University of Technology Sydney
Faculty of Engineering and Information Technology
School of Electrical and Data Engineering

Deep Neural Network for Anomaly Detection

Ly Thi Vu

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

under the supervision of

A/Prof. Diep N. Nguyen
A/Prof. Hoang Dinh
Prof. Eryk Dutkiewicz

Sydney, Australia

June 2024

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Ly Thi Vu, declare that this thesis is submitted in fulfillment of the requirements for the award of DOCTOR OF PHILOSOPHY, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This thesis has not been submitted for qualifications at any other academic institution.

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree at any other academic institution except as fully acknowledged within the text. This thesis is the result of a Collaborative Doctoral Research Degree program with Le Quy Don Technical University.

This research is supported by the Australian Government Research Training Program.

Student Name: Ly Thi Vu

Signature:

Production Note:
Signature removed prior to publication.

Date: 01/06/2024

ABSTRACT

Deep Neural Network for Anomaly Detection

by

Ly Thi Vu

The rapid growth in diverse network devices (e.g., Internet of Things/IoT devices) and new cyber-physical systems (CPSs) services create new surfaces for cyberattacks. To safeguard these CPSs, anomaly detection (AD) that detects potential attacks/adversarial behaviors plays a pivotal role. This thesis aims to design novel deep neural models to handle four challenges of the AD problem to deal with new/unknown attacks, imbalanced data, the lack of labelled data, and the vulnerability to data poisoning attacks.

First, to detect new/unknown anomalies (attacks) effectively, the thesis proposes a novel representation learning method, i.e., AutoEncoders (AEs) based models, that better represents unknown attacks, facilitating supervised learning-based AD methods. An AE consists of an encoder and a decoder component. The encoder compresses the input data into a lower-dimension representation, while the decoder attempts to reconstruct the original input from this compressed representation. Specifically, we develop three regularized AEs variants to learn a latent representation from the input data. In the new feature space, the normal and the attack data are more effectively separated. Therefore, the accuracy of detecting both known and unknown attacks is improved significantly.

Second, the thesis introduces two deep generative models to handle the imbalanced data. The first model, Conditional Denoising Adversarial AutoEncoder (CDAAE), generates specific types of attack samples. The second model (CDAEE-KNN) is a hybrid of CDAAE and the K-nearest Neighbor algorithm to generate borderline attack samples. By training on the augmented datasets, the accuracy of

the AD problems is enhanced significantly.

Third, the thesis designs a Deep Transfer Learning (DTL) model to build an effective AD system from both labelled and unlabelled data. Specifically, we develop a DTL model based on two AEs. The first AutoEncoder (AE) is trained on the source datasets (source domains) in the supervised mode using the label information, and the second AE is trained on the target datasets (target domains) in an unsupervised manner without label information. As a result, the latent representation of the second AE can be used to detect attacks in the target domain effectively.

Fourth, to reduce the influence of data poisoning attacks that are damaging and popular to low-end IoT devices, the thesis proposes a novel Federated Learning (FL) system with the Shrink Denoising AutoEncoder (FL-SDAE). The reconstruction term of the loss function helps Shrink Denoising AutoEncoder (SDAE) reconstruct the original data from its corrupted version. Therefore, the proposed SDAE model makes FL-SDAE robust to data poisoning attacks.

Dedication

To my beloved husband, children, parents, parents-in-law, university, and country, Vietnam.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors A/Prof. Diep N. Nguyen, A/Prof. Hoang Dinh, and Prof. Eryk Dutkiewicz for all the guidance, encouragement, and enduring patience. My supervisors have taught me a lot on both the scientific and personal sides of the research journey. Their guidance and support go far beyond this thesis, and I have been greatly fortunate to be supervised by them.

I would especially like to thank A/Prof. Quang Uy Nguyen from Le Quy Don Technical University for his insightful advice and great support. I would like to thank the Institute of Information and Communication Technology - Le Quy Don Technical University (LQDTU) for giving me the Collaborative Research Degree Program Scholarship. Furthermore, many thanks to all the staff from the School of Electrical and Data Engineering (SEDE), Faculty of Engineering and Information Technology, University of Technology Sydney (UTS), and UTS Library for the various forms of help they gave to me. I also would like to thank all my colleagues and friends at the UTS for their support, discussion, and friendship.

Last but most importantly, my deepest gratitude, love and respect offer to my whole family. My beloved husband and my loving children, Dao Gia Khanh and Dao Vu Khanh Chi have been supporting me through all the difficult times during this long journey. Without their infinite love, inspiration and patience, this work could never been accomplished. Their love, sacrifice, and patience will be the constant source of motivation throughout my life.

Contents

List of Publications

Journal Papers/Published

J-1. **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.

J-2. **Ly Vu**, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang and E. Dutkiewicz, "Deep Generative Learning Models for Cloud Intrusion Detection Systems," in *IEEE Transactions on Cybernetics*, doi: 10.1109/TCYB.2022.3163811.

J-3. **Ly Vu**, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang and E. Dutkiewicz," Deep Transfer Learning for IoT Attack Detection," in *IEEE Access*, PP. 1-1. 10.1109/ACCESS.2020.3000476.

Conference Papers/Published

C-1. **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". In: *IEEE International Conference on Communications (ICC)*, pp. 1–6 (2019).

C-2. **Ly Vu**, Hoang van Thuy, Quang Uy Nguyen, Nguyen Ngoc Tran, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz, "Time Series Analysis for Encrypted Traffic Classification: A Deep Learning Approach," in *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*, Bangkok, Thailand, 2018, pp. 121-126, doi: 10.1109/ISCIT.2018.8587975..

Journal/Under Review

B-1. **Ly Vu**, T. P. Tran, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang and E. Dutkiewicz, "A Robust Federated Learning System Against Poisoning Attacks in IoT Networks," on revision after submitting to *IEEE Internet Of Things Journal*, July, 2023.

List of Figures

List of Tables

Abbreviation

AAE	Adversarial AutoEncoder
ACGAN	Auxiliary Classifier Generative Adversarial Network
ACK	Acknowledgment
AD	Anomaly Detection
AE	AutoEncoder
AUC	Area Under the Receiver Operating Characteristics Curve
CDAAE	Conditional Denosing Adversarial
CNN	Convolutional Neural Network
CTU	Czech Technical University
CVAE	Conditional Variational AutoEncoder
DAAE	Denosing Adversarial AutoEncoder
DAE	Denoising AutoEncoder
DBN	Deep Beleif Network
DDoS	Distributed Deny of Service
De	Decoder
Di	Discriminator
DT	Decision Tree
DTL	Deep Transfer Learning
En	Encoder
FL	Federated Learning
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GAN	Generative Adversarial Network
Ge	Generator

IoT	Internet of Things
IP	Internet Protocol
KL	Kullback-Leibler
KNN	K-nearest Neighbor
LR	Linear Regression
MAE	Multi-Distribution AutoEncoder
MDAE	Multi-Distribution Denoising AutoEncoder
MMD	Maximum Mean Discrepancy
MVAE	Multi-Distribution Variational AutoEncoder
NAD	Network Attack Detection
NCT	Nearest CenTroid
PCT	PerCepTron
R2L	Remote to Login
RE	Reconstruction Error
RF	Random Forest
RG	Regularization Phase
RP	Reconstruction Phase
ReLU	Rectified Linear Unit
SAAE	Supervised Adversarial AutoEncoder
SAE	Shrink AutoEncoder
SDAE	Shrink Denoising AutoEncoder
SKL-AE	Deep Transfer Learning method using the KL metric and transferring task is executed on the AE's bottleneck layer
SMD-AE	Deep Transfer Learning method using the MMD metric and transferring task is executed on the AE's bottleneck layer
SMD-AE	Deep Transfer Learning method using the MMD metric and transferring task is executed on the encoding layers of AE
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
SYN	Synchronize
TCP	Transmission Control Protocol

TL	Transfer Learning
TN	True Negative
TP	True Positive
TPR	True Positive Rate
U2L	User to Login
UDP	User Datagram Protocol
VAE	Variational AutoEncoder

Chapter 1

INTRODUCTION AND LITERATURE REVIEW

1.1 Motivation

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 [81]. The data generated from these network devices often contains sensitive information of users, such as passwords, phone numbers, photos and locations, which usually attracts hackers [52]. 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 or anomalies [121]. Detecting anomalies in the networks plays a crucial role to protect network devices and service against network attacks [42, 15, 20, 52, 58, 66, 137].

An anomaly detection (AD) system monitors the network traffic to identify abnormal activities in the network environments such as computer networks, clouds, and Internet of Things (IoT). There are three popular approaches for analyzing network traffic to detect anomalies [135], i.e., knowledge/signature-based methods, statistic-based methods, and machine learning (ML)-based methods. First, to detect anomalies, the knowledge-based methods define anomalies, i.e., network attack rules or signatures of abnormal behaviors. Thus, the network traffic, which is matched with these signatures, is identified as an anomaly. The knowledge-based methods can detect attacks robustly in a short time. However, they need high-quality prior

knowledge of anomalies. Moreover, they are unable to detect unknown attacks.

Second, the statistic-based methods define an anomaly score to distinguish between normal and abnormal network traffic. The anomaly score is calculated by statistical methods on the currently observed network traffic data. If the score is more significant than the anomaly score, it will raise an alarm of anomaly [135]. Examples of these statistical methods are information entropy, conditional entropy, and information gain [128]. These methods explore network traffic distributions by capturing the essential features of network traffic data. Then, the distribution is compared with the predefined distribution of normal traffic to detect anomalies.

Third, ML-based methods, especially deep learning-based methods, for AD have received increasing attention in the research community due to their outstanding advantages [40, 101, 104, 115, 137]. These methods build ML models to detect anomalies using training datasets. Depending on the availability of data labels, ML-based methods for AD can be categorized into three main approaches: supervised learning, semi-supervised learning, and unsupervised learning [122].

Although ML, especially deep learning, has achieved remarkable success in AD, there are still 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 ML 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 ML algorithms are often biassed and inaccurate. Third, in 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 not function well on the anomaly detection tasks 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. Fourth, transferring training data of the AD problem through the network faces the data privacy problem. Due to the privacy issue while collecting data to a centralized server for training, the ML-based AD system is often trained with distributed data. However, this distributed nature of the training process is particularly vulnerable to data poisoning attacks that can severely compromise the performance of the AD system.

Given the above, first, this thesis aims to design novel deep neural models to deal with new/unknown attacks. The previous approaches usually utilize the unsupervised and semi-supervised learning methods to detect unknown attacks. However, they may not be as proficient as supervised approaches in identifying previously known attacks. Consequently, the thesis's objective is to propose a novel supervised learning approach that effectively addresses both known and unknown attacks. Second, to address the issue of imbalanced data, the thesis employs generative models to synthesize additional samples for the minor classes that have fewer data samples in the training dataset. The focus is on generating challenging data samples that are difficult for ML algorithms to classify correctly. This approach helps to overcome the biases inherent in the original imbalanced dataset and improve the ML model's ability to learn robust representations. Third, the thesis tackles the challenge of limited labeled data by designing a novel deep neural network (DNN) architecture that can learn effectively from both labeled and unlabeled data. This approach allows the DNN model to leverage the wealth of available unlabeled data to enhance its performance, even when labeled samples are scarce. Fourth, the thesis introduces a federated learning (FL) scheme built upon a novel DNN design. This approach

aims to address the vulnerability to data poisoning attacks that can occur during the training process. By leveraging FL, DNN is able to learn in a distributed manner while maintaining robustness to potential malicious data contributions, strengthening the overall security of the training pipeline.

1.2 Literature Review and Contributions

1.2.1 Unknown Anomaly Challenge and Learning Latent Representation Approach

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 or anomalies [121]. Identifying anomalies in the IoT environment with a massive number of devices and services would be a challenging, especially with the fast and continuous evolution of anomalies [15, 20, 52, 58, 66, 137]. Compared with known anomalies, unknown anomalies are the more dangerous but more difficult to detect since these anomalies could surpass most of the advanced security techniques and cause serious devastation to network systems [119, 121]. Moreover, anomaly data is very heterogeneous and complex, leading to inaccurate AD systems. Recently, deep learning-based AD has received more considerable attention from researchers and industry [78, 106, 121, 125, 137, 145] to learn the representation of network anomaly data.

AD methods can be categorized into signature-based and semantic-based methods [101, 53, 41]. The signature-based methods rely on observing frequently occurring strings or token sequences from anomalous traffic [20, 15, 66, 58]. To detect anomaly traffic, Zhang et al. [20] carried out a deep packet inspection to find anomaly signatures in the network traffic. They also proposed a lightweight and low-complexity algorithm to prevent DDoS attacks. Their work aims to enable IoT devices (working nodes) to intelligently detect and avoid DDoS attacks. However, this technique is limited by the scarce resources at each *bot*. In [15], Dietz et

al. aimed to proactively block the spreading of anomalies by automatically scanning vulnerable IoT devices and isolate them from the system. The authors of [66] proposed a host-based intrusion detection and mitigation (IoT-IDM) method using Software Defined Network with OpenFlow protocols to address malicious behaviors and block intruders from accessing the IoT devices. When an attack occurs at the network-level, IoT-TDM will generate policies to block and isolate infected hosts. Nevertheless, this technique is not scalable as adding IoT devices would require one to manually tailor protocols. Ceron et al. [58] introduced a solution to handle the traffic generated by IoT malware. This solution uses malware's actions to modify the traffic at the network layer. Generally, the signature-based approaches require prior knowledge of the behaviors of known IoT anomalies. Consequently, these approaches are unable to detect unknown attacks which usually cause more serious consequences than known attacks [121].

The semantic-based approach relies on heuristic methods to analyze the anomalous behaviors. Specifically, features of traffic, e.g., the range of packet lengths, inter-packet arrival times, flow size, duration size are used to classify the benign and attack traffic. To that end, ML based methods prove themselves as effective solutions for anomaly detection [137, 40, 104, 115, 86]. Bahsi et al. [40] used the decision tree and K-nearest neighbor algorithms to identify the Mirai botnet family and the Gafgyt botnet family. Chawathe [104] applied a number of ML algorithms including ZeroR and OneR, rule-based, and tree-based classifiers (J48 and Random Forest) to detect anomalies. However, these approaches are ineffective in detecting new types of botnets. Nomm et al. [115] proposed a solution for identifying anomalies in which the datasets are re-sampled before using the Local Outlier Factor (LOF) and One Class Support Vector Machine (OCSVM) [115] 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 LOF and OCSVM. Omar et

al. [86] proposed a framework combining feature selection methods (feature ranking and clustering-based data partitioning techniques) and classifications for botnet intrusion detection systems. Although this framework and other aforementioned showed a great potential in identifying known botnets, using supervised learning classifiers makes them less effective in detecting unknown botnets.

Recently, deep learning approaches have attracted paramount interest in detecting anomaly in cybersecurity, e.g., [137, 121, 65, 145, 125, 78, 106], in which AEs play pivotal roles, e.g., [137, 121, 94, 53, 85]. Meidan et al. [137] 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. [53] presented an online and real-time unsupervised network anomaly detection algorithm which uses a discrete time-sliding window to continuously update the feature space and an incremental grid clustering. Ibidunmoye et al. [85] estimated an underlying temporal property of the stream via adaptive learning, and then used statistically robust control charts to recognize deviations. However, this approach requires frequent adjustment of the threshold value.

To detect new/unknown attacks, Cao et al. [121] 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 anomaly detection methods in dealing with high-dimension data. Specifically, the regularizer in [121] helps SAE and DVAE learn (in a semi-supervised manner) to project the normal class in a small region at the origin. This is based on an assumption that only normal samples are available for training. They did not use any information of the anomalous class to train the representation models. In many scenarios, however, a certain type of attacks can be collected and labelled. In this case, supervised learning-based methods are usually better than semi-supervised methods in detecting known anomalies. Therefore, the thesis aims to develop a novel latent representation that

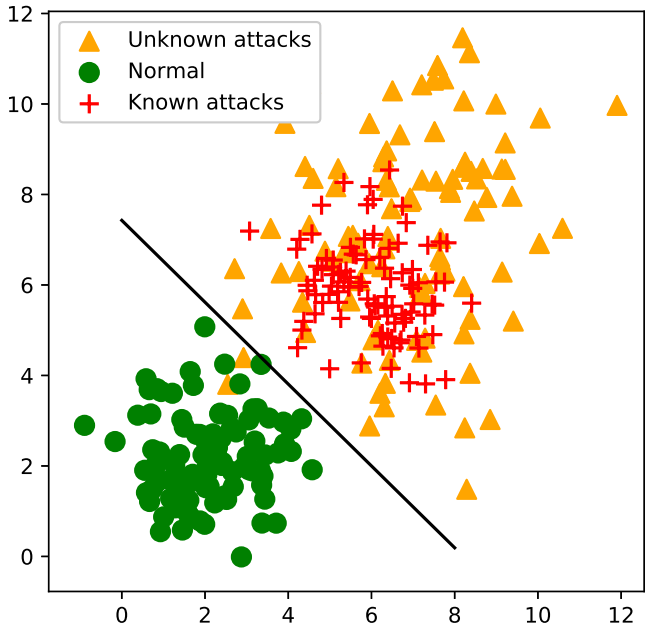


Figure 1.1 : Known and unknown abnormal data samples are separated from normal samples in the latent representation space.

facilitates supervised learning-based anomaly detection 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 [121] cannot do.

In the new representation space, normal data and known anomalies will be forced into two tightly separated regions, called the normal region (green circle points in Fig. 1.1) and anomalous region (red plus points in Fig. 1.1), respectively. We hypothesize that unknown anomalies will appear closer to the anomalous region (yellow triangle points in Fig. 1.1) as they may share 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 AD 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. In this thesis, we propose a novel learning approach that can inherit the strength of supervised learning methods in detecting known anomalies and the ability to identify unknown anomalies of unsupervised methods.

The major contributions of the learning latent representation approach are as follows:

- Introduce a new latent feature representation to enhance the ability to detect unknown network anomalies of supervised learning-based AD methods.
- Propose three novel 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 the 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 compared to learning from the original features or using latent representations produced by other AEs.
- Conduct a thorough analysis of the characteristics of the latent representation in detecting unknown anomalies, 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.

1.2.2 Imbalance Data Challenge and Deep Generative Model Approach

Today, the global cloud industry revenue exceeded $626.4 billion in 2023 and is projected to reach approximately $727.9 billion by the end of 2024, reflecting a year-over-year growth rate of 16.2%. [38]. Nevertheless, the wide adoption of cloud computing also resulted in the cloud systems being very vulnerable to many types of anomalies, i.e., cyber attacks. Among approaches to protecting the cloud systems, Anomaly Detection (AD) systems play a crucial role in early detecting and preventing security attacks [17, 91], especially, DDoS attacks [124]. This is due to the widespread of DDoS attacks and the serious impact they cause on the availability and reputation of cloud providers. Among DDoS attacks, low-rate attacks and application layer-level or gateway-level attacks are among the most dangerous anomalies.

These anomalies often attempt to conceal themselves by mimicking the normal network pattern. For example, DDoS low rate attacks inject low-volume legitimate traffic at a very slow rate and these attacks can be conducted using a number of machines. Since the traffic volume of these anomalies is very low and they often appear to be legitimate, the traditional detection methods may fail to detect them [80]. Application layer attacks present another sophisticated version of DDoS attacks in which the attack traffic attempts to be more similar to normal user traffic and hence pose a serious challenge in how they can be identified. Attackers often use the requests of legitimate users to hide the attacks. As a result, most of the defence techniques at the network layer and application layer fail to detect these attacks [4]. Moreover, these anomalies can be executed using multiple protocols at the application layer, both connection-oriented and connectionless, making them even more dangerous. ML has proven its great potential to detect these types of attacks [33, 97, 92].

However, using ML to build a trustworthy and robust AD on the cloud remains practically challenging. One of the reasons is the rapid development of various and sophisticated cloud attacks. Another reason is the lack of labelled malicious samples to construct an effective ML model. In the cloud environments, the majority of collected traffic samples are normal and only a few samples are intrusions. Subsequently, most of the intrusion datasets on the cloud are imbalanced. When being trained on the skewed datasets, the predictive model developed using conventional ML algorithms could be biased and hence inaccurate. This is because ML algorithms are usually designed to improve the accuracy by reducing the error. Thus, they do not take into account the class distribution/proportion or the balance of classes.

ML 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 [102, 105]. Wang et al. [102] proposed a loss function that independently calculates error for classes and then averages them together. Khan et al. [105] proposed a cost-sensitive deep neural network which 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 randomly under-sampling the majority or over-sampling the minority [5]. However, these techniques have potentially downsides. While under-sampling can lose useful information about the majority class data, over-sampling does not actually increase any information since the exact copies of the minority class are replicated randomly [5]. To overcome this limitation, Synthetic Minority Over-sampling Technique (SMOTE) [82] generates the synthesized minority class by extrapolating and interpolating minority samples from the neighborhood ones. An extension of SMOTE is SMOTE-Support Vector Machine (SMOTE-SVM) [43] that synthesizes samples located in the borderline be-

tween classes by only generating samples for the support vectors of a Support Vector Machine (SVM) model trained on the original dataset.

Ensemble methods (hybrid methods) combine a re-sampling method with a classifier to discover the distribution of the majority and minority class. BalanceCascade [134] develops an ensemble of classifiers to systematically select which majority class samples to under-sample. The number of classifiers are applied in sub-datasets which include majority samples and minority samples with a balanced rate. The majority samples will be removed if they are classified correctly from classifiers. EasyEnsemble [134] learns different aspects of the original majority class in an unsupervised manner. First, a number of 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. Although SMOTE-SVM, BalanceCascade, and EasyEnsemble have been widely adopted and their effectiveness has been well-evidenced [5], the shortcoming of these methods is that the distribution of the original data can be lost [99]. In other words, the generated data samples may have a very different distribution from the original data. Subsequently, the performance of ML algorithms trained on the augmented dataset may be hurt.

Recently, researchers have paid more attention to using deep generative models to address the imbalance problem. For example, Yuan et al. [22] used an Auxiliary Classifier Generative Adversarial Network (ACGAN) to generate synthesized samples to augment the intrusion detection datasets. Xu et al. [133] applied a Conditional Variational AutoEncoder (CVAE) with the log-cosh function to solve the class imbalance for an intrusion detection system (IDS). Yu and Lam [89] introduced a data generation model based on a Supervised Adversarial AutoEncoder (SAAE) for tackling the data imbalance in the Learning to Rank problem. However, in the above researches [22, 133, 89], ACGAN and CVAE are only examined on a small number of IDS datasets while SAAE has not been applied to the IDS problem.

Moreover, the properties of the generated data by ACGAN, CVAE and SAAE have not been well-analyzed.

This thesis proposes two (Adversarial AutoEncoder) AAE-based models for handling imbalanced datasets in the cloud IDSs. The first model is CDAAE that incorporates the label information into both the encoder and the decoder to effectively generate malicious samples for any specific attacks. Moreover, the input to the encoder of CDAAE is a noisy version of the original input allowing it to learn a more robust generator. Thanks to these properties, the synthesized samples of CDAAE are more correlated to the original data distribution than those of the previous approaches, i.e., ACGAN, CVAE and SAAE. The second model, CDAAE-KNN, generates dificult classified samples that lie in the borderline of different classes.

The main contributions for handling imbalance data are as follows:

- We propose two models based on the Adversarial AutoEncoder (AAE) [3] to create synthesized anomaly samples, i.e., malicious samples. The first model is Conditional Denoising Adversarial AutoEncoder (CDAAE) which can generate data samples for any specific attack. The second model is a hybrid between CDAAE and the K-Nearest Neighbor (KNN) algorithm (referred to as CDAAE-KNN) to generate malicious borderline samples that further improve the classification performance.

- We conduct extensive experiments to evaluate the effectiveness of the proposed solutions. The experimental results show that our methods significantly improve the accuracy of ML for AD compared to the state-of-the-art methods [3, 82, 93, 134, 144]. Moreover, CDAAE and CDAAE-KNN also improve the accuracy of ML models in detecting two challenging anomalies, i.e., application layer DDoS and low-bandwidth DDoS.

- We investigate various characteristics of CDAAE and CDAAE-KNN on the

rate of the balanced data, the quality of the synthesized samples and the processing time. This analysis sheds light on the practical applications of the proposed models.

1.2.3 Lack of Label Information Challenge and Deep Transfer Learning Approach

The ML-based methods only perform well under an important assumption, i.e., the distributions of the training data and the predicting data are similar [64]. Nevertheless, in many practical applications, this assumption may not be always the case [112, 57]. Especially, in network security, new types of attacks (e.g., zero-day attacks) can be found daily [121]. As such, the practical IoT data for ML models (in the predicting/online phase) is usually very different from the data 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 [6, 109]. It, thus, limits the practical deployment of ML-based methods in detecting IoT attacks for various scenarios.

