박 사 학 위 논 문
Ph.D. Dissertation

무선 네트워크에서 침입 탐지 성능 향상을 위한 심층 추상화 기법 연구

Utilizing Deep Abstraction for Higher Intrusion-Detection in Wireless Networks

2018

아미난또 무함마드 에르자 (Aminanto, Muhamad Erza)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

무선 네트워크에서 침입 탐지 성능 향상을 위한 심층 추상화 기법 연구

2018

아미난또 무함마드 에르자

한 국 과 학 기 술 원

전산학부

무선 네트워크에서 침입 탐지 성능 향상을 위한 심층 추상화 기법 연구

아미난또 무함마드 에르자

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2018년 05월 10일

심사위원장	Kwangjo Kim	(인)
심 사 위 원	MyungChul Kim	(인)
심 사 위 원	SoonJu Hyun	(인)
심 사 위 원	Ho-jin Choi	(인)
심 사 위 원	Seungwon Shin	(인)

Utilizing Deep Abstraction for Higher Intrusion-Detection in Wireless Networks

Muhamad Erza Aminanto

Advisor: Kwangjo Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
May 10, 2018

Approved by

Kwangjo Kim
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS
20145562

. 무선 네트워크에서 침입 탐지 성능 향상을 위한 심층 추상화 기법 연구. 전산학부 . 2018년. 90+iv 쪽. 지도교수: 김광조. (영문 논문)
Muhamad Erza Aminanto. Utilizing Deep Abstraction for Higher Intrusion-Detection in Wireless Networks. School of Computing . 2018. 90+iv pages. Advisor: Kwangjo Kim. (Text in English)

초 록

최근 모바일 기술의 발전으로 인해 IoT 지원 장치가 보급되고 일상 생활에 통합되었습니다. 연결된 장치는 유비쿼터스이며 거대하고 고차원적이고 복잡한 데이터를 생성합니다. 이러한 거대한 데이터에서 정상적인 동작과 다른 악의적 인 활동을 관찰하는 것은 어려운 작업입니다. 그러나 기능 학습은이 작업을 해결하는 하나의 솔루션이 될 수 있습니다. 따라서이 논문은 딥 피쳐 추출 및 선택 (D-FES)과 완전히 감독되지 않는 방법을 사용하여 새로운 피쳐 학습 기법을 제안한다. D-FES는 스택 피쳐 추출과 가중치 피쳐 선택을 결합합니다. 누적 자동 인코딩은 원시 입력에서 관련 정보를 재구성하여보다 의미있는 표현을 제공 할 수 있습니다. 표현은 클러스터링 방법으로 활용 될 수도 있습니다. 그런 다음이를 기존의 얕은 구조화 된 기계 학습에서 영감을 얻은 수정 된 가중치 적용 선택과 결합합니다. 우리는 마침내 기계 학습 모델의 편향과 교육 및 테스트의 계산 복잡성을 줄이기 위해 응축 된 기능 세트의 기능을 입증합니다. 우리는 Aegean Wi-Fi Intrusion Dataset (AWID)이라 불리는 Wi-Fi 네트워크 데이터 세트에서 우리의 제안 된 계획이 99.918%의 가장 검출 정확도와 false를 달성함으로써 D-FES의 유용성과 유용성을 입증하는지 검증합니다 알람 비율 0.012%. D-FES는 모든 종류의 공격 탐지 중에 99.910%의 정확도를 달성했습니다.

