	TL	Transfer Learning
51	TN	True Negative
52	TP	True Positive
53	TPR	True Positive Rate
54	U2L	User to Login
55	UDP	User Datagram Protocol
56	VAE	Variational AutoEncoder

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1. Motivation

Over the last few years, we have been experiencing an explosion in communications and information technology in network environments. Cisco predicted that the Global Internet Protocol (IP) traffic will increase nearly threefold over the next five years, and will increase 127-fold from 2005 to 2021 [4]. Furthermore, IP traffic will grow at a Compound Annual Growth Rate of 24% from 2016 to 2021. The unprecedented development of communication networks has significant contributions for human beings but also places many challenges for information security problems due to the diversity of emerging cyberattacks. According to a study in [5], 53 % of all network attacks resulted in financial damages of more than US$500,000, including lost revenue, customers, opportunities, and so on. As a result, early detecting network attacks plays a crucial role in preventing cyberattacks and ensuring confidentiality, integrity, and availability of information in communication networks [6].

A network attack detection (NAD) monitors the network traffic to identify abnormal activities in the network environments such as computer networks, cloud, and Internet of Things (IoT). There are three popular approaches for analyzing network traffic to detect intrusive behaviors [7], i.e., knowledge-based methods, statistic-based methods, and machine learning-based methods. First, in order to detect network 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 of attacks. Moreover, they are unable to detect unknown 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 [7]. There are some statistical methods, such as information entropy, conditional entropy, information gain [8]. 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 for NAD have received increasing attention in the research community due to their outstanding advantages [9–13]. The main idea of applying machine learning techniques for NAD is to build a detection model based on training datasets automatically. Depending on the availability of data labels, machine learning-based NAD can be categorized into three main approaches: supervised learning, semi-supervised learning, and unsupervised learning [14].

Although machine learning, especially deep learning, has achieved remarkable success in NAD, there are still some unsolved problems that can affect the accuracy of detection models. First, the network traffic is heterogeneous and complicated due to the diversity of network environments. Thus, it is challenging to represent the network traffic data that fascinates machine learning classification algorithms. Second, to train a good detection model, we need to collect a large amount of network attack data. However, collecting network attack data is often harder than those of normal data. Therefore, network attack datasets are usually highly imbalanced. When being trained on such skewed datasets, conventional machine learning algorithms are often biassed and inaccurate.

Third, in some network environments, e.g., IoTs, we are often unable to collect the network traffic from all IoT devices for training the detection model. The reason is due to the privacy of IoTs devices. Subsequently, the detection model trained on the data collected from one device may be used to detect the attacks on other devices. However, the data distribution in one device may be very different from that in other devices and it affects to the accuracy of the detection model.

2. Research Aims

The thesis aims to develop deep neural networks for analyzing security data. These techniques improve the accuracy of machine learning-based models applied in NAD. Therefore, the thesis attempts to address the above challenging problems in NAD using models and techniques in deep neural networks. Specifically, the following problems are studied.

First, to address the problem of heterogeneity and complexity of network traffic, we propose a representation learning technique that can project normal data and attack data into two separate regions. Our proposed representation technique is constructed by adding a regularized term to the loss function of AutoEncoder (AE). This technique helps to significantly enhance the accuracy in detecting both known and unknown attacks.

Second, to train a good detection model for NAD systems on an imbalanced dataset, the thesis proposes a technique for generating synthesized attacks. These techniques are based on two well known unsupervised deep learning models, including Generative Adversarial Network (GAN) and AE. The synthesized attacks are then merged with the collected attack data to balance the skewed dataset.

Third, to improve the accuracy of detection models on IoTs devices that do not have label information, the thesis develops a deep transfer learning (DTL) model. This model allows transferring the label information of the data collected from one device (a source device) to another device (a target device). Thus the trained model can effectively identify

attacks without the label information of the training data in the target domain.

3. Research Methodology

Our research method includes both researching academic theories and doing experiments. We study and analyze previous related research. This work helps us find the gaps and limitations of the previous research on applying deep learning to NAD. Based on this, we propose various solutions to handle and improve the accuracy of the NAD model.

We conduct a large number of experiments to analyze and compare the proposed solutions with some baseline techniques and state-of-the-art methods. These experiments prove the effectiveness of our proposed solutions and shed light on their weakness and strength.

4. Scope Limitations

Although machine learning has been widely used in the field of NAD [9–13], this thesis focuses on studying three issues when applying machine learning for NAD. These include representation learning to detect both known and unknown attacks effectively, the imbalance of network traffic data due to the domination of normal traffic compared with attack traffic, and the lack of label information in a new domain in the network environment. As a result, we propose several deep neural networks-based models to handle these issues.

Moreover, this thesis has experienced in more than ten different kinds of network attack datasets. They include three malware datasets, two intrusion detection in computer network datasets, and nine IoT attack datasets. In the future, more diversity datasets should be tested with the proposed methods.

Many functional research studies on deep neural networks in other fields, which are beyond this thesis's scope, can be found in the literature. However, this thesis focuses on AE-based models and GAN-based models due to their effectiveness in the network traffic data. When conducting experiments with a deep neural network, some parameters

(initialization methods, number of layers, number of neurons, activation functions, optimization methods, and learning rate) need to be considered. However, this thesis is unable to tune all different settings of these parameters.

5. Contributions

The main contributions of this thesis are as follows:

- The thesis proposes three latent representation learning models based on AEs namely Multi-distribution Variational AutoEncoder (MVAE), Multi-distribution AutoEncoder (MAE), and Multi-distribution Denoising AutoEncoder (MDAE). These proposed models project normal traffic data and attack traffic data, including known network attacks and unknown network attacks to two separate regions. As a result, the new representation space of network traffic data fascinates simple classification algorithms. In other words, normal data and network attack data in the new representation space are distinguishable from the original features, thereby making a more robust NAD system to detect both known attacks and unknown attacks.

- The thesis proposes three new deep neural networks namely Auxiliary Classifier GAN - Support Vector Machine (ACGAN-SVM), Conditional Denoising Adversarial AutoEncoder (CDAAE), and Conditional Denoising Adversarial AutoEncoder - K Nearest Neighbor (CDAAE-KNN) for handling data imbalance, thereby improving the accuracy of machine learning methods for NAD systems. These proposed techniques developed from a very new deep neural network aim to generate network attack data samples. The generated network attack data samples help to balance the training network traffic datasets. Thus, the accuracy of NAD systems is improved significantly.

- A DTL model is proposed based on AE, i.e., Maximum Mean Discrepancy-AutoEncoder (MMD-AE). This model can transfer the knowledge from a source domain of network traffic data with label information

to a target domain of network traffic data without label information. As a result, we can classify the data samples in the target domain without training with the target labels.

The results in the thesis have been published and submitted to seven papers. Three international conference papers (one Rank B paper and two SCOPUS papers) were published. One domestic scientific journal paper, one SCIE-Q1 journal paper and one SCI-Q1 journal paper were published. One SCI-Q1 journal paper is under review in the firsts round.

6. Thesis Overview

The thesis includes four main content chapters, the introduction, and the conclusion and future work parts. The rest of the thesis is organized as follows.

Chapter 1 presents the fundamental background of the NAD problem and deep neural techniques. Some characteristics of network behaviors in several networks such as computer networks, IoT, cloud environments are presented. We also survey techniques used to detect network attacks recently, including deep neural networks, and some network traffic datasets used in this thesis. In the sequel, several deep neural networks, which are used in the proposed techniques, are presented in detail. Finally, this chapter describes evaluation metrics that are used in our experiments.

Chapter 2 proposes a new latent representation learning technique that helps network attacks to be detected more easily. Based on that, we propose three new representation models representing network traffic data in more distinguishable representation spaces. Consequently, the accuracy of detecting network attacks is improved impressively. Nine IoT attack datasets are used in the experiments to evaluate the newly proposed models. The effectiveness of the proposed models is assessed in various experiments with in-depth discussions on the results.

Chapter 3 presents new generative deep neural network models for handling the imbalance of network traffic datasets. Here, we introduce generative deep neural network models used to generate high-quality

attack data samples. Moreover, the generative deep neural network model's variants are proposed to improve the quality of attack data samples, thereby improving supervised machine learning methods for the NAD problem. The experiments are conducted on well-known network traffic datasets with different scenarios to assess newly proposed models in many different aspects. The experimental results are discussed and analyzed carefully.

Chapter 4 proposes a new DTL model based on a deep neural network. This model can adapt the knowledge of label information of a domain to a related domain. It helps to resolve the lack of label information in some new domains of network traffic. The experiments demonstrate that using label information in a source domain (data collected from one IoT device) can enhance the accuracy of a target domain without labels (data collected from a different IoT device).

Chapter 1
BACKGROUNDS

This chapter presents the theoretical backgrounds and the related works of this thesis. First, we introduce the NAD problem and related work. Next, we describe several deep neural network models that are the fundamental of our proposed solutions. Here, we also assess the effectiveness of one of the main deep neural networks used in this thesis, i.e., AutoEncoder (AE), for NAD published in (iii). Finally, the evaluation metrics used in the thesis are presented in detail.

1.1. Introduction

The Internet becomes an essential function in our living. Simultaneously, while the Internet does us excellent service, it also raises many security threats. Security attacks have become a crucial portion that restricts the growth of the Internet. Network attacks that are the main threats for security over the Internet have attracted particular attention.

Recently, security attacks have been examined in several different domains. Zou et al. [15] first reviewed the security requirements of wireless networks and then presented a general overview of attacks confronted in wireless networks. Some security threats in cloud computing are presented and analyzed in [16]. Attack detection methods have received considerable attention recently to guarantee the security of information systems.

Security data indicate the network traffic data that can be used to detect security attacks. It is the main component in attack detection, no matter whether at a training or detecting stage. Many kinds of approaches are applied to examine security data to detect attacks. Usually, NAD methods take the knowledge of network attacks from network traf-

fic datasets. The next section will present some common network traffic datasets used in the thesis.

1.2. Experiment Datasets

This section presents the experimental datasets. To evaluate the effectiveness of the proposed models, we do the experiments in several well-known security datasets, including two network datasets (i.e., NSL-KDD and UNSW-NB15) and three malware datasets from the CTU-13 dataset system, IoT attack datasets.

In the thesis, we mainly use nine IoT attack datasets because they have various attacks and been published more recently. Especially, they are suitable to represent the effectiveness of DTL techniques. The reason is that the network traffic collected in different IoT devices are related domain. This matches with the assumption of a DTL model. However, for handling imbalance dataset, we need to choose some other common datasets that are imbalance, such as NSL-KDD, UNSW-NB15, CTU-13.

Table 1.1: Number of training data samples of network attack datasets.

NSL-KDD		**UNSW-NB15**	
Classes	**Number**	**Classes**	**Number**
Normal	**67373**	**Normal**	**37000**
Attack	**58630**	**Attack**	**45332**
DoS	45927	Generic	18871
U2L	52	Exploits	11132
R2L	995	Fuzzers	6062
Probing	11656	DoS	4089
		Reconnaissance	3496
		Analysis	677
		Backdoor	583
		Shellcode	378
		Worms	44

Table 1.2: Number of training data samples of malware datasets.

Menti		**NSIS.ay**		**Virus**	
Classes	**No.**	**Classes**	**No.**	**Classes**	**No.**
Benign	518904	Benign	292485	Benign	37000
Malware	230	Malware	1420	Malware	37000

1.2.1. NSL-KDD

NSL-KDD is a network attack dataset [17], which is used to solve some inherent problems of the KDD'99 dataset. Each sample has 41 features and is labeled as either a type of attack or normal. The training set contains 24 attack types, and the testing set includes additive 14 types of attacks. The simulated attack samples belong to one of the following four categories: DOS, R2L, U2R, and Probing. The details of the datasets are presented in Table 1.1.

1.2.2. UNSW-NB15

UNSW-NB15 is created by utilizing the synthetic environment as the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre of Cyber Security [18]. There are nine categories of attacks, which are Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. Each data sample has 49 features generated using the Argus, Bro-IDS tools, and their twelve algorithms to analyze characteristics of network packets. The details of the datasets are presented in Table 1.1.

1.2.3. CTU13s

The CTU-13 dataset is a publicly available malware dataset that was captured in the Czech Technical University (CTU), Czech Republic, in 2011 [19]. The data includes normal traffic combined with many kinds of real-world botnets. There are thirteen botnet scenarios, and each of them involves a specific type of malware with several protocols and different actions. We choose three scenarios in the dataset that correspond to three kinds of malware, including Menti, NSIS.ay, and Virus. The details of the datasets are presented in Table 1.2.

1.2.4. Bot-IoT Datasets (IoT Datasets)

We also use nine IoT attack-related datasets introduced by Y. Meidan et al. [9] for evaluating our proposed models. These data sam-

ples were collected from nine commercial IoT devices in their lab with two most well-known IoT-based botnet families, Mirai and BASHLITE (Gafgyt). Each of the botnet family contains five different IoT attacks. Among these IoT attack datasets, there are three datasets, namely Ennio_Doorbell (IoT-3), Provision_PT_838_Security_Camera (IoT-6), Samsung_SNH_1011_N_Webcam (IoT-7) containing only one IoT botnet family (five types of botnet attacks). The rest of these datasets consist of both Mirai and Gafgyt (ten types of DDoS attacks).

After pre-process the raw features by one-hot encoding and removing identify features ('saddr', 'sport', 'daddr', 'dport'), each data sample has 115 attributes, which are categorized into three groups: stream aggregation, time-frame, and the statistics attributes. The details of the datasets are presented in Table 1.3.

Table 1.3: The nine IoT datasets.

Dataset	Device Name	Training Attacks	Training size	Testing size
IoT-1	Danmini_Doorbell	combo, ack	239488	778810
IoT-2	Ecobee_Thermostat	combo, ack	59568	245406
IoT-3	Ennio_Doorbell	combo, tcp	174100	181400
IoT-4	Philips_B120N10_Baby _Monitor	tcp, syn	298329	800348
IoT-5	Provision_PT_737E _Security_Camera	combo, ack	153011	675249
IoT-6	Provision_PT_838 _Security_Camera	ack, udp	265862	261989
IoT-7	Samsung_SNH_1011 _N_Webcam	combo, tcp	182527	192695
IoT-8	SimpleHome_XCS7_1002 _WHT_Security_Camera	combo, ack	189055	674001
IoT-9	SimpleHome_XCS7_1003 _WHT_Security_Camera	combo, ack	176349	674477

1.3. Deep Neural Networks

In this section, we will present the mathematical backgrounds of several deep neural network models that will be used to develop our proposed models in the next chapters.

A deep neural network is an artificial neural network with multiple layers between the input and output layers. This network aims to approxi-

mate function of f^*. For example, this defines a mapping $y = f(x, \theta)$ and learns the parameters θ to approach the best approximation [1]. Deep neural networks provide a robust framework for supervised learning. A deep neural network aims to map an input vector to an output vector where the output vector is easier for other machine learning tasks. This mapping is done by given large models and large labeled training data samples [1].

1.3.1. AutoEncoders

This section presents the structure of the AutoEncoder (AE) model and the proposed work that exploits the AE's representation.

1.3.1.1. Structure of AE

An AE is a neural network trained to copy the network's input to its output [20]. This network has two parts, i.e., encoder and decoder (as shown in Fig. 1.2 (a)). Let $\mathbf{W}$, $\mathbf{W}'$, $\mathbf{b}$, and $\mathbf{b}'$ be the weight matrices and the bias vectors of the encoder and the decoder, respectively, and $\mathbf{x} = \{x^1, x^2, \ldots, x^n\}$ be a training dataset. Let $\phi = (\mathbf{W}, \mathbf{b})$ and $\theta = (\mathbf{W}', \mathbf{b}')$ be parameter sets for training the encoder and the decoder, respectively. Let q_ϕ denote the encoder, z^i be the representation of the input sample x^i. The encoder maps the input x^i to the latent representation z^i (in Eq. 1.1). The latent representation of the encoder is typically referred to as a "bottleneck". The decoder p_θ attempts to map the latent representation z^i back into the input space, i.e, $\hat{x}^i$ (in Eq. 1.2).

$$z^i = q_\phi(x^i) = a_e(\mathbf{W}x^i + \mathbf{b}), \tag{1.1}$$

$$\hat{x}^i = p_\theta(z^i) = a_d(\mathbf{W}'z^i + \mathbf{b}'), \tag{1.2}$$

where a_e and a_d are the activation functions of the encoder and the decoder, respectively.

For a single sample of x^i, the loss function of an AE is the difference between x^i and the output $\hat{x}^i$. The loss function of an AE for a dataset

is often calculated as the mean squared error (MSE) overall data samples [21] as in Eq. 1.3.

$$\ell_{AE}(\mathbf{x}, \phi, \theta) = \frac{1}{n}\sum_{i=0}^{n}\left(x^i - \hat{x}^i\right)^2. \tag{1.3}$$

1.3.1.2. Representation of AE

The effectiveness of NAD models based on AEs can be depended on the type of activation functions used in the AEs. Each kind of activation functions can only learn some specific characteristics of input data and different activation functions may result in significant different performance of AEs. Recently, researchers has paid attention to combine activation functions in AE models to learn more information of the input data [22]. In [22], they combined the hyperbolic Tangent (Tanh) and logistic (Sigmoid) functions to enhance the accuracy of the latent representation for a classification problem. However, due to the vanishing gradient problem[1], the Sigmoid function is very ineffective in the AE with many layers training on a large dataset like an IoT anomaly dataset.

We have proposed a work [i] to exploit the effectiveness of AE in the NAD problem. To understand the latent representation of AE, we combine two useful activation functions, i.e., Relu and Tanh, to present network traffic in higher-level representation space. We also conducted an analysis on the properties of three popular activation functions, i.e., Sigmoid, Tanh, and Relu to explain why Tanh and Relu be more suitable for learning characteristics of IoT anomaly data than Sigmoid. The detail of this proposed method is described as following.

We design two AE models that have same network structure namely AE_1. Let's denote the encoder and decoder of AE_1 as En_1 and De_1, respectively, and those of AE_2 as En_2 and De_2, respectively. Let's denote W_{En_1}, b_{En_1} and W_{En_2}, b_{En_2} as the weight matrix and bias vector of the

[1]When the output of an activation function go to it's saturated area, it's gradient will come to zero. Thus, gradient cannot be updated. This is called as a vanishing gradient problem.

encoders of AE_1 and AE_2, respectively. Those of the decoders are W_{De_1}, b_{De_1} and W_{De_2}, b_{De_2}, respectively. The outputs of encoder and decoder for AE_1 are z_1 (Eq. 1.4) and $\tilde{x}_1$ (Eq. 1.5), respectively. Those values for AE_2 are z_2 (Eq. 1.6) and $\tilde{x}_2$ (Eq. 1.7), respectively.

$$z_1 = f(W_{En_1}x + b_{En_1}), \tag{1.4}$$

$$\tilde{x}_1 = f(W_{De_1}x + b_{De_1}), \tag{1.5}$$

$$z_2 = g(W_{En_2}x + b_{En_2}), \tag{1.6}$$

$$\tilde{x}_2 = g(W_{De_2}x + b_{De_2}), \tag{1.7}$$

where f and g are two different activation functions which are the Tanh and Relu in the proposed model.

The AE_1 and AE_2 models use f and g as their activation functions for every hidden layers excepting the bottleneck layer and the last layer where they use the Sigmoid function. Both AE models are trained separately on each batch size of the training data. Eq. 1.8 and Eq. 1.9 present the loss functions of AE_1 and AE_2 over all data samples x^i for i in range $1 \ldots n$, respectively. Here, we use a loss function as MSE.

$$RE_1\left(x^i, W, b\right) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \tilde{x}_1^i\right)^2, \tag{1.8}$$

$$RE_2\left(x^i, W, b\right) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \tilde{x}_2^i\right)^2, \tag{1.9}$$

where n is the number of training samples, (W, b) is the learning parameter set of the AE model.

After training, we use the encoder part of each AE model, i.e., En_1 and En_2 to generate the latent representations, i.e., z_1 and z_2. The combination of z_1 and z_2 is the input of classification algorithms instead of the original data x. Thus, the representation of the original data x has benefits of both Tanh and Relu functions. As a result, the accuracy of classification algorithms is improved significantly.

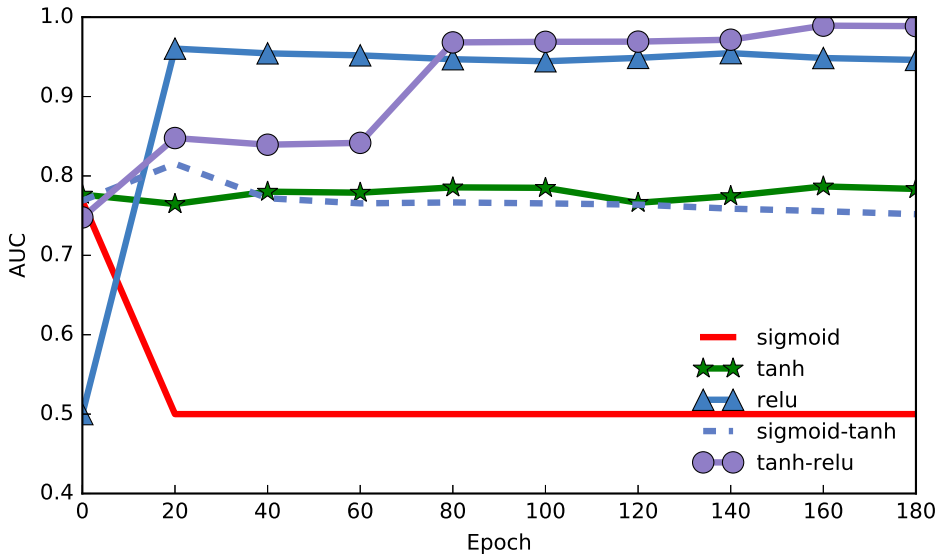


Figure 1.1: AUC comparison for AE model using different activation function of IoT-4 dataset.

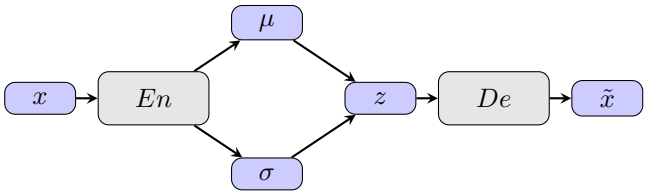
We visualize AUC scores in the training process. Fig. 1.1 presents the comparison of the AUC score of Support Vector Machine (SVM) on the representation of five AE-based models in the IoT-4 dataset. This figure shows that SVM is unable to classify the representation generated by the AE-based model with the Sigmoid function (Sigmoid-based model) due to its AUC score approximately at 0.5[2]. The AUC score of the Tanh-based model is nearly 0.8. However, the combination of the Sigmoid-Tanh-based model is not higher than the Tanh-based model due to the inefficient Sigmoid-based model. Thus, using the Sigmoid function in the AE model for IoT anomaly detection is not as effective as problems presented in [22].

Fig. 1.1 also shows the AUC score of the Relu-based model, which is relatively high (over 0.9) in the training process. Moreover, the combination of Relu and Tanh activation can enhance extremely high performance after several epochs of training. The reasons can be that in the AE model, using the Tanh function can reduce the limitation of the dying problem of the Relu function and using the Relu function model

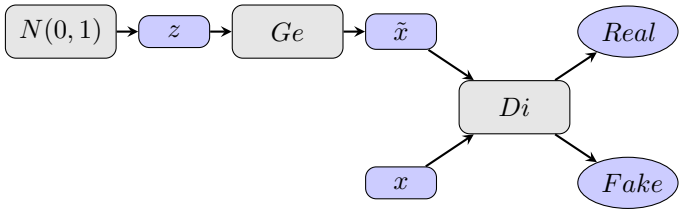
[2]A random classifier has an AUC score at 0.5. The detail description of AUC will be presented in section 1.5

to handle the vanishing problem of Tanh function.