Based on using deep learning techniques, TL can be categorized into four groups such as instances-based TL, mapping-based TL, network-based TL, and adversarial-based TL. Instances-based TL approach selects samples from the source domain to supplement into the training set in the target domain. These samples are assigned appropriate weights in the training set. [16] proposed 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, [141] introduced a metric TL framework together with learning a distance between two domains. This framework makes knowledge to transfer between domains more ef-

fectively. An ensemble TL introduced in [130] can leverage samples from the source domain to train on the target domain.

Mapping-based TL aims to map samples from both 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 [31] 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 [73] to transfer the data distribution in different domains and improve previous works. Moreover, the work [69] proposed Wasserstein distance to used as a distance measurement of domains to achieve a better representation space.

Adversarial-based TL refers to use adversarial networks inspired by generative adversarial nets (GAN) [48] to get a representation space that is suitable to both the source domain and target domain. [30] introduced a new loss function of GAN combining with a discriminative model to a TL method. A randomized multi-linear adversarial networks was introduced in [74] to find the multiple feature layers.

Network-based TL reuses the network structure and parameters trained in the source domain for training the target domain. The work [71] reused front-layers of Convolutional Neural Network (CNN) trained on the ImageNet dataset to map images in other datasets to an intermediate image representation. It helps to transfer knowledge to other object recognition tasks with a small training set size. The work [75] introduced an approach to learn adaptive classifiers with a residual function. Several TL approaches based on AEs was introduced in [64, 35, 18]. They used different AE-based models such as AE [35], denoising AE [18], and sparse AE [64]

for some specific applications.

Moreover, transferring higher level of features from the source domain with labels to the target domain without labels help to improve the classifying tasks of the target domain. This approach helps to improve the accuracy of learning tasks on the target domain with the limited samples and no labels.

Our proposed DTL model in this thesis, i.e., MMD-AE, leverages a non-linear mapping, i.e., AE, to improve the performance of IoT attack detection on the target domain. The key idea of our proposed DTL (compared with previous AE-based DTL methods [64, 35]) is that the knowledge of features in *every encoding layers* (instead of the only bottleneck layer in previous works) is transferred to the target domain. This helps to force the latent representation of the target domain similarly to the latent representation of the source domain. The experimental results illustrate the effectiveness of our proposed DTL model on the IoT attack detection task in the target domain.

Recently, Deep Transfer Learning (DTL) techniques have been used to handle the above issues of ML methods where training data from a source domain and test data from a target domain are drawn from different distributions. A DTL model attempts to reduce the distribution divergence between the source domain and the target domain [141]. As a result, the trained knowledge of a learning task (e.g., classification) on the source domain can be used to support the learning task on the similar target domain [141, 112, 61, 19]. Gou et al. [110] applied an instance-based DTL approach in network intrusion detection that requires label information from the target domain. Zhao et al.[59] proposed the feature-based DTL technique to project the source and the target domain into the latent subspace via linear transformations, i.e., Principal Component Analysis (PCA) for network attack detection. However, PCA is a linear mapping technique that only works well

with a simple data feature set [39].

This thesis aims to leverage a non-linear mapping, i.e., AE, to improve the performance of AD on the target domain. The key idea of our proposed DTL (compared with previous AE-based DTL methods [64, 35]) is that the knowledge of features in *every encoding layers* (instead of the only bottleneck layer in previous works) is transferred to the target domain. This helps to force the latent representation of the target domain similarly to the latent representation of the source domain.

Given the above, we propose a novel DTL approach based on AutoEncoder (AE) to enable further applications of ML 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 labelled samples (in the source domain) and unlabelled 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 is trained with the labelled data while AE_2 is trained with the unlabelled 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 unlabelled IoT data in the target domain.

The major contributions of the DTL approach are as follows:

- We propose a novel DTL model based on AEs, i.e., MMD-AE, that allows to transfer of knowledge, i.e., labelled information, from the source domain to the target domain. This model helps to lessen the problem of "lack of label information" in collected network traffic datasets from IoT devices.

- We introduce 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 IoT anomaly detection systems.

- We experiment with our proposed method using nine IoT anomaly datasets and compare its performance with the canonical deep learning model and the state-of-the-art DTL models [64, 35]. The experimental results demonstrate the advantage of our proposed model against the other tested methods.

1.2.4 Data Privacy Challenge and Federated Learning Approach

In anomaly detection for IoT networks, AutoEncoder (AE) is considered to be one of the most effective approaches [121, 137, 79, 9, 114]. The AE architecture allows to transform of the raw IoT data into a new data representation that better exposes the underlying characteristics of the input data. Thus, it facilitates the downstream classifiers to separate attack samples from normal samples.

However, collecting and centring data from IoT networks to train an effective AE is a challenging task [121, 137, 9]. This is due to user data privacy concerns [90]. IoT users tend to avoid divulging data that could compromise their privacy or privileged information of their systems. Subsequently, Federated Learning (FL) has recently emerged as the best solution for developing anomaly detection for IoT networks. FL does not require IoT devices or networks to share their local data with the centralized server. Instead, distributed nodes (also referred to as clients) only need to share their local gradients with the centralized server that in turn aggregates into the global gradient and shares it with the local nodes [36, 118, 13, 126, 11, 24, 140].

In order to train a robust FL model, one needs to assume that the training processes of all clients are honestly conducted. However, such an assumption is not always realistic [37, 139, 111]. For example, the attackers may take control of a number of IoT devices in the FL system, i.e., genuine clients, and then inject the poisoning data during the local training phase [139, 36]. Two common types of data

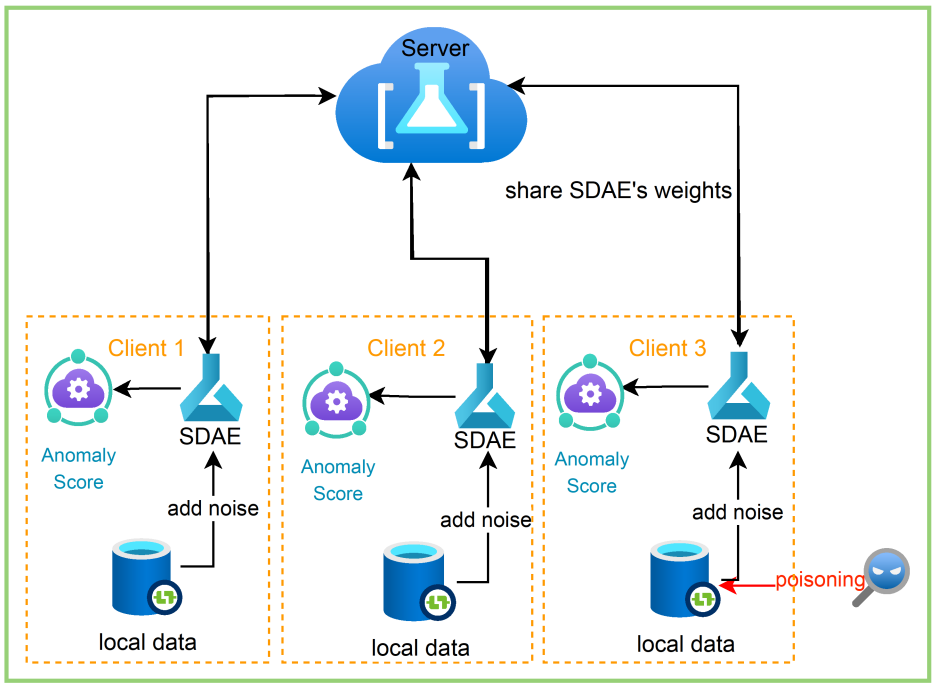


Figure 1.2 : The architecture of proposed FL model. The local data in each client is added with noise and then used to train a novel autoencoder model (SDAE). SDAE attempts to transform the input data into a compact region at the origin. Thus, it helps the aggregation of the weights at the server side more effectively. Moreover, SDAE is trained with noisy data, and thus, FL-SDAE is robust against data poisoning attacks in which attackers try to inject poisoning data into the local data of clients (e.g., Client 3).

poisoning attacks are dirty label attacks and clean label attacks [37]. *Dirty label attacks* are based on flipping the label of the training data while the clean label attacks add noise to the training data. Data poisoning attacks first degrade the effectiveness of the impacted clients and consequently deteriorate the performance of the whole FL system. Therefore, building a robust FL system for IoT anomaly detection that mitigates data poisoning attacks is of great desirable. This is even more critical given the vulnerabilities of IoT devices that are easily to be tampered in a large number.

In the data poisoning attacks, the attackers may tamper the subset training data of clients by flipping or adding noisy data [98]. Zhang et al. [56] indicated that the data poisoning attacks including label flipping and backdoor can reduce the effectiveness of almost FL model. These studies indicated that the poisoning attacks are dangerous factors for an FL scheme. Thus, to protect the integrity of the FL scheme, the ML models trained on clients need to be robust against poisoning attacks.

To handle the data poisoning attacks of FL, we propose a FL system based on Shrink Denoising AutoEncoder (illustrated in Fig. 1.2), namely FL-SDAE, for IoT anomaly detection that is robust against the data poisoning attacks. First, the Shrink Denoising AutoEncoder (SDAE) aims to transform benign data in different IoT devices into the same latent representation space. This makes the aggregation of clients's weights into the global weights more effective. Thus, the proposed model, i.e., FL-SDAE, enhances the accuracy in detecting anomalies when data poisoning attacks are not conducted. Second, the SDAE model is trained with noisy data that mimics the poisoning data. This makes the FL-SDAE system robust against data poisoning attacks.

The main contributions of handling data poisoning attacks for FL are as follows:

- We propose the SDAE model for the FL-based anomaly detection system that is trained on noisy data to transform the raw data of different clients into the same space.

- We design a new FL system based on SDAE that is capable of mitigating poisoning attacks.

- We conduct extensive experiments to evaluate the effectiveness of FL-SDAE for the anomaly detection problem on five datasets, i.e., N-BaIoT, CICIDS, NSL-KDD, Spambase, and CTU13-08.

1.3 Thesis Organization

The rest of the thesis is organised as follows:

- Chapter 2 presents the fundamental background of the AD problem and deep neural networks. In particular, in section 2.1, we describe main deep neural network (DNN) models that are the fundamental of our proposed solutions Section 2.2 presents the fundamentals of transfer learning techniques. Next, the fundamental of federated learning is described in Section 2.3. Section 2.4 presents common evaluation metrics used in the thesis. The experimental datasets of the proposed methods are presented in Section 2.5. Finally, the conclusion is drawn in Section 2.6.

- Chapter 3 proposes a novel representation learning method to enhance the accuracy of deep learning in detecting network attacks, especially unknown attacks. In particular, the proposed models are presented in Section 3.1. The description of applying the proposed model for the network AD detection is discussed in Section 3.2. Section 3.3 presents the experimental settings. Section 3.4 discusses and analyzes results obtained from the proposed models. Finally, in Section 3.5, we conclude the contributions of this work.

- Chapter 4 presents new generative deep neural network models for handling the imbalance of network traffic datasets. Particularly, the proposed models for handling the imbalanced data are discussed in Section 4.1. In Section 4.2, we describe the datasets used to evaluate our proposed solutions. The performance of CDAAE and CDAAE-KNN is compared with other tested models in Section 4.3. Section 4.4 analyses properties of the tested models. Finally, conclusions are drawn in Section 4.5.

- Chapter 5 proposes a novel DTL for the anomaly detection problem that can adapt the knowledge of label information of a domain to a related domain. In particular, the proposed DTL model is presented in Section 5.1. Section 5.2 discusses the experiment settings and Section 5.3 provides detailed analysis and discussion of experimental results for the anomaly detection problem. Finally, Section 5.4 outlines the contributions of this work.

- Chapter 6 presents a new FL framework to solve the data poisoning attack problem for the AD problem. In particular, the proposed FL models are presented in detail in Section 6.1. Section 6.2 discusses the experimental settings for FL models. The results and discussions of experiments are represented in Section 6.3. Finally, conclusions are summarized in Section 6.4.

- Chapter 7 outlines the conclusion and draws potential research directions of this thesis.

Chapter 2

FUNDAMENTAL BACKGROUND

This chapter presents the theoretical backgrounds related to the proposed models of this thesis. In section 2.1, we describe deep neural network (DNN) models that are the fundamental of our proposed solutions. Section 2.2 presents fundamental to transfer learning techniques. Next, the fundamentals of federated learning is described in Section 2.3. Section 2.4 provides common evaluation metrics used in the thesis. The experimental datasets of the proposed methods are discussed in Section 2.5. Finally, the conclusion is presented in Section 2.6.

2.1 Deep Neural Network

Deep neural networks (DNNs) provide a robust framework for supervised learning. A DNN 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 giving large models and large labelled training data samples [46]. In this section, we will present the mathematical backgrounds of DNN models, i.e., AutoEncoder (AE), Denoising AutoEncoder (DAE), Generative Adversarial Network (GAN), Adversarial AutoEncoder (AAE), that will be used to develop our proposed solutions in the next chapters. Table 2.1 presents general notations used to define in a deep neural network.

Notation	Meaning	Shape
n	number of data samples in a dataset	1
d	number of features of a data sample	1
$\mathbf{x}$	input dataset	$n \times d$
x^i	one data sample in a dataset	$1 \times d$
$\mathbf{W}$	weight matrix	depending on number of layer and the number of neurons
$\mathbf{b}$	bias vector	number of layers

Table 2.1 : General notations of a deep neural network.

2.1.1 AutoEncoder

An AE is a neural network trained to copy the network's input to its output [12]. This network has two parts, i.e., encoder and decoder (as shown in Fig. 2.1 (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. 2.1). 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. 2.2).

$$z^i = q_\phi(x^i) = a_e(\mathbf{W}x^i + \mathbf{b}), \tag{2.1}$$

$$\hat{x}^i = p_\theta(z^i) = a_d(\mathbf{W}'z^i + \mathbf{b}'), \tag{2.2}$$

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 [121] as in Eq. 2.3.

$$\ell_{AE}(\mathbf{x}, \phi, \theta) = \frac{1}{n} \sum_{i=0}^{n} \left(x^i - \hat{x}^i\right)^2. \tag{2.3}$$

2.1.2 Variational AutoEncoder

A Variational AutoEncoder (VAE) [28] is a variant of an AE that also consists of two parts: encoder and decoder (Fig. 2.1 (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, i.e., $\ell_{VAE}(x^i, \theta, \phi)$, for a datapoint x^i includes two terms as follows:

$$\begin{aligned} \ell_{VAE}(x^i, \theta, \phi) = &- \mathbf{E}_{q_\phi(\mathbf{z}|x^i)} \left[log\, p_\theta(x^i|\mathbf{z})\right] \\ &+ D_{KL}(q_\phi(\mathbf{z}|x^i)||p(\mathbf{z})). \end{aligned} \tag{2.4}$$

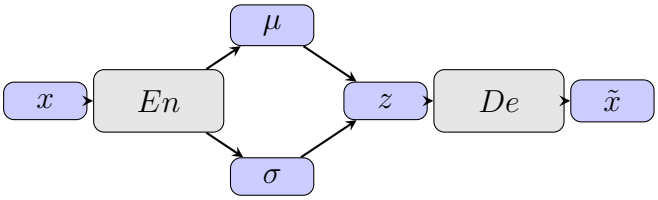
In Eq. 2.4, the first term is the expected negative log-likelihood of the i-th data sample. This term is also called the reconstruction error (RE) of VAE since it forces the decoder to learn to reconstruct the input data. The second term is the Kullback-Leibler (KL) divergence between the encoder's distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the expected distribution $p(\mathbf{z})$. This divergence measures how close q is to p [28]. In the VAE, $p(\mathbf{z})$ is specified as a standard Gaussian 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 a standard Gaussian 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:

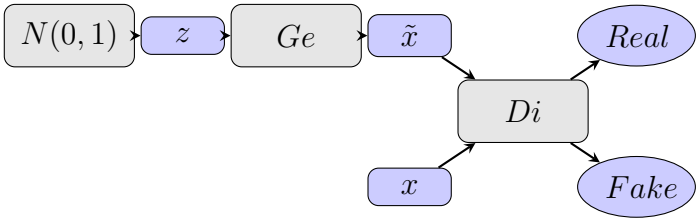
$$\begin{aligned}\ell_{VAE}(x^i, \theta, \phi) = &- \frac{1}{K}\sum_{k=1}^{K} \log p_\theta(x^i|z^{i,k}) \\ &+ D_{KL}(q_\phi(\mathbf{z}|x^i)||p(\mathbf{z})),\end{aligned} \tag{2.5}$$

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.

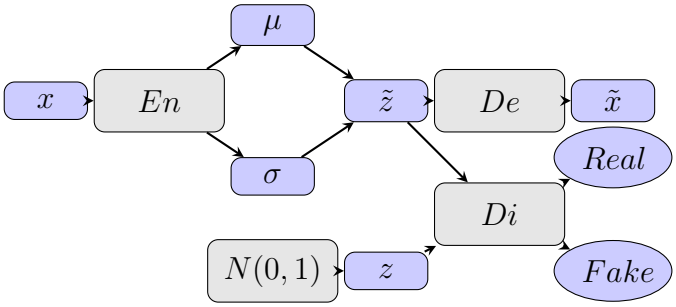
(a) AutoEncoder (AE).



(b) Variational AutoEncoder (VAE).



(c) Generative Adversarial Network (GAN).



(d) Adversarial AutoEncoder (AAE).

Figure 2.1 : Structure of generative models (a) AE, (b) VAE, (c) GAN, and (d) AAE.

2.1.3 Denoising AutoEncoder

Denoising AutoEncoder (DAE) is a regularized AE that aims to reconstruct the original input from a noised version of the input [87]. Thus, DAE can capture the true distribution of the input instead of learning the identity [46, 10]. The additive isotropic Gaussian noise is the most common one for adding noise to the input data [87].

Let's an additive isotropic Gaussian noise $C(\tilde{\mathbf{x}}|\mathbf{x})$ be the conditional distribution over a corrupted sample set $\tilde{\mathbf{x}}$, given a data sample set $\mathbf{x}$. Let $\mathbf{x_{noise}}$ to be the noise data sample set drawn from the Gaussian 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:

$$C(\tilde{\mathbf{x}}|\mathbf{x}) = \mathbf{x} + \mathbf{x_{noise}}. \tag{2.6}$$

Let's $\tilde{x}^i$ to be the corrupted version of an input data sample x^i obtained from $C(\tilde{\mathbf{x}}|\mathbf{x})$. Note that the corruption process is performed stochastically on the original input and each data sample x^i is calculated separately. Based on the loss function of AE, the loss function of DAE can be written as follows:

$$\ell_{DAE}\left(\mathbf{x}, \tilde{\mathbf{x}}, \phi, \theta\right) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - p_\theta(q_\phi(\tilde{x}^i))\right)^2, \tag{2.7}$$

where q_ϕ and p_θ are the encoder and decoder parts of DAE, respectively. n is the number of data samples in a dataset.

2.1.4 Generative Adversarial Network

A Generative Adversarial Network (GAN) [48] has two neural networks which are trained in an opposite way (Fig. 2.1(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 the difference between $\tilde{\mathbf{x}}$ and $\mathbf{x}$.

$$L_{GAN} = E_{\mathbf{x}}[\log Di(\mathbf{x})] + E_{\mathbf{z}}[\log(1 - Di(Ge(\mathbf{z})))]. \tag{2.8}$$

The loss function of GAN is presented in Eq. 2.8 in which Di is the discriminator used for predicting its input as a real or fake data sample. $Ge(\mathbf{z})$ is the output of Ge when given noise $\mathbf{z}$. $E_{\mathbf{x}}$ and $E_{\mathbf{z}}$ are the expected value (average value) of overall real and fake data samples, respectively. Di is trained to maximize this loss function, 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.

2.1.5 Adversarial AutoEncoder

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 [3]. As illustrated in Fig. 2.1(d), training AAE has two phases, i.e., reconstruction and regularization. In the reconstruction phase (RP), the latent sample $\tilde{\mathbf{z}}$ is drawn from the generator Ge. The 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 by the difference between $\mathbf{x}$ and $\tilde{\mathbf{x}}$ as in Eq. 2.9.

$$L_{RP} = -\,\mathbb{E}_{\mathbf{x}}\left[\log p(\mathbf{x}|\tilde{\mathbf{z}})\right]. \tag{2.9}$$

In the regularization phase (RG), the discriminator Di 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

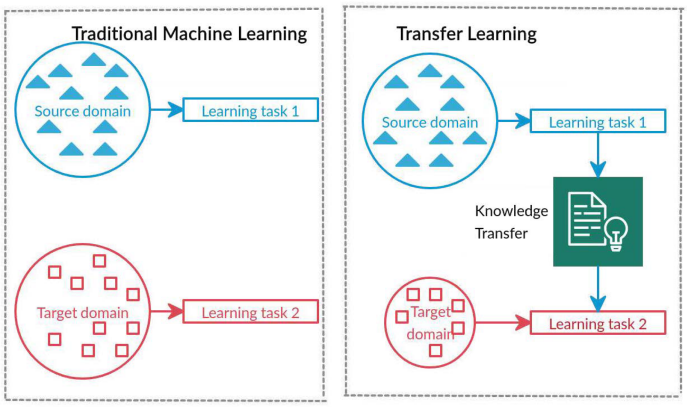


Figure 2.2 : Traditional machine learning vs. transfer learning.

the fake sample, i.e., $\tilde{\mathbf{z}}$, as similar as the real sample, i.e., $\mathbf{z}$, by minimizing the second term in Eq. 2.10. The discriminator then attempts to distinguish between $\tilde{\mathbf{z}}$ and $\mathbf{z}$ by maximizing this equation. The generator is also the encoder portion of the AE. Therefore, the training process in the regularization phase is the same as that of GAN.

$$L_{RG} = \mathbb{E}_{\mathbf{z}} \left[\log Di(\mathbf{z})\right] + \mathbb{E}_{\mathbf{x}} \left[\log(1 - Di(En(\mathbf{x})))\right]. \tag{2.10}$$

2.2 Transfer Learning

2.2.1 Definition of Transfer Learning

Transfer Learning (TL) refers to the situation what has been learned in one learning task is exploited to improve generalization in another learning task [46]. Fig. 2.2 compares traditional machine learning and TL. In traditional machine learning (ML), the datasets and training processes are separated for different learning tasks. Thus, no knowledge is retained/accumulated nor transferred from one ML model to another. In TL, the knowledge (i.e., weights) from previously trained models of a source domain is used for training models of a target domain. Moreover, TL

can even handle the lack of data problem in the target domain.

In our studies, the TL method defines 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 data samples are given, i.e., the source data sample $D_S = (X_S, Y_S) = (x_S^i, y_S^i)_{i=1}^{n_S}$ and the target data 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 of the target domain is built based on the source domain D_S with label information Y_S and the target domain D_T without label information to do the classification task of the target domain. The TL systems based on DNN architectures are Deep Transfer Learning (DTL) systems.

2.2.2 Maximum mean discrepancy

Similar to KL divergence [35], Maximum mean discrepancy (MMD) is used to estimate the discrepancy between two distributions. However, MMD is more flexible than KL by the ability to estimate the nonparametric. distance [8]. Moreover, MMD can avoid computing the intermediate density of the distributions. The definition of MMD can be formulated as Eq. 2.11.

$$MMD(X_S, X_T) = |\frac{1}{n_S}\sum_{i=1}^{n_S}\xi(x_S^i) - \frac{1}{n_T}\sum_{i=1}^{n_T}\xi(x_T^i)|, \tag{2.11}$$

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.

2.3 Federated Learning

Federated Learning (FL) is a promising distributed machine learning paradigm for privacy preservation [14]. The demonstration of FL is shown in Fig. 2.3. A shared global model is trained from a federation of participating devices under the control

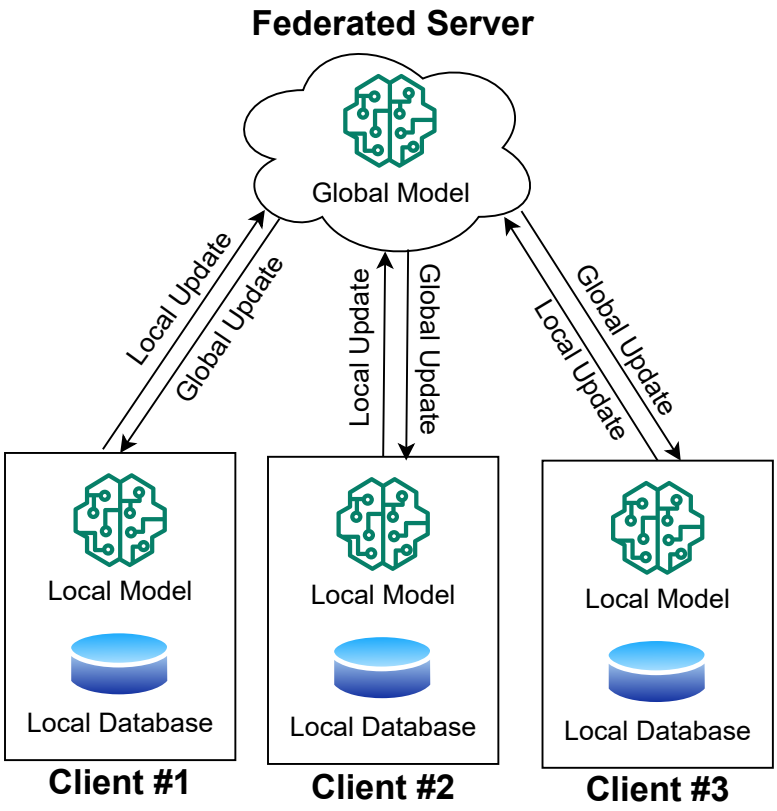


Figure 2.3 : Federated learning (FL) framework.

of a central server. FL enables mobile devices to collaborate on training a shared model while preserving all training data on the device, ensuring the protection of personal data and minimizing system latency.

he FL operation process consists of three phases, i.e., initialization, local model training and updating, and global model aggregation and updating [127]. The server broadcasts an initialized global model to all the clients during the initialization phase. Following that, each client trains the model and updates the model's weights locally by using its local dataset. At the end of this phase, the clients send the updated model's weights to the server. The server aggregates the local models and then broadcasts the updated global model back to the clients again during the aggregation phase. To aggregate the models of clients, the averaging model operation has been widely used in the literature [140, 138].