핵 심 낱 말 중첩된 오토 인코더, 심층 추상화, 추출, 선택, 클러스터링, 침입 탐지, 위장 공격

Abstract

The recent advances in mobile technologies have resulted in IoT-enabled devices becoming more pervasive and integrated into our daily lives. The connected devices are ubiquitous, generating huge, high-dimensional and complex data. Observing malicious activities, which deviate from normal behavior, in such colossal data is a challenging task. Feature learning, however, can be one solution to solve this task. This dissertation thus proposes novel feature-learning schemes using Deep-Feature Extraction and Selection (D-FES) and fully unsupervised method. D-FES combines stacked feature extraction and weighted feature selection. The stacked auto-encoding is capable of providing abstractions (representations) that are more meaningful by reconstructing the relevant information from its raw inputs. The representations could also be leveraged as a clustering method. We then combine this with modified weighted feature selection inspired by an existing shallow-structured machine learning. We finally demonstrate the ability of the condensed set of features to reduce the bias of a machine learning model and the computational complexity of training and testing. We verify our proposed schemes on a Wi-Fi network dataset, called, the Aegean Wi-Fi Intrusion Dataset (AWID), prove the usefulness and the usability of the D-FES by achieving an impersonation detection accuracy of 99.918% and a false alarm rate of 0.012%. D-FES also achieved 99.910% of accuracy during all class of attacks detection.

Keywords Stacked auto encoder, Deep abstraction, Extraction, selection, clustering, Intrusion detection, Wireless networks, Impersonation attack

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Research Background

The rapid growth of the Internet has led to a significant increase in wireless network traffic in recent years. According to a worldwide telecommunication consortium, Mobile and Wireless Communications Enablers for the Twenty-Twenty Information Society (METIS) [11], a proliferation of 5G and Wi-Fi networks is expected to occur in the next decades. They believe that avalanche of mobile and wireless traffic volume will occur due to the development of society needs to be fulfilled. Applications such as e-learning, e-banking, and e-health would spread and become more mobile. By 2020 [1] wireless network traffic is anticipated to account for two-thirds of total Internet traffic — with 66% of IP traffic expected to be generated by Wi-Fi and cellular devices only as shown in Fig. 1.1. Cyber-attacks have become an immense growing rate as Internet of Things (IoT) are widely used these days [12].

IBM [1] reported an enormous account hijacked during 2016 and spam emails are four times higher than the previous year. Common attacks noticed in the same report including brute-force, malvertising, phishing, SQL injection, DDoS, malware, *etc* as depicted in Fig. 1.2. Majority of malwares are accounted as a ransomware (85% of malwares existed in a year are a ransomware). These attacks might leak sensitive data or disrupt normal operations which leads to an enormous financial loss. The most popular companies impacted by security incidents are financial services-related companies. Followed by information and communications, manufacture, retail, and healthcare [1]. Wireless networks such as IEEE 802.11 have been widely deployed to provide users with mobility and flexibility in the form of high-speed local area connectivity. However, other issues such as privacy and security have raised. The rapid spread of IoT-enabled devices has resulted in wireless networks becoming to both passive and active attacks, the number of which has grown dramatically [12]. Examples of these attacks are impersonation, flooding, and injection attacks. The wide and rapid spread of computing devices using Wi-Fi networks creates complex, large, and high-dimensional data, which cause confusion when capturing attack properties and force us to strengthen our security measures in our system.

An Intrusion Detection System (IDS) is one of the most common components of every network security infrastructure [13] including wireless networks [14]. Machine-learning techniques have been well adopted as the primary detection algorithm in IDS owing to their model-free properties and learnability [15]. Leveraging the recent development of machine-learning techniques such as deep learning [16] can be expected to bring significant benefits concerning improving existing IDSs particularly for detecting impersonation attacks in large-scale networks. Based on the detection method, IDS can be classified into three types; misuse, anomaly, and specification-based IDS. A misuse-based IDS also known as a signature-based IDS [17] detects any attack by checking whether the attack characteristics match previously stored signatures or patterns of attacks. This type of IDS is suitable for detecting known attacks; however, new or unknown attacks are difficult to detect.

The vast and rapid spread of computing devices using Wi-Fi networks creates complex, large, and high-dimensional data, which confuse when capturing attack properties. Feature learning acts as an

[1] Cisco Visual Networking Index: Forecast and Methodology 2015—2020, published at www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