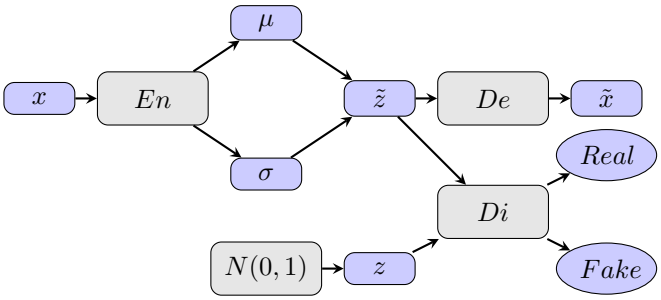
(a) AutoEncoder (AE).



(b) Variational AutoEncoder (VAE).



(c) Generative Adversarial Network (GAN).



(d) Adversarial AutoEncoder (AAE).

Figure 1.2: Structure of generative models (a) AE, (b) VAE, (c) GAN, and (d) AAE.

1.3.2. Denoising AutoEncoder

Denoising AutoEncoder (DAE) is a regularized AE that aims to reconstruct the original input from a noised version of the input [23]. Thus, DAE can capture the true distribution of the input instead of learning the identity [1, 24]. There are several methods adding noise to the input data, and the additive isotropic Gaussian noise is the most common one.

Let define an additive isotropic Gaussian noise $C(\tilde{\mathbf{x}}|\mathbf{x})$ to be a conditional distribution over a corrupted sample $\tilde{\mathbf{x}}$, given a data sample $\mathbf{x}$. Let define $\mathbf{x_{noise}}$ to be the noise component drawn from the normal distribution with the mean is 0 and the standard deviation is σ_{noise}, i.e.,

$\mathbf{x_{noise}} \sim \mathcal{N}(0, \sigma_{noise})$. The denoising criterion with the Gaussian corruption is presented as follows:

$$C(\tilde{\mathbf{x}}|\mathbf{x}) = \mathbf{x} + \mathbf{x_{noise}}. \tag{1.10}$$

Let define $\tilde{x}^i$ to be the corrupted version of the input data x^i obtained from $C(\tilde{\mathbf{x}}|\mathbf{x})$. Note that the corruption process is performed stochastically on the original input each time a point x^i is considered. Based on the loss function of AE, the loss function of DAE can be written as follows:

$$\ell_{DAE}(\mathbf{x}, \tilde{\mathbf{x}}, \phi, \theta) = \frac{1}{n} \sum_{i=1}^{n} \left(x^i - p_\theta(q_\phi(\tilde{x}^i))\right)^2, \tag{1.11}$$

where $\tilde{x}^i$ is the corrupted version of x^i drawn from $C(\tilde{\mathbf{x}}|\mathbf{x})$. q_ϕ and p_θ are the encoder and decoder parts of DAE, respectively. n is the number of data samples in a dataset.

1.3.3. Variational AutoEncoder

A Variational AutoEncoder (VAE) [25] is a variant of an AE that also consists of two parts: encoder and decoder (Fig. 1.2 (b)). The difference between a VAE and an AE is that the bottleneck of the VAE is a Gaussian probability density ($q_\phi(\mathbf{z}|\mathbf{x})$). We can sample from this distribution to get noisy values of the representations $\mathbf{z}$. The decoder inputs a latent vector $\mathbf{z}$ and attempts to reconstruct the input. The decoder is denoted by $p_\theta(\mathbf{x}|\mathbf{z})$.

The loss function of a VAE $\ell_{VAE}(x^i, \theta, \phi)$ for a datapoint x^i includes two terms as follows:

$$\begin{aligned} \ell_{VAE}(x^i, \theta, \phi) = &- \mathbf{E}_{q_\phi(\mathbf{z}|x^i)}\left[log\, p_\theta(x^i|\mathbf{z})\right] \\ &+ D_{KL}(q_\phi(\mathbf{z}|x^i)||p(\mathbf{z})). \end{aligned} \tag{1.12}$$

The first term is the expected negative log-likelihood of the i-th data point. This term is also called the reconstruction error (RE) of VAE since

it forces the decoder to learn to reconstruct the input data. The second term is the Kullback-Leibler (KL) divergence between the encoder's distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the expected distribution $p(\mathbf{z})$. This divergence measures how close q is to p [25]. In the VAE, $p(\mathbf{z})$ is specified as a standard Normal distribution with mean zero and standard deviation one, denoted as $\mathcal{N}(0,1)$. If the encoder outputs representations z that are different from those of standard normal distribution, it will receive a penalty in the loss. Since the gradient descent algorithm is not suitable to train a VAE with a random variable $\mathbf{z}$ sampled from $p(\mathbf{z})$, the loss function of the VAE is re-parameterized as a deterministic function as follows:

$$\begin{aligned}\ell_{VAE}(x^i,\theta,\phi) = &-\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k}) \\ &+ D_{KL}(q_\phi(\mathbf{z}|x^i)||p(\mathbf{z})).\end{aligned} \tag{1.13}$$

where $z^{i,k} = g_\phi(\epsilon^{i,k}, x^i)$. g is a deterministic function, ϵ^k denotes $\mathcal{N}(0,1)$. K is the number of samples that is used to reparameterize $\mathbf{z}$ for the sample x^i.

After training, the latent layers (i.e., the bottleneck layers or the middle hidden layers) of AEs (AE, DAE, and VAE) can be used for a classification task. The original data is passed through the encoder part of AEs to generate the latent representation. A classification algorithm is then applied to the latent representation instead of the original input.

1.3.4. Generative Adversarial Network

A Generative Adversarial Network (GAN) [26] has two neural networks which are trained in an opposite way (Fig. 1.2(c)). The first neural network is a generator (Ge) and the second neural network is a discriminator (Di). The discriminator Di is trained to maximize the difference between a fake sample $\tilde{\mathbf{x}}$ (comes from the generator) and a real sample $\mathbf{x}$ (comes from the original data). The generator Ge inputs a noise sample $\mathbf{z}$ and outputs a fake sample $\tilde{\mathbf{x}}$. This model aims to fool

the discriminator Di by minimizing difference between $\tilde{\mathbf{x}}$ and $\mathbf{x}$.

$$L_{GAN} = E_{\mathbf{x}}[\log Di(\mathbf{x})] + E_{\mathbf{z}}[\log(1 - Di(Ge(\mathbf{z})))]. \tag{1.14}$$

The loss function of GAN is presented in Eq. 1.14 in which $Di(\mathbf{x})$ is the the probability of Di to predict a real data instance $\mathbf{x}$ is real, $Ge(\mathbf{z})$ is the output of Ge when given noise $\mathbf{z}$, $Di(Ge(\mathbf{z}))$ is the probability of Di to predict a fake instance ($Ge(\mathbf{z})$) is real, $E_{\mathbf{x}}$ and $E_{\mathbf{z}}$ are the expected value (average value) overall real and fake instances, respectively. Di is trained to maximize this equation, while Ge tries to minimize its second term. After the training, the generator (Ge) of GAN can be used to generate synthesized data samples for attack datasets. However, since two neural networks are trained oppositely, there is no guarantee that both networks are converged simultaneously [27]. As a result, GAN is often difficult to train.

Auxiliary Classifier Generative Adversarial Network (ACGAN) [28] is an extension of GAN by using the class label in the training process. ACGAN also includes two neural networks operating in a contrary way: a Generator (Ge) and a Discriminator (Di). The input of Ge in ACGAN includes a random noise z and a class label c instead of only random noise z as in the GAN model. Therefore, the synthesized sample of Ge in ACGAN is $X_{fake} = Ge(c, z)$, instead of $X_{fake} = Ge(z)$. In other words, ACGAN can generate data samples for the desired class label.

1.3.5. Adversarial AutoEncoder

One drawback of VAE is that it uses the KL divergence to impose a prior on the latent space, $p(\mathbf{z})$. This requires that $p(\mathbf{z})$ is a Gaussian distribution. In other words, we need to assume that the original data follows the Gaussian distribution. Adversarial AutoEncoder(AAE) avoids using the KL divergence to impose the prior by using adversarial learning. This allows the latent space, $p(\mathbf{z})$, can be learned from any distribution [29].

Fig. 1.2(d) shows how AAEs work in detail. Training an AAE has two phases: reconstruction and regularization. In the reconstruction phase (RP), a latent sample $\tilde{\mathbf{z}}$ is drawn from the generator Ge. Sample $\tilde{\mathbf{z}}$ is then sent to the decoder (denoted by $p(\mathbf{x}|\tilde{\mathbf{z}})$) which generates $\tilde{\mathbf{x}}$ from $\tilde{\mathbf{z}}$. The RE is computed as the error between $\mathbf{x}$ and $\tilde{\mathbf{x}}$ (Eq. 1.15) and this is used to update the encoder and the decoder.

$$L_{RP} = -\mathbb{E}_{\mathbf{x}}\left[\log p(\mathbf{x}|\tilde{\mathbf{z}})\right]. \tag{1.15}$$

In regularization phase (RG), the discriminator receives $\tilde{\mathbf{z}}$ from the generator Ge and $\mathbf{z}$ is sampled from the true prior $p(\mathbf{z})$. The generator tries to generate the fake sample, $\tilde{\mathbf{z}}$, similar to the real sample, $\mathbf{z}$, by minimizing the second term in Eq. 1.16. The discriminator then attempts to distinguish between $\tilde{\mathbf{z}}$ and $\mathbf{z}$ by maximizing this equation. An interesting note here is that the adversarial network's generator is also the encoder portion of the AE. Therefore, the training process in the regularization phase is the same as that of GAN.

$$L_{RG} = \mathbb{E}_{\mathbf{z}}\left[\log Di(\mathbf{z})\right] + \mathbb{E}_{\mathbf{x}}\left[\log(1 - Di(En(\mathbf{x})))\right]. \tag{1.16}$$

An extension of AAE is Supervised Adversarial AutoEncoder (SAAE) [29] where the label information is concatenated with the latent representation ($\mathbf{z}$) to form the input of Di. The class label information allows CAAE to generate the data samples for a specific class. Another version of AAE is Denosing AAE (DAAE) [30] that attempts to match the intermediate conditional probability distribution $q(\tilde{\mathbf{z}}|\mathbf{x_{noise}})$ with the prior $p(\mathbf{z})$ where $\mathbf{x_{noise}}$ is the corrupted version of the input $\mathbf{x}$ as in Eq. 1.10. Because DAAE reconstructs the original input data from the corrupted version of input data (input data with noise), its latent representation is often more robust than the representation in AAE [23].

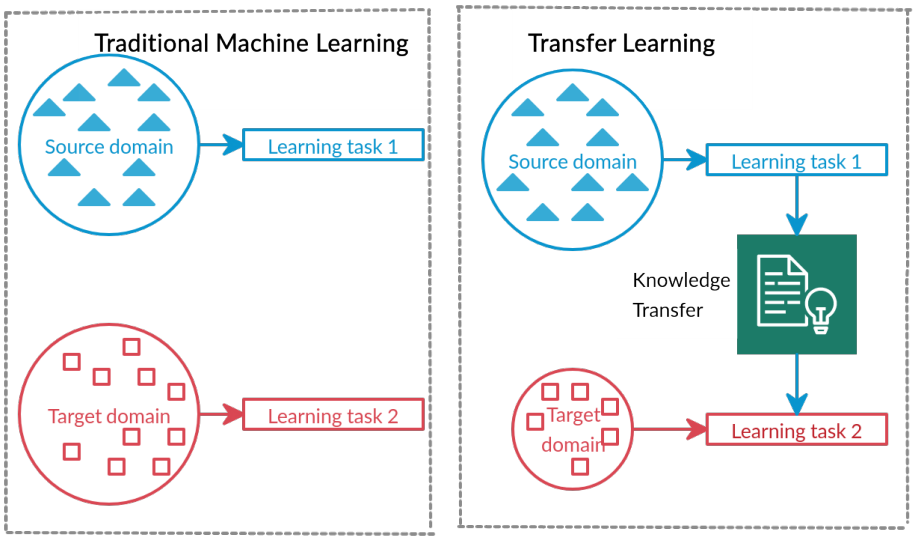


Figure 1.3: Traditional machine learning vs. transfer learning.

1.4. Transfer Learning

Transfer Learning (TL) is used to transfer knowledge learned from a source domain to a target domain where the target domain is different with the source domain but they are related data distributions. This section will present the definition of TL and the distance metric between two data distributions used in the thesis.

1.4.1. Definition

TL refers to the situation where what has been learned in one learning task is exploited to improve generalization in another learning task [1]. Fig. 1.3 compares traditional machine learning methods, including traditional machine learning and TL models. In traditional machine learning, the datasets and training processes are separated for different learning tasks. Thus, no knowledge is retained/accumulated nor transferred from one model to another. In TL, the knowledge (i.e., features, weights) from previously trained models in a source domain is used for training newer models in a target domain. Moreover, TL can even handle the problems of having less data or no label information in the target domain.

We consider the TL method with an input space X and its label space Y. Two domain distributions are given, such as a source domain D_S and a target domain D_T. Two corresponding samples are given, i.e., the source sample $D_S = (X_S, Y_S) = (x_S^i, y_S^i)_{i=1}^{n_S}$ and the target sample $D_T = (X_T) = (x_T^i)_{i=1}^{n_T}$. n_S and n_T are the number of samples in the source domain and the target domain, respectively. The learning task builds a model based on the source domain with label information and the target domain without label information to classify the target domain.

1.4.2. Maximum mean discrepancy (MMD)

Similar to KL divergence [2], MMD is used to estimate the discrepancy of two distributions. However, MMD is more flexible than KL by the ability of estimating the nonparametric distance [31]. Moreover, MMD can avoid computing the intermediate density of the distributions. The definition of MMD can be formulated as Eq. 1.17.

$$MMD(X_S, X_T) = |\frac{1}{n_S}\sum_{i=1}^{n_S}\xi(x_S^i) - \frac{1}{n_T}\sum_{i=1}^{n_T}\xi(x_T^i)|, \tag{1.17}$$

where n_S and n_T are the number of samples of the source and target domain, respectively, ξ presents the representation of the original data x_S^i or x_T^i.

1.5. Evaluation Metrics

In this section, we present two evaluation metrics that will be used to evaluate the performance of our proposed models. These include the AUC score and the model's complexity .

The AUC score is a main performance evaluation metric used to measure the effectiveness of our proposed models. Besides, for each scenario of experiment, we additionally use some other metrics to assess various sides of the proposed models. In such cases, we shortly describe these metrics before using. For example, we use the Parzen window-based log-likelihood of generative models to evaluate the quality of generating samples.

1.5.1. AUC Score

AUC stands for Area Under the Receiver Operating Characteristics Curve [32] that is created by plotting the True Positive Rate (TPR) or Sensitivity against the False Positive Rate (FPR) at various threshold settings. The True Positive Rate (TPR) score measures how many actual positive observations are predicted correctly (Eq. 1.18). FPR is the proportion of real negative cases that are incorrectly predicted (Eq. 1.19).

$$TPR = \frac{TP}{TP + FN}, \tag{1.18}$$

$$FPR = \frac{FP}{TN + FP}, \tag{1.19}$$

where TP and FP are the number of correct and incorrect predicted samples for the positive class, respectively, and TN and FN are the number of corrected and incorrect predicted samples for the negative classes. The advantage of these metrics is that they are very intuitive and easy to implement. However, they make no distinction between classes that are sometime insufficient to measure a classifier especially for imbalanced datasets [32].

A perfect classifier will score in the top left hand corner ($FPR = 0$, $TPR = 100\%$). A worst-case classifier will score in the bottom right hand corner ($FPR = 100\%$, $TPR = 0$). The space under the ROC curve is represented as the AUC score. It measures the average quality of the classification model at different thresholds. A random classifier has an AUC value of 0.5, and the value of the AUC score for a perfect classifier is 1.0. Therefore, the most classifier has the values of AUC score between 0.5 and 1.0.

1.5.2. Complexity of Models

In general, model complexity can be defined as a function of a number of trainable parameters[3]: the more trainable parameters a model

[3]The parameters are updated in the training process of a neural network.

has, the more complex the model is [33, 34]. The function of calculating trainable parameters are slightly different between neural network types. Commonly, for the fully connected layers, the number of trainable parameters can be calculated by $(n+1) \times m$, where n is the number of input units and m is the number of output units, and the $+1$ term presents the bias terms. These can be used to represent the model size of a neural network [33].

As discussed in [34], given equivalent accuracy, one neural network architecture with fewer parameters has several advantages such as more efficient distributed training, less overhead when exporting new models to clients, and embedded development. Therefore, we use the number of trainable parameters of deep neural network based models to compare their model sizes or complexity. Besides, we also report the inference time in each proposed model for comparison. Moreover, the same computing platform (Operating system: Ubuntu 16.04 (64 bit), Intel(R) Core(TM) i5-5200U CPU, two cores and 4GB RAM) was used in every experiment in this thesis.

1.6. Review of Network Attack Detection Methods

After selecting appropriate features to represent network traffic, a NAD system will determine network attacks based on analysing the security data [35]. This system can identify malicious network traffic generated by network attacks by using various methods. Generally, NAD methods can be grouped into three categories, i.e., knowledge-based methods, statistical-based methods, and machine learning-based methods [7, 35].

1.6.1. Knowledge-based Methods

This approach requires a knowledge of some specific network attacks to pre-defined attack rules or signatures of known attack samples. If incoming network traffic matches the pre-defined attack signatures, this traffic is considered a kind of attack. This is the earliest technique used

for the NAD problem [35]. The attack signatures are pre-defined by a type of string. The incoming network traffic is extracted and then matched to the pre-defined signatures. If the information is matched, the incoming traffic will be detected as an attack [36]. This work is a simple method and can detect known attacks accurately. However, with a large number of string rules, the matching process is a massive computation.

Other approaches of the knowledge-based methods for NAD are based on the description of language or an expert system. For example, [37] used a finite state machine to represent and control the execution flow. A state monitors the history data. Then, the finite state machine represents the normal network behaviours. Any observed deviation reported by the finite state machine is considered as an attack. The work [38] defined a language syntax of rules to represent characteristics of attacks. Besides that, the rules can be defined by the combination working of a knowledge engineer and a domain expert as in [39]. These approaches can detect widespread network attacks quickly and accurately. However, it is unable to detect unknown attacks that are more serious for the information security of network systems [40].

1.6.2. Statistical-based Methods

In statistics-based methods, the distribution of normal traffic is built for normal behaviors. In the sequel, the NAD system detects low probability behaviors (i.e., anomalous behaviors) as attacks. These methods use statistical metrics, such as mean, median, and standard deviation of network packets to define a threshold. For an incoming network behavior, if the statistical metric is passed over this threshold, the network behavior is detected as a network attack [7].

Ye et al. [41] created the normal profile for an only univariate and multivariate metric of behavior in computer systems. The NAD system monitors anomalous in each individual metric. Besides, Viinikka et al. [42] aggregate the time-series features to alert anomalous. Qingtao et

al. [43] presented a technique to detect anomaly samples by the abrupt variation of the time-series data. Bhuyan et al. [44] discussed that the statistical-based methods are simple and less accurate.

Both knowledge-based and statistics-based approaches are simple, thereby less time-consuming. However, they usually need a large amount of prior knowledge of network attacks before they can detect these attacks. It is not suitable for the development of the network environment today. With increasing network size and complexity, network attacks are evolved quickly. Therefore, we are unable to have enough prior knowledge before they harm network systems. Moreover, we also cannot detect zero-day attacks (i.e., new attacks).

1.6.3. Machine Learning-based Methods

To handle the issues of knowledge-based and statistics-based methods, the machine learning-based approach is developed dramatically to resolve NAD problems effectively. Machine learning-based methods use machine learning models to detect network attacks by extracting knowledge from enormous network traffic data. Machine learning models can be used to find new characteristics of network traffic data. As a result, they can predict network behaviors as normal or certain types of attacks. The NAD problem based on machine learning is currently receiving considerable interest from the research community. One of the appealing properties of a machine learning-based NAD system is its ability to detect new or unknown attacks. It is discussed more detail in the sequel. Machine learning-based methods can be divided into three groups, i.e., unsupervised learning, semi-supervised learning, and supervised learning.

The ability of building models without collecting attack data from unsupervised and semi-supervised samples is considered in many NAD systems. [45] proposed to use the K-mean clustering algorithm to reduce the network packet payload size. Then, the packet payload content is classified more effectively. Hongchun et al. [46] proposed a framework

for attack detection in wireless sensor networks. In this framework, an unsupervised machine learning method, i.e., Mean Shift Clustering algorithm, was used to detect anomalous patterns that reflect the anomalous behaviors from the normal context in wireless sensor networks. In the sequel, they used SVM for maximizing the margin between normal and anomalous features. Nomm et al. [13] proposed a semi-supervised approach for identifying IoT attacks in which the datasets are re-sampled before using the Local outlier factor and One-Class SVM to detect malicious samples. The drawback of this work is that the sampling technique can change the original data distribution, thereby reducing the effectiveness of Local outlier factor and One-Class SVM.

The advantage of unsupervised and semi-supervised learning is that they can detect unknown attacks without any prior knowledge. Moreover, they do not require label attacks when building models. However, they do not adequately detect known attacks due to training models without knowledge about attacks. Moreover, we usually need to set threshold values that are used to distinguish normal and anomalous data samples manually. It reduces learning automatically from the data of a machine learning model. The next sub-sections will present the previous work related to three approaches of the thesis.

1.6.3.1. Machine Learning Methods for Network Attack Detection

The first popular machine learning technique is used for NAD problems is SVM. The idea of SVM is that input vectors are non-linearly mapped to a higher dimensional feature space, which is a linear decision surface [47]. The effectiveness of SVM in the NAD problem has proven in many previous works [13, 48–50]. For expressly, [50] analyzed SVM-based NAD techniques. Their experimental results showed that Linear SVM, Quadratic SVM, Fine Gaussian SVM, and Medium Gaussian SVM provide very high accuracy on the NAD problem. Besides that, SVM can be used for feature selection to extract essential features for attack detection systems [51].

The second machine learning approaches are based on tree architectures Nadiammai et al. [52] analyzed the performance of tree-based algorithms, including Decision Stump, Bloom filter Tree, Iterative Dichotomiser 3 (ID3), J48, Logical Analysis of Data (LAD), Random Tree, REP Tree, Random Forest (RF), and Simple Cart algorithms for the NAD problem. They proved that RF is the best classifier for identifying network attacks based on the NSL-KDD dataset. RF is constructed by bagging ensembles of random trees [53]. Paulo et al. [54] provided the survey of using RF for NAD. [55] also implemented RF for feature selection and attack detection on the NSL-KDD dataset. These proved that RF is one of the popular tree-based classifiers for NAD systems. Besides, Bahsi et al. [11] used the decision tree and K-nearest neighbor algorithms to identify the Mirai botnet family and the Gafgyt botnet family. Chawathe [12] applied a number of machine learning algorithms, including ZeroR and OneR, rule-based, and tree-based classifiers (J48 and Random Forest) to detect IoT anomalies.

The above machine learning approaches can detect known attacks effectively due to using the information of both normal data and known attacks for training. However, these approaches are ineffective in detecting new types of attacks. Moreover, the accuracy of a machine learning model is affected by an imbalanced data problem.

Besides effectiveness of machine learning-based methods for NAD, these methods face with three challenges presented in the INTRODUCTION part of the thesis. First, the network traffic is heterogeneity and complexity. Second, the network traffic datasets are usually imbalanced. Third, there are many types of networks in which we are difficult to collect the data with label information. Therefore, this thesis aims to resolve these challenges in the NAD model using machine learning methods.