2.4 Evaluation Metrics

In this section, we present two evaluation metrics that will be used to evaluate the performance of our proposed models.

2.4.1 AUC Score

The the Area Under the Receiver Operating Characteristics Curve (AUC) score is the main performance evaluation metric used to measure the effectiveness of our proposed models. This score is calculated based on the Sensitivity, False Positive Rate (FPR) and Specificity. The Sensitivity score measures how many actual positive observations are predicted correctly (as in Eq. 2.12). FPR is the proportion of real negative cases that are incorrectly predicted (as in Eq. 2.13), and Specificity is the rate of real negative cases that are correctly predicted (as in Eq. 2.14).

$$Sensitivity = TPR = \frac{TP}{TP + FN}, \tag{2.12}$$

$$FPR = \frac{FP}{TN + FP}, \tag{2.13}$$

$$Specificity = \frac{TN}{TN + FP}, \tag{2.14}$$

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 of imbalanced datasets [23].

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 ($FPR =$

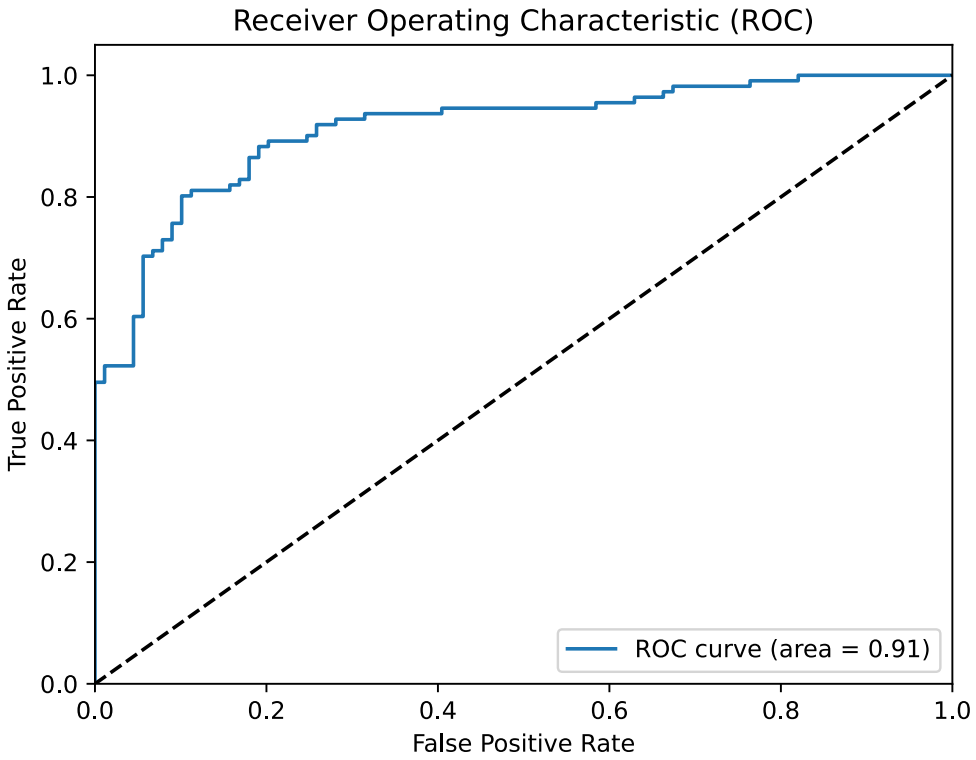


Figure 2.4 : Example of ROC curve.

100%, $TPR = 0$). The space under the ROC curve is represented as the AUC score. As illustrated in Fig. 2.4, the space under the ROC curve is 0.91. It measures the average quality of the classification model at different thresholds. The value of the AUC score for a perfect classifier is 1.0.

2.4.2 Geometric Mean Score

Geometric mean (GEO) is the root of the product of class-wise Sensitivity and Specificity [77]. This metric tries to maximize the accuracy of each class while keeping this accuracy balanced. For binary classification, GEO is calculated as in Eq. 2.15). The best value is 1, and the worst value is 0. If at least one class is unrecognized by the classifier, the GEO score value is 0.

$$GEO = \sqrt{Sensitivity \times Specificity}, \tag{2.15}$$

where *Sensitivity* and *Specificity* are calculated by Eq. 2.12) and Eq. 2.14, respectively.

2.4.3 False Alarm Rate and Miss Detection Rate

False Alarm Rate (FAR) and Miss Detection Rate (MDR) are metrics which focus on the false classification results. FAR is defined as the number of false alarms of negative samples per the total number of real negative data samples. It is calculated by Eq. 2.16. MDR is defined as the number of miss detection of positive samples per total of real positive samples as in Eq. 2.17.

$$FAR = \frac{FP}{FP + TN}, \tag{2.16}$$

$$MDR = \frac{FN}{FN + TP}, \tag{2.17}$$

where TP, FP, TN, and FN are defined in Section 2.4.1.

2.4.4 Complexity

In general, model complexity can be defined by the number of trainable parameters*: the more trainable parameters a model has, the more complex the model is [34, 76]. As discussed in [34], given equivalent accuracy, one neural network architecture with fewer parameters has 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 DNN-based models to compare their model sizes or complexity.

2.5 Exprimental Datasets

2.5.1 NBIoT dataset

This dataset was introduced by Y. Meidan et al. [137]. The data samples of this dataset were collected from nine commercial IoT devices with two most well-known IoT-based botnet families, i.e., Mirai and BASHLITE (Gafgyt). Thus, this dataset consists of nine IoT datasets corresponding to nine IoT devices. Each of the botnet family contains five different IoT attacks. Among these IoT attack datasets, there are three IoT attack 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 the IoT attack datasets consist of both Mirai and BASHLITE botnets which include ten types of DDoS attacks. Each data sample has 115 attributes which are categorized into three groups: stream aggregation, time-frame, and statistics attributes. In Chapter 3, we aim to evaluate the ability to detect unknown anomalies, thus, we divide the training and testing datasets as presented in Table 2.2 where the attacks

*The parameters are updated in the training process of a neural network.

in the testing sets are not represented in the training sets. The experiments of other chapters using these IoT attack datasets will be described later in the chapters.

Table 2.2 : The nine IoT datasets.

Dataset	**Device Name**	**Traing Attacks**	**Traing 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

2.5.2 NSL-KDD dataset

NSL-KDD is an anomaly dataset [84] which is used to solve intrinsic problems of the KDD'99 dataset. The NSL-KDD dataset contains $148,517$ records in total, which is divided into the training set ($125,973$ data samples) and the testing set ($22,544$ data samples). Each sample has 41 features and is labelled as either a type of attack or normal. The training set contains 24 specific attack types, and the testing set has 14 new types of attacks that are not presented in the training set. The simulated attack samples belong to one of four categories, i.e., DOS, R2L, U2R, and Probing. Three categorical features, i.e., *protocol* type, *service*, and *flag*, are preprocessed by one-hot-encoding which increases the number of features to 122.

Table 2.3 : Number of samples of the network IDS datasets.

Datasets	Classes	# samples	# synthesized samples
NSL-KDD	Normal	67373	0
	DoS	45927	21446
	U2L	52	520
	R2L	995	9520
	Probing	11656	55717
UNSW-NB15	Normal	37000	0
	Generic	18871	18129
	Exploits	11132	25868
	Fuzzers	6062	30938
	DoS	4089	32911
	Reconnaissance	3496	34960
	Analysis	677	6770
	Backdoor	583	5830
	Shellcode	378	3780
	Worms	44	440

The number of data samples of each class is presented in Table 2.3.

2.5.3 UNSW-NB15 dataset

The dataset is created by utilizing the synthetic environment as the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre of Cyber Security (ACCS) [83]. The number of data samples in the training set and the testing set are $175,341$ records and $82,332$ records, respectively. There are nine categories of attacks in UNSW-NB15, which are Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. Each data sample has 49 features which have been generated by using the Argus, Bro-IDS tools. The categorical attributes, such as *protocol*, *service*, and *state*, are pre-processed by one-hot encoding which increases the number of features to 196. The number of data samples of this dataset are presented in Table 2.3.

Table 2.4 : Number of samples of the Cloud IDS datasets.

Attack types	Datasets	# normal samples	# anomaly samples	Rate	# synthesized samples
TCP Land	Cloud IDS	3077	34	0.011	340
PingOfDeadth	Cloud IDS	3077	48	0.016	480
Slowloris	Cloud IDS	3077	293	0.095	2930
Slowloris	CIC-IDS 17	219068	580	0.003	5800
SlowHttp	CIC-IDS 17	219068	550	0.003	5500
BruteForce	CIC-IDS 17	219068	1384	0.006	13840
SSH-Patator	CIC-IDS 17	219068	590	0.003	5900
Botnet	CIC-IDS 17	219068	195	0.0009	1950

2.5.4 Cloud datasets

The cloud IDS datasets are published by Kumar et al. [95] as a part of developing their DDoS attack on the cloud environment. They stimulated the cloud environment and used an open-source platform to launch 15 normal Virtual machines (VMs), 1 target VM and 3 malicious VMs. The extracted datasets contain 24 time-based traffic flow features. The CIC-IDS 2017 dataset consists of benign and the most up-to-date common attacks, which are similar to the true real-world data. This dataset contains many types of DDoS attacks including low-rate DDoS attacks and application layer DDoS attacks [50]. The number of features for each data sample is 80. The number of original samples and the synthesized samples in each class of the cloud IDS datasets and the CIC-IDS 2017 datasets are presented in Table 2.4. The cloud IDS datasets are divided into training and testing sets with a ratio of 7/3. For the CIC-IDS 2017 datasets, the training and testing sets are used as in the original datasets.

2.6 Conclusion

This chapter presents the fundamental backgrounds related to the proposed models of this thesis. The DNN models discussed in this chapter are AE, DAE, VAE,

GAN, AAE with their architectures, loss functions, and applications. Moreover, the TL and FL frameworks are also analyzed to handle problems related to the lack of training data and the security of training processes. Besides, we also define the common evaluation metrics and the experimental datasets to evaluate the proposed solutions for the AD problems. From next chapter, we will clarify our proposed solutions to handle the challenges of the AD problems.

Chapter 3

LEARNING LATENT REPRESENTATION FOR ANOMALY DETECTION

To effectively detect new/unknown anomalies, i.e., 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 rest of this chapter is as follows. The proposed models are presented in Section 3.1. The description of applying the proposed model for the network AD detection is discussed in Section 3.2. Section 3.3 presents the experimental settings. Section 3.4 discusses and analyzes results obtained from the proposed models. Finally, in Section 3.5, we conclude the contributions of this work.

3.1 Representation Learning Models for Anomaly Detection

In this section, we describe the proposed latent representation that facilitates supervised learning-based AD methods in identifying attacks, especially unknown attacks. We then present 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. We introduce 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, we introduce new regularizers to a classical AE and a DAE to form three 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 [121]. Specifically, the regularized AEs in [121] can learn to represent only normal class into a tight region at the origin in a semi-supervised manner.

3.1.1 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, we incorporate 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

*With each class label, the representation space of data is shrinked as a tight region.

class y^i. The loss function of MVAE on x^i can be calculated as follows:

$$\begin{aligned}\ell_{MVAE}(x^i, y^i, \theta, \phi) = &-\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k}, y^i) \\ &+ D_{KL}(q_\phi(z^i|x^i, y^i)||p(z^i|y^i)),\end{aligned} \tag{3.1}$$

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. 2.5. 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, $p(z^i|y^i) = \mathcal{N}(\mu_{y^i}, 1)$[†]. 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. 3.1

[†]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 [121].

is rewritten as follows:

$$\begin{aligned} &D_{KL}(q_\phi(z^i|x^i,y^i)||p_\theta(z^i|y^i)) \\ &\qquad = D_{KL}(\mathcal{N}(\mu^i,(\sigma^i)^2)||\mathcal{N}(\mu_{y^i},1)). \end{aligned} \tag{3.2}$$

Let D, μ_j^i and σ_j^i denote the dimension of z^i, the j-th element of μ^i and σ^i, respectively; $\mu_{y_j^i}$ 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:

$$\begin{aligned} &D_{KL}\left(q_\phi(z|x^i,y^i)\|p_\theta(z|y^i)\right) \\ &\qquad = \frac{1}{2}\sum_{j=1}^{D}\left((\sigma_j^i)^2+(\mu_j^i-\mu_{y_j^i})^2-1-\log((\sigma_j^i)^2)\right). \end{aligned} \tag{3.3}$$

Taking Multi-KL term in Eq. 3.3, the loss function of MVAE in Eq. 3.1 finally is rewritten as follows:

$$\begin{aligned} \ell_{MVAE}(x^i,y^i,\theta,\phi) = &-\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k},y^i) \\ &+\lambda\frac{1}{2}\sum_{j=1}^{D}((\sigma_j^i)^2+(\mu_j^i-\mu_{y_j^i})^2-1-\log((\sigma_j^i)^2)), \end{aligned} \tag{3.4}$$

where λ is a parameter to control the trade-off between two terms in Eq. 3.4 as discussed in [121]. 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.

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.

3.1.2 Multi-distribution AutoEncoder

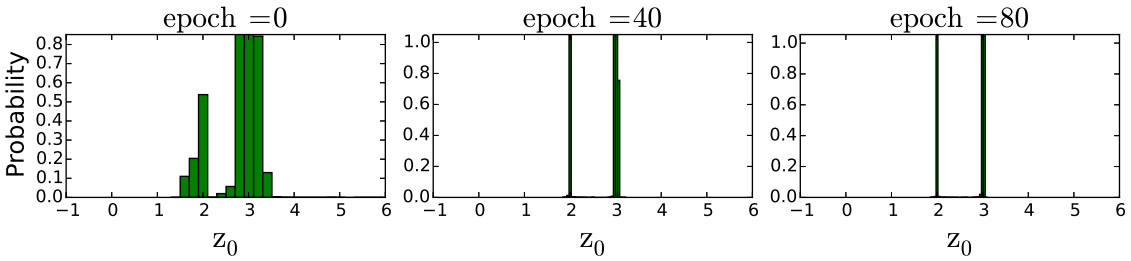


Figure 3.1 : The probability distribution of the latent data ($\mathbf{z}_0$) of MAE at epoch 0, 40 and 80 in the training process.

This subsection describes how to integrate a regularizer to an AE into 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 tight region. Specifically, we incorporate 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. 3.5.

$$\Omega(\mathbf{z}) = \left|\left|\mathbf{z} - \mu_{y^i}\right|\right|^2, \tag{3.5}$$

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, the new regularized AE is named as Multi-distribution AE.

In the MAE loss function, we also use a parameter λ to control the trade-off between the reconstruction error (RE) and $\Omega(\mathbf{z})$ terms as discussed in Section 3.1.1. Thus, the loss function of MAE can be defined as follows:

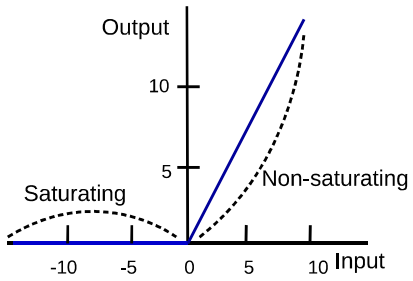
$$\ell_{MAE}(\theta, \phi, \mathbf{x}, \mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \hat{x}^i\right)^2 + \lambda\frac{1}{n}\sum_{i=1}^{n}\left|\left|z^i - \mu_{y^i}\right|\right|^2, \tag{3.6}$$

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. 3.6 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.

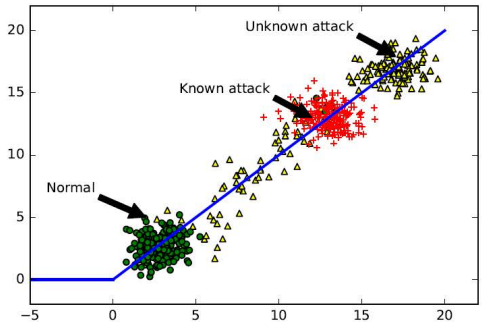
To visualize the probability distribution of the latent representation of MAE, i.e., $\mathbf{z}$, we calculate the histogram of one feature of the latent data $\mathbf{z_0}$. Fig. 3.1 presents the probability distribution of $\mathbf{z_0}$ of normal class and known attacks during the training process 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.

3.1.3 Multi-distribution Denoising AutoEncoder

In this subsection, we discuss the details of the multi-distribution Denoising AutoEncoder (MDAE). We employ DAE proposed in [87] to develop MDAE. For each data sample x^i, this model can draw its corrupted version $\tilde{x}^i$ using Eq. 2.6. 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 μ_{y^i}. The loss function of



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

MDAE can be presented in Eq. 3.7.

$$\begin{aligned}\ell_{MDAE}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{z}, \phi, \theta) =& \frac{1}{n}\sum_{i=0}^{n}\left(x^i - p_\theta(q_\phi(\tilde{x}^i))\right)^2 \\ &+ \lambda\frac{1}{n}\sum_{i=1}^{n}\left|\left|z^i - \mu_{y^i}\right|\right|^2,\end{aligned} \tag{3.7}$$

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. 3.7.

This chapter explores the motivation behind the proposed models by examining activation functions in neural networks. Activation functions are commonly used in neural networks to introduce non-linear computation and enable the network to learn complex patterns and relationships in the data. However, the issue of vanishing gradients arises when an activation function becomes saturated, causing gradients during backpropagation to diminish significantly and impede the network's learning

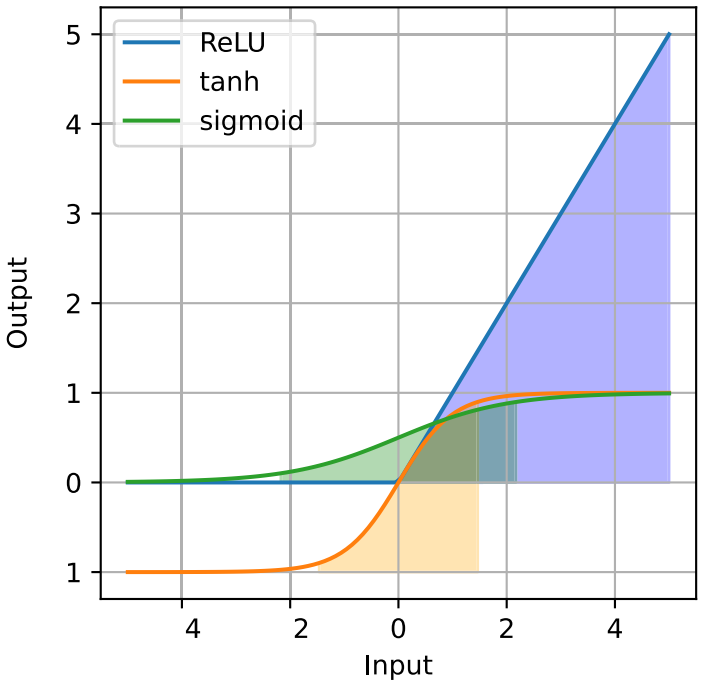


Figure 3.3 : Visualization of non-saturated areas of activation functions.

process. The non-saturated region of an activation function is highly desirable as it facilitates substantial changes in output values and fosters larger gradients during backpropagation. Notably, while saturated areas approach the function's bounds, the non-saturated region contains distinct output values. Activation functions exhibit different non-saturated regions as shown in Fig. 3.3; for instance, ReLU is non-saturated for positive inputs (the blue area), whereas sigmoid (the green area) and tanh (the orange area) have larger non-saturated regions near their input range center.

To 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. 3.2 (a)). Thus, μ_{y^i} needs to be assigned to a positive value under the ReLu activation function. In our experiments,

we set $\mu_{y^i} = 2$ for a normal class and $\mu_{y^i} = 3$ for an abnormal class[‡]. These values of μ_y^i are used in all the IoT-based datasets in our 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 [121]. The unknown attacks are predicted to project different regions towards the known anomaly region on the non-saturated area of the activation function. Fig. 3.2 (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.

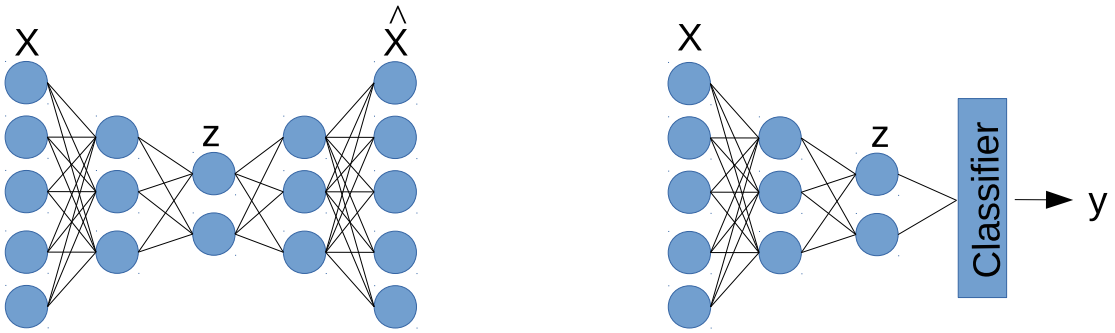
3.2 Using Proposed Models for Network Attack Detection

This section presents the training and predicting processing when applying the proposed latent representation learning models for AD. These processes can be applied to 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.

3.2.1 Training Process

Algorithm 1 presents the training process when using the AE-based model for AD. First, we train 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

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



(a) An AE-based model.

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

Figure 3.4 : Illustration of an AE-based model (a) and using it for classification (c,d).

Algorithm 1 Training a AE-based model.

INPUT:
x, y: Training data samples and corresponding labels.
An AE-based model with its hyper-parameters.
A classifier with its hyper-parameters.
OUTPUT: Trained AE-based model, Trained classifier.
BEGIN:
1. Put x, y to input of an AE-based model.
2. Training to minimize the loss function of an AE-based model as described in Section 3.1.
3. Put x, y to the trained AE-based model. To get the latent representation of x is z.
4. Train a classifier with input as z, y.
return Trained AE-based model, Trained classifier.
END.

minimize the loss function[§] of the AE-based model. This step tries to train a latent representation learning model based on AE as illustrated in Fig. 3.4 (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. 3.4 (b). Finally, we have training results, including the trained AE-based model and the trained classifier.

3.2.2 Predicting Process

Algorithm 2 Predicting process based on representation learning model.

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

Algorithm 2 describes the process of sample prediction for AD using 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. 3.4 (b). Therefore, the classifier identifies a network traffic sample based on its latent representation instead of its original representation.

[§]The computations of loss functions are presented in Section 3.1.

3.3 Experimental Setting

3.3.1 Dataset

We use nine IoT attack-related datasets (N-BaIoT) introduced by Y. Meidan et al. [137] for evaluating our proposed models. This dataset is described in Section 2.5.1. We split each of these datasets into a training set and a testing set based on the scenarios presented in Section 3.3.3. We randomly select 10% of the training data to create validation sets for model selection [142].

3.3.2 Hyper-parameter Setting

Table 3.1 : Hyper-parameters for AE-based models.

Hyper-parameter	**Value**
The number of hidden layers	5
The size of the bottleneck layer	$[1 + \sqrt{n}]$ [121]
Batch size	100
Learning rate	10^{-4}
Initialized method	Glorot [131]
Optimization algorithm	ADAM [26]
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[¶]. The common hyper-parameters of AE-based models are presented in Table 3.1 [121]. 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 [121], 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

[¶]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 their regularizer.