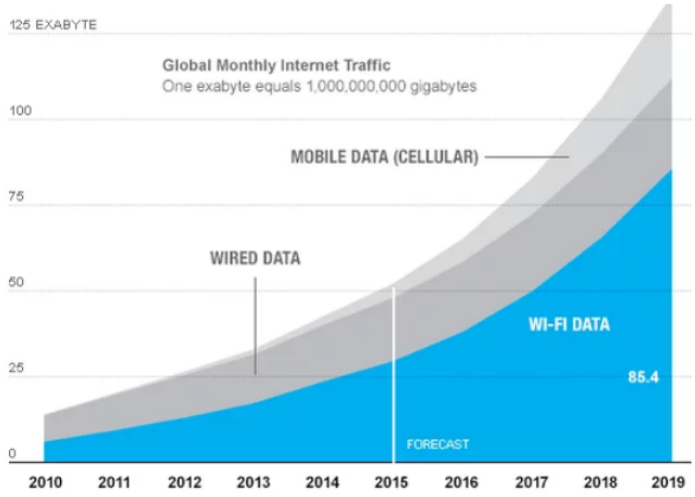


Figure 1.1: Forecast of Internet Data Traffic by CISCO

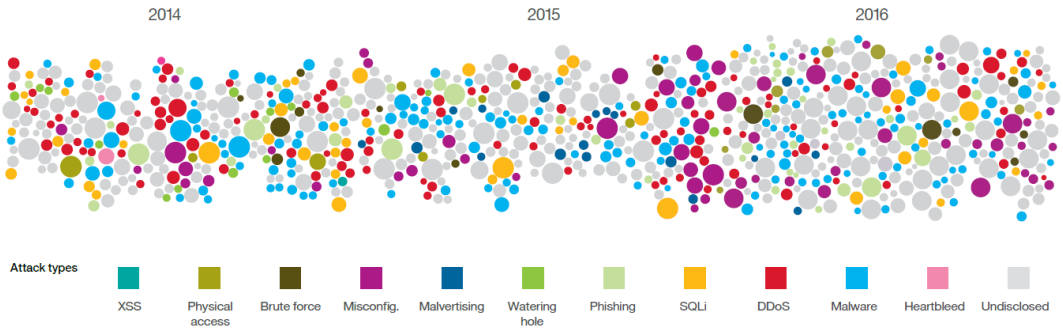


Figure 1.2: Sampling by Security Incidents by IBM [1]

essential tool for improving the learning process of a machine-learning model. It consists of feature construction, extraction, and selection. Feature construction expands the original features to enhance their expressiveness, whereas feature extraction transforms the original features into a new form and feature selection eliminates unnecessary features [18]. Feature learning is a key to improve the performance of existing machine-learning-based IDSs. Therefore, we leverage deep abstraction by building a stacked autoencoder network to improve feature selection and clustering tasks.

1.2 Our Contribution

Large-scale Wi-Fi network leads to a huge, complex and high-dimensional data. In [2], impersonation attack is claimed to be the most difficult to detect as shown by low detection rate in Fig. 1.3 which is 4,419 out of 20,079 (22%) impersonation instances only can be detected. Therefore, we focus on the following problem:

- How to improve impersonation detector in Wi-Fi network dataset?
- How to develop a fully unsupervised IDS using deep learning approach?
- How to achieve compact and efficient feature representation for complex and high-dimensional data like AWID dataset, to achieve highest detection rate and lowest false alarm rate?
- How to develop a general model that can detect not only impersonation attack, but also other attacks?

To answer above questions, we propose enhanced feature extraction (as a deep abstraction), feature selection and clustering schemes for reducing the data complexity and learning hidden meaning from massive data. We verify our scheme by detecting impersonation attacks in a large wi-fi networks dataset, AWID dataset. Owing to the scale and complexity of recent data, building a machine-learning-based IDS has become a daunting task. One of the critical contributions of this study is the introduction of novel Deep-Feature Extraction and Selection (D-FES), which improves its feature learning process by combining stacked feature extraction with weighted feature selection. The feature extraction of Stacked Auto Encoder (SAE) is capable of transforming the original features into a more meaningful representation by reconstructing its input and providing a way to check that the relevant information in the data has been captured. SAE can be efficiently used for unsupervised learning on a complex dataset. Unlike supervised learning, unsupervised learning does not require a labeled dataset for training. Unsupervised learning capability is of critical importance as it allows a model to be built to detect new attacks without creating costly labels or dependent variables. SAE is constructed by stacking additional unsupervised feature learning layers and can be trained by using greedy methods for each extra layer. We train a new hidden layer by training a standard supervised neural network with one hidden layer. The output of the previously trained layer is taken as pre-training initialization for the next layer and as the extracted features.