1.6.3.2. Machine Learning Methods for Handling Imbalanced Data

Techniques for handling the imbalance problem can be grouped into two categories: cost-sensitive learning and data re-sampling. The first group operates at the algorithmic level by assigning higher miss classification costs to the minority class than to the majority [56–59]. Zhang et al. [57] divided the data into smaller balanced subsets using an intelligent sampling technique. After that, a cost-sensitive SVM learning is applied to deal with the imbalance problem. Li et al. [59] proposed an approach for dealing with imbalanced data by setting a higher weight to the minority class during training an ensemble machine learning algorithm (Adaboost).

Recently, many new approaches have been proposed to address the imbalance problem in deep neural networks. Chung et al. [58] introduced a new regression loss function for multi-class deep learning to handle the imbalance problem. Wang et al. [60] and Raj et al. [61] proposed loss functions that independently calculate the error for classes and then averages them together. Khan et al. [56] proposed a cost-sensitive deep neural network that can automatically learn robust feature representations for both the majority and minority classes.

The second group aims to alter the training data distribution to balance the majority and minority classes, usually by random under-sampling the major or over-sampling the minor [62, 63]. However, these techniques have some potential downsides. While under-sampling can lose useful information about the majority class data, over-sampling does not increase any information since the exact copies of the minority class are replicated randomly [63]. To overcome this limitation, Synthetic Minority Over-sampling Technique (SMOTE) [64] generates the synthesized minority class by extrapolating and interpolating minority samples from the neighborhood ones. It has the effect of creating clusters around each minority observation. An extension of SMOTE is SMOTE-SVM [65] that synthesizes samples located in the borderline between

classes by only generating samples for the support vectors of an SVM model trained on the original dataset.

Ensemble methods (hybrid methods) combine a classifier's re-sampling technique to discover the majority and minority classes. BalanceCascade [66] develops an ensemble of classifiers to select which majority of class samples to under-sample systematically. The number of classifiers is applied in sub-datasets, which include majority and minority samples with a balanced rate. The majority of samples will be removed if they are classified correctly from classifiers. EasyEnsemble [66] learns different aspects of the original majority class in an unsupervised manner. First, many balanced training sets are created using a random under-sampling technique. Next, a model is trained on each dataset, and the prediction is combined as in the bagging technique [67]. Tomek Link [68] removes samples from the negative class that is close to the positive region to return a dataset that presents a better separation between the two classes.

Although SMOTE-SVM, BalanceCascade, and EasyEnsemble have been widely adopted and their effectiveness has been well-evidenced [62, 69], the shortcoming of these methods is that the distribution of the original data can be lost [70]. In other words, the generated data samples may have a very different distribution from the original data. Subsequently, the performance of machine learning algorithms trained on the augmented dataset may be hurt.

Recently, generative models are widely used to generate data samples. A Generative Adversarial Network (GAN) was used to create images with various styles [26]. To generate samples for a specific class, some extensions of generative models have been proposed. Auxiliary Classifier Generative Adversarial Network (ACGAN) [28] is an extension of GAN by using the class label in the training process. ACGAN can generate data samples for the desired class label. Similar to GAN, the training process in ACGAN using a gradient descent algorithm may not be

converged. Since two cost functions are updated independently, there is no guarantee to enhance the training error for both neural networks in ACGAN [27]. Supervised Adversarial AutoEncoder (SAAE) [29] and Conditional Variational AutoEncoder (CVAE [71]) are the extensions of AAE and VAE, respectively, which are also used to generate the images with a specific class. However, the AAE-based model has the advantage of training processes to enhance the better quality samples [29].

Therefore, in Chapter 3, we propose a novel method for generating the synthesized malicious samples of various network attacks. Notably, the synthesized samples of our method are proved to be stronger correlated to the original data distribution compared with those of the previous approaches.

1.6.3.3. Deep Neural Network Methods for Representation Learning

For learning representation data, deep learning has attracted paramount interest in network attack detection, e.g., [9,21,72–74], in which AEs play pivotal roles, e.g., [9,21,75,76]. Meidan et al. [9] proposed an AE model to train with the normal data samples. It then sets a RE threshold to classify an unseen data sample to be a normal or malicious one. Juliette et al. [75] presented an online and real-time unsupervised NAD algorithm which uses a discrete time-sliding window to continuously update the feature space and an incremental grid clustering. Ibidunmoye et al. [76] estimated an underlying temporal property of the stream via adaptive learning, and then used statistically robust control charts to recognize deviations. However, these approaches require frequent adjustment of the threshold value.

More recently, Cao et al. [21] proposed two AE-based models, namely Shrink AE (SAE) and Dirac Delta VAE (DVAE), to learn a latent representation. This latent representation aims to facilitate one-class NAD methods in dealing with high-dimension data. Specifically, the regularizer in [21] helps SAE and DVAE learn (in a semi-supervised manner) to project normal class in a small region at the origin. It is based on the

assumption that only normal samples are available for training. They did not use any information about the attacked class to train the representation models. Besides that, Wang et al. [73] and Li et al. [72] presented a learning representation method based on Convolution Neural Network. However, this network is rarely used for NAD due to the 1-D dimension of security data. Moreover, Deep Belief Network (DBN) was proposed as a very effective representation learning model that can be used in NAD [77].

In many scenarios, however, a particular type of IoT attacks can be collected and labeled. In this case, supervised learning-based methods are usually better than semi-supervised methods in detecting known attacks. Therefore, in Chapter 2, our work aims to develop a novel latent representation that facilitates supervised learning-based NAD methods in detecting unknown/new attacks. Moreover, our proposed regularizers in this work can also be expanded to multi-classification problems that the models in [21] cannot do.

1.6.3.4. Deep Neural Networks for Transfer Learning

The approaches using deep learning techniques to isolate the attack samples (anomaly samples) have been used in several previous researches for NAD [9, 21, 75, 76]. Juliette et al. [75] presented a new online and real-time unsupervised network anomaly detection algorithm that uses a discrete time-sliding window to update continuously the feature space and an incremental grid clustering to detect anomalies. In [76], an underlying temporal property of the stream is estimated via transfer learning (TL), and then statistically robust control charts are applied to recognize deviations. However, this approach highly depends on the chosen threshold value. Another work represents the original data in a new space that encourages the classification tasks [21]. However, general machine learning models usually need to assume that the data distribution of training datasets is similar to the data distribution of testing datasets. However, many real-world applications are unable to make this assumption [78,79].

Therefore, deep transfer learning (DTL) techniques are recently leveraged widely. The works in [78, 80, 81] give details of categories and applications of DTL. 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.

The instance-based DTL approach selects samples from the source domain to supplement the training set in the target domain. These samples are assigned appropriate weight values in the training set. [82] 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 appropriate weights. To learn sample weights, [83] introduced a metric DTL framework together with learning a distance between two domains. This framework makes knowledge transfer between domains more effectively. An ensemble DTL introduced in [84] can leverage samples from the source domain to train on the target domain.

Mapping-based DTL 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 be considered as a training set of a deep neural network. The work in [85] introduced an adaptation layer and an additional domain confusion loss based on MMD to learn a new representation space. In this new representation space, the source domain and target domain are invariant. To measure the distance of joint distributions, joint MMD (JMMD) was introduced in [86] to transfer the data distribution in different domains and improve previous works.

Adversarial-based DTL refers to use adversarial networks inspired by generative adversarial nets (GAN) [26] to get a representation space that is suitable to both the source domain and target domain. [87] introduced a new loss function of GAN combining with a discriminative model to

a DTL method. A randomized multi-linear adversarial network was introduced in [88] to find the multiple feature layers. It can allow both DTL and discriminative adversarial DTL.

Network-based DTL reuses the network structure and parameters trained in the source domain for training the target domain. [89] reused front-layers of Convolutional Neural Network (CNN) trained on the ImageNet dataset to map images in other datasets to intermediate image representation. It helps to transfer knowledge to other object recognition tasks with a small training set size. [90] introduced an approach to learning adaptive classifiers with a residual function. Several DTL approaches based on AEs were introduced in [2, 3, 91]. They used different AE-based models such as AE [2], denoising AE [91], and sparse AE [3] for some specific applications. In previous work based on AE, the transferring process is executed only on the bottleneck layer. They may use the KL metric [2] or MMD metric [3] 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 for enhance attack detection in IoT. More specifically, transferring the higher level of features from the source domain with labels to the target domain without labels helps to improve the classifying tasks of the target domain. This approach improves the accuracy of learning tasks on the target domain with limited samples and no labels. Therefore, we propose a DTL model in Chapter 4 that uses the network-based DTL approach to enhance the classifying task's performance on the target samples without target labels.

Our proposed DTL model in Chapter 4 leverages a non-linear mapping of a neural network (i.e., AE) to improve the performance of attack detection on the target domain. The key idea of our proposed DTL model (compared with previous AE-based DTL methods [2, 3]) is that

the knowledge of features in *every encoding layer* (instead of the only bottle layer in previous works) is transferred to the target domain. It helps to force the latent representation of the target domain is similar to the latent representation of the source domain. The experimental results illustrate the effectiveness of our proposed DTL model on the target domain's attack detection task.

1.7. Conclusion

This chapter presents the fundamental backgrounds of related work to this thesis. We define the NAD problem as well as the common security datasets used for NAD methods. In the sequel, approaches for this problem are shown including statistical-based methods, knowledge-based methods, and machine learning-based methods. Between these methods, machine learning-based methods are more powerful methods and more widely used in the NAD problem.

In the NAD problem, both known attacks and unknown attacks are dangerous for network systems. Therefore, we aim to make the machine learning approach more effectively to detect both known and unknown attacks. First, we will develop the deep neural network models that help to represent network traffic in an appropriate feature space where classification algorithms can distinguish normal samples and attack samples more accurately. Second, we will develop a deep neural network to handle the imbalanced data problem, which usually reduces the accuracy of a machine learning model. Finally, we will handle "the lack of label" information on a new domain of network traffic by developing a DTL technique. It can improve NAD in a domain of network traffic, which has no label information, with the help of label information of a related domain.

Chapter 2
LEARNING LATENT REPRESENTATION FOR NETWORK ATTACK DETECTION

The Internet has emerged as a cutting-edge technology that is changing human life. However, the rapid and widespread applications of Internet services make cyberspace more vulnerable, especially to network attacks in which network devices are used to launch attacks on cyber-physical systems. Thus, detecting and preventing these network attacks is critical. However, this task is very challenging due to the continuous and fast evolving of attackers. Among network attacks, unknown ones are far more devastating as these attacks could surpass most of the current security systems and it takes time to detect them and "cure" the systems.

To effectively detect new/unknown attacks, in this chapter, we propose a novel representation learning method to enhance the accuracy of deep learning in detecting network attacks, especially unknown attacks. Specifically, we develop three regularized versions of AutoEncoders (AEs) to learn a latent representation from the input data. The bottleneck layers of these regularized AEs trained in a supervised manner using normal data and known network attacks in the IoT environment will then be used as the new input features for classification algorithms. The contributions of this chapter are published and accepted to the work [ii] and [vi] of our publications.

2.1. Introduction

With the rapid development of network devices and services, networks have been providing enormous benefits to many aspects of our life, such as healthcare, transportation, and manufacturing [92]. The data trans-

mitted from/to these network devices often contains sensitive information of users, such as passwords, phone numbers, photos and locations, which usually attracts hackers [93]. Moreover, the rapid growth in the number of diverse network devices and services can lead to a dramatic increase in the number of emerging network attacks (also referred to as [21]. Identifying attacks in the IoT environment with a massive number of devices and services would be a challenging, especially with the fast and continuous evolution of network attacks [9, 93–98].

Machine learning has proven its great potential in the NAD problem [9–13]. Depending on the availability of data labels, machine learning-based NAD can be categorized into three main approaches: supervised learning, semi-supervised learning, and unsupervised learning [14]. Supervised learning methods assume that both labeled normal and abnormal data are available for constructing predictive models. They attempt to model the distinction between normal behaviors and anomalous activities. The downside of supervised learning methods is that they require a sufficient number of normal and abnormal instances to achieve good performance. Moreover, these methods are often ineffective in detecting unknown attacks, i.e., the attacks which are not contained in the training data. Semi-supervised learning methods assume that only labeled normal data is available for training models. These methods construct generative models representing normal behaviors and measure how well an unseen sample fits the models. Unsupervised methods require no labeled training data and rely on the assumption that the number of attack samples is far fewer than that of the normal samples [14]. Since unsupervised and semi-supervised approaches do not require abnormal data to construct predictive models, they are often more robust to unknown attacks and hence widely applied to NAD [21, 44, 99]. The limitation of semi-supervised and unsupervised approaches is, however, that they might not be as effective as supervised approaches in identifying the previously known attacks. In this chapter, we develop a novel approach

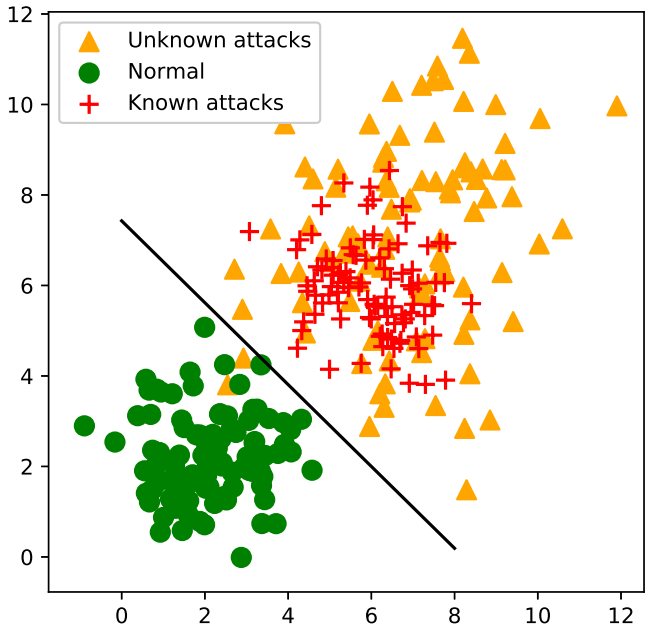


Figure 2.1: Visualization of our proposed ideas: Known and unknown abnormal samples are separated from normal samples in the latent representation space.

that performs effectively on both known and unknown attacks.

Among many types of attacks, unknown ones are the most dangerous but difficult to detect since these attacks could surpass most of the advanced security techniques and cause serious devastation to network systems [21, 100]. Moreover, network attack data is very heterogeneous and complex, leading to inaccurate NAD systems. Recently, deep learning-based NAD has received more considerable attention from researchers and industry [9, 21, 72–74, 101] to learn the representation of network attack data. Among deep learning-based techniques, AEs are widely used for NAD [21]. In this chapter, we propose a novel learning approach that can inherit the strength of supervised learning methods in detecting known IoT attacks and the ability to identify unknown attacks of unsupervised methods.

In the new representation space, normal data and known network attacks will be forced into two tightly separated regions, called normal

region (green circle points in Fig. 2.1) and anomalous region (red plus points in Fig. 2.1), respectively. We hypothesize that unknown attacks will appear closer to the anomalous region (yellow triangle points in Fig. 2.1) as they may share some common characteristics with known ones. Hence, they can be easily detected. To obtain the feature representation, we develop two new regularized AEs, namely Multi-distribution AE (MAE) and Multi-distribution Denoising AE (MDAE). These AEs will learn to construct the desired feature representation at their bottleneck layers (also called "latent feature space"). The resulting representation can facilitate supervised learning-based NAD methods, such as Linear Support Vector Machine (SVM), Perceptron (PCT), Nearest Centroid (NCT), and Linear Regression (LR). Because experiments aim to emphasize the effectiveness of representation methods, thus, we prefer to use the linear/simpler classifiers, e.g., SVM, PCT, NCT, and LR , to lessen the impact of classifiers on the results.

The major contributions of this chapter are as follows:

- Introduce a new latent feature representation to enhance the ability to detect unknown network attacks of supervised learning-based NAD methods.
- Propose three novels regularized AEs to learn the new latent representation. A new regularizer term is added to the loss function of these AEs to separate normal samples from abnormal samples in the latent space. This latent representation is then used as the input to classifiers to identify abnormal samples.
- Perform extensive experiments using nine latest IoT botnet datasets to evaluate our models. The experimental results show that our learning representation models help simple classifiers perform much better when comparing to learning from the original features or using latent representations produced by other AEs.
- Conduct a thorough analysis of the characteristics of the latent rep-

resentation in detecting unknown attacks, performing on a cross-dataset test, and its robustness with various values of hyper-parameters. This analysis sheds light on the practical applications of the proposed models.

The rest of the chapter is organized as follows. The proposed models are presented in Section 2.2. Section 2.4 presents the experimental settings. Section 2.5 discusses and analyzes results obtained from the proposed models. Finally, in Section 2.6, we conclude and suggest potential future work.

2.2. Proposed Representation Learning Models

In this section, the chapter describes the proposed latent representation that facilitates supervised learning-based NAD methods in identifying attacks, especially unknown attacks. This chapter then presents three novel regularized AEs that can learn to construct the new latent representation of data.

In the latent representation, normal samples and known attacked samples are forced to distribute into two tightly separated regions, the normal region, and the anomalous region, respectively. Unknown attacks that may share some common attributes with known attacks can be identified as closer to the anomaly than the normal region. Our approach is to develop the AE-based models that can learn to construct the new feature representation at the bottleneck layer to achieve the feature representation. This chapter introduces new regularized terms to the loss functions of AEs. Data labels are incorporated into the regularizers to compress normal and known attacked data into two tiny separated regions associated with each class of data in the latent representation. The latent representation is then used as the input of binary classifiers, such as SVM and LR. The output of these classifiers is the final score to determine the abnormality of the input data sample.

As such for effective learning the latent representation, this chapter

introduces new regularizers to a classical AE and a DAE to form two regularized AEs. These AEs are named as Multi-variational AE (MVAE), Multi-distribution AE (MAE) and Multi-distribution DAE (MDAE). Our proposed models are also very different from the regularized AEs presented in [21]. Specifically, the regularized AEs in [21] can learn to represent only normal class into a small region at the origin in a semi-supervised manner.

2.2.1. Muti-distribution Variational AutoEncoder

Muti-distribution Variational AutoEncoder (MVAE) is a regularized version of VAE, aiming to learn the probability distributions representing the input data. To that end, this chapter incorporates the label information into the loss function of VAE to represent data into two Gaussian distributions with different mean values. Given a data sample x^i with its associated label y^i, μ_{y^i} is the distribution centroid for the class y^i. The loss function of MVAE on x^i can be calculated as follows:

$$\begin{aligned}\ell_{MVAE}(x^i, y^i, \theta, \phi) = &-\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k}, y^i) \\ &+ D_{KL}(q_\phi(z^i|x^i, y^i)||p(z^i|y^i)),\end{aligned} \tag{2.1}$$

where $z^{i,k} = g_\phi(\epsilon^{i,k}, x^i)$. g is a deterministic function and $\epsilon^k \sim \mathcal{N}(0, 1)$; K and y^i are the number of samples used to reparameterize x^i and the label of the sample x^i, respectively.

The loss function of MVAE consists of two terms. The first term is RE or the expected negative log-likelihood of the i-th data point to reconstruct the original data at its output layer. The second term is created by incorporating the label information to the posterior distribution $q_\phi(z^i|x^i)$ and the prior distribution $p(z^i)$ of VAE in Eq. 1.13. Therefore, the second term is the KL divergence between the approximate distribution $q_\phi(z^i|x^i, y^i)$ and the conditional distribution $p(z^i|y^i)$. The objective of adding the label information to the second term is to force the samples from each class data to reside in each Gaussian distribution conditioned

on the label y^i. Moreover, $p(z^i|y^i)$ follows the normal distribution with the mean μ_{y^i} and the standard deviation 1.0, $p(z^i|y^i) = \mathcal{N}(\mu_{y^i}, 1)$[1]. The posterior distribution $q_\phi(z^i|x^i, y^i)$ is the multi-variate Gaussian with a diagonal covariance structure. In other words, $q_\phi(z^i|x^i, y^i) = \mathcal{N}(\mu^i, (\sigma^i)^2)$, where μ^i and σ^i are the mean and standard deviation, respectively, are sampled from the sample x^i. Thus, the Multi-KL term in Eq. 2.1 is rewritten as follows:

$$\begin{aligned} &D_{KL}(q_\phi(z^i|x^i, y^i)||p_\theta(z^i|y^i)) \\ &\qquad = D_{KL}(\mathcal{N}(\mu^i, (\sigma^i)^2)||\mathcal{N}(\mu_{y^i}, 1)). \end{aligned} \tag{2.2}$$

Let D, μ^i_j and σ^i_j denote the dimension of z^i, the j-th element of μ^i and σ^i, respectively; $\mu_{y^i_j}$ is the j-th element of μ_{y^i}. Then, applying the computation of the KL divergence, the Multi-KL term is rewritten as follows:

$$\begin{aligned} &D_{KL}\left(q_\phi(z|x^i, y^i)\|p_\theta(z|y^i)\right) \\ &\qquad = \frac{1}{2}\sum_{j=1}^{D}\Big((\sigma^i_j)^2 + (\mu^i_j - \mu_{y^i_j})^2 - 1 - \log((\sigma^i_j)^2)\Big). \end{aligned} \tag{2.3}$$

Taking Multi-KL term in Eq. 2.3, the loss function of MVAE in Eq. 2.1 finally is rewritten as follows:

$$\begin{aligned} \ell_{MVAE}(x^i, y^i, \theta, \phi) = &-\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k}, y^i) \\ &+ \lambda\frac{1}{2}\sum_{j=1}^{D}((\sigma^i_j)^2 + (\mu^i_j - \mu_{y^i_j})^2 - 1 - \log((\sigma^i_j)^2)), \end{aligned} \tag{2.4}$$

where λ is a parameter to control the trade-off between two terms in Eq. 2.4 as discussed in [21]. The trade-off parameter λ is approximated by the ratio of two loss terms, i.e., the RE and Multi-KL terms, in the loss function of MVAE. This helps to reduce both two loss terms more efficiently.

[1]This chapter has tested several small values ($10^{-3}, 10^{-2}$ and 10^{-1}) for the covariance of the Gaussian distributions of the two classes in order to shorten "tails" of these distributions. At the early iterations of the MVAE training process, the Multi-KL term of MVAE is extremely large compared to the RE term, which makes MVAE difficult to reconstruct the input data. In the later iterations, the Multi-KL term is small, but both of the two terms fluctuate substantially [21].

The mean values for the distributions of the normal class and attack class are chosen to make these distributions located far enough from each other. In our experiments, the mean values are 4 and 12 for the normal class and attack class, respectively. These values are calibrated from the experiments for the good performance of MVAE. In this chapter, the distribution centroid μ_{y^i} for the class y^i, and the trade-off parameter λ are determined in advance. The hyper-parameter μ_{y^i} can receive two values associated with the normal class and the attack class. The trade-off parameter λ is approximated by the ratio of two loss terms in the loss function of our proposed models.

2.2.2. Multi-distribution AutoEncoder

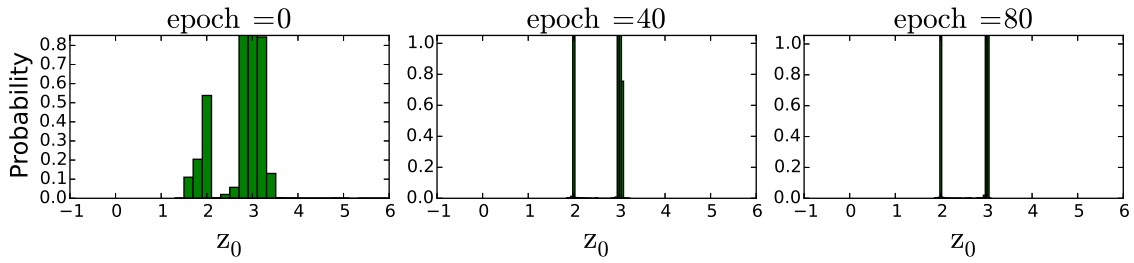


Figure 2.2: The probability distribution of the latent data ($\mathbf{z}_0$) of MAE at epoch 0, 40 and 80 in the training process.