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. [131] to facilitate the convergence. We employ the ADAM optimization algorithm [26] 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 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 [32]. The DBN-based model has three layers as in [100] and is implemented by [51], where the number of neurons in each layer is similar to the AEs-based models in our experiments.

3.3.3 Experimental Scenario

Four linear classification algorithms, including Support Vector Machine (SVM), Perceptron (PCT), Nearest Centroid (NCT), and Linear Regression (LR) [113], 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 perform very fast and (2) to express the strength 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 [113].

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 AD, Random Forest (RF) [54, 103]; (2) classifiers using the latent representations of AutoEncoder (AE), Denoising AutoEncoder (DAE), Variational AutoEncoder (VAE), Convolutional Neuron Network (CNN) and Deep Belief Network (DBN) [100]. 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 number of parameters.

All experiments are done using the N-BaIoT datasets as described in Section 3.3.1.

Table 3.2 : 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

3.4 Result 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.

3.4.1 Ability to Detect Unknown Attacks

This section presents the main experimental results of our proposed models. We evaluate the proposed models as the ability to detect unknown 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 2.2. 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 3.2.

In Table 3.2, we can observe that the classifiers are unable to detect unseen IoT attacks (the AUC scores approximate 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. This observation is also obtained in the previous research [121].

It can also be seen from Table 3.2 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.

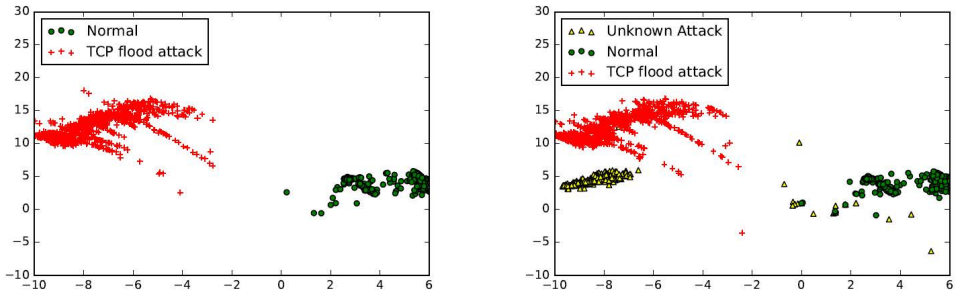
It can be seen from Table 3.2 that the latent representations resulting from MVAE, MAE, and MDAE help four classifiers achieve higher classification accuracy in comparison to those using the original data. For example, the AUC scores of SVM, PCT, NCT, and LR working on the latent representation of MAE 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

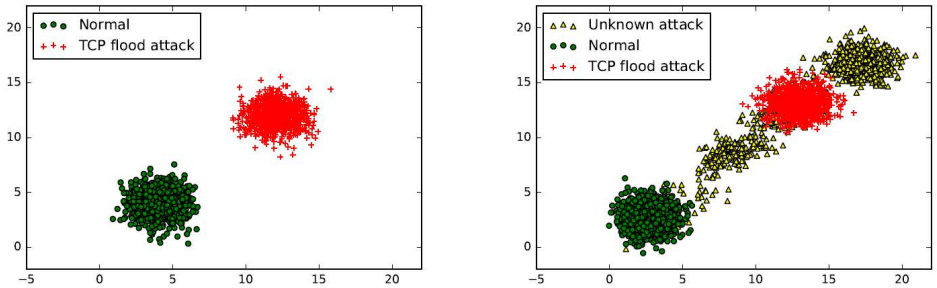
that the proposed models help to project the non-linear datasets in the original space into linearly separable data in the latent space.

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. 3.5, we plot 1000 random samples of the training and the testing data in the hidden space of AE and MAE.

Fig. 3.5 (a) and Fig. 3.5 (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 3.2. 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 more easily. 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 of constraining the normal data and known attack data into two compact areas in 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 applied to the latent features of MAE.



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

3.4.2 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 [72]. 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) [68, 72].

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 attack data.

This experiment aims to examine 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[‖] are shown in Table 3.3. 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 in detecting attacks presented in Table 3.2.

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

[‖]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.

Table 3.3 : 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

to produce 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 poor performance for the NCT classifier (as shown in the first three rows of Table 3.3). On the other hand, the latent representations of MVAE, MAE, and MDAE are designed to reserve some regions being close to the anomaly region for unknown IoT attacks. Thus, these AEs help NCT to identify unknown attacks more effectively and perform well in both scenarios (as observed in the last three rows of Table 3.3). 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.

3.4.3 Influence of Parameter

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, and the hyper-parameters in SVM and NCT classifiers.

Influence of the Noise Factor

This experiment examines the impact of the noise factor on the MDAE's performance. In this proposed model, the Gaussian noise function in Eq. 2.6 is employed

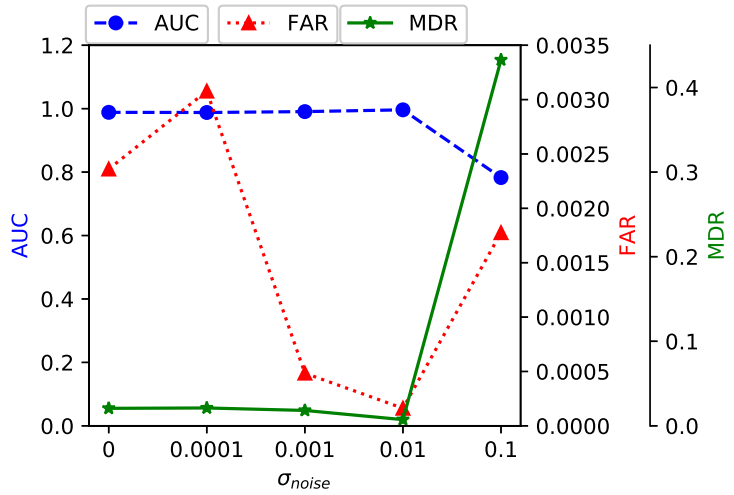


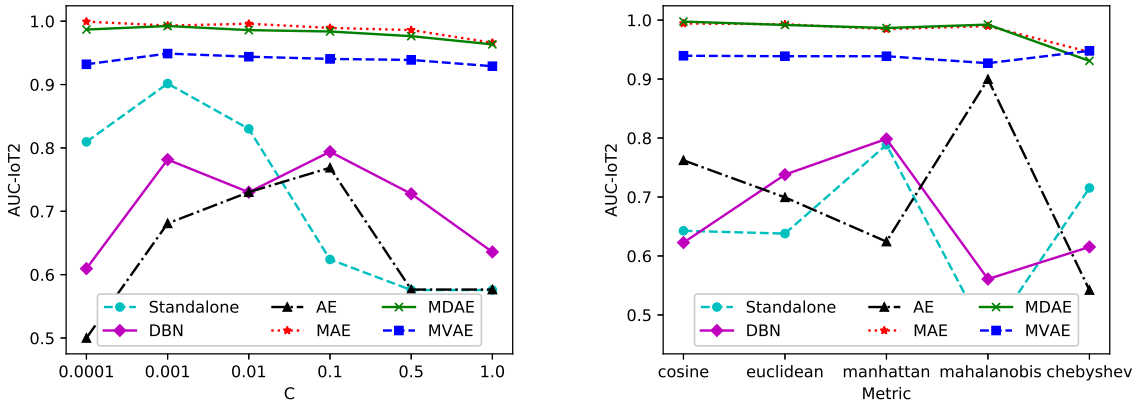
Figure 3.6 : Influence of noise factor on the performance of MDAE measuring by the average of AUC scores, FAR, and MDR produced from SVM, PCT, NCT and LR on the IoT-1 dataset. The noise standard deviation value at $\sigma_{noise} = 0.01$ results in the highest AUC, and lowest FAR and MDR.

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. 3.6 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** and Miss Detection Rate (MDR) scores†† curves go in the opposite direction. These results imply that MDAE achieves the best performance when the value of σ_{noise} is 0.01.

**FAR is defined as the number of false alarms of negative samples per the total number of real negative data samples.

††MDR is defined as the number of miss detection of positive samples per total of real positive samples.



(a) SVM. (b) NCT.

Figure 3.7 : AUC scores of (a) the SVM classifier and (b) the NCT classifier with different parameters on the IoT-2 dataset.

Influence of the parameter of Classifier

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 including AE, DBN, MVAE, MAE and MDAE. We conduct experiments on two well-known classification algorithms, i.e., SVM and NCT[‡‡]. 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 [21]. This hyper-parameter presents the generalization ability to detect unseen attacks of the SVM classifier. Fig. 3.7 (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

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

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 [113].

In machine learning, we might possibly get better results when using different distance metrics [1]. 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 the dataset has discrete and/or binary attributes, the *manhattan* distance can work quite well. Besides, the *mahalanobis* distance takes co-variance into account which helps in measuring the strength/similarity between two different data objects. Finally, the *chebyshev* distance is often used in warehouse logistics as it closely resembles the time an overhead crane takes to move an object.

Fig. 3.7 (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 their performance is less sensitive to hyper-parameters.

3.4.4 Complexity

Table 3.4 : Complexity of AE-based models trained on the IoT-1 dataset.

Models	No. Trainable Parameters
AE	25117
VAE	25179
DAE	25117
MVAE	25179
MAE	25117
MDAE	25117

Table 3.4 presents 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 trainable parameters of AE-based models with the same architecture are similar. Therefore, adjusting the loss functions of the proposed models does not affect much to the model size of AE-based models.

3.4.5 Assumption and Limitation

Although our proposed models possess many advantages in learning 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 the proposed models 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 [123] to train the models proposed in this chapter. Second, the advantages of representation learning models come with the

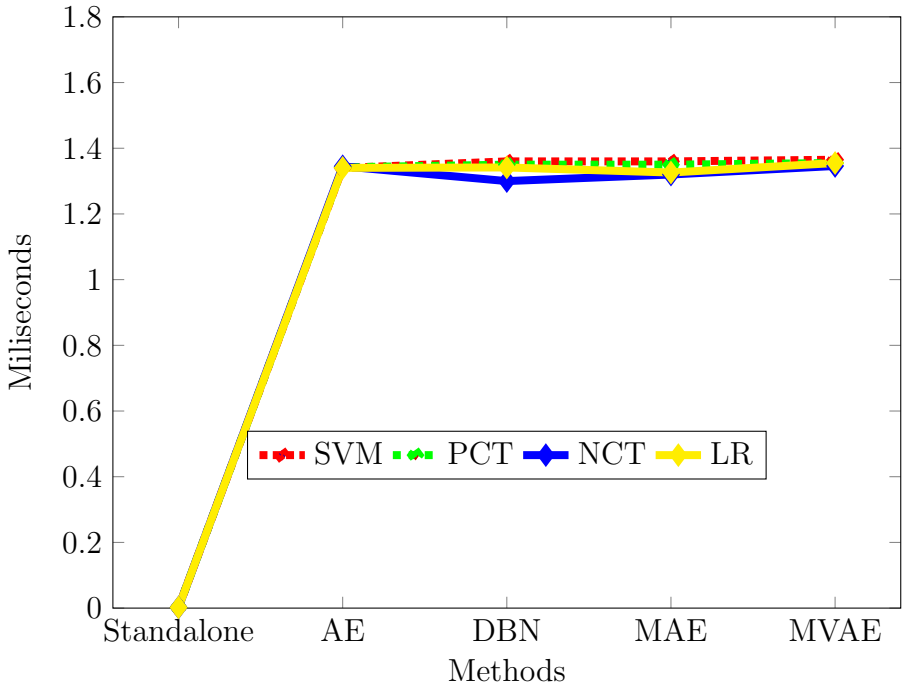


Figure 3.8 : Average testing time for one data sample of four classifiers with different representations on IoT-9.

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. 3.8 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 lives, such as smart cities, home automation, and data security [62, 96]. Specifically, alert systems in 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 [96]. 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.

3.5 Conclusion

In this chapter, we have designed three novel AE-based models for learning a new latent representation to enhance the accuracy in AD. 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 attack 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 in 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.

Chapter 4

DEEP GENERATIVE LEARNING MODELS FOR CLOUD ANOMALY DETECTION

In Chapter 3, we 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 anomalies, especially new/unknown anomalies. However, this approach only performs well with the assumption that we can collect enough labelled 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, e.g., IoT, cloud, collecting attack traffic is much more complicated than those of normal traffic. For example, the network attacks in the cloud environment are difficult to collect due to the sensitive applications of providers and end-users. Moreover, the wide spread of cloud attacks and serious impact causes to the availability and the reputation of cloud providers. 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 cloud anomaly using a type of deep neural network so called a generative model. Compared with previous data generation techniques, our solution uses the generative models with noisy

input and label information. Moreover, the proposed technique focuses on generating the data samples which are difficult for classifiers. Specifically, we develop two deep generative models to synthesize malicious samples on the cloud systems. The first model, Conditional Denosing Adversarial AutoEncoder (CDAAE), is used to generate specific types of malicious samples. The second model (CDAEE-KNN) is a hybrid of CDAAE and the K-nearest Neighbor algorithm to generate malicious borderline samples* that further help improve the accuracy of cloud anomaly detection. The synthesized samples are merged with the original samples to form the augmented datasets. Three machine learning algorithms are trained on the augmented datasets and their effectiveness is analyzed. The experiments conducted on four popular intrusion datasets show that our proposed techniques significantly improve the accuracy of the cloud anomaly detection system compared with the baseline technique and previous approaches. Moreover, our models also enhance the accuracy of machine learning algorithms in detecting some currently challenging DDoS attacks including low-rate DDoS attacks and application layer DDoS attacks.

The rest of the chapter is organized as follows. The proposed models are presented in Section 4.1. Section 4.2 presents the experimental settings. Section 4.3 and Section 4.4 discuss and analyze the performance results and properties of the proposed models. Finally, in Section 4.5, we conclude the contributions of this chapter.

4.1 Proposed Generative Models for Cloud Anomaly Detection

This section proposes two generative models for generating malicious samples for the cloud AD, i.e., CDAAE and CDAAE-KNN.

*The samples are positioned close to the decision boundary in order to distinguish between the two classes.

4.1.1 Conditional Denoising Adversarial AutoEncoder

CDAAE aims to leverage the effectiveness of SAAE [3] in generating malicious samples. It is different from SAAE in two folds. First, the label information (y) is incorporated into both decoder De and encoder En instead of only De in SAAE. This makes CDAAE easier to generate synthesized samples for any specific class. Second, the input to CDAAE is the noisy version of the original input. This helps the En to learn the more robust latent representation. In the CDAAE model, $\mathbf{x_{noise}}$ is generated from $\mathbf{x}$ using the Gaussian corruption function [7] in Eq. 4.1.

$$f(\mathbf{x_{noise}}|\mathbf{x}) = \mathcal{N}(\mathbf{x}, \sigma^2 I), \tag{4.1}$$

where the mean and standard deviation values of the Gaussian noise are $\mathbf{x}$ and σ, respectively.

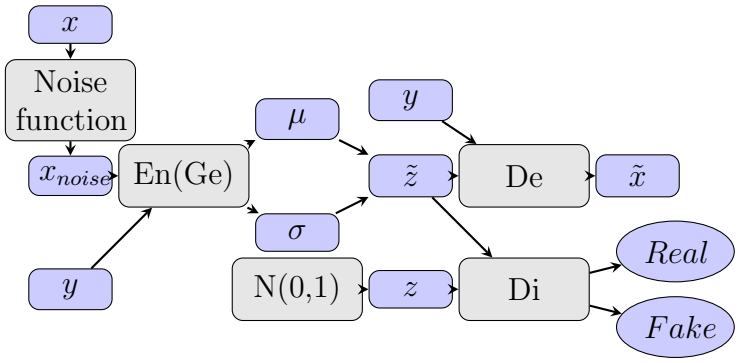


Figure 4.1 : Structure of CDAAE.

Fig. 4.1 describes the architecture 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 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. 4.2 and Eq. 4.3, respectively.

$$\tilde{z}^i = f_{En}(\mathbf{W_{En}}(x^i_{noise} \mid y^t + \mathbf{b_{En}}), \tag{4.2}$$

$$\tilde{x}^i = f_{De}(\mathbf{W_{De}}(z^i \mid y^t) + \mathbf{b_{De}}), \tag{4.3}$$

where $\mid$ is concatenation operator and f_{En} and f_{De} are the activation functions of the encoder En and the decoder De, respectively. y^t is the label of the data sample x^i, x^i_{noise} is generated from x^i by Eq. 2.6. 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. 4.4.

$$R_{loss} = \frac{1}{n}\sum_{i=0}^{n}\left(x^i - \hat{x}^i\right)^2. \tag{4.4}$$

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:

$$d^i_{real} = f_{Di}(\mathbf{W_{Di}}z^i + \mathbf{b_{Di}}), \tag{4.5}$$

$$d^i_{fake} = f_{Di}(\mathbf{W_{Di}}\tilde{z}^i + \mathbf{b_{Di}}), \tag{4.6}$$

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 Eq. 4.7 whereas the generator Ge (or En) tries to generate $\tilde{\mathbf{z}}$ to fool the discriminator Di by maximizing Eq. 4.8.

$$D_{loss} = -\frac{1}{n}\sum_{i=0}^{n}\left(\log(d^i_{real}) + \log(1 - d^i_{fake})\right), \tag{4.7}$$

$$G_{loss} = \frac{1}{n}\sum_{i=0}^{n}\log(d^i_{fake}). \tag{4.8}$$

The total loss function of CDAAE is the combination of the three above losses as in Eq. 4.9 and CDAAE is trained to minimise this loss function.

$$L_{CDAAE} = R_{loss} + D_{loss} - G_{loss}. \tag{4.9}$$

4.1.2 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 in this area potentially makes it more beneficial than performing on the whole minority class [45]. Nguyen et al. [43] uses SVMs to learn the boundary of classes and over-sampling the minor samples around this boundary. 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 [55]

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 cloud ADs. In this model, KNN is used to select the generated samples by CDAAE that are

close to the borderline.

Algorithm 3 CDAAE-KNN algorithm.

INPUT:
X: original training set
$\tilde{X}$: generated data samples
m: number of nearest neighbours
d(x,y): Euclidean 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.

Algorithm 3 describes the details of CDAAE-KNN. The algorithm is divided into two phases: generation and selection. In the generation phase (Steps 1 and 2), we train a CDAAE model on the original dataset X. Then, we 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 $X \cup \tilde{X}$ to find the k nearest neighbours 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 which 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}.

We consider the K nearest neighbor data samples of the generated data of a minor class by CDAAE-KNN. Out of these neighbor samples, 85% belong to the minor

class and 15% belong to the major class. As a result, CDAAE-KNN can generate minor data samples that are closely located to major data samples. These samples are known as borderline data samples, which often pose challenges for classifiers. Hence, it is reasonable to expect that CDAAE-KNN may generate misclassified samples with a certain probability.

4.2 Experimental Settings

This section describes the datasets and the experimental settings used in this chapter.

4.2.1 Datasets

To evaluate the effectiveness of the proposed methods, we performed experiments using four datasets including cloud DoS attack datasets, low-rate DDoS attack datasets, application layer DDoS attack datasets, and network AD datasets. These attacks are drawn from the cloud AD dataset [95], the CIC-IDS 2017 dataset [50]), the NSL-KDD dataset [84] and the UNSW-NB15 dataset [83]. The reason for using these datasets is to demonstrate the effectiveness of the proposed solutions in detecting attacks in three different scenarios: (1) the most popular attack on the cloud (the cloud DDoS datasets), (2) the two most challenging attacks on the cloud (the low-rate DDoS datasets and the application level datasets), (3) the general network attacks (the network AD datasets). The descriptions of these datasets are presented in Section 2.5. The number of synthesized samples is calibrated at 10 times compared to the number of attacks in the original data. We have also tested with different values: 5, 10, 15, 20, 50, and 100.

4.2.2 Experimental Scenario

We divided our experiments into three sets. The first set (presented in detail in Appendix A.1) is to find a set of appropriate values for the hyper-parameters

of the deep network models on each dataset. For this, we applied the Bayesian Optimization [70] technique to automatically find the appropriate values for the hyper-parameters of the tested models on each dataset. We then used the values obtained by the Bayesian Optimization to train the deep network models on the training data.

After training, the model is used to generate the synthesized data to balance the current skewed dataset. 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 [113], is used. To minimize the impact of experimental parameters on the performance of the classification algorithms, we also implemented the grid search technique for each classifier algorithm. The range of values for each parameter tuned by the grid search technique is shown in Table 4.1. 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 4.1 : Values of the parameters used in the 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	$n - estimators = 5, 10, 20, 40, 80, 150$

The second experimental set is to compare the performance (i.e. accuracy and time) of our proposed models with the previous generative models. The tested generative models include SMOTE-Support Vector Machine (SMOTE-SVM), Auxiliary Classifier Generative Adversarial Network (ACGAN), Conditional Variational AutoEncoder (CVAE), and Supervised Adversarial AutoEncoder (SAAE). We also evaluate the accuracy of the original method (referred to as ORIGINAL) in which

the classifiers are trained on the original datasets without adding synthesized attack samples. The last set is to analyze the three properties of CDAAE. This experiment helps to shed some light on the performance of CDAAE. The results of the second and third experiment sets are presented in the next two sections.

4.3 Performance Analysis

This section compares the proposed models with other tested models on two measures, i.e., the accuracy and processing time.

4.3.1 Accuracy on Cloud DDoS Attack Datasets

First, we compare the effectiveness of CDAAE and CDAAE-KNN for generating synthesized attacks with the previous generative models on the cloud DDoS attack datasets. Table 4.2 presents the AUC and GEO of three classifiers when they are trained on the original version and the augmented versions of the cloud DDoS attack datasets.

Overall, all tested machine learning algorithms perform well in detecting cloud DDoS attacks. The reason is that DDoS attacks usually have a huge volume of traffic comparing to the normal traffic. Thus, the statistical features such as *number of occurrence for an incoming IP, bytes received per unique IP* are very distinguishable between the normal traffic and the attack traffic. Moreover, using generative models to generate synthesized attacks can enhance the accuracy of classifiers on this problem. For example, the AUC score of SVM trained on the original version of the PingOfDeath dataset is 0.972. This value is improved when SVM is trained on the augmented datasets of the generative models and reaches 1.000 with CVAE and SAAE. Among the five deep network models (ACGAN, CVAE, SAAE, CDAAE, CDAAE-KNN), the table shows that their effectiveness is mostly equal on the two datasets.

Importantly, it can be observed that these DoS attacks can achieve AUC scores of 1.000 using various generative models. This indicates that DoS attacks can be easily detected due to their significant deviation from normal network traffic in terms of volume. However, this phenomenon may not occur in the case of low-rate DoS attacks, which will be discussed in the following section.

Table 4.2 : Results on the cloud DDoS datasets.

Alg.	**Dataset**	**Cloud IDS**			
	Attack	**TCP land**		**PingOfDeath**	
	Metric	**AUC**	**GEO**	**AUC**	**GEO**
SVM	ORIGINAL	0.990	0.991	0.972	0.972
	SMOTE-SVM[82]	1.000	1.000	0.973	0.973
	ACGAN[144]	0.981	0.981	0.987	0.987
	CVAE[93]	0.977	0.977	1.000	1.000
	SAAE[3]	1.000	1.000	1.000	1.000
	CDAAE	1.000	1.000	0.988	0.988
	CDAAE-KNN	1.000	1.000	0.999	0.999
DT	ORIGINAL	0.950	0.951	0.982	0.982
	SMOTE-SVM[82]	0.999	0.999	0.973	0.973
	ACGAN[144]	1.000	1.000	1.000	1.000
	CVAE[93]	1.000	1.000	1.000	1.000
	SAAE[3]	1.000	1.000	1.000	1.000
	CDAAE	1.000	1.000	1.000	1.000
	CDAAE-KNN	0.999	0.999	0.998	0.999
RF	ORIGINAL	0.928	0.927	0.970	0.969
	SMOTE-SVM[82]	0.999	0.999	0.973	0.973
	ACGAN[144]	1.000	1.000	1.000	1.000
	CVAE[93]	1.000	1.000	1.000	1.000
	SAAE[3]	1.000	1.000	1.000	1.000
	CDAAE	1.000	1.000	1.000	1.000
	CDAAE-KNN	1.000	1.000	1.000	1.000

4.3.2 Accuracy on Low-rate DDoS Attack Datasets

This subsection analyzes the tested methods on the low-rate DDoS attack datasets. The results are presented in Table 4.3. First, it can be seen that detecting the low-rate DDoS attacks is much harder than detecting the normal DDoS attacks. This is reflected by the fact that the AUC and GEO of the classifiers in Table 4.3 are

smaller than those in Table 4.2. Second, the accuracy of the classifiers is improved significantly with the generative models. On the Slowloris of the cloud AD dataset, the AUC scores of SVM, DT, and RF trained on the original version are improved from 0.907, 0.915, and 0.911 to 0.925, 0.994, and 0.992, respectively, when they are trained on the augmented datasets by CDAAE. Those values also are increased from 0.962, 0.961 and 0.995 to 0.993, 0.980, and 0.996, respectively, on the augmented CIC-IDS17 Slowloris dataset by CDAAE.