In summary, the main contributions of this study are as follows:

- Design of an impersonation attack detector using a short set of features without modifying any protocol and without using any additional measurement.

Normal	Flooding	Injection	Impersonation	Classified As
497199	8971	11899	12716	Normal
2123	5974	0	0	Flooding
3027	0	13655	0	Injection
14187	1473	0	4419	Impersonation

Figure 1.3: The best detection performance for impersonation attack [2]

- Abstraction of raw features using a deep learning technique. The extracted features are measured once more using weighted feature selection techniques such that a measure-in-depth technique is proposed.
- Design of the proposed D-FES, which can be implemented in a real wireless network setup because an unbalanced dataset is used for testing purposes.
- Design of the generalized of D-FES that can detect any attack classes.
- Design of an enhanced clustering method using a stacked autoencoder that improves a classical clustering method, k-means clustering significantly.

1.3 Dissertation Structure

The remainder of this dissertation is organized as follows. In Chapter 2, we briefly explain IDS terminology and classification. We investigates various deep learning techniques in Chapter3. We describe our proposed feature extraction and selection and fully unsupervised IDS in Chapter 4. In Chapter 5, we evaluate our proposed methods. Chapter 6 introduces previous work which related to our work. Chapter 7 concludes this dissertation along future research direction of this work.

Chapter 2. Intrusion Detection Systems

2.1 Definition

Intrusion Detection System (IDS) becomes a standard security measure in computer networks. Unlike a firewall, IDS usually located inside the network to monitor all internal traffics as shown in Fig. 2.1. One may consider having both a firewall and IDS to protect the network. IDS is defined as automation of intrusion detection process which is a process of finding events of violation of security policies or standard security practices in computer networks [19]. Besides identifying the security incidents, IDS also has other functions: documenting existing threats and deterring adversaries [19]. IDS requires particular properties which acts as a passive countermeasure, monitors whole or part of networks only and aims high attack detection rate and low false alarm rate.

2.2 Classification

We can divide IDSs based on their placement in the network and methodology used. By the positioning of the IDS module in the network, we might distinguish IDSs to 3 classes: network-based, host-based, and hybrid-based IDSs. The first IDS, network-based IDS as shown in Fig. 2.2 puts the IDS module in the network which can monitor whole the network traffics. This IDS has a big picture of the network makes it has a better understanding the network in overall. On the other hand, Fig. 2.3 shows the host-based IDS which places the IDS module on each client of the network. The module can only see the ingoing or outgoing traffics of the corresponding client leads to detail monitoring of the particular client. Two types of IDSs have specific drawbacks– the network-based IDS might burden of the workload then misses some malicious activities, while the host-based IDS does not have the overview of the whole network but having less workload than the network-based IDS. Therefore, the hybrid-based IDS as shown in Fig. 2.4 places IDS modules in the network as well as clients to monitor both specific clients and network overview at the same time.

In the latter case, based on the detection method, IDSs can be divided into three different types: misuse, anomaly, and specification-based IDSs. A misuse-based IDS, known as a signature-based IDS [17], looks for any malicious activities by matching the known signatures or patterns of attacks with the monitored traffics. This IDS suits a known attack detection; however, new or unknown attacks (also