This subsection describes how to integrate a regularizer to an AE to create Multi-distribution AutoEncoder (MAE). The regularizer is a multi-distribution penalty, called $\Omega(\mathbf{z})$, on the latent representation $\mathbf{z}$. The penalty $\Omega(\mathbf{z})$ encourages the MAE to construct a new latent feature space in which each class of data is projected into a small region. Specifically, this chapter incorporates class labels into $\Omega(\mathbf{z})$ to restrict the data samples of each class to lie closely together centered at a pre-determined value. The new regularizer is presented in Eq. 2.5.

$$\Omega(\mathbf{z}) = ||\mathbf{z} - \mu_{y^i}||^2, \tag{2.5}$$

where $\mathbf{z}$ is the latent data at the bottleneck layer of MAE, and μ_{y^i} is a distribution centroid of class y^i in the latent space. The label y^i used in

$\Omega(\mathbf{z})$ maps the input data into its corresponding region defined by μ_{y^i} in the latent representation. The latent feature space is represented by multiple distributions based on the number of classes. Thus, this chapter names the new regularized AE to be Multi-distribution AE.

In the MAE loss function, this chapter also uses a parameter λ to control the trade-off between the reconstruction error (RE) and $\Omega(\mathbf{z})$ terms as discussed in Sub-section 2.2.1. Thus, the loss function of MAE can be defined as follows:

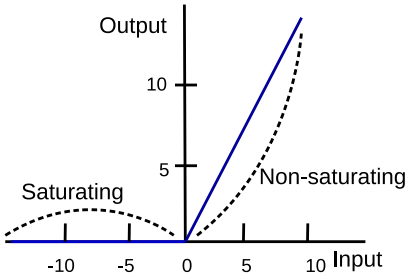
$$\ell_{MAE}(\theta, \phi, \mathbf{x}, \mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \hat{x}^i\right)^2 + \lambda\frac{1}{n}\sum_{i=1}^{n}\left|\left|z^i - \mu_{y^i}\right|\right|^2, \tag{2.6}$$

where x^i, z^i and $\hat{x}^i$ are the i-th element of the input samples, its corresponding latent data and reconstruction data, respectively. y^i and μ_{y^i} are the label of the sample x^i and the centroid of class y^i, respectively. n is the number of training samples. The first term in Eq. 2.6 is the RE that measures the difference between the input data and its reconstruction. The second term is the regularizer used to compress the input data to the separated regions in the latent space.

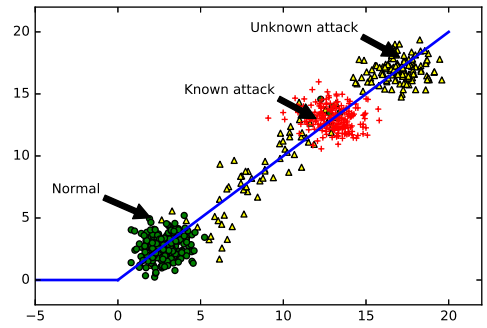
To visualize the probability distribution of the latent representation of MAE, i.e., $\mathbf{z}$, we calculate the histogram of one feature of the latent data $\mathbf{z_0}$. Fig. 2.2 presents the probability distribution of $\mathbf{z_0}$ of normal class and known attacks during the training process of MAE on the IoT-1 dataset. After some epochs, the latent data is constrained into two tight regions in the latent representation of MAE.

2.2.3. Multi-distribution Denoising AutoEncoder

In this subsection, this chapter discusses the details of the multi-distribution Denoising AutoEncoder (MDAE). In this chapter, this chapter employs DAE proposed in [23] to develop MDAE. For each data sample x^i, this chapter can draw its corrupted version $\tilde{x}^i$ using Eq. 1.10. MDAE learns to reconstruct the original input x^i from a corrupted data $\tilde{x}^i$, and also penalizes the corresponding latent vector z^i to be close to



(a) Description of saturating and non-saturating areas of the ReLu activation function.



(b) Illustration of the output of ReLU: two separated regions for normal data and known attacks; unknown attacks are hypothesized to appear in regions toward known attacks.

Figure 2.3: Using non-saturating area of activation function to separate known and unknown attacks from normal data.

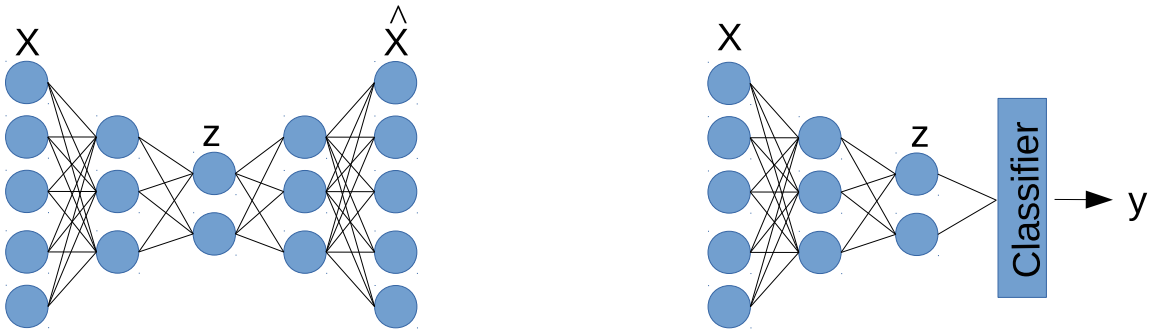
μ_{y^i}. The loss function of MDAE can be presented in Eq. 2.7.

$$\begin{aligned}\ell_{MDAE}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{z}, \phi, \theta) =& \frac{1}{n}\sum_{i=0}^{n}\left(x^i - p_\theta(q_\phi(\tilde{x}^i))\right)^2 \\ &+ \lambda\frac{1}{n}\sum_{i=1}^{n}\left|\left|z^i - \mu_{y^i}\right|\right|^2,\end{aligned} \tag{2.7}$$

where z^i is the latent vector of the data sample x^i. μ_{y^i} is the predefined distribution centroid of the class y^i in the latent feature space of MDAE. q_ϕ and p_θ are the encoder and decoder parts as in DAE, respectively. n is the number of training samples. The hyper-parameter λ controls the trade-off between two terms in Eq. 2.7.

To be better separate the normal data and known attacks and to encourage unknown attacks moving toward the anomaly region, μ_{y^i} of MAE and MDAE is selected in the non-saturated area of activation. The non-saturated area is the steep slope area of the graph of the activation function (as shown in Fig. 2.3 (a)). Thus, μ_{y^i} needs to be assigned to a positive value under the ReLu activation function. In our experiments, this chapter sets $\mu_{y^i} = 2$ for a normal class and $\mu_{y^i} = 3$ for an abnormal class[2]. These values of μ_y^i are used in all the IoT-based datasets in our

[2]These values are calibrated from the experiments for the good performance of the proposed model. The data points within each data class are forced to be very close to each class's distribution centroid. Thus, it is sufficient to separate two normal and anomalous regions with a pair of centroids values 2 and 3. The pair of centroid values also keep almost



(a) An AE-based model.

(b) Using an AE-based model to detect network attacks.

Figure 2.4: Illustration of an AE-based model (a) and using it for classification (c,d).

experiments.

Given the assigned values of μ_{y^i}, the regularized term will attempt to project the two classes (normal data and known attacks) into two tightly separated regions on the non-saturated area. If an attack is different from normal data points in the input feature space, it tends to differ greatly in the latent representation space [21]. The unknown attacks are predicted to project to different regions towards the known anomaly region on the non-saturated area of the activation function. Fig. 2.3 (b) illustrates our idea of learning the latent representation using the ReLU activation function. In this representation, normal data and known attacks are represented in two separate regions, and unknown attacks are predicted to appear in regions closer to known attacks.

2.3. Using Proposed Models for Network Attack Detection

This section presents the training and predicting processing when applying the proposed latent representation learning models for NAD. These processes can be applied for our proposed models such as MVAE, MAE, and MDAE. Thus, we use the AE-based model term for all these proposed models in this sub-section.

2.3.1. Training Process

latent vectors being not much larger than the input and the output of MAE and MDAE, resulting in an easy training process.

Algorithm 1 Training a AE-based model.

INPUT:
x, y: Training data samples and corresponding labels.
The AE-based model with its hyper-parameters.
The classifier with its hyper-parameters.
OUTPUT: Trained AE-based model, Trained classifier.
BEGIN:
1. Put x, y to the input of the AE-based model.
2. Training to minimize the loss function of the AE-based model as described in Section 2.2.
3. Put x, y to the trained AE-based model. To get the latent representation of x is z.
4. Train the classifier with input as z, y.
return Trained AE-based model, Trained classifier.
END.

Algorithm 1 presents the training process when using the AE-based model for NAD. First, this chapter trains the AE-based model on the original network attack dataset x, y. This training process is executed by an optimization method (e.g., Adam) to minimize the loss function[3] of the AE-based model. This step tries to train a latent representation learning model based on AE as illustrated in Fig. 2.4 (a). Second, the latent representation z of the original data is received by fitting the original data to the trained AE-based model. Third, a classifier is trained on the latent representation data z in a supervised manner as described in Fig. 2.4 (b). Finally, we have training results, including the trained AE-based model and the trained classifier.

2.3.2. Predicting Process

Algorithm 2 Predicting process based on representation learning models.

INPUT:
x^i: Testing data samples in the target domain
Trained AE-based model
Trained classifier
OUTPUT:
y^i: Label of x^i
BEGIN:
1. Putting x^i to the input of the trained AE-based model to get the corresponding latent representation as z^i.
2. Putting z^i to the trained classifier to get the output y^i
return y^i
END.

Algorithm 2 describes the process of sample prediction for NAD using

[3]The computations of loss functions are presented in Section 2.2.

our proposed models. First, we have a latent representation of an original data sample x^i ,i.e., z^i, by fitting it to the trained AE-based model. Second, the trained classifier predicts the label y^i with the input as z^i as illustrated in Fig. 2.4 (b). Therefore, the classifier identifies a network traffic sample based on its latent representation instead of its original representation.

2.4. Experimental Settings

The experiments in this chapter are based on the IoT datasets presented in chapter 1. This section presents the experimental sets and hyper-parameter setting used in this chapter.

2.4.1. Experimental Sets

Four linear classification algorithms, including Support Vector Machine (SVM), Perceptron (PCT), Nearest Centroid (NCT), and Linear Regression (LR) [102], are applied to the latent representation produced from MVAE, MAE, and MDAE. We choose these linear and simple classifiers due to two reasons, i.e., (1) to performed very fast and (2) to express the strengthen of representation models. Thus, these algorithms are appropriate to use in IoTs networks where the device's computing resource is often constrained. The experiments were conducted on nine IoT datasets (specified below). All techniques were implemented in python using Tensorflow, and Scikit learn frameworks [102].

To highlight the strength of the proposed models, the performances of the four classifiers trained on the latent representation of MVAE, MAE, and MDAE are compared with those from: (1) standalone classifiers (without using latent representation), including a very effective and widely use for NAD, RF [103, 104]; (2) classifiers using the latent representations of AE, DAE, VAE, Convolutional Neuron Network (CNN) and Deep Belief Network (DBN) [77]. We carried out four experiments to investigate the properties of the latent representations obtained by MVAE, MAE, and MDAE.

- *Ability to detect unknown attacks:* Evaluate the accuracy of the four classifiers that are trained on the latent representation of the proposed models compared to those working with AE, DAE, VAE, CNN and DBN, and on the original input data with RF.
- *Cross-datasets evaluation:* Investigate the influence of the various attack types used for training models on the accuracy classifiers in detecting unknown attacks.
- *Influence of parameters*
  - *Influence of the noise factor*: Measure the influence of the noise level on the latent representation of MDAE.
  - *Influence of the hyper-parameters of classifiers:* Investigate the effects of hyper-parameters on the accuracy of the classifiers working on different latent representations.
- *Complexity of AE-based models*: Assess the complexity of AE-based models based on the training time and the number of parameters.

We split each of IoT datasets into a training set and a testing set based on the scenarios presented in Section 2.5. We randomly select 10% of the training data to create validation sets for model selection [105].

2.4.2. Hyper-parameter Settings

Table 2.1: Hyper-parameters for AE-based models.

Hyper-parameter	**Value**
The number of hidden layers	5
The size of the bottleneck layer	$[1 + \sqrt{n}]$ [21]
Batch size	100
Learning rate	10^{-4}
Initialized method	Glorot [106]
Optimization algorithm	ADAM [107]
Activation function	Relu

The configuration of AE-based models, including AE, MAE, MDAE, and MVAE is as follows. The parameter for balancing between the RE term and the regularized term λ is set at 1 for MAE and MDAE, and

at 1000 for MVAE[4]. The common hyper-parameters of AE-based models are presented in Table 2.1 [21]. The number of hidden layers is 5, and the size of the bottleneck layer m is calculated using the rule $m = [1 + \sqrt{n}]$ in [21], where n is the number of the input features. The number of layers is usually smaller than other problems based on deep learning. The reason is that the input data of network traffic is relatively low dimension. The batch size is 100, and the learning rate is set at 10^{-4}. The weights of these AEs are initialized using the methods proposed by Glorot et al. [106] to facilitate the convergence. We employ the ADAM optimization algorithm [107] for training these networks. In these AEs, the Identity and Sigmoid activation functions are used in the bottleneck layers and the output layers, respectively. The rest of the layers use the ReLu activation function.

We use the validation sets to evaluate our proposed models at every 20 epoch for early-stopping. If the average of the AUC scores[5] produced from the four classifiers, SVM, PCT, NCT, and LR decreases for a certain amount consecutively over a number of epochs, the training process will be stopped. The hyper-parameters of these classifiers are set by default values as in [108]. The DBN-based model has three layers as in [77] and is implemented by [109], where the number of neurons in each layer is similar to the AEs based models in our experiments.

2.5. Results and Analysis

This section describes in detail the main experiments and the investigation of the proposed latent representation models. More importantly, we try to explain the experimental results.

Table 2.2: AUC scores produced from the four classifiers SVM, PCT, NCT and LR when working with standalone (STA), our models, DBN, CNN, AE, VAE, and DAE on the nine IoT datasets. In each classifier, we highlight top three highest AUC scores where the higher AUC is highlighted by the darker gray. Particularly, RF is chosen to compare STA with a non-linear classifier and deep learning representation with linear classifiers.

Class-ifiers	Models	**Datasets**								
		IoT-1	IoT-2	IoT-3	IoT-4	IoT-5	IoT-6	IoT-7	IoT-8	IoT-9
RF	STA	0.979	0.963	0.962	0.670	0.978	0.916	0.999	0.968	0.838
SVM	STA	0.839	0.793	0.842	0.831	0.809	0.934	0.999	0.787	0.799
	DBN	0.775	0.798	0.950	0.941	0.977	0.822	0.960	0.772	0.757
	CNN	0.500	0.500	0.702	0.878	0.815	0.640	0.996	0.809	0.845
	AE	0.845	0.899	0.548	0.959	0.977	0.766	0.976	0.820	**0.997**
	VAE	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	DAE	0.849	0.990	0.569	0.968	0.980	0.803	0.982	0.818	0.996
	MVAE	0.914	0.948	0.978	0.985	0.932	0.950	**0.998**	0.826	0.858
	MAE	**0.999**	0.997	**0.999**	0.987	**0.982**	**0.999**	**0.999**	0.846	0.842
	MDAE	**0.999**	**0.998**	0.999	**0.992**	**0.982**	**0.999**	**0.999**	**0.892**	0.902
PCT	STA	0.768	0.834	0.568	0.835	0.809	0.933	0.998	0.753	0.802
	DBN	0.995	0.786	0.973	0.954	0.697	0.847	0.957	0.783	0.755
	CNN	0.500	0.500	0.674	0.877	0.812	0.635	0.996	0.797	0.844
	AE	0.849	0.892	0.498	0.965	0.977	0.813	0.977	0.814	0.815
	VAE	0.503	0.501	0.499	0.501	0.507	0.497	0.500	0.500	0.499
	DAE	0.882	0.903	0.534	0.969	0.982	0.862	0.984	0.857	0.849
	MVAE	0.954	0.947	0.972	0.986	0.923	0.923	0.997	0.823	0.849
	MAE	**0.996**	0.996	**0.999**	**0.998**	**0.989**	**0.999**	**0.999**	0.833	**0.991**
	MDAE	**0.996**	**0.997**	**0.999**	**0.998**	**0.989**	**0.999**	**0.999**	**0.889**	**0.991**
NCT	STA	0.743	0.747	0.498	0.785	0.692	0.570	0.993	0.770	0.748
	DBN	0.994	0.786	0.954	0.938	0.961	0.927	0.859	0.781	**0.964**
	CNN	0.500	0.500	0.680	0.877	0.767	0.632	0.977	0.777	0.799
	AE	0.985	0.767	0.498	0.834	0.835	0.997	0.945	0.746	0.767
	VAE	0.501	0.506	0.511	0.487	0.499	0.505	0.500	0.488	0.479
	DAE	0.989	0.770	0.580	0.882	0.863	0.997	0.966	0.806	0.788
	MVAE	0.846	0.939	0.973	0.984	0.927	0.937	0.998	0.822	0.796
	MAE	**0.998**	0.996	**0.999**	0.987	0.982	**0.999**	**0.999**	0.828	0.799
	MDAE	0.996	**0.998**	0.998	**0.992**	**0.985**	**0.999**	**0.999**	**0.887**	0.889
LR	STA	0.862	0.837	0.565	0.829	0.802	0.932	0.998	0.791	0.800
	DBN	0.776	0.939	0.960	0.955	0.961	0.837	0.962	0.779	0.755
	CNN	0.500	0.500	0.710	0.878	0.811	0.636	0.997	0.801	0.843
	AE	0.850	0.894	0.498	0.958	**0.987**	0.743	0.996	0.795	**0.998**
	VAE	0.500	0.499	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	DAE	0.871	0.902	0.587	0.966	0.982	0.801	0.996	0.810	0.988
	MVAE	0.921	0.989	0.981	0.985	0.933	0.955	**0.999**	0.828	0.858
	MAE	**0.999**	0.997	**0.999**	0.988	0.984	**0.999**	**0.999**	0.835	0.840
	MDAE	0.996	**0.998**	0.998	**0.992**	0.985	**0.999**	**0.999**	**0.887**	0.889

2.5.1. Ability to Detect Unknown Attacks

This section presents the main experimental results of our chapter. We evaluate the proposed models based on the ability to detect unknown

[4]The reason for setting the much higher value of λ for MVAE than MAE and MDAE is that the RE value of MVAE is often much higher than the regulazier value of MVAE while the RE value of MAE and MDAE is mostly equal to

attacks of the four classifiers trained on the latent representation. As mentioned above, each of the nine IoT datasets has five or ten specific types of botnet attacks. For each IoT dataset, we randomly select two types of IoT attacks, and 70% of normal traffic for training, and the rest of IoT attacks and normal data are used for evaluating our models. The training attacks in this experiment are shown in Table 1.3. As seen in this table, we only use two types of DDoS attacks for training, and the rest is for testing. This guarantees that there are some types of IoT attacks used for evaluating models that have not been seen in the training process. These types of attacks are considered as unknown attacks. The results produced from the four classifiers working with our proposed models are also compared with those working with the original input space and the latent feature space of AE, DAE, VAE, CNN, and DBN. We also compare the results from all linear classifiers with one non-linear classifier (i.e., RF) that is trained on the original feature. The main experimental results (AUC scores) are shown in Table 2.2.

In Table 2.2, we can observe that the classifiers are unable to detect unseen IoT attacks (the AUC scores approximates 0.5) on the representation resulting from the VAE model. The reason is that the VAE model aims to generate data samples from the normal distribution instead of building a robust representation for a classification task. It can also be seen from Table 2.2 that the performances of the four classifiers working with all latent representation models on the IoT-9 dataset are not consistent as those on other datasets. When observing the latent representation of AE and DAE, LR and SVM can perform very well on the IoT-9 dataset while PCT and NCT can not. On the contrary, LR and SVM perform less efficiently than PCT and NCT when working on the latent representation of our proposed models.

It can be seen from Table 2.2 that the latent representations resulting from MVAE, MAE, and MDAE help four classifiers achieve higher clas-

their regularizer.

[5]AUC metric was described in Chapter 1

sification accuracy in comparison to those using the original data. For example, the AUC scores of SVM, PCT, NCT, and LR working on the latent representation of MAE are increased from 0.839, 0.768, 0.743, and 0.862 to 0.999, 0.996, 0.998, and 0.999 with those working on the original data on the IoT-1 dataset, respectively. The increase in the classification accuracy can also be observed from MDAE and MVAE. Moreover, our proposed models also help the linear classifiers achieve higher AUC scores (the fifth and sixth rows) than those using the latent representations of the AE and DBN (the third and fourth rows). Among the linear classifiers, PCT working with the latent representation of MVAE, MAE, and MDAE enhances the accuracy of all the IoT datasets, including IoT-9. Finally, four classifiers trained on the latent representations of MAE and MDAE tend to produce more consistent results than the previous one (MVAE).

Comparing the accuracy of linear classifiers with a non-linear classifier (e.g., RF), the table shows that RF is often much better than all linear classifiers when they are trained on the original features. This evidences that these datasets are not linearly separable in the original space. However, by training on the latent representation of MVAE, MAE, and MDAE, the accuracy of all linear classifiers is considerably improved and often much greater than that of RF. The exception only occurs in IoT-8, where none of the linear classifiers can outperform RF. This result verifies that the proposed models help to project the non-linear datasets in the original space into a linearly separable data in the latent space.

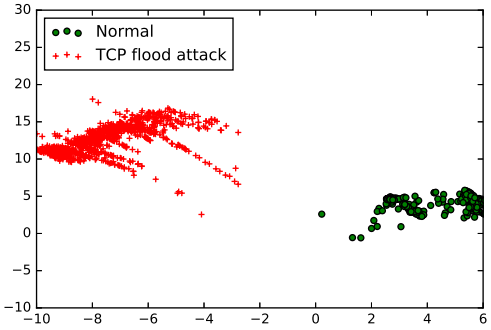
We also carried out an experiment to explain why our models can support conventional classifiers to detect unknown attacks efficiently. In this experiment, we train the AE and MAE on normal data and TCP attacks. The size of the hidden layer of AE and MAE is set at 2 to facilitate the visualization. After training, we test these models on the testing data containing normal samples, the TCP attacks (known attacks), and the UDP attacks (unknown attacks). In Fig. 2.5, we plot 1000 random

samples of the training and the testing data in the hidden space of AE and MAE. Fig. 2.5 (a) and Fig. 2.5 (b) show that the representation of AE still can distinguish the normal samples, known attack samples and unknown attack samples. This is the main reason for the high performance of classifiers on the AE's representation presented in Table 2.2. However, while MAE can compress normal and known attack samples into two very compact areas on both the training and testing data, AE does not obtain this result. The normal and known attacks in the training data of AE spread significantly wider than the samples of MAE. More interestingly, the samples of unknown attacks in the testing data of MAE are mapped closely to the region of known attacks. Hence, they can be distinguished from normal samples easier. By contrast, the samples of unknown attacks in AE are very close to the normal data, and hence they are difficult to separate from the normal samples (benign samples). This result evidences that our proposed model, i.e., MAE, achieves its objective in constraining the normal data and known attack data into two compact areas at the hidden space. Moreover, the unknown attacks are also projected close to the region of known attacks. Subsequently, both attacks (known and unknown) can be effectively identified using some simple classifiers applying on the latent features of MAE.