Most importantly, the table shows that the proposed models, i.e., CDAAE and CDAAE-KNN often achieve the highest AUC and GEO among all the tested generative methods. For instance, the AUC of DT on the datasets augumented by CVAE and SAAE are 0.986 and 0.977, respectively and these values are increased to 0.994 and 0.998, respectively, with CDAAE and CDAAE-KNN. The results with other configurations are also similar in the sense that they show the benefit of using CDAAE and CDAAE-KKN in improving the ability of machine learning models in detecting low-rate DDoS, one of the most challenging DDoS attacks.

4.3.3 Accuracy on Application Layer Attack Datasets

This subsection compares the effectiveness of CDAAE and CDAAE-KNN for generating synthesized attacks for application/gateway-level attacks. The results of SVM, DT and RF on this datasets are presented in Table 4.4. These results are consistent with the results in Table 4.3 in two folds. First, using generative models to generate synthesized samples for the minor classes helps to improve the accuracy of the classifiers significantly. The GEO score considers both the ability of the classifier to correctly identify positive data samples (precision) and its ability to capture all positive data samples (recall). Thus, it is a robust metric that provides a balanced assessment of the classifier's ability to handle both the majority and minority classes. With the Botnet dataset, the GEO scores of SVM, DT, and RF

Table 4.3 : Results on low-rate DDoS attack datasets.

Alg.	Dataset	Cloud IDS		CIC-IDS 2017			
	Attack	Slowloris		Slowloris		Slowhttp	
	Metric	AUC	GEO	AUC	GEO	AUC	GEO
SVM	ORIGINAL	0.907	0.989	0.962	0.962	0.958	0.958
	SMOTE-SVM[82]	0.922	0.921	0.980	0.980	0.993	0.992
	ACGAN[144]	0.923	0.915	0.963	0.962	0.993	0.992
	CVAE[93]	0.904	0.899	0.967	0.965	0.992	0.992
	SAAE[3]	0.897	0.894	0.993	0.994	0.994	0.993
	CDAAE	0.925	0.922	0.993	0.993	0.995	0.994
	CDAAE-KNN	0.999	0.999	0.993	0.993	0.995	0.994
DT	ORIGINAL	0.915	0.915	0.961	0.960	0.975	0.975
	SMOTE-SVM[82]	0.960	0.957	0.962	0.962	0.991	0.990
	ACGAN[144]	0.956	0.963	0.995	0.994	0.980	0.980
	CVAE[93]	0.986	0.985	0.966	0.966	0.979	0.979
	SAAE[3]	0.977	0.977	0.975	0.975	0.994	0.993
	CDAAE	0.994	0.994	0.980	0.980	0.996	0.996
	CDAAE-KNN	0.998	0.999	0.982	0.981	0.996	0.996
RF	ORIGINAL	0.911	0.966	0.995	0.995	0.992	0.991
	SMOTE-SVM[82]	0.976	0.976	0.995	0.995	0.996	0.996
	ACGAN[144]	0.995	0.950	0.995	0.995	0.996	0.995
	CVAE[93]	0.983	0.983	0.995	0.995	0.994	0.993
	SAAE[3]	0.989	0.989	0.996	0.995	0.997	0.996
	CDAAE	0.992	0.992	0.996	0.996	0.997	0.996
	CDAAE-KNN	0.926	0.928	0.996	0.995	0.998	0.996

trained on the original version are improved from 0.670, 0.885, and 0.884 to 0.702, 0.982, and 0.940, respectively, when they are trained on the augmented datasets by CDAAE. These values are also increased from 0.757, 0.472 and 0.947 to 0.824, 0.882, and 0.999, respectively, with the Brute Force dataset. These proves that the proposed solution for balancing datasets strongly improves the effectiveness of ML models for AD.

Second, Table 4.4 also shows that the AUC score and GEO score of classifiers based on the generative models are usually higher than those of the traditional technique (SMOTE-SVM). For example, comparing SAAE and SMOTE-SVM, the AUC scores are increased from 0.724, 0.980, and 0.912 to 0.730, 0.980, and 0.930, respec-

tively, on the Botnet dataset corresponding to SVM, DT, and RF. These values are changed from 0.802, 0.835, and 0.943 to 0.824, 0.837, and 0.997, respectively, on the Brute Force dataset. Among all techniques for synthesizing data, it can be seen that our models (CDAAE and CDAAE-KNN) often achieve the best results. For example, the AUC scores of RF with CDAAE and CDAAE-KNN are 0.942 and 0.945, respectively, and these are the highest values among all generative methods.

Table 4.4 : Results of application level attack detection.

Alg.	**Dataset**	**CIC-IDS 2017**					
	Attack	**Botnet**		**Brute Force**		**SSH-Patator**	
	Metric	**AUC**	**GEO**	**AUC**	**GEO**	**AUC**	**GEO**
SVM	ORIGINAL	0.724	0.670	0.786	0.757	0.750	0.708
	SMOTE-SVM[82]	0.724	0.665	0.802	0.809	0.760	0.758
	ACGAN[144]	0.735	0.722	0.812	0.812	0.760	0.758
	CVAE[93]	0.724	0.670	0.811	0.810	0.755	0.752
	SAAE[3]	0.730	0.701	0.824	0.823	0.761	0.760
	CDAAE	0.755	0.702	0.828	0.824	0.764	0.762
	CDAAE-KNN	0.755	0.702	0.832	0.825	0.764	0.762
DT	ORIGINAL	0.888	0.885	0.611	0.472	0.999	0.999
	SMOTE-SVM[82]	0.980	0.980	0.835	0.820	0.999	0.999
	ACGAN[144]	0.981	0.980	0.837	0.821	0.999	0.999
	CVAE[93]	0.975	0.972	0.837	0.821	0.999	0.999
	SAAE[3]	0.980	0.981	0. 837	0.821	0.999	0.999
	CDAAE	0.980	0.982	0.838	0.822	0.999	0.999
	CDAAE-KNN	0.980	0.982	0.838	0.822	0.999	0.999
RF	ORIGINAL	0.891	0.884	0.949	0.947	0.998	0.998
	SMOTE-SVM[82]	0.912	0.911	0.943	0.940	0.999	0.999.
	ACGAN[144]	0.942	0.940	0.996	0.995	0.999	0.999
	CVAE[93]	0.935	0.930	0.997	0.993	0.999	0.999
	SAAE[3]	0.930	0.928	0.997	0.995	0.999	0.999
	CDAAE	0.942	0.940	0.999	0.999	0.999	0.999
	CDAAE-KNN	0.945	0.942	0.999	0.999	0.999	0.999

4.3.4 Accuracy on Network AD Datasets

Table 4.5 presents the results of SVM, RF, and DT, on two network AD datasets (NSL-KDD and UNSW-NB15). It can be seen that these datasets are much harder than the previous datasets. This is reasonable since these experiments address

multi-classification problems instead of binary classification. The difficulty of the problems is reflected by the fact that the GEO value is 0 in many configurations. The 0 GEO score evidences that the machine learning algorithms completely misclassify some attacks. For example, on the NSL-KDD dataset, the classifiers usually cannot detect R2L and U2L attacks. On UNSW-NB15, the classifiers trained on the original version often cannot detect 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 GEO scores of SVM, DT, and RF are improved from 0, 0, and 0.442 to 0.763, 0.668, and 0.644, respectively, when they are trained on the augmented datasets by CDAAE. These values are increased from 0 to approximately 0.452, 0.502, and 0.451, respectively, on the augmented UNSW-NB15 dataset by CDAAE-KNN.

4.3.5 Processing Time Analysis

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 two datasets, PingOfDeath and TCPLand. All experiments were conducted on the same computing platform, i.e., Operating system: Ubuntu 18.04.2 LTS (64 bit), CPU: Intel(R) Core(TM) i5-5200U, and RAM 8GB. The results are presented in Table 4.6.

It can be seen that all approaches based on deep neural networks require a significant time to train the models whereas the traditional approach, i.e., SMOTE-SVM do not require training the generative models. However, for the generation time, i.e., the time to generate synthesized samples, the table shows that the difference between the traditional approaches and the deep networks is negligible. Among the

Table 4.5 : Results on the network AD datasets.

Alg.	**Dataset**	**NSL-KDD**		**UNSW-NB15**	
	Attack	**Network attack**		**Network attack**	
	Metric	**AUC**	**GEO**	**AUC**	**GEO**
SVM	ORIGINAL	0.570	0	0.129	0
	SMOTE-SVM[82]	0.688	0.585	0.218	0.148
	ACGAN[144]	0.794	0.534	0.496	0.322
	CVAE[93]	0.707	0.627	0.365	0.300
	SAAE[3]	0.742	0.698	0.349	0
	CDAAE	0.812	0.763	0.371	0.453
	CDAAE-KNN	0.804	0.684	0.640	0.452
DT	ORIGINAL	0.430	0	0.221	0
	SMOTE-SVM[82]	0.446	0.361	0.348	0.228
	ACGAN[144]	0.745	0.613	0.436	0.437
	CVAE[93]	0.750	0.653	0.458	0.460
	SAAE[3]	0.735	0.663	0.462	0.455
	CDAAE	0.759	0.668	0.492	0.485
	CDAAE-KNN	0.733	0.671	0.499	0.502
RF	ORIGINAL	0.760	0.442	0.357	0
	SMOTE-SVM[82]	0.780	0.564	0.436	0.407
	ACGAN[144]	0.679	0.421	0.485	0.443
	CVAE[93]	0.790	0.352	0.554	0.398
	SAAE[3]	0.785	0.516	0.463	0.433
	CDAAE	0.779	0.644	0.556	0.503
	CDAAE-KNN	0.788	0.494	0.649	0.451

Table 4.6 : Processing time of training and generating samples processes in seconds.

Method	**PingOfDeath**		**TCP land**	
	Train	Generate	Train	Generate
SMOTE-SVM[82]	0	0.100	0	0.178
ACGAN[144]	157.182	1.132	134.037	0.011
CVAE[93]	280.112	0.015	288.776	0.015
SAAE[3]	223.886	0.112	241.386	0.017
CDAAE	277.525	0.713	215.241	0.769
CDAAE-KNN	330.714	0.998	297.803	2.185

tested deep network models, we can see that CDAAE-KNN often requires a longer time to generate data than those of the others. However, the overhead is not significant. Moreover, both the 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 (i.e., time to detect attacks) of the classification algorithms.

Overall, the results in this section show that the generative models can be used to generate meaningful samples for the minor classes on the cloud AD. 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 approach (SMOTE-SVM) is that the traditional approaches create samples that do not follow the original data distribution, while this problem is mitigated by using the generative models. Additionally, the methods based on CDAAE (e.g., CDAAE and 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 usually yields more successful results compared with CDAAE thanks to the usage of KNN to select important samples which contribute more to the classifiers.

4.4 Properties Analysis

This section deeper analyzes and compares three properties of the CDAAE including the impact of balancing skewed datasets, the convergence of loss function in the training process and the quality of the generated samples.

4.4.1 Effect of Balancing Data

We investigate whether adding the synthesized samples to the minor classes can improve the performance of classifiers by conducting an experiment on the Slowloris dataset. The ratio between the minor class (attack data) and the major class (normal data) in the original data is 0.095. We train CDAAE on the original dataset and use the trained model to generate the minor samples. We tested five scenarios in which the rate between the attack data and the normal data increases from 0.195

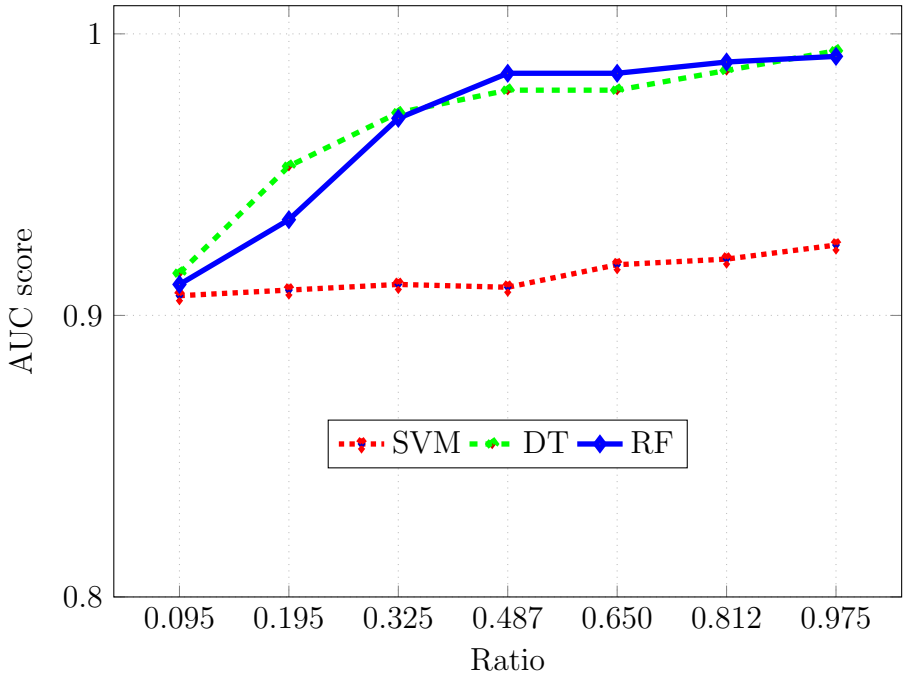


Figure 4.2 : Effect of using CDAAE to generate minor samples.

to 0.975 (as shown in Fig. 4.2).

Fig. 4.2 presents the AUC scores of SVM, DT and RF on the tested data. It can be seen that the accuracy of the three classifiers is consistently improved when the number of minor samples increases. For example, the AUC score of SVM is improved from 0.907 to 0.925 when the balancing rate is increased from 0.095 to 0.975. More impressively, the AUC score of DT and RF is improved from 0.915 to 0.994 and from 0.911 to 0.992, respectively, corresponding to the above balancing rate. This result evidences that using CDAAE to balance the skewed dataset in the cloud AD can significantly improve the accuracy of the detection algorithms.

4.4.2 Loss Visualization

This subsection visualizes the reconstruction error in the training process of three deep network models including CVAE, SAAE, and CDAAE [†]. The reconstruction

[†]For a fair comparison, we visualize the CDAAE model without noise.

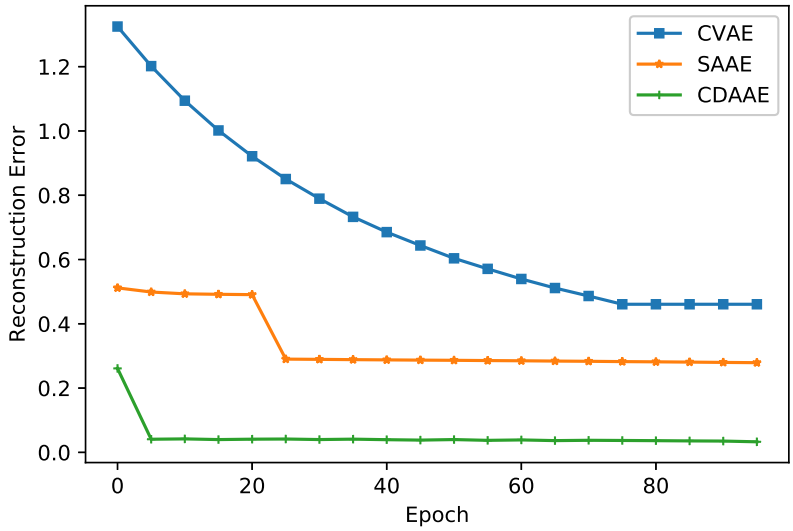


Figure 4.3 : Reconstruction error in training process of CVAE, SAAE, and CDAAE without noise.

error loss value represents the ability of the models for generating data samples that are similar to the input data samples. The smaller reconstruction error loss value presents a better model for generating data samples.

It can be observed in Fig. 4.3 that the AAE-based models, i.e., SAAE and CDAAE, reach lower reconstruction error values in the training process compared with the VAE-based model, i.e., CVAE. Moreover, incorporating the label to both *En* and *De* in the CDAAE model helps its RE loss to converge more quickly than those of CVAE and SAAE. The reason for that is that incorporating label information into both *En* and *De* can impose restrictions on how the *En* and *De* may use the label attribute [44]. Only incorporating the label attribute to *De* as in CVAE and SAAE may not guarantee these restrictions. Overall, our proposed model CDAAE converges the lowest reconstruction error value in the training process. This subsequently helps CDAAE generate more meaningful attack samples compared to the others.

4.4.3 Quality of Synthesized Samples

This subsection qualitatively measures whether the generated data of the generative models converges to the true data distribution. We used the Parzen window method [48] 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 Parzen window and the log-likelihood of each generative model under the true distribution is reported [48]. The window size parameter of the Gaussian is obtained by cross-validation on the validation set. Experimental results are reported in Table 4.7[‡] where a higher value presents a better generative model.

Table 4.7 : Parzen window-based log-likelihood estimates of generative models on the Cloud AD datasets.

Model	**Slowloris**	**TCP Land**	**PingOfDeath**
SMOTE-SVM [82]	43.33 ± 1.44	12.32 ± 1.94	60.06 ± 0.40
ACGAN [144]	44.18 ± 1.32	24.96 ± 1.45	68.54 ± 1.59
CVAE [93]	52.29 ± 1.78	33.59 ± 1.29	46.89 ± 1.78
SAAE [3]	52.69 ± 1.78	34.36 ± 1.46	60.87 ± 2.67
CDAAE	56.34 ± 1.54	35.99 ± 1.65	69.24 ± 1.89

It can be seen that the log-likelihood of all generative models is always greater than the value of SMOTE-SVM on all tested datasets. The reason is that 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 greatest 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 compared to those trained on the augmented datasets of the other generative models.

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

4.5 Conclusion

This chapter has proposed two novel models, i.e., CDAAE and CDAAE-KNN, to enrich the data and address the imbalance problem of AD datasets in the cloud environment. The CDAAE model is used to generate samples for a specific class label and CDAAE-KNN is 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 four classes of cloud AD datasets and the results have demonstrated that using our proposed models to generate malicious data helps the classification algorithms achieve higher performance in detecting cyberattacks on the cloud environment. The experiments also show that CDAAE and CDAAE-KNN support classification algorithms to enhance their accuracy in detecting two of the most challenging DDoS attacks namely application layer DDoS and low-bandwidth DDoS.

We have also quantitatively measured the quality of the generated samples of CDAAE and compared them 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 better performance of the classifiers when they are trained on the augmented datasets of CDAAE and CDAAE-KNN.

Chapter 5

TRANSFER LEARNING FOR IOT ANOMALY DETECTION

In Chapter 3 and Chapter 4, we have proposed two solutions to leverage the effectiveness of deep learning models in AD. These solutions are based on the assumption that we can collect labelled data of both normal and anomaly classes. However, the labelling process is usually performed manually by humans. Such a process is time-consuming and expensive. Thus, in practical 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 labelled 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 labelled. 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 rest of this chapter is organized as follows. The proposed model is presented

in Section 5.1. Section 5.2 discusses the experiment settings and Section 5.3 provides detailed analysis and discussion related to experimental results. Finally, the conclusion is drawn in Section 5.4.

5.1 Proposed Transfer Learning Model for IoT Cyberattack Detection

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

5.1.1 System Structure

Fig. 5.1 presents the system structure that uses DTL for IoT anomaly detection. First, the data collection module gathers data from all IoT devices. The training data consists of both labelled and unlabelled data. The labelled data is collected from some IoTs devices which are dedicated to labelling data. The labelling process is usually executed in two steps [109], i.e., each data sample is extracted from captured packets using the Tcptrace tool [116], then the data sample is labelled as a normal sample or an attack sample by manually analyzing the flow using Wireshark software [129]. Usually, the number of labelling IoT devices is much smaller than the number of unlabelling IoT devices. Second, the collected data is passed to the DTL model for training. The training process attempts to transfer the knowledge information learnt from the data with label information to data without label information. This 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. A detailed description of the DTL model is presented in the next subsection.

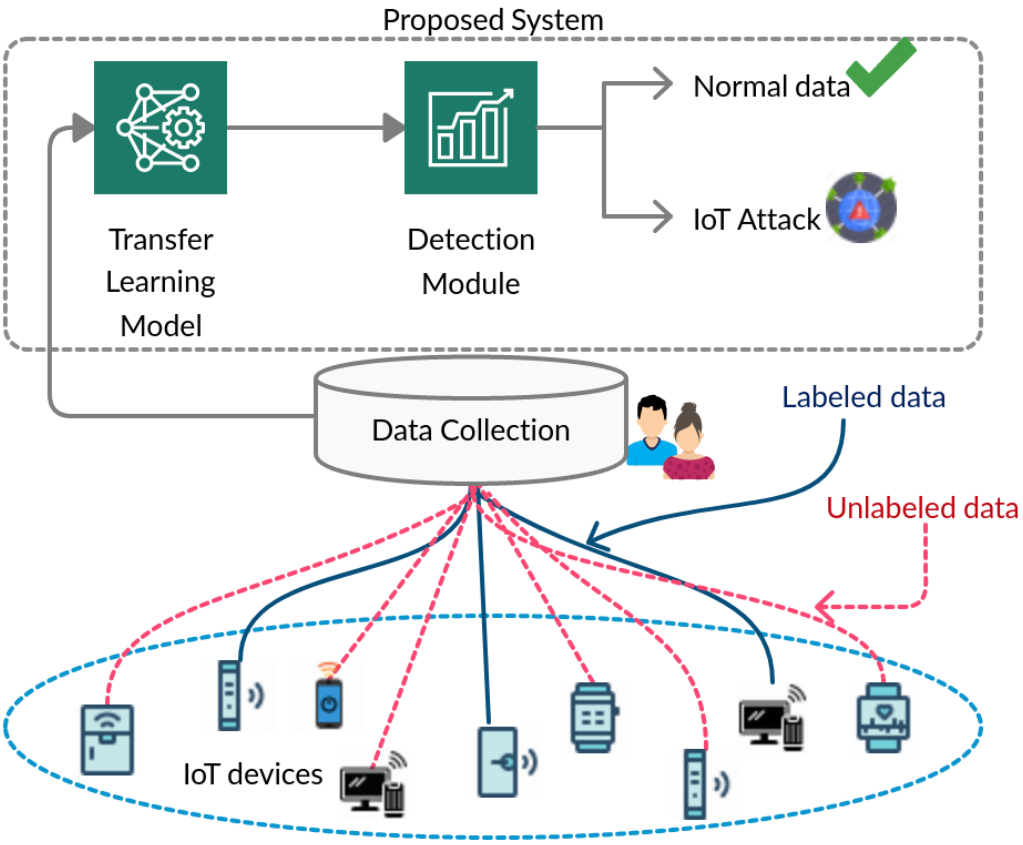


Figure 5.1 : Description of Deep Transfer Learning system.

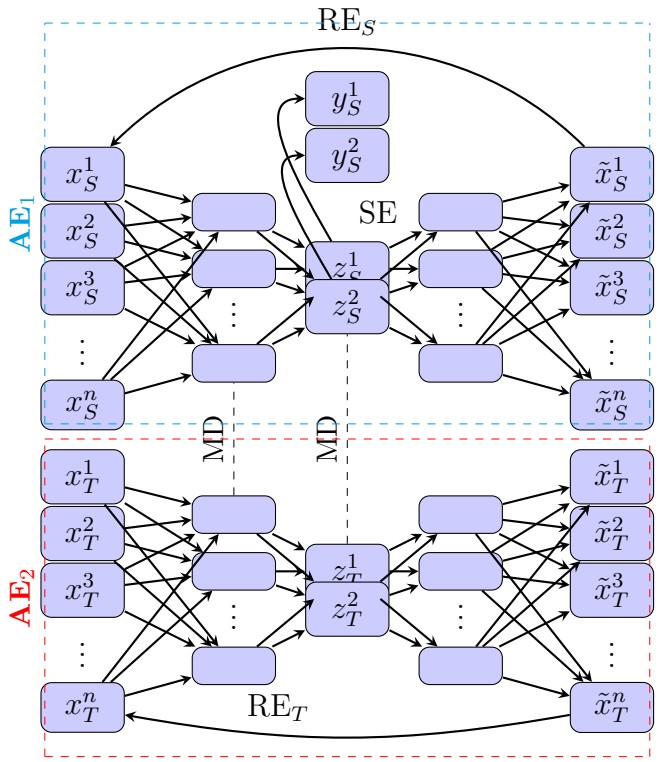


Figure 5.2 : Architecture of MMD-AE.