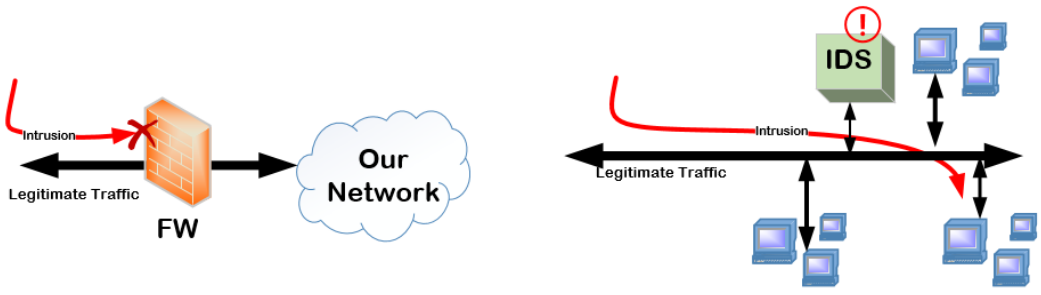


Figure 2.1: IDS vs Firewall

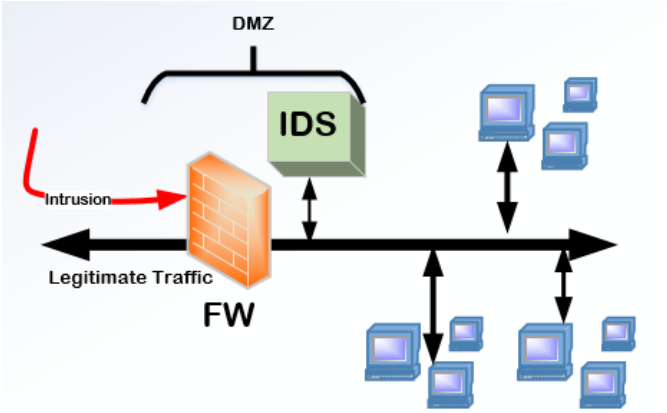


Figure 2.2: Network-based IDS

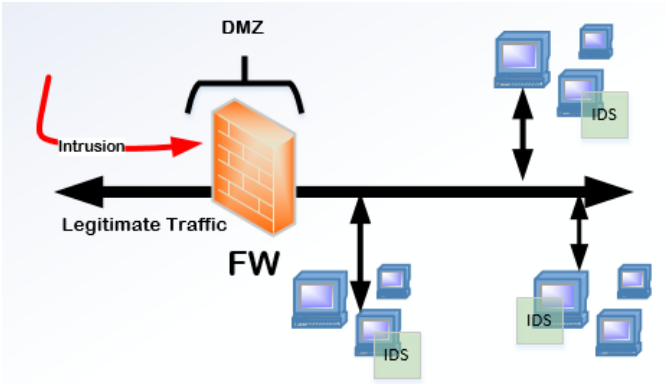


Figure 2.3: Host-based IDS

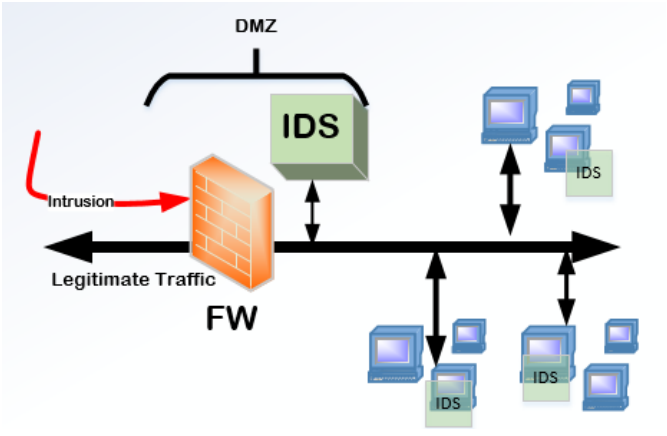


Figure 2.4: Hybrid-based IDS

Table 2.1: Comparison of IDS Types based on the Methodology

	Misuse-based	Anomaly-based	Specification-based
Method	Identify known attack patterns	Identify unusual activity patterns	Identify violation of pre-defined rules
Detection Rate	High	Low	High
False Alarm Rate	Low	High	Low
Unknown Attack Detection	Incapable	Capable	Incapable
Drawback	Updating signatures is burdensome	Computing any machine learning is heavy	Relying on expert knowledge during defining rules is undesirable

called as a zero-day exploit) are difficult to be detected. An anomaly-based IDS detects an attack by profiling normal behavior and then triggers an alarm if there is any deviation from it. The strength of this IDS is its ability for unknown attack detection. However, misuse-based IDS usually achieves higher detection performance for known attacks than anomaly-based IDS. A specification-based IDS manually defines a set of rules and constraints to express the normal operations. Any deviation from the rules and constraints during execution is flagged as malicious [20]. Table 2.1 summarizes the comparison of IDS types based on the methodology.