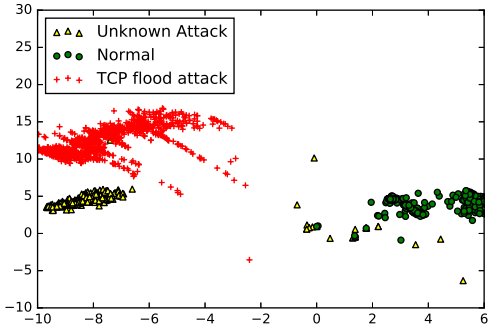
2.5.2. Cross-datasets Evaluation

Among the two tested botnet families, the Gafgyt botnet family is a lightweight version of the Internet Relay Chat model. Thus, the DDoS attacks in Gafgyt are often the traditional SYN, UDP, and ACK Flooding attacks [110]. However, the Mirai botnet is usually a more dangerous IoT malware. It can exploit devices based on several architectures and can perpetuate a wide range of DDoS attacks based on different protocols (e.g., TCP, UDP, and HTTP) [110, 111].

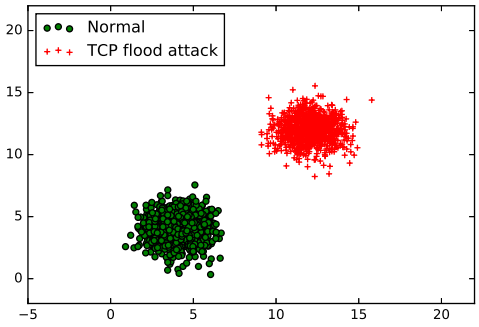
Each botnet family, Gafgyt or Mirai, can generate several DDoS attacks. Different botnets can create different network traffic transmitted from *bots* to infected devices, which results in different feature values of



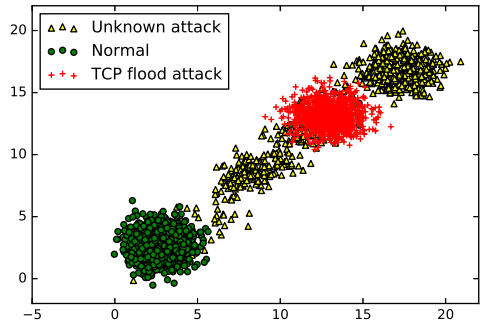
(a) AE representation of training samples.



(b) AE representation of testing samples.



(c) MAE representation of training samples.



(d) MAE representation of testing samples.

Figure 2.5: Latent representation resulting from AE model (a,b) and MAE model (c,d).

attack data.

This experiment aims to exam the stability of the latent representation produced by MVAE, MAE, and MDAE when training on one botnet family and evaluating the other. We consider two scenarios: (1) training data is Mirai, and testing data is Gafgyt, and (2) Gafgyt is chosen for training, and Mirai is used for testing. These scenarios guarantee that the testing attack family has not been seen in the training phase. We use the NCT classifier for investigating our models, and the experimental results of NCT trained on IoT-2[6] are shown in Table 2.3. In this table, the second row represents the models trained on Gafgyt botnets and evaluated on Mirai botnets. In the third row, Mirai is used for training and Gafgyt is used for testing. We do not do this experiment for CNN and VAE due to their ineffectiveness for detecting attacks presented in Table 2.2.

Table 2.3: AUC score of the NCT classifier on the IoT-2 dataset in the cross-datasets experiment.

Train/Test botnets	**STA**	**DBN**	**AE**	**MVAE**	**MAE**	**MDAE**
Gafgyt/Mirai	0.747	0.732	0.717	0.943	0.974	0.988
Mirai/Gafgyt	0.747	0.720	0.628	0.999	0.999	0.999

This table shows that when the training data and testing data come from different botnet families, it is difficult for the NCT classifier to detect unknown botnet attacks. Both the standalone NCT and NCT, with the representation of AE and DBN, tend to produce a poor performance in both scenarios. The reason is that the AE and DBN can only capture the input data's useful information once they gather sufficient data. In this case, the training attacks and testing attacks come from totally different botnet families. The trained AE and DBN may be unable to represent the attacks that have not been seen in the training phase, which results in a poor performance for the NCT classifier (as shown in the first three rows of Table 2.3). On the other hand, the latent representations of MVAE, MAE, and MDAE are designed to reserve some regions being

[6]Due to the space limitation, we only present the results of the NCT classifier on one dataset, i.e., IoT-2. The results of the other classifiers on the rest of datasets are similar to the results in this subsection.

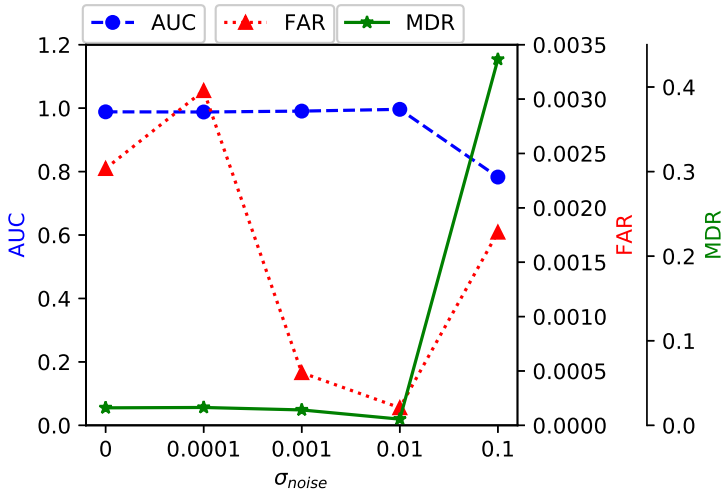


Figure 2.6: Influence of noise factor on the performance of MDAE measuring by the average of AUC scores, FAR, and MDR produced from SVM, PCT, NCT and LR on the IoT-1 dataset. The noise standard deviation value at $\sigma_{noise} = 0.01$ results in the highest AUC, and lowest FAR and MDR.

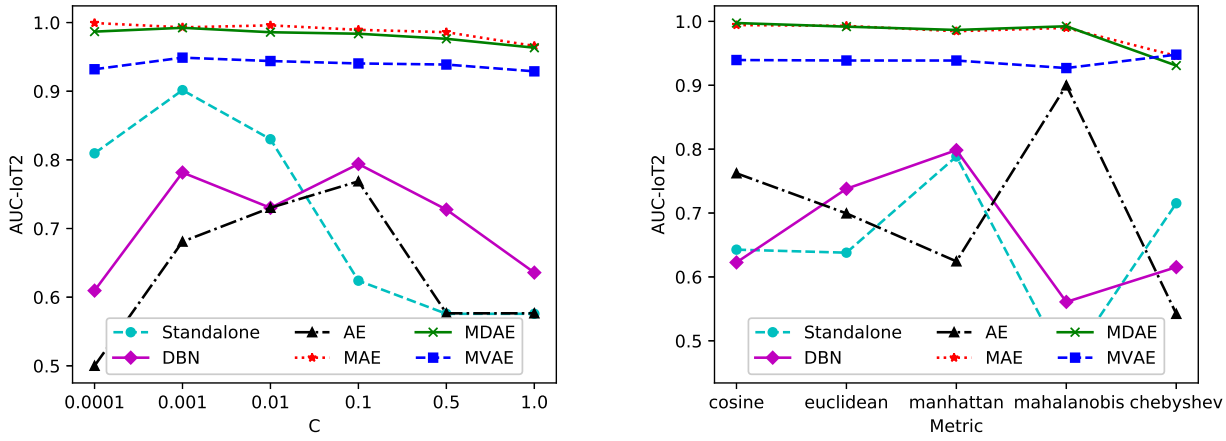
close to the anomaly region for unknown IoT attacks. Thus, these AEs help NCT to identity unknown attacks more effectively and perform well on both scenarios (as observed in the last three rows of Table 2.3). For example, the AUC scores of NCT to predict the Mirai botnet increase from 0.747 with the original data to 0.943, 0.974, and 0.988 with representations of MVAE, MAE and MDAE, respectively. These results confirm that our learning representation models can enhance the ability to detect unknown IoT attacks of simple classifiers.

2.5.3. Influence of Parameters

This subsection analyzes the impact of several important parameters to the performance of the proposed models. The analyzed parameters include the noise factors in MDAE, the hyper-parameters in SVM and NCT classifiers.

2.5.3.1. Influence of the Noise Factor

This experiment examines the impact of the noise factor on the MDAE's performance. In this chapter, the Gaussian noise function in Eq. 1.10



(a) SVM. (b) NCT.

Figure 2.7: AUC scores of (a) the SVM classifier and (b) the NCT classifier with different parameters on the IoT-2 dataset.

is employed to add noise to the input of MDAE. We will analyze the characteristics of MDAE when the standard deviation σ_{noise} of Gaussian function is varied in the range of $[0.0, 0.1]$ on the IoT-1 dataset. The value of the standard deviation σ_{noise} presents the amount of information of the input data which is noised.

Fig. 2.6 presents the influence of the noise factor on MDAE observed by measuring the classification accuracy average of the four classifiers. This figure shows that the mean of AUC tends to be stable with $\sigma_{noise} \leq 0.01$, reaches a peak at $\sigma_{noise} = 0.01$, and then decreases gradually when $\sigma_{noise} > 0.01$. At the same time, a major part of the False Alarm Rate (FAR) scores[7] and Miss Detection Rate (MDR) scores[8] curves go in the opposite direction. These results imply that MDAE achieves the best performance when the value of σ_{noise} is 0.01.

2.5.3.2. Influence of the Hyper-parameters of Classifiers

This experiment investigates the influence of the hyper-parameters on the performance of classifiers when they are trained on the original feature space and the latent representation of five deep learning models

[7]FAR is defined as the number of false alarms of negative samples per the total number of real negative data samples.
[8]MDR is defined as the number of miss detection of positive samples per total of real positive samples.

including AE, DBN, MVAE, MAE and MDAE. We conduct experiments on two well-known classification algorithms, i.e., SVM and NCT[9]. The IoT-2 dataset is chosen for this experiment.

The first parameter is analyzed as the hyper-parameter C of SVM. The hyper-parameter C is a regularizer that controls the trade-off between the complexity of the decision plane and the frequency of error in the SVM algorithm [47]. This hyper-parameter presents the generalization ability to detect unseen attacks of the SVM classifier. Fig. 2.7 (a) presents the influence of C on the performance of SVM (AUC scores). It can be seen that the AUC scores of SVM training on the original feature space and the latent feature spaces of the AE and DBN vary considerably when C is varied in the range from 10^{-4} to 10^{0}. By contrast, the MVAE, MAE, and MDAE models support the SVM to produce very high and consistent AUC scores over a wide range of C values. It suggests that the proposed models generate more robust latent representations than the previous ones. It makes the SVM training process consistent/insensitive on a wide range of hyper-parameter settings.

The second parameter is the distance *metric* used in the NCT classifier. The five common distance metrics including *cosine*, *euclidean*, *manhattan*, *mahalanobis* and *chebyshev* are used to measure distances in NCT. The *metric* hyper-parameter is used to calculate distances between data samples in a feature array [102].

In machine learning, we might possibility get better results when using different distance metrics [112]. Usually, the *cosine* metric is usually used when we have high-dimensional data and when the magnitude of the vectors is not of importance. Oppositely, the *euclidean* distance works well when data has low-dimension and the magnitude of the vectors is important. When dataset has discrete and/or binary attributes, the *manhattan* distance can work quite well. Besides, the *mahalanobis* distance takes co-variance in account which helps in measuring the strength/similarity between two different data objects. Fi-

[9]The performance of SVM and NCT is often strongly influenced by some important hyper-parameters.

nally, the *chebyshev* distance is often used in warehouse logistics as it closely resembles the time an overhead crane takes to move an object.

Fig. 2.7 (b) shows that the NCT classifier working with MVAE, MAE, and MDAE tends to yield high and stable AUC scores over the five different values of the *metric* parameter. On the other hand, the AUC scores of NCT training on the original feature space and the AE and DBN feature spaces are much lower and unpredictably changed with different values of *metric*.

The experiments in this subsection clearly show that our proposed models (i.e., MVAE, MAE, and MDAE) can support classifiers to perform consistently on a wide range of hyper-parameter settings. Perhaps, the reason for these results is that the latent representations of our models can map normal data and attacks into two separate regions. Thus, linear classifiers can easily distinguish attacks from normal data, and its performance is less sensitive to hyper-parameters.

2.5.4. Complexity of Proposed Models

Table 2.4: Complexity of AE-based models trained on the IoT-1 dataset.

Models	**Training Time**	**No. Trainable Parameters**
AE	370.868	25117
VAE	420.969	25179
DAE	405.188	25117
MVAE	408.047	25179
MAE	354.990	25117
MDAE	424.166	25117

Table 2.4 presents the training time and the number of trainable parameters in each AE-based model. This information is exported from the training process of these models on the IoT-1 dataset. As presented in the training time and the trainable parameters of AE-based models with the same architecture are similar. Therefore, adjusting loss functions of the proposed models does not affect much to training time and model size of AE-based models.

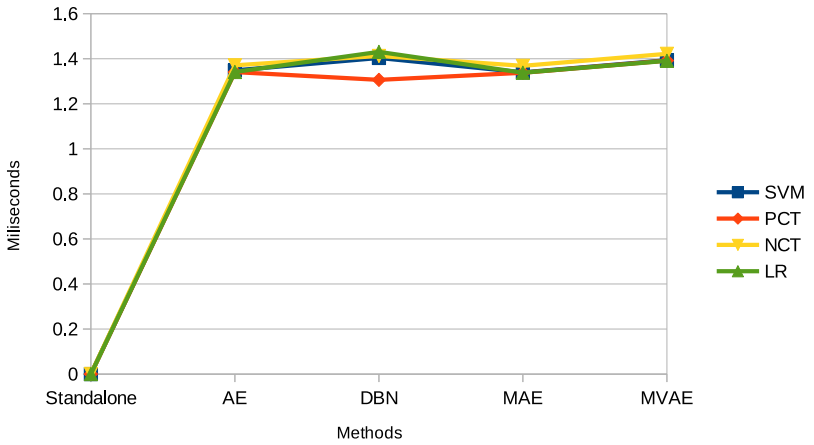


Figure 2.8: Average testing time for one data sample of four classifiers with different representations on IoT-9.

2.5.5. Assumptions and Limitations

Although our proposed models possess many advantages to learn a new representation of the IoTs datasets, they are subject to few limitations. First, we assume that there is sufficient data to train good models for learning representation. In this chapter, the number of training samples for both normal and attack data is more than ten thousand samples. If we do not have enough data for training (only a few hundred), it will be difficult to train a good model for mapping input features to a new feature space. In the future, we will study techniques to generate synthesized data [61] to train the models proposed in this chapter. Second, the advantages of representation learning models come with the cost of running time. Since, a neural network is used to project the input data into a new presentation, the executing time of these models is often much longer than using classifiers on the original feature spaces. Fig. 2.8 presents the average time for processing one data sample of all tested methods. It can be seen that the processing time of all deep learning models is roughly equal and much longer than that of the standalone method.

IoT applications appear in many aspects of our life, such as smart city, home automation, data security [113, 114]. Specifically, alert systems in

the data security applications require usage and pattern analysis for all data across all systems, in real-time. Moreover, only stream processing is able to filter, aggregate, and analyze continuous data collection in milliseconds [114]. As a result, no behaviors of IoT traffic data get overseen or outdated. Thus, the running time of these models is still acceptable (about 1.3 milliseconds) for most applications in the real world.

2.6. Conclusion

In this chapter, we have designed three novel AE based models for learning a new latent representation to enhance the accuracy in NAD. As far as we have known, our work is the first research that attempts to design regularized versions of AE for learning a latent representation in a supervised learning manner. In our models, normal data and known attacks are projected into two narrow separated regions in the latent feature space. To obtain such a latent representation, we have added new regularized terms to three AE versions, resulting in three regularized models, namely the MVAE, MAE and MDAE. These regularized AEs are trained on the normal data and known IoT attacks, and the bottleneck layer of the trained AEs was then used as the new feature space for linear classifiers.

We have carried out extensive experiments to evaluate the strength and examine different aspects of our proposed models. The experimental results demonstrate that our proposed models can map the non-linear separable normal and attacks data in the original space into linear and isolated data in their latent feature space. Moreover, the results also showed that unknown attacks tend to appear close to known attacks in the latent space. The distribution of the data in the latent space makes the classification tasks much easier than executing them on the original feature space. Specifically, linear classifiers working with the latent feature produced from our models often significantly outperform those working with the original features and the features created by AE

and DBN on the nine IoT attack datasets. The new data representation also helps the classifiers perform consistently when training on different datasets and varying a wide range of hyper-parameter settings.

In the future, one can extend our current work in several directions. First, the proposed models in this chapter are only examined on two-class classification problems. In the future, it is interesting to extend these models and test them on multi-class classification problems. Second, the distribution centroid (μ_{y^i} in Eq. 2.1) are currently determined through trial and error. It is desirable to find an approach to select good values for each dataset automatically. Last but not least, the regularized AE models are only tested on a number of IoT datasets. It is also more comprehensive to experiment with them on a wider range of problems.

Chapter 3
DEEP GENERATIVE LEARNING MODELS FOR NETWORK ATTACK DETECTION

In Chapter 2, we have proposed a representation learning method that projects the normal traffic and abnormal traffic into distinguishable spaces. The proposed representation learning method helps to improve the accuracy of machine learning in detecting network attacks, especially new/unknown attacks. However, this approach only performs well with the assumption that we can collect enough labeled data from both normal traffic and anomalous traffic. Nevertheless, in many practical applications, this assumption may not always be the case. In many network environments, collecting attack traffic is much more complicated than those of normal traffic. Subsequently, most of the network attack datasets are imbalanced. On such skewed datasets, the predictive model developed using conventional machine learning algorithms could be biased and inaccurate.

In this chapter, we propose a novel solution to enable robust NAD systems using deep neural networks. Specifically, we develop deep generative models to synthesize malicious samples on the network systems. First, a hybrid of Auxiliary Conditional Generative Adversarial Network (ACGAN) and Support Vector Machine (ACGAN-SVM) is used to generate specific types of malicious samples where SVM is used for selecting borderline samples. Second, Conditional Denoising Adversarial AutoEncoder (CDAAE) is proposed instead of the ACGAN-based model to create particular kinds of malicious samples due to its effectiveness in the training process. The third model is a hybrid of CDAAE with the K-nearest Neighbor algorithm (CDAEE-KNN) to generate samples that further improve NAD's accuracy. The synthesized samples are merged

with the original samples to form the augmented datasets. The contributions of this chapter are published and submitted to the work of [i] and [v] of our publications and one paper in under review.

3.1. Introduction

Among many NAD methods, machine learning-based methods are proven to be the most effective methods to detect network attacks (presented in Chapter 1). However, using machine learning to build a trustworthy and robust NAD in the network environment remains practically challenging. One of the main reasons is the lack of labeled malicious samples. In the network environments, the majority of collected traffic samples are normal, and only a few samples are attacks. Subsequently, most of the network attack datasets are imbalanced. The predictive model developed using conventional machine learning algorithms could be biased and inaccurate when trained on the skewed datasets. This is because machine learning algorithms are usually designed to improve accuracy by reducing the error. Thus, they do not take into account the class distribution/proportion or the balance of classes.

A possible solution to address the above imbalance problem of NAD datasets is collecting more malicious samples. However, in the network environments, collecting attack samples is extremely difficult due to the privacy and security concerns of Internet users [21]. Internet providers tend to avoid divulging data that could compromise the privacy of their clients or privileged information of their systems. Therefore, re-sampling (over-sampling and under-sampling) methods are often used to balance the skewed datasets [62]. Nevertheless, these techniques have some intrinsic drawbacks. While under-sampling can potentially lose useful information, over-sampling is prone to cause over-fitting, when exact copies of the minority class are replicated randomly. Moreover, it does not solve the fundamental "lack of data" problem.

In this chapter, we propose a novel solution to address the "lack of

attack data" problem of NAD datasets. We propose three generative network-based models, i.e., Auxiliary Conditional Generative Adversarial Network - Support Vector Machine (ACGAN-SVM), Conditional Denoising Adversarial AutoEncoder (CDAAE), and a hybrid between CDAAE and the K-Nearest Neighbor algorithm (CDAAE-KNN) to create attack samples that further improve the classification performance. The synthesized malicious data is merged with the original data to form the augmented training datasets. Three conventional machine learning algorithms are trained on the augmented datasets including SVM, Decision Tree (DT), and Random Forest (RF). We choose these classfiers due to their effectiveness on the NAD problem presented in Subsection 1.6.3 in Chapter 1.

The rest of the chapter is organized as follows. Our proposed model is discussed in Section 3.2. In Section 3.4, we describe the experimental settings for our proposed models. The performance of ACGAN-SVM, CDAAE, and CDAAE-KNN is analyzed in Section 3.5. Finally, conclusions with future work are drawn in Section 3.6.

3.2. Deep Generative Models for NAD

3.2.1. Generating Synthesized Attacks using ACGAN-SVM

The first method for generating artificial data is ACGAN-SVM. ACGAN-SVM attempts to produce samples that are nearby the borderline area defined by the SVM model. This method is inspired by SMOTE-SVM [64]. However, unlike SMOTE-SVM that produces data samples by extrapolating or interpolating from the original samples [64], ACGAN-SVM uses a generative model to generate new samples. We hypothesize that the samples made by ACGAN-SVM will preserve the original dataset's underlying distribution better than SMOTE-SVM. Subsequently, the augmented dataset of ACGAN-SVM will improve the performance of classification algorithms.

Algorithm 3 presents a detailed description of using ACGAN-SVM

Algorithm 3 ACGAN-SVM algorithm for oversampling.

Inputs: original training set X, generated data samples X', number of nearest neighbors m, Euclid distance between vector x and vector y d(x,y).
Outputs: new sampling set X_{new}.
begin
Training ACGAN on X to have trained Generator G Set X' contains minority samples generated by G network Train SVM model on X to have the set of support vectors SVs
foreach $sv_i \in SVs$ **do**
Compute m nearest neighbors in X Compute d_i that is average of Euclid distance from m nearest neighbors to sv_i
foreach $x_j \in X'$ **do**
if $d(x_j, sv_i) \leq d_i$ **then**
$X_{new} = X \cup \{x_j\}$
return X_{new}.

for generating synthesized data. The technique is divided into two main phases, i.e., generation and selection. In the generation phase, the ACGAN network[1] is trained on the training dataset X. After that, the generator network (G) of ACGAN is used to generate synthesized samples X'. In the selection phase, the SVM model is trained on the training dataset X, and the set of support vectors of this model is called SV_s. For each support vector $sv_i \subset SV_s$, we calculate the average Euclidean distance d_i of m nearest neighbor samples of sv_i in X to sv_i. If a generated sample $x_j \subset X'$ has the distance to sv_i smaller than d_i, this sample is kept. Conversely, if the distance from x_i to sv_i is greater than d_i, the sample is removed. The algorithm will stop when the augmented dataset is balanced for every class. The augmented dataset is then used to train three classification algorithms as in ACGAN.

3.2.2. Conditional Denoising Adversarial AutoEncoder

The disadvantage of GAN-based models (e.g., ACGAN, ACGAN-SVM) is that the training process is difficult to convergence [27]. Specifically, the generator collapses to a parameter setting where it always emits the same point. Thus, the gradient of the discriminator may point the same directions for many similar points. Consequently, the gradient descent technique is unable to separate the identical outputs. As a

[1]ACGAN is described in Sub-section 1.3.4 of Chapter 1.

result, the algorithm cannot converge to a distribution with the correct amount of entropy.

Thus, we improve the generative model based on ACGAN by proposing a new generative model, i.e., CDAAE and CDAAE-KNN. The DAAE model[2] is only trained in the unsupervised learning mode that does not consider the class label of data samples. Therefore, this model cannot be used to generate data samples for a specific class. To address this issue, we propose a Conditional Denoising Adversarial AutoEncoder (CDAAE), which can generate data samples of any specific class. This model will be used to generate malicious samples for each type of network attack. Fig. 3.1 describes CDAAE, which is an ensemble of a Denosing AutoEncoder (DAE)[3] and an Adversarial Network. In CDAAE, $\mathbf{x_{noise}}$ is generated from $\mathbf{x}$ using the Gaussian corruption function in Eq. 1.10.