5.1.2 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. 5.2. 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. 5.2, 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:

$$\ell_{\text{RE}}(x_S^i, \phi_S, \theta_S, x_T^i, \phi_T, \theta_T) = l(x_S^i, \hat{x}_S^i) + l(x_T^i, \hat{x}_T^i), \tag{5.1}$$

where l function is the MSE function [121], 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 labelled 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 [46] 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 distinguishing the latent representation space from separated class labels. Formally, this loss is defined as follows:

$$\ell_{\text{SE}}(x_S^i, y_S^i, \phi_S.\theta_S) = -\sum_{j=1}^{C} y_S^{i,j} \log(z_S^{i,j}), \tag{5.2}$$

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. The transferring process is executed by minimizing the MMD distances [8] 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:

$$\ell_{\text{MMD}}(x_S^i, \phi_S, \theta_S, x_T^i, \phi_T, \theta_T) = \sum_{k=1}^{K} \text{MMD}(\xi_S^k(x_S^i), \xi_T^k(x_T^i)), \tag{5.3}$$

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. 2.11.

The final loss function of MMD-AE that is presented in Eq. 5.4 combines the loss terms in Eq.5.1, Eq.5.2, and Eq. 5.3. After training, the AE_1 can do AD by minimizing both ℓ_{SE} and ℓ_{RE}, the AE_2 also can do AD on the data of the target domain due to minimizing both ℓ_{MMD} and ℓ_{RE} terms.

$$\ell = \ell_{\text{SE}} + \ell_{\text{RE}} + \ell_{\text{MMD}}. \tag{5.4}$$

Algorithm 4 presents the pseudo-code for training our proposed DTL model. The training samples with labels in the source domain are input to AE_1 while the training samples without labels in the target domain are input to AE_2. The training process attempts to minimize the loss function in Eq. 5.4). After training, AE_2 is used to classify the testing samples in the target domain as in Algorithm 5.

Compared with the previous DTL model [35, 64], the proposed model, i.e., MMD-AE, is to transfer the knowledge not only in the bottleneck layer but also in *all encoding layers* from the source domain, i.e., AE_1, to the target domain, i.e., AE_2. As a result, MMD-AE allows to transfer more knowledge from the source domain to the target domain. One possible limitation of MMD-AE is that it may incur 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

Algorithm 4 Training the proposed DTL 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. $\xi_k(x_S)$ is the representation of x_S at the layer k of AE_1
4. z_S is the representation of x_S at the bottleneck layer of AE_1
5. $\xi_k(x_T)$ is the representation of x_T at the layer k of AE_2
6. Training the TL model by minimizing the loss function in Eq. 5.4
return Trained models: AE_1, AE_2.
END.

Algorithm 5 Classifying on the target domain.

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

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.

5.2 Experimental setting

5.2.1 Dataset and Metric

To evaluate the performance of MMD-AE we used nine IoT attack detection datasets from Meidan et al. [137]. These datasets are presented in Section 2.5.1. To evaluate the effectiveness of the proposed model, we use a popular performance metric, i.e., Area Under the Curve (AUC) score as presented in Section 2.4.

5.2.2 Hyper-parameters setting

The same configuration is used for all AE-based models in our experiments. This configuration is based on the AE-based models for detecting network attacks in the literature [136, 137, 121]. 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 number of layers including both the encoding layers and the decoding layers is 5. The ADAM algorithm [26] 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. For all datasets, we select 10% of training data as the validation sets for early stopping. This technique helps to stop the training process automatically. The performance of each model is evaluated on the validation set at the end of each 10 epoch. If the AUC score is reduced, the training procedure will be stopped.

5.2.3 Experimental Scenario

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 [60].

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 [35, 64]. These DTL models have same architecture which is based on AE. 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), the DTL method of using the MMD metric at the bottleneck layer (i.e., SMD-AE), and our model (MMD-AE).

The third set is to measure the processing time of the training and the predicting process of the above evaluated methods. The detailed results of three experimental sets are presented in the next section.

5.3 Results

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

5.3.1 Effectiveness of transferring information in MMD-AE

MMD-AE implements multiple transfers between encoding layers of AE_1 and AE_2 to force the latent representation of AE_2 closer to the latent representation of

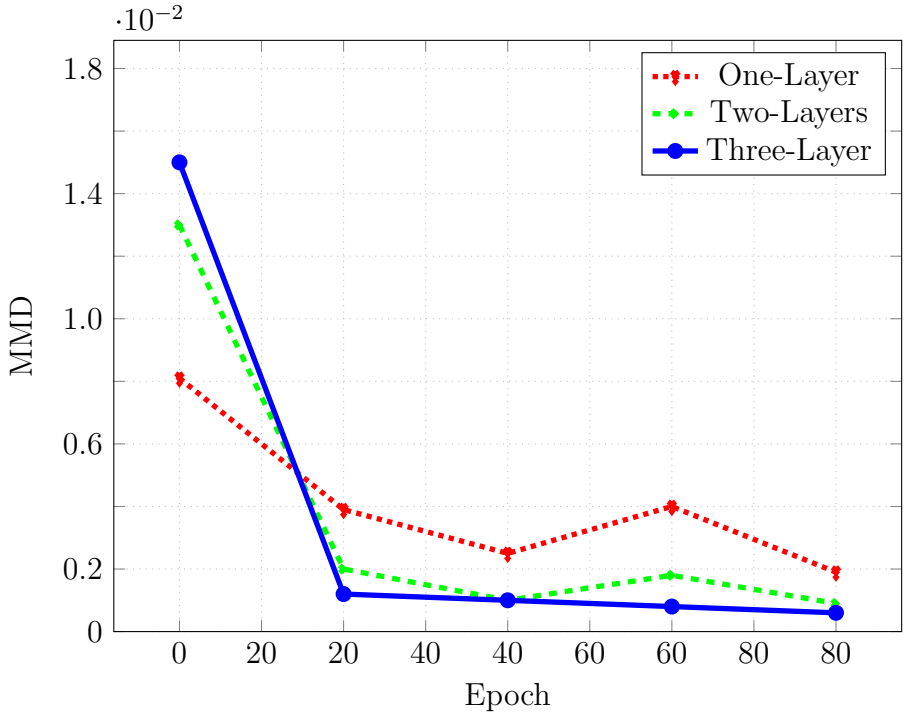


Figure 5.3 : MMD of latent representations of the source (IoT-1) and the target (IoT-2) when transferring task on one, two, and three encoding layers.

AE_1. To evaluate if MMD-AE achieved its objective we conducted an experiment in which, IoT-1 was 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 the distance is, the more information is transferred from the source domain (AE_1) to the target domain (AE_2). The result is presented in Fig. 5.3.

The figure shows that transferring task implemented on more layers results in a 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 more encoding layers. This result evidences that our proposed solution, MMD-AE, is more effective than the previous DTL models performing the transferring task only at the bottleneck layer of AE.

5.3.2 Performance Comparison

Table 5.1 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. Apparently, when an AE is trained on an IoT dataset (the source) and evaluated on other IoT datasets (the target), its performance is not effective. The reason for this ineffective result is that the predicting data in the target domain is far different from the training data in the source domain.

Conversely, the results of the three DTL models are much better than those 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 in 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*. For example, the AUC score is 0.937 compared to $0.600, 0.745$, and 0.764 of AE, SKL-AE and SMD-AE, respectively, when the source dataset is IoT-1 and the target dataset is IoT-3. The results on the other datasets are also similar to the results on IoT-3. These results demonstrate that implementing the transferring task in multiple layers of MMD-AE helps the model to transfer the label information from the source to the target domain more effectively. Subsequently, MMD-AE often achieves better results compared to AE, SKL-AE and SMD-AE in detecting IoT attacks in the target domain.

*The AUC scores of the proposed model in each scenario are presented by the bold text style.

Table 5.1 : AUC scores of AE, SKL-AE, SMD-AE, 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 5.2 : Training and testing of AE, SKL-AE, SMD-AE, and MMD-AE when the source domain is IoT-2 the target domain is IoT-1.

Models	AE	SKL-AE	SMD-AE	MMD-AE
Training Time (hours)	0.001	0.443	3.693	11.057
Predicting Time (seconds)	1.001	1.112	1.110	1.108

5.3.3 Processing Time Analysis

Table 5.2 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[†]. In this figure, 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 at every iteration while this calculation is not required in AE. Moreover, the training time of MMD-AE is much higher than that of SKL-AE and SMD-AE since MMD-AE needs to calculate the MMD distance between every encoding layer whereas SKL-AE and SMD-AE only calculate the distance metric in the bottleneck layer.

However, it is important to note that the predicting time of all DTL methods is mostly equal to that of AE. The reason is that the testing samples are only fitted to one AE in all tested models. For example, the 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 $778,810$ testing samples of the IoT-1 dataset.

5.4 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

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

problem of the "lack of labelled information" for the training detection model in ubiquitous IoT devices. Specifically, the labelled data and unlabelled data are 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. Compared to the previous DTL models, MMD-AE can operate at all the encoding layers instead of only the bottleneck layer.

We have carried out the 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, operating transformation at all the levels of encoding layers of the AEs helps to improve the effectiveness of the transferring process. Thus, the proposed model is meaningful when having label information in the source domain but no label information in the target domain. While the proposed DTL model requires more training time compared to previous models, the predicting time for each input data sample remains similar. It is important to note that the training process is typically conducted offline. Therefore, the slightly longer training time and the relatively equal predicting time are acceptable when considering the overall processing time measurement for the proposed DTL model.

Chapter 6

FEDERATED LEARNING FOR IOT ANOMALY DETECTION

The exponential proliferation of diverse IoT devices has resulted in a significant rise in the occurrence of emerging anomalies or attacks. From previous chapters, we have observed that the Deep Neural Network (DNN) models are able to detect network anomalies effectively. However, training DNN models for anomaly detection usually requires a lot of data available on the centre server. It may cause information leakage, especially with IoT networks. Federated Learning (FL) approaches can handle this problem by collaboratively training DNN models without sharing data. However, these approaches face the problem of data poisoning attacks that damages the training data of low-end IoT devices in the training process of FL.

To mitigate data poisoning attacks, this chapter proposes a novel FL system with the Shrink Denoising AutoEncoder (FL-SDAE). The loss function of SDAE includes two terms: a compressing term and a reconstruction term. The compressing term aims to project data of different IoT devices into the same latent representation space. Hence, aggregating different SDAE models on the server side is more effective. The reconstruction term helps SDAE reconstruct the original data from its corrupted version. As a result, the proposed SDAE model makes FL-SDAE robust to data poisoning attacks.

The rest of the chapter is organized as follows. The proposed models are pre-

sented in detail in Section 6.1. Section 6.2 shows the experimental settings for this chapter. The results and discussions of experiments are represented in Section 6.3. Finally, the conclusions are summarized in Section 6.4.

6.1 Federated Learning based on Shrink Denoising AutoEncoder

This section presents our proposed an FL system for IoT anomaly detection. First, we describe the new AE model in our FL system namely Shrink Denoising AutoEncoder (SDAE). Second, the FL system based on SDAE, i.e., FL-SDAE, is presented in detail.

6.1.1 Shrink Denoising Auto Encoder

This section proposes a new AE variant namely Shrink Denoising Auto Encoder (SDAE). SDAE is inspired by Shrink Auto Encoder (SAE) [121] which projects the latent representation space near the origin. However, SDAE is designed to train with noisy data. Thus, the representation of SDAE is more robust against data poisoning attacks.

Let $\mathcal{C}(\tilde{\mathbf{x}}|\mathbf{x})$ be a random noise that is a conditional distribution over a corrupted sample $\tilde{\mathbf{x}}$, given a data sample $\mathbf{x}$. $\mathcal{C}(\tilde{\mathbf{x}}|\mathbf{x})$ is calculated as Eq.2.6. Let define $\tilde{x}^i$ to be the corrupted version of the input data x^i obtained from $C(\tilde{\mathbf{x}}|\mathbf{x})$, the loss function for SDAE is represented as in Eq. 6.1.

$$\mathcal{L}\left(\mathbf{x}, \tilde{\mathbf{x}}, \phi, \theta\right) = \frac{1}{n}\sum_{i=0}^{n}\left(\mathbf{x}^i - p_\theta(q_\phi(\tilde{\mathbf{x}}^i))\right)^2 + \lambda * |\mathbf{z}^i|^2, \tag{6.1}$$

where q_ϕ and p_θ are the encoder and decoder parts, respectively, n is the number of data samples in a dataset and $\mathbf{z}^i$ is the latent representation of the input $\mathbf{x}^i$ for this AE structure.

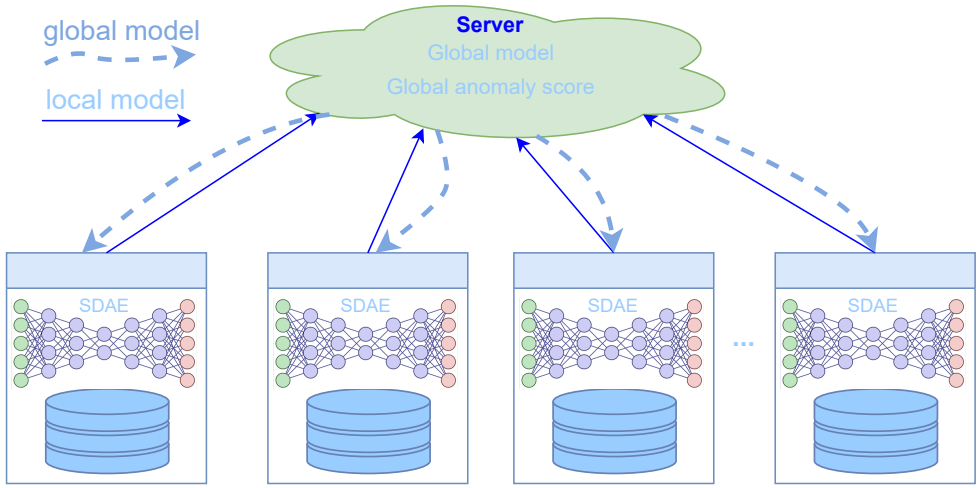


Figure 6.1 : Illustration of FL-SDAE.

The first term in the loss function of SDAE aims to reconstruct the input data $\mathbf{x}$ from the noisy input $\tilde{\mathbf{x}}^i$. The second term forces the latent representation space closely to the origin. The value of λ controls the trade-off between two loss terms. The training process of SDAE attempts to minimize this loss function.

6.1.2 Federated Learning based on SDAE

The new FL system named FL-SDAE is based on SDAE to mitigate the data poisoning attacks. The FL-SDAE architecture is illustrated in Fig. 6.1. The SDAE model is first trained on each client (e.g., an IoT device) using the benign data of that client. After that, the clients transmit their models, i.e., the model's weights, to the server for aggregating. The aggregation task is implemented by averaging the weights from the clients and then the server sends them, i.e., the aggregated weights or the global weights, back to all clients. After receiving the global weights, the clients update their weights using the global weights and continue the training process using it local data. The above process is repeated until it satisfies a termination condition. In the final round, the server calculates the final global model and shares this model with the clients. All clients calculate the anomaly score based on

the global model.

Algorithm 6 describes the training process of FL-SDAE in detail. After the initialization step, each client C_i trains its model SDAE_i and sends the model's weights w_i to the server. The sever calculates the global weights $\mathbf{w}_s = \frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_k$ and sends the global weights back to client C_i. Finally, the server and all clients can calculate the global anomaly score (AS), i.e., $\text{AS}_{\text{global}} = norm(\mathbf{w}_s^z)$ where $norm(\mathbf{w}_s^z)$ is the Euclidean norm of the weights of the hidden layer z of the global model. After the training process, the clients use $\text{AS}_{\text{global}}$ to detect the anomaly at their local. More precisely, if the value of the vector w_s^z is greater than $\text{AS}_{\text{global}}$, then this data is considered as an anomaly.

Algorithm 6 Training process of FL-SDAE.

Server: Initialization $\mathbf{w}_s$,
Clients: Initialization $\mathbf{w}_i$ (i = 1, ..., K) where K is the number of clients.
Set R: Number of training rounds,
for r = 0 to R **do**
  for i = 1 to K **do**
    Client C_i trains SDAE_i model,
    Client C_i sends its weight $\mathbf{w}_i$ to the server,
    Server updates the global weight $\mathbf{w}_s = \frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_k$,
    The server sends $\mathbf{w}_s$ to client C_i.
  end for
end for
The server and all clients calculate $\text{AS}_{\text{global}}$ is the value of vector $\mathbf{w}_s^z$.

FL-SDAE has two advantage features compared to previous FL systems for anomaly detection. First, the SDAE in FL-SDAE transforms the benign data of multiple clients into the same latent representation space allowing it to effectively aggregate the weights from multiple clients. In other words, FL-SDAE is capable of integrating various clients of different characteristics.

Second, SDAE introduces $\mathbf{x}_{\text{noise}}$ to its loss function helping to model to robust to the noisy data. Thus FL-SDAE is able to mitigate the influence of the data

Datasets	NBIoT	CICIDS	NSL-KDD	Spambase	CTU13-08
Training set	19408	260596	58910	1962	50955
Validating set	19407	45183	15747	460	7895
Testing set	38093	90365	22544	921	15790

Table 6.1 : The number of data samples in each dataset.

poisoning attacks.

6.2 Experimental Setting

6.2.1 Dataset

We evaluate the performance of the anomaly detection models using five well-known datasets, namely NBIoT [137], CICIDS [49], NSL-KDD [67], Spambase [25], CTU13-08 [108] datasets. These datasets are described in Section 2.5. Each dataset is divided into three sets, i.e., the training set, the validation set, and the testing set with the ratio 7/1/2. In the training and validating set, we only use the benign or normal samples for training and selecting the models, respectively. The anomalies of the training set are utilized to poison the training process in the poisoning attack scenarios. The testing sets have both normal and abnormal data samples. The number of data samples for each dataset is described in Table 6.1.

6.2.2 Parameter Settings

We compare the effectiveness of FL-SDAE with other FL systems including FedDetect [118], FL-based Denoising AE (FL-DAE) [47], FL-based Variational AE (FL-VAE) [27], FL-based SAE (FL-SAE) [121]. The configurations of all AE-based models are same our experiments. λ is set to 1 for both SAE and SDAE. The number of hidden layers is 5, and the size of the bottleneck layer is determined using the formula $m = [1 + \sqrt{n}]$ in [120], where n is the number of input features. The batch size is 128, and the learning rate is 10^{-4}. To promote convergence, the weights of

the AE models are initialized using the technique proposed in Glorot et al. [132]. We train these networks using the ADAM optimization algorithm [29].

The number of clients is 200 for all experiments. The maximum of training rounds is $10,000$. In addition, the early stopping approach is utilized to save training resources. Specifically, after 200 training rounds, the training process will stop if the performance of the global model on the validation dataset has not improved.

6.2.3 Experimental Scenario

All experiments have been implemented in Python using the PyTorch [88] and Scikit-Learn [107] frameworks. The same computing platform (Operating System: Ubuntu 20.04 (64 bit), Intel Core i7-10700K CPU, 16 cores, NVIDIA GeForce RTX 2060 GPU, and 16GB RAM memory) was used in experiments in this chapter. We have conducted two groups of experiments to investigate various aspects of the proposed system, i.e., the ability of to mitigate poisoning attacks and the properties of FL system.

- Ability of Mitigating Poisoning Attacks: Evaluate the influence of two popular types of poisoning attacks, i.e., the dirty label attacks and the clean label attacks on the FL systems for anomaly detection.
- Property of FL systems: Inspect some properties of using FL system and explain the reason of the effectiveness for the proposed FL system, i.e., FL-SDAE.

6.3 Experimental Result and Discussion

This section first examines the ability of FL-SDAE in mitigating poisoning attacks. After that, some characteristics of FL-SDAE are analysed.

Malicious data ratio (%)		25				50				75			
Attacked client (%)		20	40	60	80	20	40	60	80	20	40	60	80
Dataset	**Model**	**AUC score**											
NBIoT	FedDetect	0.772	0.770	0.770	0.767	0.766	0.766	0.766	0.767	0.766	0.765	0.765	0.764
	FL-VAE	0.793	0.793	0.793	0.789	0.793	0.793	0.791	0.792	0.793	0.791	0.79	0.79
	FL-DAE	0.773	0.773	0.773	0.773	0.773	0.773	0.773	0.773	0.773	0.773	0.773	0.773
	FL-SAE	0.878	0.879	0.879	0.88	0.876	0.879	0.88	0.88	0.876	0.877	0.877	0.880
	FL-SDAE	**0.941**	**0.941**	**0.942**	**0.942**	**0.942**	**0.942**	**0.942**	**0.891**	**0.941**	**0.942**	**0.889**	**0.886**
CICIDS	FedDetect	0.599	0.598	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587
	FL-VAE	0.549	0.554	0.552	0.553	0.555	0.555	0.556	0.553	0.551	0.555	0.553	0.551
	FL-DAE	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587
	FL-SAE	0.806	0.802	0.793	0.785	0.806	0.801	0.792	0.742	0.806	0.803	0.794	0.532
	FL-SDAE	**0.807**	**0.807**	**0.805**	**0.801**	**0.807**	**0.808**	**0.805**	**0.800**	**0.807**	**0.808**	**0.807**	**0.802**
NSLKDD	FedDetect	0.845	0.846	0.846	0.847	0.844	0.846	0.847	0.844	0.845	0.847	0.848	0.845
	FL-VAE	0.802	0.803	0.805	0.802	0.803	0.804	0.801	0.801	0.808	0.804	0.802	0.794
	FL-DAE	0.845	0.845	0.846	0.847	0.845	0.846	0.846	0.843	0.845	0.847	0.847	0.844
	FL-SAE	0.870	0.870	0.870	0.869	0.869	0.870	0.870	0.869	0.870	0.869	0.868	0.867
	FL-SDAE	**0.875**	**0.874**	**0.874**	**0.873**	**0.875**	**0.874**	**0.873**	**0.872**	**0.875**	**0.874**	**0.873**	**0.872**
Spambase	FedDetect	0.514	0.503	0.501	0.495	0.511	0.499	0.499	0.476	0.51	0.497	0.498	0.475
	FL-VAE	0.496	0.49	0.489	0.506	0.497	0.491	0.491	0.506	0.495	0.498	0.489	0.501
	FL-DAE	0.512	0.502	0.499	0.496	0.511	0.503	0.501	0.478	0.513	0.502	0.501	0.474
	FL-SAE	0.706	0.675	0.621	0.636	0.691	0.666	0.647	0.632	0.64	0.652	0.641	0.600
	FL-SDAE	**0.735**	**0.728**	**0.727**	**0.706**	**0.733**	**0.716**	**0.708**	**0.685**	**0.724**	**0.715**	**0.699**	**0.687**
CTU13-08	FedDetect	0.864	0.866	0.806	0.807	0.864	0.862	0.808	0.807	0.862	0.864	0.807	0.806
	FL-VAE	0.801	0.801	0.8	0.802	0.802	0.8	0.8	0.802	0.802	0.8	0.799	0.798
	FL-DAE	0.865	0.863	0.806	0.807	0.864	0.863	0.809	0.807	0.864	0.866	0.807	0.806
	FL-SAE	0.924	0.918	0.917	0.918	0.923	0.917	0.917	0.917	0.922	0.916	0.915	0.914
	FL-SDAE	**0.933**	**0.931**	**0.928**	**0.925**	**0.934**	**0.93**	**0.927**	**0.923**	**0.933**	**0.931**	**0.928**	**0.924**

Table 6.2 : Result of tested methods in clean label attacks.

6.3.1 Ability of Mitigating Poisoning Attacks

This subsection evaluates the robustness of FL-SDAE to mitigate poisoning attacks. We compare the effectiveness of FL-SDAE with other four FL systems when training processes are attacked by two types of the data poisoning attacks, i.e., the dirty label attacks and the clean label attacks.

Clean Label Attacks

The clean label attack is usually conducted by injecting noisy data into the training dataset. This is one of the most common poisoning attacks. We simulate this attack by injecting the noisy samples into the training data of each client. The amount of noisy samples for each malicious client are set at 25%, 50%, and 75% over all training samples. Moreover, the number of attacked clients, i.e., the clients are injected by the noisy data, is set at 20%, 40%, 60%, and 80% over a totally 200 clients. Amount of poisoning data rate is from 5% (for 25% malicious data of 20% attacked clients) to 60% (for 75% malicious data of 80% attacked clients). The results of this experiment are shown in Table 6.2.

First, this table shows that all FL frameworks are influenced by the data poisoning attacks. When the amount of poisoning attacks is increased, the accuracy of the FL systems is decreased. For example, the AUC score of FedDetect reduces from 0.772 to 0.767 when the rate of attacked clients increases from 20% to 80% and the rate of noisy samples is at 25%. A similar trend is also observed with all tested methods on all experimental configurations.