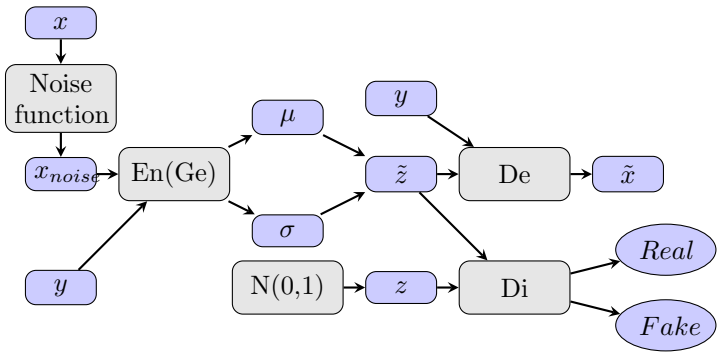


Figure 3.1: Structure of CDAAE.

Two networks in CDAAE are jointly trained in two phases, i.e., the reconstruction phase and the regularization phase. In the reconstruction phase, the encoder of CDAAE (En), receives two inputs, i.e., a noisy data $\mathbf{x_{noise}}$ and a class label $\mathbf{y}$, and generates the latent representation or the bottleneck $\tilde{\mathbf{z}}$. The latent representation $\tilde{\mathbf{z}}$ is used to guide the decoder of CDAAE to reconstruct the input at its output from a desired distribution, i.e., the standard normal distribution. Let $\mathbf{W_{En}}$, $\mathbf{W_{De}}$, $\mathbf{b_{En}}$, and $\mathbf{b_{De}}$ be the weight matrices and the bias vectors of En and

[2]DAAE is presented in Sub-section 1.3.5 Chapter 1.

[3]Using a Denoising AutoEncoder instead of an AutoEncoder allows CDAAE to learn more robust features from the original data [23].

De, respectively, and $\mathbf{x} = \{x^1, x^2, \ldots, x^n\}$ be a training dataset where n is the number of training data samples, the computations of En and De based on each training sample are presented in Eq. 3.1 and Eq. 3.2, respectively.

$$\tilde{z}^i = f_{En}(\mathbf{W_{En}}(x^i_{noise} \mid y^t + \mathbf{b_{En}}), \tag{3.1}$$

$$\tilde{x}^i = f_{De}(\mathbf{W_{De}}(z^i \mid y^t) + \mathbf{b_{De}}), \tag{3.2}$$

where $\mid$ is concatenation operator and f_{En} and f_{De} are the activation functions of the encoder and the decoder, respectively. y^t is the label of the data sample x^i, x^i_{noise} is generated from x^i by Eq. 1.10. The reconstruction phase aims to reconstruct the original data $\mathbf{x}$ from the corrupted version $\mathbf{x_{noise}}$ by minimizing the reconstruction error (R_{loss}) in Eq. 3.3.

$$R_{loss} = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \hat{x}^i\right)^2. \tag{3.3}$$

In the regularization phase, an adversarial network is used to regularize the hidden representation $\tilde{\mathbf{z}}$ of CDAAE. The generator (Ge) of the adversarial network, which is also the encoder of the Denoising AutoEncoder (En), tries to generate the latent variable $\tilde{\mathbf{z}}$ that is similar to sample $\mathbf{z}$ drawn from a standard normal distribution, $p(\mathbf{z}) = \mathbb{N}(\mathbf{z}|0, 1)$. We define d_{real} and d_{fake} as the outputs of Di with inputs $\mathbf{z}$ and $\tilde{\mathbf{z}}$, respectively. Let $\mathbf{W_{Di}}$ and $\mathbf{b_{Di}}$ be the weight matrix and the bias vector of Di, respectively. For each data point x^i, d^i_{real} and d^i_{fake} are calculated as follows:

$$d^i_{real} = f_{Di}(\mathbf{W_{Di}}z^i + \mathbf{b_{Di}}), \tag{3.4}$$

$$d^i_{fake} = f_{Di}(\mathbf{W_{Di}}\tilde{z}^i + \mathbf{b_{Di}}), \tag{3.5}$$

where f_{Di} is the activation function of Di.

The discriminator (Di) attempts to distinguish the true distribution $\mathbf{z} \sim p(\mathbf{z})$ from the latent variable $\tilde{\mathbf{z}}$ by minimizing in Eq. 3.6 whereas the generator Ge (or En) tries to generate $\tilde{\mathbf{z}}$ to fool the discriminator Di by maximizing in Eq. 3.7.

$$D_{loss} = -\frac{1}{n}\sum_{i=0}^{n}\left(\log(d^i_{real}) + \log(1 - d^i_{fake})\right), \tag{3.6}$$

$$G_{loss} = \frac{1}{n}\sum_{i=1}^{n}\log(d^i_{fake}). \tag{3.7}$$

The total loss function of CDAAE is the combination of the three above loss as in (3.8), and CDAAE is trained to minimize this loss function.

$$L_{CDAAE} = R_{loss} + D_{loss} - G_{loss}. \tag{3.8}$$

3.2.3. Borderline Sampling with CDAAE-KNN

In machine learning, data samples around the borderline between classes are often prone to be misclassified. Therefore, these samples are more important for estimating the optimal decision boundary. Generating synthesized samples on this area potentially makes more benefit than performing on the whole minority class [115]. As presented in Section 3.2.1, SVMs are also used to learn the boundary of classes and over-sampling the minor samples around these boundaries. However, when dealing with imbalanced datasets, the class boundary learned by SVMs may be skewed toward the minority class, thereby increasing the misclassified rate of the minority class [116].

In this chapter, we determine the borderline samples using the K nearest neighbor algorithm (KNN). Specifically, we propose a hybrid model between CDAAE and KNN (shorted as CDAAE-KNN) for generating malicious data for the NAD. In this model, KNN is used to select the generated samples by CDAAE that are close to the borderline.

Algorithm 4 describes the details of CDAAE-KNN. The algorithm divides into two phases: generation and selection. In the generation phase (Step 1 and 2), we train a CDAAE model on the original dataset X. We then use the decoder network *De* of CDAAE to generate new samples for the minority classes. The generated dataset is called $\tilde{X}$. In the selection phase, the KNN algorithm is executed in the sample set

Algorithm 4 CDAAE-KNN algorithm.

INPUT:
X: original training set
$\tilde{X}$: generated data samples
m: number of nearest neighbours
d(x,y): Euclide distance between vector x and vector y
OUTPUT:
X_{new}: new sampling set
BEGIN:
1. Training CDAAE on X to have the trained decoder network (De)
2. Set $\tilde{X}$ contains minority samples generated by De
3. Run KNN algorithm on set $X \cup \tilde{X}$
for each $x_i \in \tilde{X}$ **do**
  - Set m = number of majority class samples in k nearest neighbours
  - Set n = number of minority class samples in k nearest neighbours
  if $m \geq \alpha_1$ and $n \geq \alpha_2$ **then**
    $X_{new} = X \cup \{x_i\}$
  end if
end for
return X_{new}
END.

$X \cup \tilde{X}$ to find the k nearest neighbors of the samples $x_i \in \tilde{X}$. For each x_i, we calculate the number of nearest samples which belong to minority classes n and the number of nearest samples that belong to majority classes m. If m and n are larger than thresholds α_1 and α_2, respectively, the sample x_i will be considered as the borderline samples, and it is added to the output set X_{new}. In our experiment, α_1 and α_2 are set at 5 and k is set at 50. These values are calibrated from the experiments for the good classifier performance[4].

A possible problem with CDAAE-KNN is that this approach may generate the misclassified samples, i.e., the generated samples for the minor class may be closer to the major class than the minor class. We have conducted an experiment to examine if this happens by calculating the number of neighbor samples of the borderline samples in CDAAE-KNN and found that most of the neighbor (about 85%) of the borderline samples belong to the same class with the borderline samples. Therefore, the probability of generating misclassified samples of CDAAE-KNN will be legitimated.

[4]We have also conducted more experiments in which α_1 and α_2 are set at 10, 15, and k is set at 40, 60, and found that the performance of CDAAE-KNN declined with these values.

3.3. Using Proposed Generative Models for Network Attack Detection

3.3.1. Training Process

Algorithm 5 Training the generative model (ACGAN-SVM, CDAAE, and CDAAE-KNN) or generative-based models.

INPUT:
x, y: Training data samples and corresponding labels.
The generative model with its hyper-parameters
The classifier with its hyper-parameters.
OUTPUT: Trained classifier.
BEGIN:
1. Training the generative model to form the augmented dataset x_{aug}, y_{aug} (including the original samples and generated samples) as described in Section 2.2 by fitting x, y.
2. Training the classifier with input as x_{aug}, y_{aug}
return Trained classifier.
END.

Algorithm 5 presents the training process by applying a generative model to enrich the data. First, we train a generative model (e.g., ACGAN-SVM, CDAAE, or CDAAE-KNN) to synthesize the data samples of minority classes. The process of generating samples is described in Section 3.2. The new synthesized samples and the original samples are combined to form an augmented dataset. Second, a classifier (e.g., SVM, DT, or RF) is trained on the augmented dataset. The final result is the trained classifier.

3.3.2. Predicting Process

Algorithm 6 Predicting process based on generative model.

INPUT:
x^i: A data sample that need to be classified.
Trained classifier.
OUTPUT: y^i: Label of x^i.
BEGIN:
Putting x^i to the trained classifier to get the output y^i.
return y^i.
END.

Algorithm 6 presents the predicting process for NAD. First, a new network traffic data sample x^i is put to the trained classifier. Then, the classifier predicts the output y^i for the corresponding x^i. This predicting

process is similar with a classification task. Therefore, this is very quick to ensure the speed of networks.

3.4. Experimental Settings

We do the experiments to evaluate the generative models on five datasets, i.e., NSL-KDD, UNSW-NB15, three CTU13s datasets, including CTU13.6-Menti, CTU13.12-NSIS.ay, and CTU13.13-virut. This section presents the hyper-parameter settings and experimental sets that we used to evaluate the proposed models.

3.4.1. Hyper-parameter Setting

The generative models were implemented using a popular deep learning framework, Tensorflow. The same network structure is used for all generative models. In each model, the decoder and the discriminator have two layers, and the encoder has three layers, including the latent layer. The latent layer size is set based on the number of features of the datasets [21]. Let n be the dimension of input data, then the number of neural in the latent layer is $\lceil 1 + \sqrt{n} \rceil$. All networks were trained using Adam optimization [25] with the learning rate of 10^{-3}. After training, the decoders of CVAE, SAAE, CDAAE, and the Generator of ACGAN are used to generate synthesized malicious samples for the tested datasets.

The synthesized samples are then merged with the original data to form the augmented datasets. Three popular classification algorithms, i.e., SVM, decision tree (DT), and random forest (RF), are trained on the augmented datasets. The implementation of these classification algorithms in a popular machine learning packet in Python Scikit learn [102] is used. In order to minimize the impact of experimental parameters on the performance of the classification algorithms, we implement the grid search technique for each classifier algorithm. The range of values for each parameter that is tuned by the grid search technique presented in Table 3.1. In our experiments, we used the default settings in the Scikit-learn library in which 5-fold crossover validation is used to perform the

grid search.

Table 3.1: Values of grid search for classifiers.

Classifiers	Parameters
SVM	$kernel = rbf; gamma = 0.001, 0.01, 0.1, 1.0$
DT	$max - depth = 5, 6, 7, 8, 9, 10, 50, 100$
RF	$n - estimators = 5, 10, 20, 40, 80, 150$

Table 3.2 presents our network setting for CDAAE. For every dataset, the AE part including En and De has 5 neural network layers and the Di has 3 neural network layers. The number of neuron in each layer of CDAAE is presented in Table 3.2.

Table 3.2: Hyper-parameters for CDAAE.

Datasets	Dimension	En	De	Di
NSL-KDD	41	41-15-8	15-41	8-20-40
UNSW-NB15	49	49-25-8	25-49	8-20-40
CTU13s	14	14-10-5	10-14	5-20-40

Except for NSD-KDD and UNSW-NB15, which have been divided into training and testing sets, the other tested datasets are divided as follows: training set (70%) and testing set (30%). The descriptions of datasets are presented in chapter 2. These tables show that the datasets are highly imbalanced: there are very few attack samples compared to normal samples. Two network attack datasets (NSL-KDD and UNSW-NB15) are processed to form multi-class classification problems where each attack type is one class. Other datasets are labeled as binary classification problems.

3.4.2. Experimental sets

We divided our experiments into three sets. The first set is to compare the effectiveness of our proposed models with the previous generative models. The performance of the classification algorithms trained on the augmented datasets generated by ACGAN-SVM, CDAAE, and CDAAE-KNN is compared with the version trained on the original data (namely ORIGINAL) and four other approaches for addressing imbalanced data, i.e., SMOTE-SVM [64], BalanceCascada [66], ACGAN [28], CVAE [71],

SAAE [29].

The second set is to assess the quality of the generated samples of CDAAE and compare it with other generative models. For each generative model, we calculate the Gaussian Parzen window distribution [26]. This metric presents the degree to which the generated samples of the model follow the original data distribution. The higher score implies that the generated samples have high probabilities to follow the original data distribution. The results in each experimental set are presented in Section 3.5.

The third set is to assess the processing time of training and generating processes of the models for handling imbalanced datasets. Moreover, we report the trainable parameters of the deep neural network-based models to assess the model size or model complexity of the deep neural-based networks. Here, we do the experiments on the NSL-KDD and UNSW-NB15 datasets. The results are shown in Table 3.5.

3.5. Results and Discussions

This section presents the results of three experiments and the discussions on these results.

3.5.1. Performance Comparison

This subsection compares the effectiveness of our proposed models for generating synthesized attacks with the previous generative models. The tested generative models include SMOTE-SVM, BalanceCascade, ACGAN, CVAE, SAAE.

Table 3.3 presents the results of SVM, RF, and DT, on five network attack datasets (NSL-KDD, UNSW-NB15, and three CTU13 datasets). This table evidences for the impressive performance of the classifiers when they are trained on the augmented datasets of CDAAE and CDAAE-KNN. First, we can observe that the classifiers perform well on NSL-KDD, but they seem ineffective on UNSW-NB15. The reason could be that the UNSW-NB15 dataset is more complicated than NSL-KDD.

UNSW-NB15 has more categories of attacks (ten), and each attack has fewer samples compared to NSL-KDD. Thus, when trained on the original dataset of UNSW-NB15, all three classifiers achieve the AUC scores at very low values. The machine learning algorithms completely misclassify some attacks. For example, on the NSL-KDD dataset, the classifiers usually cannot predict R2L and U2L attacks. On UNSW-NB15, the classifiers trained on the original version often can not predict Analysis, Backdoor, Shellcode, and Worms attacks.

Second, by using generative models to generate synthesized samples for the minor classes, the accuracy of the classifiers is improved considerably. On the NSL-KDD dataset, the AUC scores of SVM, DT, and RF are improved from 0.570 to 0.753, 0.660, and 0.842 when trained on the augmented datasets by CDAAE-KNN. Those values are increased from 0.129 to approximately 0.441, 0.598, and 0.623 on the UNSW-NB15 dataset.

Third, the table also shows that the AUC score classifiers based on the generative models are usually higher than those of the traditional techniques (SMOTE-SVM and BalanceCascade). For example, comparing between CDAAE and SMOTE-SVM, the AUC score is increased from 0.688, 0.446, 0.780 to 0.741, 0.650, 0.835 on NSL-KDD dataset corresponding to SVM, DT, and RF. These values are from 0.218, 0.348, 0.436 to 0.441, 0.592, 0.602, respectively on the UNSW-NB15 dataset. Among all techniques for synthesizing data, we can see that CDAAE-KNN often achieves the best results.

Overall, the results in this subsection show that the generative models can be used to generate meaningful samples for the minor classes on NAD. Moreover, our proposed models often achieve better results compared to the previous models. The reason for the better performance of the generative models compared to the traditional approaches (SMOTE-SVM and BanalceCascade) is that the traditional approaches create samples that do not follow the original data distribution. At the

Table 3.3: Result of SVM, DT, and RF of on the network attack datasets.

Algorithm	**Augmented datasets**	NSL-KDD	UNSW-NB15	CTU13.6-Menti	CTU13.12-NSIS.ay	CTU13.13-Virut
SVM	ORIGINAL	0.570	0.129	0.500	0.500	0.831
	SMOTE-SVM [64]	0.688	0.218	0.820	**0.824**	0.887
	BalanceCascade [66]	0.620	0.304	0.876	0.820	0.895
	CVAE [71]	0.736	0.325	0.927	0.630	0.950
	SAAE [29]	0.738	0.345	0.928	0.653	0.951
	ACGAN [28]	0.712	0.200	0.905	0.601	0.947
	ACGAN-SVM	0.752	0.205	0.932	0.658	0.953
	CDAAE	0.741	0.416	0.963	0.693	0.962
	CDAAE-KNN	**0.753**	**0.441**	**0.972**	0.702	**0.971**
DT	ORIGINAL	0.430	0.221	0.500	0.777	0.962
	SMOTE-SVM [64]	0.446	0.348	0.892	0.782	0.964
	BalanceCascade [66]	0.522	0.486	0.897	0.786	0.965
	CVAE [71]	0.612	0.538	0.992	0.796	0.968
	SAAE [29]	0.629	0.542	0.920	0.800	0.968
	ACGAN [28]	0.523	0.506	0.905	0.799	0.967
	ACGAN-SVM	0.601	0.584	0.909	**0.834**	0.958
	CDAAE	0.650	0.592	0.934	0.816	0.971
	CDAAE-KNN	**0.660**	**0.598**	**0.941**	0.823	**0.976**
RF	ORIGINAL	0.760	0.357	0.500	0.846	0.951
	SMOTE-SVM [64]	0.780	0.436	0.945	0.882	0.954
	BalanceCascade [66]	0.793	0.439	0.949	0.888	0.956
	CVAE [71]	0.824	0.572	0.958	0.921	0.958
	SAAE [29]	0.823	0.571	0.956	0.922	0.956
	ACGAN [28]	0.804	0.448	0.954	0.892	0.958
	ACGAN-SVM	0.834	0.589	0.962	0.901	0.965
	CDVAE	0.835	0.602	0.962	0.976	0.962
	CDVAE-KNN	**0.842**	**0.623**	**0.966**	**0.984**	**0.970**

same time, this problem is mitigated by using generative models. Additionally, the methods based on CDAAE (CDAAE, CDAAE-KNN) can lessen some limitations of the methods based on GAN (e.g., ACGAN) and the methods based on VAE (e.g., CVAE). Moreover, CDAAE-KNN also yields more successful results than CDAAE thanks to KNN to select necessary samples that contribute more to the classifiers.

3.5.2. Generative Models Analysis.

This subsection qualitatively evaluates the generative models by measuring whether their generated data converges to the true data distribution. We used the Parzen window method [26] to estimate the likelihood of the synthesized samples following the distribution of the original data. For each generative model, we fit the generated samples by a Gaussian

Table 3.4: Parzen window-based log-likelihood estimates of generative models.

Model	NSL-KDD	UNSW-NB15	CTU13.6-Menti	CTU13.12-NSIS.ay	CTU13.13-Virut
SMOTE-SVM [64]	15.87±0.69	22.84±0.67	64.07±0.96	52.80±1.36	52.15 ± 0.89
CVAE [71]	65.65±0.33	40.59±1.03	95.68±1.47	92.78±0.29	104.83± 0.89
SAAE [29]	68.20± 3.16	44.26±2.58	112.63±2.14	95.20±4.48	112.19±3.22
ACGAN [28]	60.86±4.82	33.76±2.59	89.55±5.39	82.76±7.98	90.96±2.98
CDAAE	80.34±2.56	61.48±3.95	118.32±2.03	96.67±1.56	131.69 ± 1.84

Parzen window and the log-likelihood of each generative model under the true distribution is reported [26]. The window size parameter of the Gaussian is obtained by cross-validation on the validation set. Experimental results are reported in Table 3.4[5] where a higher value presents a better generative model.

It can be seen that the log-likelihood of all generative models is always more significant than the value of SMOTE-SVM on all tested datasets. It is not surprising since the generative models are trained to learn the true distribution of the original data, while SMOTE-SVM does not do so. Moreover, the log-likelihood of CDAAE is always the highest value among all generative models. This evidences that the quality of the generated data of CDAAE is always better than the other models. This result explains why machine learning algorithms trained on the augmented datasets of CDAAE often achieve better performance than those trained on the other generative models.

3.5.3. Complexity of Proposed Models

This subsection compares the processing time of all tested approaches. We measured the computational time for training and generating synthesized samples of the methods on one the NSL-KDD and UNSW-NB15 dataset. The results are presented in Table 3.5. It can be seen that all

[5]For each model, this table presents mean and standard deviation of the log-likelihood of the generated samples.

Table 3.5: Processing time of training and generating samples processes in seconds.

Methods	NSL-KDD			UNSW-NB15		
	Train Time	Generate Time	No. Trainable Parameters	Train Time	Generate Time	No. Trainable Parameters
SMOTE-SVM [64]	109.3	0.2		98.9	0.2	
BalanceCascade [66]	134.9	1.2		112.5	1.2	
CVAE [71]	6345.6	0.3	174050	5154.4	1.4	837535
ACGAN [28]	6402.5	0.4	115320	5024.8	2.2	823868
SAAE [29]	5906.1	0.3	353967	5130.2	0.7	976058
ACGAN-SVM	9342.5	0.4	115320	8234.9	1.8	823868
CDAAE	6043.9	0.4	532946	4975.1	2.6	1132307
CDAAE-KNN	8576.7	0.9	532946	7456.8	3.7	1132307

generative approaches based on deep neural networks require a significant time to train the models. In contrast, the traditional approaches, i.e., SMOTE-SVM and BalanceCascade, do not require to train the generative models. For the time to generate synthesized samples, the table shows that the difference between the traditional approaches and the deep networks is not much. Among the tested deep network models, we can see that CDAAE-KNN often requires a longer time to generate data than the others. However, the overhead is not significant. Moreover, both training and generating processes are executed offline. Therefore, using our proposed models to enhance the accuracy of the classifiers will not affect the detection time (time to detect attacks) of the classification algorithms.

Table 3.5 also presents the number of trainable parameters (No. Parameters) in the generative models to generating samples. We can observe that the proposed models, i.e., ACGAN-SVM, CDAAE, and CDAAE-KNN, have the higher number of trainable parameters compared with the previous generative models. This proves that the proposed models increase the model size of the original ones.

3.5.4. Assumptions and Limitations

Although the proposed models have many advantages to generate network traffic samples, they are subject to several limitations. First, the original data need to be assumed to follow a Gaussian distribution, but that is not usually the case. In the future, we will test other distributions that may better represent the original data distribution. Second, the CDAAE model can learn from the external information instead of the label of data only. We expect that by adding some attributes of malicious behaviors to CDAAE, the synthesized data will be more similar to the original data.

3.6. Conclusion

This chapter proposed three novel models, i.e., ACGAN-SVM, CDAAE, and CDAAE-KNN, to enrich the data and address the imbalance problem of network attack datasets. The CDAAE model is used to generate samples for a specific class label where ACGAN-SVM and CDAAE-KNN are used to synthesize samples that are close to the borderline between classifiers. The augmented datasets are used to train three popular classification algorithms, e.g., SVM, DT, and RF. The experiments were conducted on five attack datasets, and the results have demonstrated that using our proposed models to generate malicious data can help the classification algorithms achieve higher performance in detecting cyber-attacks on the network environment.