Second, Table 6.2 also shows that the proposed model, i.e. FL-SDAE, outperforms all other models based on AE in all experimented datasets. For instance, the AUC score of FL-SDAE is 0.941 when the rate of poisoning sample is 25% and the rate of the attacked client is 20%. This value is much higher compared to the AUC score of FedDetect (0.772), FL-VAE (0.793), FL-DAE (0.773) and FL-SAE (0.878)

Malicious data ratio(%)		25				50				75			
Attacked client(%)		20	40	60	80	20	40	60	80	20	40	60	80
Dataset	**Model**	**AUC score**											
NBIoT	FedDetect	0.771	0.677	0.677	0.677	0.677	0.677	0.677	0.676	0.677	0.677	0.677	0.676
	FL-VAE	0.789	0.678	0.678	0.678	0.679	0.679	0.679	0.678	0.679	0.678	0.678	0.678
	FL-DAE	0.772	0.772	0.771	0.769	0.773	0.772	0.771	0.766	0.772	0.772	0.770	0.765
	FL-SAE	0.864	0.849	0.836	0.828	0.862	0.841	0.819	0.796	0.86	0.837	0.811	0.788
	FL-SDAE	**0.918**	**0.866**	**0.85**	**0.841**	**0.917**	**0.859**	**0.841**	**0.827**	**0.907**	**0.857**	**0.838**	**0.808**
CICIDS	FedDetect	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587
	FL-VAE	0.549	0.554	0.552	0.553	0.555	0.555	0.556	0.553	0.551	0.555	0.553	0.554
	FL-DAE	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587	0.587
	FL-SAE	**0.799**	**0.788**	**0.765**	**0.748**	**0.793**	**0.761**	**0.742**	**0.723**	**0.787**	**0.747**	**0.725**	0.667
	FL-SDAE	0.797	0.777	0.753	0.739	0.79	0.754	0.721	0.68	0.786	0.744	0.681	**0.675**
NSLKDD	FedDetect	0.863	0.860	**0.861**	0.812	0.815	0.811	0.811	0.811	0.818	0.811	0.812	0.812
	FL-VAE	0.807	0.804	0.802	0.806	0.802	0.806	0.800	0.805	0.802	0.805	0.802	0.803
	FL-DAE	0.852	0.86	0.861	0.812	0.855	**0.861**	0.811	0.811	0.857	**0.861**	0.811	0.812
	FL-SAE	0.869	0.855	0.848	0.837	0.857	0.851	0.838	0.816	0.856	0.844	0.832	0.811
	FL-SDAE	**0.873**	**0.863**	0.847	**0.838**	**0.876**	0.856	**0.837**	**0.828**	**0.875**	0.846	**0.834**	**0.821**
Spambase	FedDetect	0.511	0.511	0.510	0.510	0.512	0.510	0.508	0.504	0.511	0.509	0.497	0.497
	FL-VAE	0.504	0.509	0.505	0.506	0.506	0.486	0.481	0.480	0.505	0.485	0.494	0.48
	FL-DAE	0.511	0.511	0.510	0.509	0.512	0.509	0.507	0.500	0.512	0.510	0.504	0.496
	FL-SAE	0.703	0.664	0.629	0.625	0.696	0.632	0.630	0.606	0.686	0.633	0.619	0.589
	FL-SDAE	**0.729**	**0.712**	**0.683**	**0.668**	**0.717**	**0.689**	**0.662**	**0.635**	**0.709**	**0.668**	**0.635**	**0.611**
CTU13-08	FedDetect	0.911	0.902	0.903	0.901	0.901	0.918	0.901	0.866	0.900	0.900	0.893	0.859
	FL-VAE	0.802	0.803	0.802	0.804	0.801	0.802	0.802	0.803	0.801	0.804	0.802	0.802
	FL-DAE	0.911	0.921	0.911	0.912	0.919	0.918	0.911	0.900	0.915	0.912	0.893	0.859
	FL-SAE	0.921	0.917	0.908	0.901	0.920	0.902	0.901	0.893	0.913	0.901	0.891	0.840
	FL-SDAE	**0.933**	**0.928**	**0.919**	**0.912**	**0.931**	**0.923**	**0.914**	**0.905**	**0.927**	**0.922**	**0.911**	**0.891**

Table 6.3 : Results of tested methods in dirty label attacks.

on the same configuration. The reason for the superior performance of FL-SDAE is that FL-SDAE is trained with noisy data instead of clean data. This makes FL-SDAE more robust to noisy samples injected by the clean label attack. The results in this table prove that FL-SDAE is less influenced by the clean label attack than the other tested approaches.

Dirty Label Attacks

The dirty label attack is often conducted by flipping the label of data samples in the training dataset. This is also one of the most common poisoning attacks. We simulate this attack by flipping the data samples in the training data of each client. The number of attacked clients and the ratio of malicious data per attacked client are similar to the configurations in Section 6.3.1. The results of this experiment are shown in Table 6.3.

First, the results in Table 6.3 show that all tested FL systems are more seriously affected by the dirty label attack compared to the clean label attack. Thus, the AUC scores of these systems are considerably reduced when the rate of the attack increases. For example, the AUC score of FedDetect is reduced from 0.771 to 0.677 when the rate of attacked clients increases from 20% to 80% and the rate of noisy samples is at 25%. Similar results are also observed for the other methods.

Second and more importantly, the table shows that FL-SDAE is still more effective than the other FL systems against this attack. Specifically, the AUC score of FL-SDAE is the highest value in all experiments. We also observe that two methods, i.e., FL-SAE and FL-SDAE are often better than the three others. The reason is that FL-SAE and FL-SDAE attempt to project the normal data closely to the origin and their representation is more robust against the dirty attack. Moreover, the AUC scores of FL-SDAE are higher than that of FL-SAE. This is because FL-SDAE is trained with noisy samples instead of clean samples as FL-SAE. Thus FL-SDAE is

better than FL-SAE in neutralizing the noisy data attacks as the dirty label attack.

6.3.2 Characteristics of the FL-SDAE

This subsection inspects some characteristics of FL-SDAE to explain its superior performance. First, we investigate FL-SDAE in a scenario where poisoning attacks do not exist. Second, we analyze the representation of FL-SDAE to explain its effectiveness in mitigating the poisoning attack.

The Performance of FL-SDAE without Poisoning Attacks

This subsection investigates the accuracy of FL systems when they are trained on datasets without data poisoning attacks. Fig. 6.2 presents the AUC scores of the tested FL systems. This figure illustrates that FL-SAE and FL-SDAE achieve better results compared with the others. The reason is that both FL-SAE and FL-SDAE are trained to constrain the benign data of all clients into a compact region at the origin. Thus, the global anomaly scores of these system are aggregated more effectively. As a result, the FL-SAE and FL-SDAE are better than the other systems when the poisoning attacks do not exist. Moreover, the AUC score of FL-SDAE and that of FL-SAE are mostly equal in this experiment. This is different from the results in Section 6.3.1 where the AUC score of FL-SDAE is always significantly higher than the value of FL-SAE. This confirms that FL-SDAE is specifically effective against the data poisoning attacks.

Visualization

This section visualizes the representation of FedDetect, FL-SAE, and FL-SDAE when they are trained on the NBIoT dataset. To facilitate the visualization, the number of neurons at the bottle layer of these models is set at 2. After training, the models are used to represent the data in the testing set NBIoT dataset in the

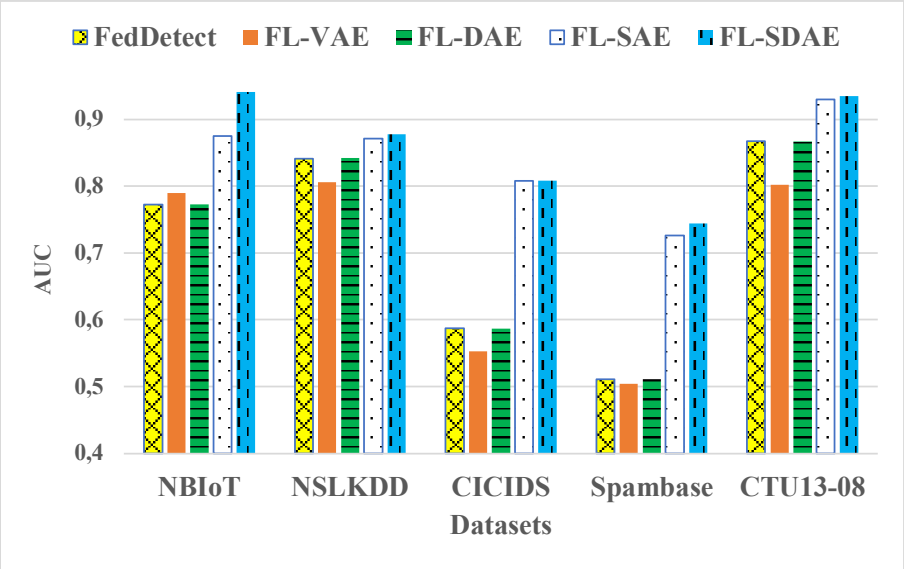
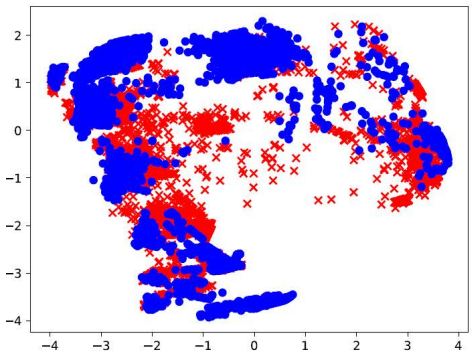


Figure 6.2 : The accuracy of FL systems without data poisoning attacks.

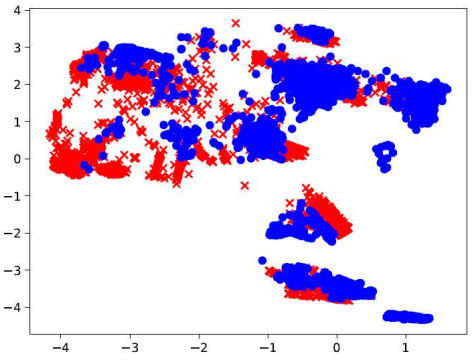
2D latent representation space. The representations of benign and abnormal data samples are shown in the blue circle points and red cross points, respectively.

Figs. 6.3(a), 6.3(c), and 6.3(e) show the representation of data samples of FedDetect, FL-SAE, and FL-SDAE without the data poisoning attacks, respectively. We can observe that FedDetect projects the benign and abnormal data samples stochastically in the latent representation space. Each client may represent its data differently in the latent representation space. Conversely, all clients of FL-SAE and FL-SDAE compress the benign data samples into the origin. However, the abnormal data samples are not trained and FL-SAE and FL-SDAE project them far from the region of origin. As a result, all clients share the same AS to distinguish benign and abnormal data samples. This helps the servers of FL-SAE and FL-SDAE aggregate the AS more effectively than others.

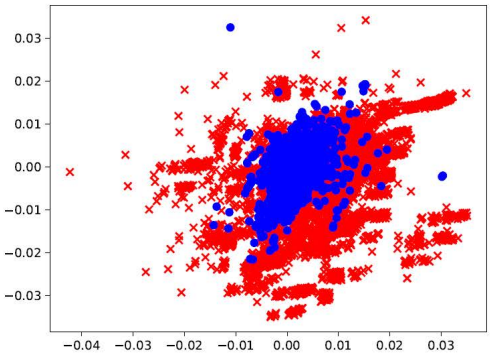
Moreover, Figs. 6.3(b), 6.3(d), and 6.3(f) illustrate the data representation of FedDetect, FL-SAE, and FL-SDAE, respectively when the data poisoning attacks persist. In this experiment, the number of malicious clients is 20% and 25% training



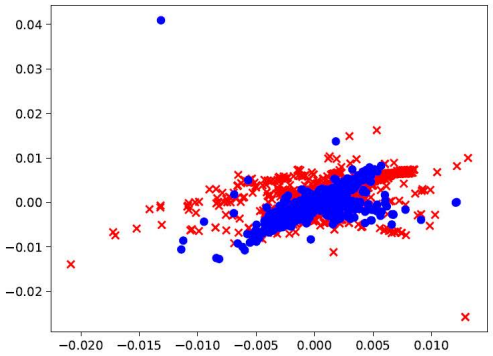
(a) FedDetect



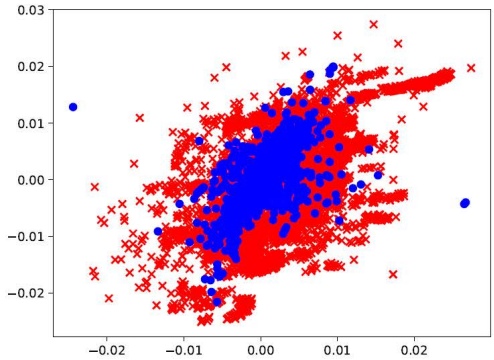
(b) Attack FedDetect.



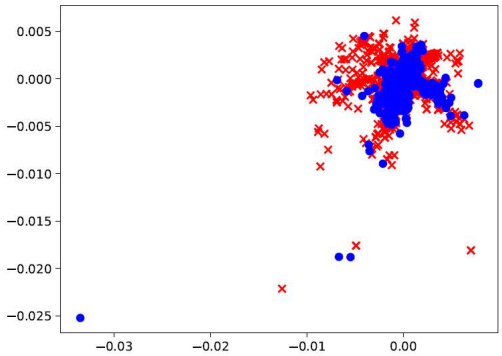
(c) FL-SAE.



(d) Attack FL-SAE.



(e) FL-SDAE.



(f) Attack FL-SDAE.

Figure 6.3 : Illustration of latent representation of AE-based models trained in FL using NBIoT.

data of each malicious client is poisoned by randomly flipping data labels. We can observe that the FedDetect system still stochastically represent the benign and abnormal of all clients. By contrast, FL-SAE and FL-SDAE still represents the benign more compact. Moreover, Fig. 6.3(d) and Fig. 6.3(f) show that the representation of the benign samples of FL-SDAE is more compact than that of FL-SAE. Thus, the FL-SDAE system is less influenced by the data poisoning attack.

6.4 Conclusion

This chapter has proposed a novel FL system for the IoT anomaly detection problems, named as Federated Learning based on Shrink Denoising AutoEncoder (FL-SDAE) that is robust to data poisoning attacks. Our proposed FL-SDAE framework can collaboratively train machine learning models for multiple training clients. This ensures the data privacy of the IoT network because data transmission between IoT devices is unnecessary. In the FL-SDAE system, the proposed SDAE model projects the benign data of multiple clients into the same latent representation space. This helps aggregate the global SDAE model at the server more effectively. Moreover, the SDAE model is less influenced by noisy data. Accordingly, the FL-SDAE system is more robust against the data poisoning attacks compared with other FL systems. The effectiveness of FL-SDAE is demonstrated via extensive experiments in five AD datasets.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Contribution

This thesis aims to design novel DNN models to handle four challenges of the network anomaly detection (AD) problem. The proposed solutions aim to deal with new/unknown attacks, imbalanced data, the lack of labelled data, and the vulnerability to data poisoning attacks in the training process.

In the first study, we propose a novel representation learning models to facilitate the subsequent machine learning-based AD. Specifically, we develop three regularized versions of AEs to learn a latent representation from the network data samples. The bottleneck layers of these regularized AEs, i.e., MVAE, MAE, and MDAE, trained in a supervised manner using normal data and known network attacks will then be used as the new input data features to classify normal and abnormal data samples. The experimental results demonstrate that the new latent representation can significantly enhance the performance of supervised learning methods in detecting unknown network attacks. This proposed solution can be applied for AD when we have enough training data and groundtruth information.

The second study aims to handle the imbalance problem of network AD datasets. Usually, the AD datasets are often dominated by normal data, and machine learning models trained on those imbalanced datasets are ineffective in detecting anomalies. Thus, we propose generative models to generate synthesized abnormal data samples

including CDAAE and CDAAE-KNN. The synthesized attacks are merged with the original data to form the augmented dataset. This improves the accuracy of AD on the imbalanced datasets. We can use this proposed solution to generate network attacks that are difficult to collect for building AD systems.

In the third study, we resolve "the lack of label information" in the AD problem by using the DTL technique. In many 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 similar. Thus, we develop a DTL technique named as MMD-AE 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 DTL technique can help classifiers to identify anomalies more accurately. This proposed study can be applied on the IoT anomaly detection where we are unable to collect data from all types of IoT devices.

In the fourth study, we propose a novel FL system for the IoT AD problem, named as FL-SDAE, that is robust to data poisoning attacks. Our proposed models ensure the data privacy of the IoT network because the data transmission between IoT devices is unnecessary. In the FL-SDAE system, the proposed SDAE model projects the benign data of multiple clients into the same latent representation space. This helps aggregating the global SDAE model at the server more effectively. Moreover, the SDAE model is trained on the noisy data. Thus, the proposed model is more robust against the data poisoning attacks compared with other FL systems. When the data privacy is considered more seriously, we can utilize the proposed FL system to enhance the robust AD models on all clients without transferring training data.

7.2 Limitation

However, the thesis remains two limitations. First, the advantages of representation learning models come with the cost of running time. When using a neural network to learn the representation of input features, the executing time is often much longer than using only classifiers on the original feature spaces. The proposed representation learning models proposed in Chapter 3 also face with this drawback. However, it can be seen in Chapter 3 that the average time of predicting one sample of the representation learning models is acceptable in real applications.

Second, in the CDAAE model proposed in Chapter 4, 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. Thus, the proposed generative models may be less effective with other data distributions different from the Gaussian distribution. Moreover, generative models may struggle to generate network traffic data samples that accurately capture the complexity and nuances of real-world network behavior. Additionally, generative models usually have the mode collapse problem that refers to a situation where the generative model fails to capture the full diversity of the target distribution and instead produces a limited set of similar samples. In the context of network traffic data, this means that the generated samples may exhibit a limited range of behaviors and fail to represent the wide variety of anomalies and network dynamics that occur in real-world scenarios. This leads to reducing the quality of generated data samples.

Third, the denoising component of SDAE makes the training process more robust against noise of training data, i.e., clean label attacks. Thus, the FedSDAE model, introduced in Chapter 6, exhibits greater robustness against clean label attacks compared to dirty label attacks. Dirty label attacks involve replacing the data labels in the training data, thereby diminishing the effectiveness of the Federated Learning

(FL) schemes. Furthermore, one limitation of using FL based on AEs for AD is the potential loss of global knowledge. In FL, each client trains its own local AE model using its own data. While this decentralized approach helps preserve data privacy and security, it can result in a fragmented view of the overall network behavior. Additionally, the performance of the FL heavily relies on the participation and data quality of the individual clients. If some clients have limited or biased data, it can negatively impact the overall accuracy and generalizability of the model. Moreover, the communication overhead and resource constraints associated with federated learning can pose challenges in terms of scalability and real-time anomaly detection in large-scale networks.

7.3 Future work

Building upon this research, there are a number of directions for future work arising from the thesis. First, in Chapter 3, there are hyper-parameters of the proposed representations of AE-based models (i.e., μ_{y^i}) which 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 proposed in Chapter 4, 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 attributes of anomaly behaviors to CDAAE, the synthesized data will be more similar to the original data. One can extend our work in different ways.

Third, the current DTL model is developed based on AutoEncoder (AE) in Chapter 5. In the future, we will attempt to extend this model based on other neural networks such as Deep Adaptation Network (DAN), Adversarial Discriminative Domain Adaptation (ADDA), Maximum Classifier Discrepancy (MCD), and Con-

ditional Domain Adversarial Network (CDAN) [143]. Additionally, pre-training the AE models on large-scale datasets, such as publicly available network traffic data or other related datasets, can help the model learn general representations of normal network behavior. This pre-trained models can then be fine-tuned using the client's specific data to adapt to the unique characteristics of the target network.

Fourth, the FedSDAE model, introduced in Chapter 6, is more robust on the clean label attacks than on the dirty label attacks. In the future, we will conduct a thorough analysis of the impact of dirty label attacks on FL schemes. Subsequently, the thesis will enhance the robustness of deep learning models trained on clients to better withstand these attacks. Furthermore, the proposed FL assumes that the training data has the same distribution across all clients. However, this assumption does not always hold true in real network environments due to various types of network attacks. Consequently, we will develop personalized FL schemes to handle the different data distributions present in the training data. Additionally, FL often involves limited communication bandwidth and resource-constrained clients. Employing model compression techniques such as quantization, pruning, or knowledge distillation can help reduce the communication overhead and model size without significantly sacrificing performance. Optimizing the FL algorithms for efficiency can also improve scalability and real-time anomaly detection capabilities.

Appendix A

Supplement in Chapter 4

This Appendix presents the application of the Bayesian optimization to tuning the hyper-parameters of the deep generative models in Chapter 4. The tested models include Auxiliary Classifier Generative Adversarial Network (ACGAN), Conditional Variational AutoEncoder (CVAE), Supervised Adversarial AutoEncoder (SAAE), Conditional Denoising Adversarial AutoEncoder (CDAAE), and Conditional Denoising Adversarial AutoEncoder-K Nearest Neighbor (CDAAE-KNN).

A.1 Hyper-parameter Tuning using Bayesian Optimization

Selecting appropriate hyper-parameters is crucial for the performance of deep neural networks. In Chapter 4, we conducted an experiment to find the appropriate hyper-parameters for each network model on each dataset using the Bayesian Optimization [70] technique in which 10% of each training dataset is used for the validation. The Bayesian Optimization is a model-based method for finding the minimum of a function. This technique uses a surrogate model, e.g., a Gaussian Process, to select the most promising set of hyper-parameters to assess. Thus it allows to reduce the number of times the objective function needs to be evaluated. Recently, the Bayesian optimization method has been applied to tuning the hyper-parameters of deep neural networks [2, 117].

For all tested models, we tune five parameters, i.e., the optimization methods, the learning rate, the number of layers in each sub-neural network in the tested

models (i.e., En, De, Ge, and Di), the hidden size, and the batch size. These are considered as the most important parameters to the performance of deep neural networks [63]. For our proposed models, we also tune α_1, α_2, k for CDAAE-KNN and the noise factor for both CDAAE and CDAAE-KNN. The range of the values for tuning in each parameter is presented in Table A.1.

The process of using the Bayesian Optimization technique for tuning the hyper-parameters of a deep neural network on a dataset including the following steps:

- Step 1: The range of values for each hyper-parameter in Table A.1 and the deep network model are input to the Bayesian Optimization.
- Step 2: The Bayesian Optimization generates a promising set of specific values for each parameter including the optimization method and the learning rate in Table A.1.
- Step 3: The deep neural network is trained on the training data using the set of values given by the Bayesian Optimization.
- Step 4: After training the deep neural network, the Bayesian Optimization evaluates the effectiveness of the current set of values based on the accuracy of the model on the validation set.
- Step 5: Repeat step 2 to step 4 until the termination condition is met.

A.2 Result of Hyper-parameter tuning for the deep generative learning models

After applying the Bayesian Optimization, we obtained the set of appropriate values for the hyper-parameters on each dataset. These values are presented in Table A.2. We then used these values to train the deep network models on the

Table A.1 : Range of values for tuning in each hyper-parameter.

Hyper-parameters	**Value Range**
Optimization methods	Adam, Gradient Descent (GD), RMSprop, Adadelta
Learning rate (LR)	$0.0001 \rightarrow 0.1$
Number of layers	$2 \rightarrow 5$
Hidden size	$5 \rightarrow 50$
Batch size	$32 \rightarrow 256$
Noise factor	$0 \rightarrow 0.1$
α_1	$1 \rightarrow 50$
α_2	$1 \rightarrow 50$
k	$20 \rightarrow 100$

training data. After training, the model is used to generate the synthesized data to balance the current skewed dataset. Values obtained by Bayesian Optimization for the hyper-parameters on each dataset (N means that the parameter is not used in the corresponding model and (1), (2), (3), (4) present the Cloud IDS, CIC-IDS 2017, NSL-KDD, UNSW-NB15 datasets, respectively).

Table A.2 : Values obtained by Bayesian Optimization for the hyper-parameters.