We quantitatively measured the quality of the generated samples of CDAAE and compared it with the previous models using the Gaussian Parzen window. The results showed that the generated samples of CDAAE better follow the original data distribution than the other tested models. These results shed light on the classifiers' better performance when trained on the augmented datasets of CDAAE and CDAAE-KNN. However, the proposed models increase the model size compared with the previous generative models.

Chapter 4
DEEP TRANSFER LEARNING FOR NETWORK ATTACK DETECTION

In Chapter 2 and Chapter 3, we have proposed two solutions to leverage the effectiveness of deep learning models in NAD. These solutions are based on the assumption that we can collect labeled data of both normal and attack classes. However, the labeling 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 data for all samples collected from multiple devices. In other words, it is desirable to develop detection models that can be used to detect attacks on IoT devices without labeled information.

In this chapter, we propose a novel deep transfer learning (DTL) method that allows us to learn from data collected from multiple IoT devices in which not all of them are labeled. Specifically, we develop a DTL model based on the hybrid of two AutoEncoders (AEs). The first AE is trained on the source domain with label information, while the second AE is trained on the target domain without label information. The training process aims to transfer the knowledge of the source domain with label information to the target domain. As a result, the second AE can enhance the accuracy of the target domain even though it has no label information. The contributions of this chapter are publishes to the work [iv] of our publications.

4.1. Introduction

As presented in previous chapters, machine learning-based methods have received a great attention in NAD. They attempt to learn the fea-

tures of normal and malicious data in the training/offline phase. In the predicting/online phase, these models are used to detect attacks in the incoming traffic. Thanks to the capability to automatically and progressively learn useful information/features from collected data, machine learning-based methods can early detect various attacks [9, 21, 75, 76].

However, the machine learning-based methods only perform well under the critical assumption, i.e., the distributions of the training data and the predicting data are similar [3]. Nevertheless, in many real-world applications, this assumption may not always be the case [78, 79]. Especially in network security, new types of attacks (e.g., zero-day attacks) can be found daily [21]. Moreover, in some network environment as IoT, the data collected from different IoT devices can be different. As such, the practical traffic data for machine learning models (in the predicting/online phase) is usually very different from that used during the training/offline phase. To alleviate the above problem, a large volume of training data with labels from multiple IoT devices is often required. However, manually labeling a huge volume of data is very time consuming and expensive [117, 118]. It, thus, limits the practical deployment of machine learning-based methods in detecting attacks for various scenarios.

Given the above, this chapter proposes a novel deep transfer learning (DTL) approach based on AE to enable further applications of machine learning in IoT attack detection. The proposed model is referred to as Multi-Maximum Mean Discrepancy AE (MMD-AE). MMD-AE can be trained on a dataset including both labeled samples (in the source domain) and unlabeled samples (in the target domain). After training, MMD-AE is used to predict IoT attacks in the incoming traffic in the target domain. Specifically, MMD-AE consists of two AEs: AE_1 and AE_2. AE_1 in trained on labeled data while AE_2 is trained on the unlabeled data. The whole model, i.e., MMD-AE, is trained to drive the latent representation of AE_2 closely to the latent representation of AE_1.

As a result, the latent representation of AE_2 can be used to classify the unlabeled IoT data in the target domain.

The main contributions of this chapter are as follows:

- Propose a novel DTL model based on AEs, i.e., MMD-AE, that allows the transfer of knowledge, i.e., labeled information, from the source domain to the target domain. This model helps to lessen the problem of "lack label information" in network traffic.
- Introduce the use of the Maximum Mean Discrepancy (MMD) metric to minimize the distance between multiple hidden layers of AE_1 and multiple hidden layers of AE_2. This metric helps to improve the effectiveness of knowledge transferred from the source to the target domain in attack detection systems.
- Experiment with our proposed method using nine IoT attack datasets and compare its performance with the canonical deep learning model and the state-of-the-art TL models [2, 3]. The experimental results demonstrate the advantage of our proposed model against the other tested methods.

The rest of this chapter is organized as follows. Section 4.2 presents the proposed DTL models for NAD. The training and predicting process with the proposed DTL model are described in Section 4.3. Section 4.4 discusses the experiment settings and Section 4.5 provides detailed analysis and discussion related to experimental results. Finally, Section 4.6 concludes with future work.

4.2. Proposed Deep Transfer Learning Model

This section presents the proposed DTL models for attack detection. We first describe the overview of the system structure. After that, the DTL model is discussed in detail.

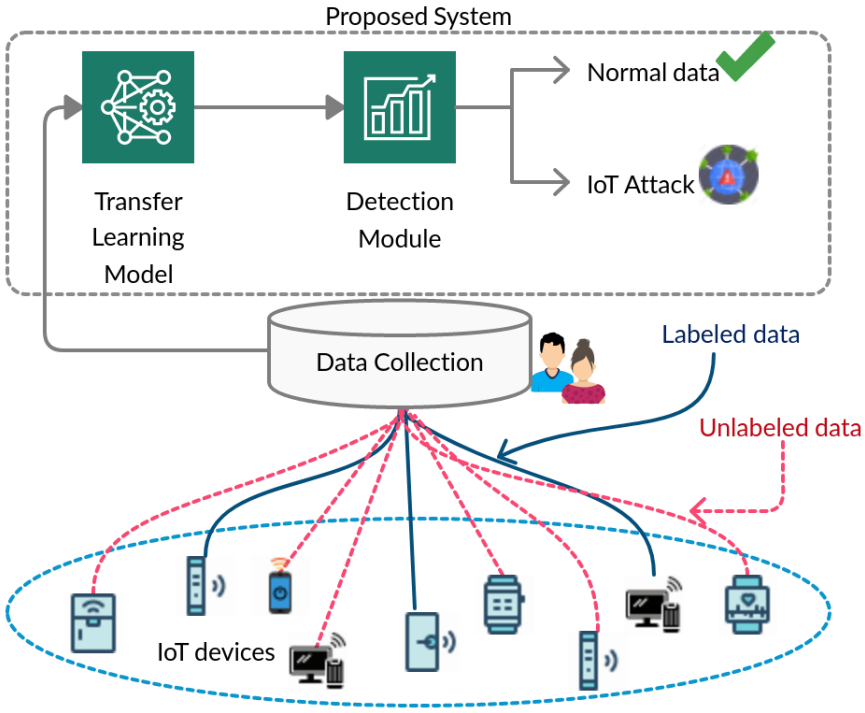


Figure 4.1: Proposed system structure.

4.2.1. System Structure

Fig. 4.1 presents the system structure that uses DTL for IoT attack detection. It can be adapted for other types of network environments. First, the data collection module gathers data from all IoT devices. The training data consists of both labeled and unlabeled data. The labeled data is collected from some IoTs devices which are dedicated to labeling data. The labeling process is usually executed in two steps [118]. Each data sample is extracted from captured packets using Tcptrace tool [119]. Then, the data sample is labeled as a normal sample or an attack sample by manually analyzing the flow using Wireshark software [120]. Usually, the number of labeling IoT devices can be smaller than the number of unlabeled IoT devices.

Second, the collected data is passed to the DTL model for training. The training process attempts to transfer the knowledge learned from

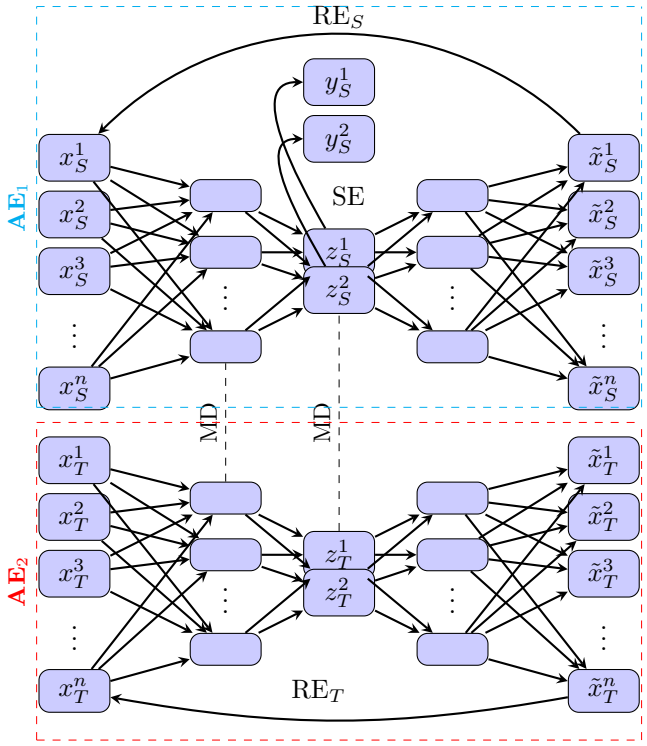


Figure 4.2: Architecture of MMD-AE.

the data to label information to data without labeling information. It is achieved by minimizing the difference between latent representations of the source data and the target data. After training, the trained DTL model is used in the detection module that can classify incoming traffic from all IoT devices as normal or attack data. The detailed description of the DTL model is presented in the next subsection.

4.2.2. Transfer Learning Model

The proposed DTL (i.e., MMD-AE) model includes two AEs (i.e., AE_1 an AE_2) that have the same architecture as Fig. 4.2. The input of AE_1 is the data samples from the source domain (x_S^i), while the input of AE_2 is the data samples from the target domain (x_T^i). The training process attempts to minimize the MMD-AE loss function. This loss function includes three terms: the reconstruction error (ℓ_{RE}) term, the supervised (ℓ_{SE}) term and the Multi-Maximum Mean Discrepancy (ℓ_{MMD}) term.

We assume that ϕ_S, θ_S, ϕ_T, θ_T are the parameter sets of encoder and

decoder of AE_1 and AE_2, respectively. The first term, ℓ_{RE} including RE_S and RE_T in Fig. 4.2 attempts to reconstruct the input layers at the output layers of both AEs. In other words, the RE_S and RE_T try to reconstruct the input data x_S and x_T at their output from the latent representations z_S and z_T, respectively. Thus, this term encourages two AEs to retain the useful information of the original data at the latent representation. Consequently, we can use latent representations for classification tasks after training. Formally, the ℓ_{RE} term is calculated as follows:

$$\ell_{\text{RE}}(x_S^i, \phi_S, \theta_S, x_T^i, \phi_T, \theta_T) = l(x_S^i, \hat{x}_S^i) + l(x_T^i, \hat{x}_T^i), \tag{4.1}$$

where l function is the MSE function [21], x_S^i, $\hat{x}_S^i$, x_T^i, $\hat{x}_T^i$ are the data samples of input layers and the output layers of the source domain and the target domain, respectively.

The second term ℓ_{SE} aims to train a classifier at the latent representation of AE_1 using labeled information in the source domain. In other words, this term attempts to map the value at two neurons at the bottleneck layer of AE_1, i.e., z_S, to their label information y_S. This is achieved by using the softmax function [1] to minimize the difference between z_S and y_S. It should be noted that, the number of neurons in the bottleneck layer must be the same as the number of classes in the source domain. This loss encourages to distinguish the latent representation space from separated class labels. Formally, this loss is defined as follows:

$$\ell_{\text{SE}}(x_S^i, y_S^i, \phi_S.\theta_S) = -\sum_{j=1}^{C} y_S^{i,j} \log(z_S^{i,j}), \tag{4.2}$$

where z_S^i and y_S^i are the latent representation and labels of the source data sample 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.

The third term ℓ_{MMD} is to transfer the knowledge of the source domain to the target domain. ℓ_{MMD} aims to present how close between two data distributions. The transferring process is executed by mini-

mizing the MMD distances between every encoding layers of AE_1 and the corresponding encoding layers of AE_2. This term aims to make the representations of the source data and target data close together. The ℓ_{MMD} loss term is described as follows:

$$\ell_{\text{MMD}}(x_S^i, \phi_S, \theta_S, x_T^i, \phi_T, \theta_T) = \sum_{k=1}^{K} \text{MMD}(\xi_S^k(x_S^i), \xi_T^k(x_T^i)), \tag{4.3}$$

where K is the number of encoding layers in the AE-based model. $\xi_S^k(x_S^i)$ and $\xi_T^k(x_T^i)$ are the encoding layers k-th of AE_1 and AE_2, respectively, $\text{MMD}(,)$ is the MMD distance presenting in Eq. 1.17.

The final loss function of MMD-AE combines the loss terms in Eq. 4.1, Eq. 4.2, and Eq. 4.3 as in Eq. 4.4.

$$\ell = \ell_{\text{SE}} + \ell_{\text{RE}} + \ell_{\text{MMD}}. \tag{4.4}$$

Our key idea in the proposed model, i.e., MMD-AE, compared with the previous DTL model [2, 3] is to transfer the knowledge not only in the bottleneck layer but also in *every encoding layer* from the source domain, i.e., AE_1, to the target domain, i.e., AE_2. In other words, MMD-AE allows transferring more knowledge from the source domain to the target domain. One possible limitation of MMD-AE is that it may incur the overhead time in the training process since the distance between multiple layers of the encoders in AE_1 and AE_2 is evaluated. However, in the predicting phase, only AE_2 is used to classify incoming samples in the target domain. Therefore, this model does not lead to increasing the predicting time compared to other AE-based models.

4.3. Training and Predicting Process using the MMD-AE Model

4.3.1. Training Process

Algorithm 7 presents the pseudocode for training our proposed DTL model, i.e., the MMD-AE model. The training samples with labels in the source domain are input to the first AE, while the training samples without labels in the target domain are input to the second AE. The

training process attempts to minimize the loss function in Eq. 4.4.

Algorithm 7 Training the MMD-AE model.

INPUT:
x_S, y_S: Training data samples and corresponding labels in the source domain
x_T: Training data samples in the target domain
OUTPUT: Trained models: AE_2.
BEGIN:
1. Put x_S to the input of AE_1
2. Put x_T to the input of AE_2
3. $\xi_j(x_S)$ is the representation of x_S at the layer j of AE_1
4. z_S is the representation of x_S at the bottleneck layer of AE_1
5. $\xi_j(x_T)$ is the representation of x_T at the layer j of AE_2
6. Training the TL model by minimizing the loss function in Eq. 4.4
return Trained models: AE_1, AE_2.
END.

4.3.2. Predicting Process

Algorithm 8 Classifying on the target domain by the MMD-AE model.

INPUT:
x_T^i: A network traffic data sample in the target domain
Trained AE_2 model
OUTPUT: y_T^i: Label of x_T^i
BEGIN:
1. Put x_T^i to the input of AE_2
2. z_T^i is the representation of x_T^i at the bottleneck layer of AE_2
3. y_T^i = softmax (z_T^i)
return y_T^i
END.

After training, AE_2 is used to classify the testing samples in the target domain as in Algorithm 8. First, a network traffic data sample in the target domain is put to the input of AE_2 to get the bottleneck layer z_T^i. Then, the the label y_T^i is calculated by applying the softmax function to z_T^i.

4.4. Experimental Settings

We use the IoT datasets presented in Chapter 1 for all experiments in this chapter. This section presents the hyper-parameter settings and the experimental sets in this chapter.

4.4.1. Hyper-parameters Setting

Table 4.1: Hyper-parameter setting for the DTL models.

Hyper-parameter	**Value**
Number of layers	5
Bottleneck layer size	2
Optimization algorithm	Adam
Activation function	Relu

The same configuration is used for all AE-based models in our experiments. Table 4.1 presents the common hyper-parameters using for the AE-based models. This configuration is based on the AE-based models for detecting network attacks in the literature [9,21,94]. As we integrate the ℓ_{SE} loss term to MMD-AE, the number of neurons in the bottleneck layer is equal to the number of classes in the IoT dataset, i.e., 2 neurons in this chapter. The reason is that we aim to classify into two classes in this bottleneck layer. The number of layers, including both the encoding layers and the decoding layers, is 5. This follows the previous research for network traffic data [21]. The ADAM algorithm [107] is used for optimizing the models in the training process. The ReLu function is used as an activation function of AE layers except for the last layers of the encoder and decoder, where the Sigmoid function is used.

4.4.2. Experimental Sets

We carried out three sets of experiments in this chapter. The first set is to investigate how effective our proposed model is at transferring knowledge from the source domain to the target domain. We compare the MMD distances between the bottleneck layer of the source domain and the target domain after training when the transferring process is executed in one, two, and three encoding layers. The smaller MMD distance, the more effective the transferring process from the source to the target domain [121].

The second set is the main result of the chapter in which we compare the AUC scores of MMD-AE with AE and two recent DTL models [2,3].

We choose two these DTL models for comparision due to two reasons: (1) these are based on AE models and the AE-based models are the most effective with network traffic datasets in many work [9, 21, 94] and (2) these DTL models are in the same transfer learning domain with our proposed model where the source dataset has label information and the target dataset has no label information. All methods are trained using the training set, including the source dataset with label information and the target dataset without label information. After training, the trained models are evaluated using the target dataset. The methods compared in this experiment include the original AE (i.e., AE), and the DTL model using the KL metric at the bottleneck layer (i.e., SKL-AE) [2], the DTL method of using the MMD metric at the bottleneck layer (i.e., SMD-AE) [3], and our model (MMD-AE).

The third set is to measure the training's processing time and the predicting process of the above-evaluated methods. Moreover, the model size reported by the trainable parameters presents the complexity of the DTL models. The detailed results of three experimental sets are presented in the next section.

4.5. Results and Discussions

This section presents the result of three sets of the experiments in this chapter.

4.5.1. Effectiveness of Transferring Information in MMD-AE

MMD-AE implements multiple transfer between encoding layers of AE_1 and AE_2 to force the latent representation AE_2 closer to the latent representation AE_1. In order to evaluate if MMD-AE achieves its objective, we conducted an experiment in which IoT-1 is selected as the source domain, and IoT-2 is the target domain. We measured the MMD distance between the latent representation, i.e., the bottleneck layer, of AE_1 and AE_2 when the transfer information is implemented in one, two and three layers of the encoders. The smaller distance, the more infor-

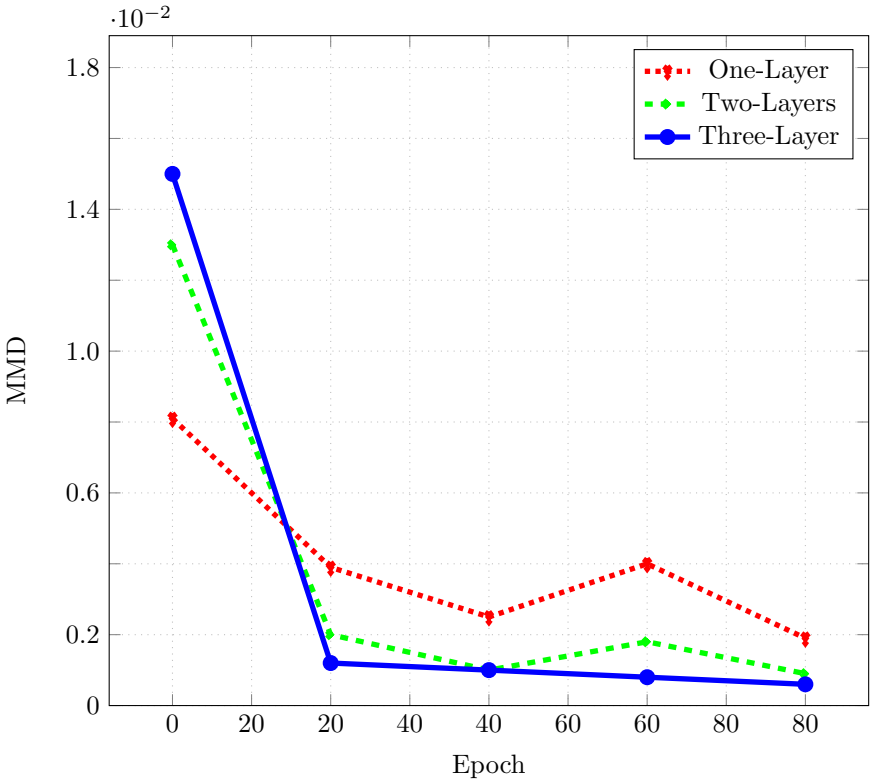


Figure 4.3: MMD of latent representations of the source (IoT-1) and the target (IoT-2) when transferring task on one, two, and three encoding layers.

mation is transferred from the source domain (AE_1) to the target domain (AE_2). The result is presented in Fig. 4.3.

The figure shows that transferring tasks implemented on more layers results in the smaller MMD distance value. In other words, more information can be transferred from the source to the target domain when the transferring task is implemented on a more encoding layer. This result evidences that our proposed solution, MMD-AE, is more effective than the previous DTL models that perform the transferring task only on the bottleneck layer of AE.

4.5.2. Performance Comparison

Table 4.2 represents the AUC scores of AE, SKL-AE, SMD-AE, and MMD-AE when they are trained on the dataset with label information in the columns and the dataset without information in the rows and tested on the dataset in the rows. In this table, the result of MMD-AE is printed in boldface. We can observe that AE is the worst method among the tested methods. When an AE is trained on an IoT dataset (the source) and evaluating on other IoT datasets (the target), its performance is not convincing. The reason for this unconvincing result is that predicting data in the target domain is far different from the training data in the source domain.

Conversely, the results of three DTL models are much better than the one of AE. For example, if the source dataset is IoT-1 and the target dataset is IoT-3, the AUC score is improved from 0.600 to 0.745 and 0.764 with SKL-AE and SMD-AE, respectively. These results prove that using DTL helps to improve the accuracy of AEs on detecting IoT attacks on the target domain.

More importantly, our proposed method, i.e., MMD-AE, usually achieves the highest AUC score in almost all IoT datasets[1]. For example, the AUC score is 0.937 compared to $0.600, 0.745, 0.764$ of AE, SKL-AE, and SMD-AE, respectively, when the source dataset is IoT-1, and the target

[1]The AUC scores of the proposed model in each scenario is presented by the bold text style.

Table 4.2: AUC scores of AE [1], SKL-AE [2], SMD-AE [3] and MMD-AE on nine IoT datasets.

Target	Model	Source								
		IoT-1	**IoT-2**	**IoT-3**	**IoT-4**	**IoT-5**	**IoT-6**	**IoT-7**	**IoT-8**	**IoT-9**
IoT-1	AE		0.705	0.542	0.768	0.838	0.643	0.791	0.632	0.600
	SKL-AE		0.700	0.759	0.855	0.943	0.729	0.733	0.689	0.705
	SMD-AE		0.722	0.777	0.875	0.943	0.766	0.791	0.701	0.705
	MMD-AE		**0.888**	**0.796**	**0.885**	**0.943**	**0.833**	**0.892**	**0.775**	**0.743**
IoT-2	AE	0.540		0.500	0.647	0.509	0.743	0.981	0.777	0.578
	SKL-AE	0.545		0.990	0.708	0.685	0.794	0.827	0.648	0.606
	SMD-AE	0.563		0.990	0.815	0.689	0.874	0.871	0.778	0.607
	MMD-AE	**0.937**		**0.990**	**0.898**	**0.692**	**0.878**	**0.900**	**0.787**	**0.609**
IoT-3	AE	0.600	0.659		0.530	0.500	0.501	0.644	0.805	0.899
	SKL-AE	0.745	0.922		0.566	0.939	0.534	0.640	0.933	0.916
	SMD-AE	0.764	0.849		0.625	0.879	0.561	0.600	0.918	0.938
	MMD-AE	**0.937**	**0.956**		**0.978**	**0.928**	**0.610**	**0.654**	**0.937**	**0.946**
IoT-4	AE	0.709	0.740	0.817		0.809	0.502	0.944	0.806	0.800
	SKL-AE	0.760	0.852	0.837		0.806	0.824	0.949	0.836	0.809
	SMD-AE	0.777	0.811	0.840		0.803	0.952	0.947	0.809	0.826
	MMD-AE	**0.937**	**0.857**	**0.935**		**0.844**	**0.957**	**0.959**	**0.875**	**0.850**
IoT-5	AE	0.615	0.598	0.824	0.670		0.920	0.803	0.790	0.698
	SKL-AE	0.645	0.639	0.948	0.633		0.923	0.695	0.802	0.635
	SMD-AE	0.661	0.576	0.954	0.672		0.945	0.822	0.789	0.833
	MMD-AE	**0.665**	**0.508**	**0.954**	**0.679**		**0.928**	**0.847**	**0.816**	**0.928**
IoT-6	AE	0.824	0.823	0.699	0.834	0.936		0.765	0.836	0.737
	SKL-AE	0.861	0.897	0.711	0.739	0.980		0.893	0.787	0.881
	SMD-AE	0.879	0.898	0.713	0.849	0.982		0.778	0.867	0.898
	MMD-AE	**0.927**	**0.899**	**0.787**	**0.846**	**0.992**		**0.974**	**0.871**	**0.898**
IoT-7	AE	0.504	0.501	0.626	0.791	0.616	0.809		0.598	0.459
	SKL-AE	0.508	0.625	0.865	0.831	0.550	0.906		0.358	0.524
	SMD-AE	0.519	0.619	0.865	0.817	0.643	0.884		0.613	0.604
	MMD-AE	**0.548**	**0.621**	**0.888**	**0.897**	**0.858**	**0.905**		**0.615**	**0.618**
IoT-8	AE	0.814	0.599	0.831	0.650	0.628	0.890	0.901		0.588
	SKL-AE	0.619	0.636	0.892	0.600	0.629	0.923	0.907		0.712
	SMD-AE	0.622	0.639	0.902	0.717	0.632	0.919	0.872		0.629
	MMD-AE	**0.735**	**0.636**	**0.964**	**0.723**	**0.692**	**0.977**	**0.943**		**0.616**
IoT-9	AE	0.823	0.601	0.840	0.851	0.691	0.808	0.885	0.579	
	SKL-AE	0.810	0.602	0.800	0.731	0.662	**0.940**	0.855	0.562	
	SMD-AE	0.830	0.609	0.892	0.600	0.901	0.806	0.886	0.626	
	MMD-AE	**0.843**	**0.911**	**0.910**	**0.874**	**0.904**	0.829	**0.889**	**0.643**	

Table 4.3: Processing time and complexity of DTL models.

Models	Training Time (hours)	Predicting Time (second)	No. Trainable Parameters
AE [1]	0.001	1.001	25117
SKL-AE [2]	0.443	1.112	150702
SMD-AE [3]	3.693	1.110	150702
MMD-AE	11.057	1.108	150702

dataset is IoT-3. The results on the other datasets are also similar to the result of IoT-3. This result proves that implementing the transferring task in multiple layers of MMD-AE helps the model transfers more effectively the label information from the source to the target domain. Subsequently, MMD-AE often achieves better results compared to AE, SKL-AE, and SMD-AE in detecting IoT attacks in the target domain.

4.5.3. Processing Time and Complexity Analysis

Table. 4.3 shows the training and the predicting time of the tested model when the source domain is IoT-2, and the target domain is IoT-1[2]. In this table, the training time is measured in **hours**, and the predicting time is measured in **seconds**. It can be seen that the training process of the DTL methods (i.e., SKL-AE, SMD-AE, and MMD-AE) is more time consuming than that of AE. One of the reasons is that DTL models need to evaluate the MMD distance between the AE_1 and AE_2 in every iteration while this calculation is not required in AE. Moreover, the training time of MMD-AE is even much higher than those of SKL-AE and SMD-AE since MMD-AE needs to calculate the MMD distance between every encoding layer. In contrast, SKL-AE and SMD-AE only calculate the distance metric in the bottleneck layer. Moreover, the training processes present the same number of trainable parameters for all the DTL models based on AE.

However, more important is that the predicting time of all DTL methods is mostly equal to that of AE. It is reasonable since the testing samples are only fitted to one AE in all tested models. For example, the

[2]The results on the other datasets are similar to this result.

total of the predicting time of AE, SKL-AE, SMD-AE, and MMD-AE are 1.001, 1.112, 1.110, and 1.108 seconds, respectively, on 778810 testing samples of the IoT-1 dataset.

4.6. Conclusion

In this chapter, we have introduced a novel DTL-based approach for IoT network attack detection, namely MMD-AE. This proposed approach aims to address the problem of "lack of labeled information" for the training detection model in ubiquitous IoT devices. The labeled data and unlabeled data are specially fitted into two AE models with the same network structure. Moreover, the MMD metric is used to transfer knowledge from the first AE to the second AE. Comparing to the previous DTL models, MMD-AE is operated on all the encoding layers instead of only the bottleneck layer.

We have carried out extensive experiments to evaluate the strength of our proposed model in many scenarios. The experimental results demonstrate that DTL approaches can enhance the AUC score for IoT attack detection. Furthermore, our proposed DTL model, i.e., MMD-AE and operating transformation at all encoding layers of the AEs, helps to improve the effectiveness of the transferring process. Thus, the proposed model is meaningful when labeling information in the source domain but with no label information in the target domain.

An important limitation of the proposed model is that it is more time consuming to train the model. However, the predicting time of MMD-AE is mostly similar to that of the other AE-based models. In the future, we will distribute the training process to the multiple IoT nodes by the federated learning technique to speed up this process.

CONCLUSIONS AND FUTURE WORK

1. Contributions

This thesis aims to develop the machine learning-based approaches for the NAD. First, to effectively detect new/unknown attacks by machine learning methods, we propose a novel representation learning method to better predictively "describe" unknown attacks, facilitating the subsequent machine learning-based NAD. Specifically, we develop three regularized versions of AEs to learn a latent representation from the input data. The bottleneck layers of these regularized AEs trained in a supervised manner using normal data and known network attacks will then be used as the new input features for classification algorithms. The experimental results demonstrate that the new latent representation can significantly enhance the performance of supervised learning methods in detecting unknown network attacks.

Second, we handle the imbalance problem of network attack datasets. To develop a good detection model for a NAD system using machine learning, a great number of attacks and normal data samples are required in the learning process. While normal data can be relatively easy to collect, attack data is much rarer and harder to gather. Subsequently, network attack datasets are often dominated by normal data, and machine learning models trained on those imbalanced datasets are ineffective in detecting attacks. In this thesis, we propose a novel solution to this problem by using generative adversarial networks to generate synthesized attack data for network attack data. The synthesized attacks are merged with the original data to form the augmented dataset. In the sequel, the supervised learning algorithms trained on the augmented datasets provide better results than those trained on the original

datasets.

Third, we resolve "the lack of label information" in the NAD problem. In some situations, we are unable to collect network traffic data with its label information. For example, we are unable to label all incoming data from all IoT devices in the IoT environment. Moreover, data distributions of data samples collected from different IoT devices are not the same. Thus, we develop a TL technique 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 experimental results demonstrate that the proposed TL technique can help classifiers to identify attacks more accurately.

In addition to a review of literature regarding to the research in this thesis, the following main contributions can be drawn from the investigations presented in the thesis:

- Three latent representation learning models are proposed based on AEs to make the machine learning models to detect both known and unknown attacks.
- Three new techniques are proposed for handling data imbalance, thereby improving the accuracy of the network attack detection system.
- A DTL technique based on AE is proposed to handle "the lack of label information" in the new domain of network traffic data.

2. Limitations

However, the thesis is subject to some limitations. First, the advantages of representation learning models come with the cost of running time. When using a neural network to lean the representation of input data, the executing time of these models is often much longer than using classifiers on the original feature spaces. The proposed representation learning models in this thesis also have these drawbacks. However, it

can be seen in Chapter 2 that the average time of predicting one sample of the representation learning models is acceptable in real applications. Moreover, the regularized AE models are only tested on a number of IoT attack datasets. It is also more comprehensive to experiment with them on a broader range of problems.

Second, in CDAAE, we need to assume that the original data distribution follows a Gaussian distribution. It may be correct with the popularity of network traffic datasets but not entire network traffic datasets. Moreover, this thesis focuses on only sampling techniques for handling imbalanced data. It is usually time-consuming due to generating data samples.

Third, training MMD-AE is more time consuming than previous DTL models due to transferring processes executed in multiple layers. However, the predicting time of MMD-AE is mostly similar to that of the other AE-based models. Moreover, the current proposed DTL model is developed based on the AE model.

3. Future work

Building upon this research, there are a number of directions for future work arisen from the thesis. First, there are some hyper-parameters of the proposed representations of AE-based models (i.e., μ_{y^i}) are currently determined through trial and error. It is desirable to find an approach to select proper values for each network attack dataset automatically.

Second, in the CDAAE model, we can explore other distributions different from the Gaussian distribution that may better represent the original data distribution. Moreover, the CDAAE model can learn from the external information instead of the label of data only. We expect that by adding some attributes of malicious behaviors to CDAAE, the synthesized data will be more similar to the original data. Last but not least, we will distribute the training process of the proposed DTL model to the multiple IoT nodes by the federated learning technique to speed up this process.

PUBLICATIONS

[i] **Ly Vu**, Cong Thanh Bui, and Nguyen Quang Uy: *A deep learning based method for handling imbalanced problem in network traffic classification.* In: Proceedings of the Eighth International Symposium on Information and Communication Technology. pp. 333–339. ACM (Dec. 2017).

[ii] **Ly Vu**, Van Loi Cao, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz: *Learning Latent Distribution for Distinguishing Network Traffic in Intrusion Detection System.* IEEE International Conference on Communications (ICC), Rank B, pp. 1–6 (2019).

[iii] **Ly Vu** and Quang Uy Nguyen: *An Ensemble of Activation Functions in AutoEncoder Applied to IoT Anomaly Detection.* In: The 2019 6th NAFOSTED Conference on Information and Computer Science (NICS'19), pp. 534–539 (2019).

[iv] **Ly Vu** and Quang Uy Nguyen: *Handling Imbalanced Data in Intrusion Detection Systems using Generative Adversarial Networks.* In: Journal of Research and Development on Information and Communication Technology. Vol. 2020, no. 1, Sept. 2020.

[v] **Ly Vu**, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz:*Deep Transfer Learning for IoT Attack Detection.* In: IEEE Access (ISI-SCIE, IF = 3.745). pp.1-10, June 2020.

[vi] **Ly Vu**, Van Loi Cao, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz: *Learning Latent Representation for IoT Anomaly Detection.* In: IEEE Transactions on Cybernetics (ISI-SCI, IF=11.079). DOI: 10.1109/TCYB.2020.3013416, Sept. 2020.

BIBLIOGRAPHY

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He, "Supervised representation learning: Transfer learning with deep autoencoders," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[3] L. Wen, L. Gao, and X. Li, "A new deep transfer learning based on sparse auto-encoder for fault diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 136–144, 2017.

[4] "Cisco visual networking index: Forecast and methodology, 2016-2021.," 2017. `https://www.reinvention.be/webhdfs/v1/docs/complete-white-paper-c11-481360.pdf`.

[5] "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`.

[6] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. C. Atkinson, and X. J. A. Bellekens, "A taxonomy and survey of intrusion detection system design techniques, network threats and datasets," *CoRR*, vol. abs/1806.03517, 2018.

[7] X. Jing, Z. Yan, and W. Pedrycz, "Security data collection and data analytics in the internet: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 586–618, 2018.

[8] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pp. 130–143, IEEE, 2001.

[9] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, pp. 12–22, Jul 2018.

[10] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 898–924, Second 2014.

[11] H. Bahşi, S. Nõmm, and F. B. La Torre, "Dimensionality reduction for machine learning based IoT botnet detection," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1857–1862, Nov 2018.

[12] S. S. Chawathe, "Monitoring IoT networks for botnet activity," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, Nov 2018.

[13] S. Nomm and H. Bahsi, "Unsupervised anomaly based botnet detection in IoT networks," *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1048–1053, 2018.

[14] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, July 2009.

[15] Y. Zou, J. Zhu, X. Wang, and L. Hanzo, "A survey on wireless security: Technical challenges, recent advances, and future trends," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1727–1765, 2016.

[16] M. Ali, S. U. Khan, and A. V. Vasilakos, "Security in cloud computing: Opportunities and challenges," *Information sciences*, vol. 305, pp. 357–383, 2015.

[17] "Nsl-kdd dataset [online]." `http://nsl.cs.unb.ca/NSL-KDD/`. Accessed: 2018-04-10.

[18] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems conference (MilCIS)*, pp. 1–6, IEEE, 2015.

[19] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[20] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, pp. 153–160, 2007.

[21] V. L. Cao, M. Nicolau, and J. McDermott, "Learning neural representations for network anomaly detection," *IEEE Transactions on Cybernetics*, vol. 49, pp. 3074–3087, Aug 2019.

[22] W. W. Ng, G. Zeng, J. Zhang, D. S. Yeung, and W. Pedrycz, "Dual autoencoders features for imbalance classification problem," *Pattern Recognition*, vol. 60, pp. 875–889, 2016.

[23] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[24] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, "Stacked convolutional denoising auto-encoders for feature representation," *IEEE Transactions on Cybernetics*, vol. 47, pp. 1017–1027, April 2017.

[25] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[27] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2226–2234, 2016.

[28] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the 34th International Conference on Machine*

Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pp. 2642–2651, 2017.

[29] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[30] A. Creswell and A. A. Bharath, "Denoising adversarial autoencoders," *IEEE Transactions on Neural Networks and Learning Systems*, no. 99, pp. 1–17, 2018.

[31] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," in *Advances in neural information processing systems*, pp. 513–520, 2007.

[32] D. Powers, "Evaluation: From precision, recall and fmeasure to roc, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, vol. 2, pp. 37–63, 01 2007.

[33] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[35] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, 2019.

[36] P. S. Kenkre, A. Pai, and L. Colaco, "Real time intrusion detection and prevention system," in *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*, pp. 405–411, Springer, 2015.

[37] N. Walkinshaw, R. Taylor, and J. Derrick, "Inferring extended finite state machine models from software executions," *Empirical Software Engineering*, vol. 21, no. 3, pp. 811–853, 2016.

[38] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *International Journal of Embedded Systems*, vol. 10, no. 1, pp. 1–12, 2018.

[39] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1690–1700, 2014.

[40] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[41] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on computers*, vol. 51, no. 7, pp. 810–820, 2002.

[42] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen, "Processing intrusion detection alert aggregates with time series modeling," *Information Fusion*, vol. 10, no. 4, pp. 312–324, 2009.

[43] Q. Wu and Z. Shao, "Network anomaly detection using time series analysis," in *Joint international conference on autonomic and autonomous systems and international conference on networking and services-(icas-isns' 05)*, pp. 42–42, IEEE, 2005.

[44] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 303–336, First 2014.

[45] S. Zanero and S. M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 412–419, 2004.

[46] H. Qu, Z. Qiu, X. Tang, M. Xiang, and P. Wang, "Incorporating unsupervised learning into intrusion detection for wireless sensor networks with structural co-evolvability," *Applied Soft Computing*, vol. 71, pp. 939–951, 2018.

[47] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[48] K. Ghanem, F. J. Aparicio-Navarro, K. G. Kyriakopoulos, S. Lambotharan, and J. A. Chambers, "Support vector machine for network intrusion and cyber-attack detection," in *2017 Sensor Signal Processing for Defence Conference (SSPD)*, pp. 1–5, Dec 2017.

[49] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," *2010 IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.

[50] B. S. Bhati and C. Rai, "Analysis of support vector machine-based intrusion detection techniques," *Arabian Journal for Science and Engineering*, pp. 1–13, 2019.

[51] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," *2003 Symposium on Applications and the Internet, 2003. Proceedings.*, pp. 209–216, 2003.

[52] G. Nadiammai and M. Hemalatha, "Performance analysis of tree based classification algorithms for intrusion detection system," in *Mining Intelligence and Knowledge Exploration*, pp. 82–89, Springer, 2013.

[53] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, no. 1, pp. 213–217, 2016.

[54] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–36, 2018.

[55] P. Negandhi, Y. Trivedi, and R. Mangrulkar, "Intrusion detection system using random forest on the nsl-kdd dataset," in *Emerging Research in Computing, Information, Communication and Applications*, pp. 519–531, Springer, 2019.

[56] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Transaction Neural Network Learning System*, vol. 29, no. 8, pp. 3573–3587, 2018.

[57] Y. Zhang and D. Wang, "A cost-sensitive ensemble method for class-imbalanced datasets," *Abstract and Applied Analysis*, vol. 2013, 2013.

[58] A. D. Pozzolo, O. Caelen, S. Waterschoot, and G. Bontempi, "Cost-aware pre-training for multiclass cost-sensitive deep learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1411–1417, 2016.

[59] K. Li, X. Kong, Z. Lu, L. Wenyin, and J. Yin, "Boosting weighted ELM for imbalanced learning," *Neurocomputing*, vol. 128, pp. 15–21, 2014.

[60] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4368–4374, July 2016.

[61] V. Raj, S. Magg, and S. Wermter, "Towards effective classification of imbalanced data with convolutional neural networks," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 150–162, Springer, 2016.

[62] A. D. Pozzolo, O. Caelen, S. Waterschoot, and G. Bontempi, "Racing for unbalanced methods selection," in *Intelligent Data Engineering and Automated Learning - IDEAL 2013 - 14th International Conference, IDEAL 2013, Hefei, China, October 20-23, 2013. Proceedings*, pp. 24–31, 2013.

[63] C. Drummond and R. C. Holte, "C4.5, class imbalance, and cost sensitivity: Why under-sampling beats oversampling," *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Datasets*, pp. 1–8, 01 2003.

[64] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[65] H. M. Nguyen, E. W. Cooper, and K. Kamei, "Borderline over-sampling for imbalanced data classification," *International Journal of Knowledge Engineering and Soft Data Paradigms*, vol. 3, no. 1, pp. 4–21, 2011.

[66] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transaction Systems, Man, and Cybernetics, Part B*, vol. 39, no. 2, pp. 539–550, 2009.

[67] N. C. Oza, "Online bagging and boosting," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2340–2345, IEEE, 2005.

[68] A. Namvar, M. Siami, F. Rabhi, and M. Naderpour, "Credit risk prediction in an imbalanced social lending environment," *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 925–935, 2018.

[69] Q. Wang, Z. Luo, J. Huang, Y. Feng, and Z. Liu, "A novel ensemble method for imbalanced data learning: Bagging of extrapolation-smote SVM," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1827016:1–1827016:11, 2017.

[70] R. Longadge and S. Dongre, "Class imbalance problem in data mining review," *arXiv preprint arXiv:1305.1707*, 2013.

[71] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.

[72] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *International Conference on Neural Information Processing*, pp. 858–866, Springer, 2017.

[73] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*, pp. 712–717, Jan 2017.

[74] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, pp. 1–14, 2019.

[75] J. Dromard, G. Roudière, and P. Owezarski, "Online and scalable unsupervised network anomaly detection method," *IEEE Transactions on Network and Service Management*, vol. 14, pp. 34–47, March 2017.

[76] O. Ibidunmoye, A. Rezaie, and E. Elmroth, "Adaptive anomaly detection in performance metric streams," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 217–231, March 2018.

[77] R. Salakhutdinov and H. Larochelle, "Efficient learning of deep boltzmann machines," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 693–700, 2010.

[78] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[79] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: a survey," *Knowledge-Based Systems*, vol. 80, pp. 14–23, 2015.

[80] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, p. 9, 2016.

[81] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International Conference on Artificial Neural Networks*, pp. 270–279, Springer, 2018.

[82] C. Wan, R. Pan, and J. Li, "Bi-weighting domain adaptation for cross-language text classification," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[83] Y. Xu, S. J. Pan, H. Xiong, Q. Wu, R. Luo, H. Min, and H. Song, "A unified framework for metric transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 6, pp. 1158–1171, 2017.

[84] X. Liu, Z. Liu, G. Wang, Z. Cai, and H. Zhang, "Ensemble transfer learning algorithm," *IEEE Access*, vol. 6, pp. 2389–2396, 2018.

[85] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.

[86] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "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.

[87] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7167–7176, 2017.

[88] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Domain adaptation with randomized multilinear adversarial networks," *arXiv preprint arXiv:1705.10667*, 2017.

[89] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "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.

[90] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," in *Advances in Neural Information Processing Systems*, pp. 136–144, 2016.

[91] C. Kandaswamy, L. M. Silva, L. A. Alexandre, R. Sousa, J. M. Santos, and J. M. de Sá, "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.

[92] N. C. Luong, D. T. Hoang, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "Data collection and wireless communication in internet of things (IoT) using economic analysis and pricing models: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 2546–2590, Fourthquarter 2016.

[93] I. Ahmed, A. P. Saleel, B. Beheshti, Z. A. Khan, and I. Ahmad, "Security in the internet of things (IoT)," in *2017 Fourth HCT Information Technology Trends (ITT)*, pp. 84–90, Oct 2017.

[94] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized IoT devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[95] C. Zhang and R. Green, "Communication security in internet of thing: Preventive measure and avoid ddos attack over IoT network," in *Proceedings of the 18th Symposium on Communications & Networking*, CNS '15, (San Diego, CA, USA), pp. 8–15, Society for Computer Simulation International, 2015.

[96] C. Dietz, R. L. Castro, J. Steinberger, C. Wilczak, M. Antzek, A. Sperotto, and A. Pras, "IoT-botnet detection and isolation by access routers," in *2018 9th International Conference on the Network of the Future (NOF)*, pp. 88–95, Nov 2018.

[97] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using openflow," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pp. 147–156, Aug 2016.

[98] J. M. Ceron, K. Steding-Jessen, C. Hoepers, L. Z. Granville, and C. B. Margi, "Improving IoT botnet investigation using an adaptive network layer," *Sensors (Basel)*, vol. 19, no. 3, p. 727, 2019.

[99] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[100] V. L. Cao, M. Nicolau, and J. McDermott, "A hybrid autoencoder and density estimation model for anomaly detection," in *International Conference on Parallel Problem Solving from Nature*, pp. 717–726, Springer, 2016.

[101] S. E. Chandy, A. Rasekh, Z. A. Barker, and M. E. Shafiee, "Cyberattack detection using deep generative models with variational inference," *Journal of Water Resources Planning and Management*, vol. 145, no. 2, p. 04018093, 2018.

[102] "Sklearn tutorial [online]." `http://scikit-learn.org/stable/`. Accessed: 2018-04-24.

[103] S. D. D. Anton, S. Sinha, and H. Dieter Schotten, "Anomaly-based intrusion detection in industrial data with svm and random forests," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, 2019.

[104] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, pp. 649–659, Sept. 2008.

[105] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[106] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[107] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[109] "Implementation of deep belief network." `https://github.com/JosephGatto/Deep-Belief-Networks-Tensorflow.`

[110] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Ddos-capable IoT malwares: Comparative analysis and mirai investigation," *Security and Communication Networks*, vol. 2018, 2018.

[111] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1093–1110, USENIX Association, Aug. 2017.

[112] "9 distance measures in data science," 2020. `https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.`

[113] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of real-time processing technologies of iot data streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[114] "Real-time stream processing for internet of things." https://medium.com/@exastax/real-time-stream-processing-for-internet-of-things-24ac529f75a3.

[115] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International Conference on Intelligent Computing*, pp. 878–887, Springer, 2005.

[116] J. Cervantes, F. García-Lamont, L. Rodríguez-Mazahua, A. López Chau, J. S. R. Castilla, and A. Trueba, "Pso-based method for SVM classification on skewed data sets," *Neurocomputing*, vol. 228, pp. 187–197, 2017.

[117] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[118] S. García, A. Zunino, and M. Campo, "Botnet behavior detection using network synchronism," in *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*, pp. 122–144, IGI Global, 2012.

[119] "Tcptrace tool for analysis of tcp dump files," 2020. http://www.tcptrace.org/.

[120] "Wireshark tool, the world's foremost and widely-used network protocol analyzer," 2020. https://www.wireshark.org/.

[121] J. Yang, R. Yan, and A. G. Hauptmann, "Cross-domain video concept detection using adaptive svms," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 188–197, 2007.