Models	**Attacks**	**Datasets**	**Optimizes**	**LR**	**# layers**	**Hidden size**	**Batch size**	**Noise factor**	α_1	α_2	k
ACGAN	TCPLand	(1)	RMSprop	0.069	2	22	162	N	N	N	N
	PingOfDeath	(1)	GD	0.056	2	45	121	N	N	N	N
	Slowloris	(1)	RMSprop	0.029	2	24	86	N	N	N	N
	Slowloris	(2)	Adam	0.022	3	42	78	N	N	N	N
	SlowHttp	(2)	Adam	0.051	3	38	69	N	N	N	N
	BruteForce	(2)	Adam	0.044	3	35	92	N	N	N	N
	SSH-Patator	(2)	Adam	0.003	2	19	255	N	N	N	N
	Botnet	(2)	Adadelta	0.042	2	22	155	N	N	N	N
	Network attacks	(3)	GD	0.023	4	5	32	N	N	N	N
	Network attacks	(4)	Adam	0.013	4	6	256	N	N	N	N
CVAE	TCPLand	(1)	Adam	0.042	3	22	99	N	N	N	N
	PingOfDeath	(1)	Adadelta	0.027	4	21	114	N	N	N	N
	Slowloris	(1)	GD	0.026	4	23	83	N	N	N	N
	Slowloris	(2)	Adam	0.012	3	25	88	N	N	N	N
	SlowHttp	(2)	Adam	0.051	2	35	68	N	N	N	N
	BruteForce	(2)	Adam	0.011	3	38	122	N	N	N	N
	SSH-Patator	(2)	Adadelta	0.032	3	42	94	N	N	N	N
	Botnet	(2)	Adam	0.055	2	44	88	N	N	N	N
	Network attacks	(3)	Adam	0.097	4	5	255	N	N	N	N
	Network attacks	(4)	Adadelta	0.044	2	50	255	N	N	N	N
SAAE	TCPLand	(1)	Adadelta	0.094	4	43	244	N	N	N	N
	PingOfDeath	(1)	GD	0.094	3	20	72	N	N	N	N
	Slowloris	(1)	Adadelta	0.084	3	6	255	N	N	N	N
	Slowloris	(2)	Adam	0.012	2	32	80	N	N	N	N
	SlowHttp	(2)	Adam	0.001	2	34	94	N	N	N	N
	BruteForce	(2)	Adam	0.010	2	38	72	N	N	N	N
	SSH-Patator	(2)	Adam	0.021	3	45	32	N	N	N	N
	Botnet	(2)	RMSprop	0.034	3	48	65	N	N	N	N
	Network attacks	(3)	Adam	0.050	3	50	32	N	N	N	N
	Network attacks	(4)	Adadelta	0.099	3	49	118	N	N	N	N
CDAAE	TCPLand	(1)	Adadelta	0.094	4	43	244	0.044	N	N	N
	PingOfDeath	(1)	RMSprop	0.094	4	32	156	0.095	N	N	N
	Slowloris	(1)	Adam	0.064	3	29	203	0.035	N	N	N
	Slowloris	(2)	Adadelta	0.014	2	38	66	0.012	N	N	N
	SlowHttp	(2)	Adam	0.032	3	35	80	0.034	N	N	N
	BruteForce	(2)	Adam	0.041	2	31	72	0.080	N	N	N
	SSH-Patator	(2)	RMSprop	0.059	3	34	89	0.011	N	N	N
	Botnet	(2)	Adam	0.054	3	32	108	0.043	N	N	N
	Network attacks	(3)	Adadelta	0.068	5	49	256	0.094	N	N	N
	Network attacks	(4)	Adam	0.013	4	50	35	0.046	N	N	N
CDAAE-KNN	TCPLand	(1)	Adam	0.047	3	20	245	0.090	39	37	31
	PingOfDeat	(1)	Adadelta	0.018	4	26	198	0.079	22	22	51
	Slowloris	(1)	RMSprop	0.071	5	24	163	0.030	46	17	92
	Slowloris	(2)	Adadelta	0.023	2	25	179	0.012	19	34	78
	SlowHttp	(2)	Adam	0.001	3	29	100	0.043	22	25	40
	BruteForce	(2)	Adam	0.068	3	36	198	0.045	15	23	38
	SSH-Patator	(2)	Adam	0.036	3	20	225	0.016	28	42	70
	Botnet	(2)	Adam	0.045	2	22	250	0.021	16	24	56
	Network attacks	(3)	RMSprop	0.059	4	6	209	0.092	10	17	74
	Network attacks	(4)	GD	0.018	2	5	246	0.098	29	44	21

BIBLIOGRAPHY

[1] *9 Distance Measures in Data Science*. https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.. 2020.

[2] Aaron Klein et al. "Fast bayesian optimization of machine learning hyperparameters on large datasets". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 528–536.

[3] Alireza Makhzani et al. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).

[4] Amit Praseed and P. Santhi Thilagam. "DDoS Attacks at the Application Layer: Challenges and Research Perspectives for Safeguarding Web Applications". In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 661–685. DOI: 10.1109/COMST.2018.2870658.

[5] Andrea Dal Pozzolo et al. "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*. 2013, pp. 24–31.

[6] Anna L Buczak and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection". In: *IEEE Communications surveys & tutorials* 18.2 (2015), pp. 1153–1176.

[7] Antonia Creswell and Anil Anthony Bharath. "Denoising adversarial autoencoders". In: *IEEE Transactions on Neural Networks and Learning Systems* 99 (2018), pp. 1–17.

[8] Arthur Gretton et al. “A kernel method for the two-sample-problem”. In: *Advances in neural information processing systems*. 2007, pp. 513–520.

[9] Ayobami S. Edun et al. “Anomaly Detection of Disconnects Using SSTDR and Variational Autoencoders”. In: *IEEE Sensors Journal* 22.4 (2022), pp. 3484–3492. DOI: 10.1109/JSEN.2022.3140922.

[10] B. Du et al. “Stacked Convolutional Denoising Auto-Encoders for Feature Representation”. In: *IEEE Transactions on Cybernetics* 47.4 (Apr. 2017), pp. 1017–1027. DOI: 10.1109/TCYB.2016.2536638.

[11] Beibei Li et al. “DeepFed: Federated Deep Learning for Intrusion Detection in Industrial Cyber–Physical Systems”. In: *IEEE Transactions on Industrial Informatics* 17.8 (2021), pp. 5615–5624. DOI: 10.1109/TII.2020.3023430.

[12] Bernhard Schölkopf, John Platt, and Thomas Hofmann. “Greedy Layer-Wise Training of Deep Networks”. In: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. 2007, pp. 153–160.

[13] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[14] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[15] C. Dietz et al. “IoT-Botnet Detection and Isolation by Access Routers”. In: *2018 9th International Conference on the Network of the Future (NOF)*. Nov. 2018, pp. 88–95. DOI: 10.1109/NOF.2018.8598138.

[16] Chang Wan, Rong Pan, and Jiefei Li. “Bi-weighting domain adaptation for cross-language text classification”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[17] Chaoqun Yang et al. "Multiple Attacks Detection in Cyber-Physical Systems Using Random Finite Set Theory". In: *IEEE Transactions on Cybernetics* 50.9 (2020), pp. 4066–4075. DOI: 10.1109/TCYB.2019.2912939.

[18] Chetak Kandaswamy et al. "Improving transfer learning accuracy by reusing stacked denoising autoencoders". In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2014, pp. 1380–1387.

[19] Chuanqi Tan et al. "A survey on deep transfer learning". In: *International Conference on Artificial Neural Networks*. Springer. 2018, pp. 270–279.

[20] Congyingzi Zhang and Robert 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. Alexandria, Virginia: Society for Computer Simulation International, 2015, pp. 8–15. ISBN: 978-1-5108-0100-4.

[21] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[22] Danni Yuan et al. "Intrusion detection for smart home security based on data augmentation with edge computing". In: *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE. 2020, pp. 1–6.

[23] David Powers. "Evaluation: From precision, recall and fmeasure to roc, informedness, markedness and correlation". In: *Journal of Machine Learning Technologies* 2 (Jan. 2007), pp. 37–63.

[24] Dawei Chen et al. "FedSVRG Based Communication Efficient Scheme for Federated Learning in MEC Networks". In: *IEEE Transactions on Vehicular Technology* 70.7 (2021), pp. 7300–7304. DOI: 10.1109/TVT.2021.3089431.

[25] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[26] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[27] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[28] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[29] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[30] Eric Tzeng et al. "Adversarial discriminative domain adaptation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7167–7176.

[31] Eric Tzeng et al. "Deep domain confusion: Maximizing for domain invariance". In: *arXiv preprint arXiv:1412.3474* (2014).

[32] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[33] Farhan Ullah et al. "Cyber Security Threats Detection in Internet of Things Using Deep Learning Approach". In: *IEEE Access* 7 (2019), pp. 124379–124389. DOI: 10.1109/ACCESS.2019.2937347.

[34] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[35] Fuzhen Zhuang et al. "Supervised representation learning: Transfer learning with deep autoencoders". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[36] Gan Sun et al. "Data Poisoning Attacks on Federated Machine Learning". In: *IEEE Internet of Things Journal* 9.13 (2022), pp. 11365–11375. DOI: 10.1109/JIOT.2021.3128646.

[37] Geming Xia et al. "Poisoning Attacks in Federated Learning: A Survey". In: *IEEE Access* 11 (2023), pp. 10708–10722. DOI: 10.1109/ACCESS.2023.3238823.

[38] *Global cloud industry outlook 2024 - A glimpse into tomorrow's Tech tapestry.* https://www.marketsandmarkets.com.. Accessed: 2024-03-10.

[39] Guhdar AA MULLA, Yıldırım DEMİR, and Masoud HASSAN. "Combination of PCA with SMOTE oversampling for classification of high-dimensional imbalanced data". In: *Bitlis Eren Üniversitesi Fen Bilimleri Dergisi* 10.3 (2021), pp. 858–869.

[40] 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)*. Nov. 2018, pp. 1857–1862.

[41] 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)*. Nov. 2018, pp. 1857–1862. DOI: 10.1109/ICARCV.2018.8581205.

[42] Hanan Hindy et al. "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets". In: *CoRR* abs/1806.03517 (2018).

[43] Hien M. Nguyen, Eric W. Cooper, and Katsuari Kamei. "Borderline oversampling for imbalanced data classification". In: *International Journal of Knowledge Engineering and Soft Data Paradigms* 3.1 (2011), pp. 4–21.

[44] Hirokazu Kameoka et al. "ACVAE-VC: Non-parallel voice conversion with auxiliary classifier variational autoencoder". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.9 (2019), pp. 1432–1443.

[45] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning". In: *International Conference on Intelligent Computing*. Springer. 2005, pp. 878–887.

[46] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[47] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[48] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[49] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." In: *ICISSp* 1 (2018), pp. 108–116.

[50] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *International Conference on Information Systems Security and Privacy (ICISSP)*. 2018, pp. 108–116.

[51] *Implementation of Deep Belief Network*. `https://github.com/JosephGatto/Deep-Belief-Networks-Tensorflow.`.

[52] Israr Ahmed et al. "Security in the Internet of Things (IoT)". In: *2017 Fourth HCT Information Technology Trends (ITT)*. Oct. 2017, pp. 84–90.

[53] J. Dromard, G. Roudière, and P. Owezarski. "Online and Scalable Unsupervised Network Anomaly Detection Method". In: *IEEE Transactions on Network and Service Management* 14.1 (Mar. 2017), pp. 34–47. DOI: 10.1109/TNSM.2016.2627340.

[54] J. Zhang, M. Zulkernine, and A. Haque. "Random-Forests-Based Network Intrusion Detection Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5 (Sept. 2008), pp. 649–659.

[55] Jair Cervantes et al. "PSO-based method for SVM classification on skewed data sets". In: *Neurocomputing* 228 (2017), pp. 187–197.

[56] Jiale Zhang et al. "PoisonGAN: Generative Poisoning Attacks Against Federated Learning in Edge Computing Systems". In: *IEEE Internet of Things Journal* 8.5 (2021), pp. 3310–3322. DOI: 10.1109/JIOT.2020.3023126.

[57] Jie Lu et al. "Transfer learning using computational intelligence: a survey". In: *Knowledge-Based Systems* 80 (2015), pp. 14–23.

[58] João Marcelo Ceron et al. "Improving IoT Botnet Investigation Using an Adaptive Network Layer". In: *Sensors (Basel)* 19.3 (2019), p. 727.

[59] Juan Zhao et al. "Transfer learning for detecting unknown network attacks". In: *EURASIP Journal on Information Security* 2019 (2019), pp. 1–13.

[60] Jun Yang, Rong Yan, and Alexander G Hauptmann. "Cross-domain video concept detection using adaptive svms". In: *Proceedings of the 15th ACM international conference on Multimedia*. 2007, pp. 188–197.

[61] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), p. 9.

[62] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. "Survey of real-time processing technologies of iot data streams". In: *Journal of Information Processing* 24.2 (2016), pp. 195–202.

[63] Klaus Greff et al. "LSTM: A search space odyssey". In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[64] Long Wen, Liang Gao, and Xinyu Li. "A new deep transfer learning based on sparse auto-encoder for fault diagnosis". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.1 (2017), pp. 136–144.

[65] Ly Vu et al. "Learning Latent Distribution for Distinguishing Network Traffic in Intrusion Detection System". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8762015.

[66] 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)*. Aug. 2016, pp. 147–156. DOI: 10.1109/ARES.2016.64.

[67] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications* (2009), pp. 1–6.

[68] Manos Antonakakis et al. "Understanding the Mirai Botnet". In: *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Aug. 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9.

[69] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[70] Matthias Feurer and Frank Hutter. "Hyperparameter optimization". In: *Automated Machine Learning*. Springer, 2019, pp. 3–33.

[71] Maxime Oquab et al. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[72] Michele De Donno et al. "DDoS-capable IoT malwares: Comparative analysis and Mirai investigation". In: *Security and Communication Networks* 2018 (2018).

[73] Mingsheng Long et al. "Deep transfer learning with joint adaptation networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2208–2217.

[74] Mingsheng Long et al. "Domain adaptation with randomized multilinear adversarial networks". In: *arXiv preprint arXiv:1705.10667* (2017).

[75] Mingsheng Long et al. "Unsupervised domain adaptation with residual transfer networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 136–144.

[76] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[77] Mohamed Bekkar, Hassiba Djema, and T.A. Alitouche. "Evaluation measures for models assessment over imbalanced data sets". In: *Journal of Information Engineering and Applications* 3 (Jan. 2013), pp. 27–38.

[78] Mohammad Lotfollahi et al. "Deep packet: A Novel Approach for Encrypted Traffic Classification using Deep Learning". In: *Soft Computing* (2019), pp. 1–14. ISSN: 1432-7643. DOI: https://doi.org/10.1007/s00500-019-04030-2.

[79] Mohammad A. Salahuddin et al. "Chronos: DDoS Attack Detection Using Time-Based Autoencoder". In: *IEEE Transactions on Network and Service Management* 19.1 (2022), pp. 627–641. DOI: 10.1109/TNSM.2021.3088326.

[80] Muraleedharan N. and Janet B. "A deep learning based HTTP slow DoS classification approach using flow data". In: *ICT Express* 7.2 (2021), pp. 210–214. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.08.005. URL: https://www.sciencedirect.com/science/article/pii/S2405959520300965.

[81] N. C. Luong et al. "Data Collection and Wireless Communication in Internet of Things (IoT) Using Economic Analysis and Pricing Models: A Survey". In: *IEEE Communications Surveys Tutorials* 18.4 (Fourth quarter 2016), pp. 2546–2590. DOI: 10.1109/COMST.2016.2582841.

[82] Nitesh V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[83] Nour Moustafa and Jill 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)*. IEEE. 2015, pp. 1–6.

[84] *NSL-KDD dataset [Online]*. http://nsl.cs.unb.ca/NSL-KDD/. Accessed: 2018-04-10.

[85] O. Ibidunmoye, A. Rezaie, and E. Elmroth. "Adaptive Anomaly Detection in Performance Metric Streams". In: *IEEE Transactions on Network and Service Management* 15.1 (Mar. 2018), pp. 217–231. DOI: 10.1109/TNSM.2017.2750906.

[86] O. Y. Al-Jarrah et al. "Data Randomization and Cluster-Based Partitioning for Botnet Intrusion Detection". In: *IEEE Transactions on Cybernetics* 46.8 (Aug. 2016), pp. 1796–1806. DOI: 10.1109/TCYB.2015.2490802.

[87] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408.

[88] *Pytorch framework*. `https://pytorch.org`. [Online; accessed 10-Jan-2022]. 2022.

[89] Qian Yu and Wai Lam. "Data augmentation based on adversarial autoencoder handling imbalance for learning to rank". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 411–418.

[90] Qinbin Li et al. "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection". In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: 10.1109/TKDE.2021.3124599.

[91] R. Vinayakumar et al. "Deep Learning Approach for Intelligent Intrusion Detection System". In: *IEEE Access* 7 (2019), pp. 41525–41550. DOI: 10.1109/ACCESS.2019.2895334.

[92] R. Sahila Devi, R. Bharathi, and P. Krishna Kumar. "Investigation on Efficient Machine Learning Algorithm for DDoS Attack Detection". In: *2023 International Conference on Computer, Electrical & Communication Engineering (ICCECE)*. 2023, pp. 1–5. DOI: 10.1109/ICCECE51049.2023.10085248.

[93] Radu Horaud. "Audio-visual Speech Enhancement Using Conditional Variational Auto-Encoder". In: *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH AND LANGUAGE PROCESSING* (2020), p. 1.

[94] Raghavendra Chalapathy and Sanjay Chawla. "Deep learning for anomaly detection: A survey". In: *arXiv preprint arXiv:1901.03407* (2019).

[95] Raneel Kumar, Sunil Pranit Lal, and Alok Sharma. "Detecting denial of service attacks in the cloud". In: *2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE. 2016, pp. 309–316.

[96] *Real-time stream processing for Internet of Things*. `https://medium.com/@exastax/real-time-stream-processing-for-internet-of-things-24ac529f75a3.`.

[97] Rohan Doshi, Noah Apthorpe, and Nick Feamster. "Machine Learning DDoS Detection for Consumer Internet of Things Devices". In: *2018 IEEE Security and Privacy Workshops (SPW)*. 2018, pp. 29–35. DOI: `10.1109/SPW.2018.00013`.

[98] Ronald Doku and Danda B. Rawat. "Mitigating Data Poisoning Attacks On a Federated Learning-Edge Computing Network". In: *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. 2021, pp. 1–6. DOI: `10.1109/CCNC49032.2021.9369581`.

[99] Rushi Longadge and Snehalata Dongre. "Class imbalance problem in data mining review". In: *arXiv preprint arXiv:1305.1707* (2013).

[100] Ruslan Salakhutdinov and Hugo Larochelle. "Efficient learning of deep Boltzmann machines". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 693–700.

[101] S. Khattak et al. "A Taxonomy of Botnet Behavior, Detection, and Defense". In: *IEEE Communications Surveys Tutorials* 16.2 (Second 2014), pp. 898–924. DOI: `10.1109/SURV.2013.091213.00134`.

[102] S. Wang et al. "Training deep neural networks on imbalanced data sets". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. July 2016, pp. 4368–4374. DOI: 10.1109/IJCNN.2016.7727770.

[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)*. 2019, pp. 1–6.

[104] S. S. Chawathe. "Monitoring IoT Networks for Botnet Activity". In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. Nov. 2018, pp. 1–8. DOI: 10.1109/NCA.2018.8548330.

[105] Salman H. Khan et al. "Cost-Sensitive Learning of Deep Feature Representations From Imbalanced Data". In: *IEEE Transaction Neural Network Learning System* 29.8 (2018), pp. 3573–3587.

[106] Sarin E Chandy et al. "Cyberattack Detection Using Deep Generative Models with Variational Inference". In: *Journal of Water Resources Planning and Management* 145.2 (2018), p. 04018093.

[107] *Scikit-learn framework*. https://scikit-learn.org/stable/. [Online; accessed 10-Jan-2022]. 2022.

[108] Sebastian Garcia et al. "An empirical comparison of botnet detection methods". In: *computers & security* 45 (2014), pp. 100–123.

[109] Sebastián García, Alejandro Zunino, and Marcelo Campo. "Botnet behavior detection using network synchronism". In: *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*. IGI Global, 2012, pp. 122–144.

[110] Shuiping Gou et al. "Distributed transfer network learning based intrusion detection". In: *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE. 2009, pp. 511–515.

[111] Siman Huang et al. "Defending against Poisoning Attack in Federated Learning Using Isolated Forest". In: *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*. 2022, pp. 224–229. DOI: 10.1109/ICCCR54399.2022.9790094.

[112] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[113] *Sklearn tutorial [Online]*. http://scikit-learn.org/stable/. Accessed: 2018-04-24.

[114] Stefano Longari et al. "CANnolo: An Anomaly Detection System Based on LSTM Autoencoders for Controller Area Network". In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1913–1924. DOI: 10.1109/TNSM.2020.3038991.

[115] Sven Nomm and Hayretdin Bahsi. "Unsupervised Anomaly Based Botnet Detection in IoT Networks". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2018), pp. 1048–1053.

[116] *TCPtrace tool for analysis of TCP dump files*. http://www.tcptrace.org/.. 2020.

[117] Tinu Theckel Joy et al. "Fast hyperparameter tuning using Bayesian optimization with directional derivatives". In: *Knowledge-Based Systems* 205 (2020), p. 106247.

[118] Tuo Zhang et al. "Federated Learning for Internet of Things". In: *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. SenSys '21. Coimbra, Portugal: Association for Computing Machinery, 2021, pp. 413–

419. ISBN: 9781450390972. DOI: 10.1145/3485730.3493444. URL: https://doi.org/10.1145/3485730.3493444.

[119] 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*. Springer. 2016, pp. 717–726.

[120] Van Loi Cao, Miguel Nicolau, and James McDermott. "A Hybrid Autoencoder and Density Estimation Model for Anomaly Detection". In: *PPSN*. 2016.

[121] Van Loi Cao, Miguel Nicolau, and James McDermott. "Learning Neural Representations for Network Anomaly Detection". In: *IEEE Transactions on Cybernetics* 49.8 (Aug. 2019), pp. 3074–3087. DOI: 10.1109/TCYB.2018.2838668.

[122] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (July 2009), 15:1–15:58. ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: http://doi.acm.org/10.1145/1541880.1541882.

[123] Vidwath Raj, Sven Magg, and Stefan Wermter. "Towards effective classification of imbalanced data with convolutional neural networks". In: *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*. Springer. 2016, pp. 150–162.

[124] Vladimir Borgiani et al. "Toward a Distributed Approach for Detection and Mitigation of Denial-of-Service Attacks Within Industrial Internet of Things". In: *IEEE Internet of Things Journal* 8.6 (2021), pp. 4569–4578. DOI: 10.1109/JIOT.2020.3028652.

[125] Wei Wang et al. "Malware traffic classification using convolutional neural network for representation learning". In: *2017 International Conference on*

Information Networking (ICOIN). Jan. 2017, pp. 712–717. DOI: 10.1109/ICOIN.2017.7899588.

[126] Wei Yang Bryan Lim et al. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* 22.3 (2020), pp. 2031–2063. DOI: 10.1109/COMST.2020.2986024.

[127] Wei Yang Bryan Lim et al. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* 22.3 (2020), pp. 2031–2063. DOI: 10.1109/COMST.2020.2986024.

[128] Wenke Lee and Dong Xiang. "Information-theoretic measures for anomaly detection". In: *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE. 2001, pp. 130–143.

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

[130] X. Liu et al. "Ensemble Transfer Learning Algorithm". In: *IEEE Access* 6 (2018), pp. 2389–2396. DOI: 10.1109/ACCESS.2017.2782884.

[131] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[132] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *AISTATS*. 2010.

[133] Xing Xu et al. "Toward effective intrusion detection using log-cosh conditional variational autoencoder". In: *IEEE Internet of Things Journal* 8.8 (2020), pp. 6187–6196.

[134] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. "Exploratory Undersampling for Class-Imbalance Learning". In: *IEEE Transaction Systems, Man, and Cybernetics, Part B* 39.2 (2009), pp. 539–550.

[135] Xuyang Jing, Zheng Yan, and Witold Pedrycz. "Security data collection and data analytics in the Internet: A survey". In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 586–618.

[136] Yair Meidan et al. "Detection of Unauthorized IoT Devices using Machine Learning Techniques". In: *arXiv preprint arXiv:1709.04647* (2017).

[137] Yair Meidan et al. "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders". In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22. DOI: 10.1109/MPRV.2018.03367731.

[138] Yang Chen, Xiaoyan Sun, and Yaochu Jin. "Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.10 (2020), pp. 4229–4238. DOI: 10.1109/TNNLS.2019.2953131.

[139] Yanjiao Chen et al. "Data Poisoning Attacks in Internet-of-Vehicle Networks: Taxonomy, State-of-The-Art, and Future Directions". In: *IEEE Transactions on Industrial Informatics* 19.1 (2023), pp. 20–28. DOI: 10.1109/TII.2022.3198481.

[140] Yi Liu et al. "Deep Anomaly Detection for Time-Series Data in Industrial IoT: A Communication-Efficient On-Device Federated Learning Approach". In: *IEEE Internet of Things Journal* 8.8 (2021), pp. 6348–6358. DOI: 10.1109/JIOT.2020.3011726.

[141] Yonghui Xu et al. "A unified framework for metric transfer learning". In: *IEEE Transactions on Knowledge and Data Engineering* 29.6 (2017), pp. 1158–1171.

[142] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[143] Yuchen Zhang et al. "Bridging Theory and Algorithm for Domain Adaptation". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 7404–7413.

[144] Z. Liu H. Chen and P. Zhang. "ACGAN-based Data Augmentation Integrated with Long-term Scalogram for Acoustic Scene Classification". In: *ArXiv, abs/2005.13146* (2020).

[145] Zhipeng Li et al. "Intrusion detection using convolutional neural networks for representation learning". In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 858–866.