An IDS that leverages machine-learning method is an example of an anomaly-based IDS [21]. There are two types of learning namely supervised and unsupervised learning. The unsupervised learning does not require a labeled dataset for training which is crucial for large network traffics recently, while the supervised learning requires a labeled dataset. Unsupervised learning capability is of critical significance as it allows a model to be built to detect new attacks without creating costly labels or dependent variables. Table 2.2 outlines the comparison between supervised and unsupervised learning.

2.3 Machine-learning based IDS

A combination of two typical methods is commonly used to build an IDS such as learning or training and classification as shown in Fig. 2.5. It is difficult and costly to obtain the bulk of labeled network connection records for supervised training in the first stage. Then feature learning or clustering might become the solution in the first place. The clustering analysis has emerged as an anomaly detection recently [22]. Clustering is an unsupervised data exploratory technique that partitions a set of unlabeled data patterns into groups or clusters such that patterns within a cluster are similar to each other but dissimilar to other clusters' pattern [22]. Meanwhile, feature learning is a tool for improving the learning process of a machine-learning algorithm. It commonly consists of feature construction, extraction, and selection. Feature construction expands the original features to enhance their expressiveness, whereas feature extraction transforms the original features into a new form and feature selection eliminates unnecessary features [18]. The classification task is a supervised method to distinguish benign and

Table 2.2: Comparison Between Supervised and Unsupervised Learning

	Supervised	Unsupervised
Definition	The dataset are labeled with pre-defined classes	The dataset are labeled **without** pre-defined classes
Method	Classification	Clustering
Example	Support Vector Machine (SVM), Decision Tree (DT)	K-means clustering, Ant Clustering Algorithm (ACA)
Known Attack Detection	High	Low
Unknown Attack Detection	Low	High

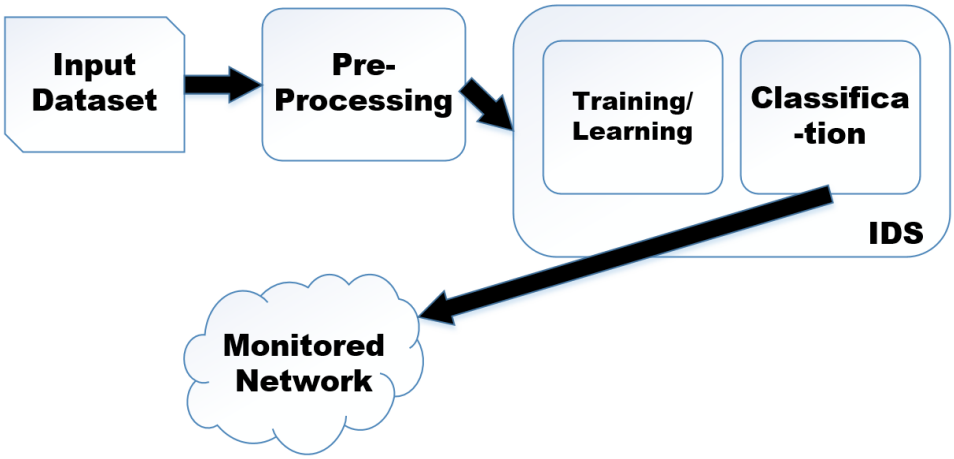


Figure 2.5: IDS Typical Scheme

malicious traffics based on provided data which usually comes from the previous step as shown in Fig. 2.5.

We can see in the Fig. 2.5 that the pre-processing step is required before entering the IDS module. The pre-processing module commonly consists of normalization and balancing steps. Data normalization is a process to output all value ranges of each attribute are equal, which is essential for proper learning by any machine learning algorithm [23]. Meanwhile, the nature of the real-world network is having benign traffics much larger than malicious traffics. This property could make it difficult for the IDS module to learn the underlying patterns correctly [24]. Therefore, a balancing process which creates the dataset with an equal ratio for both benign and malicious instances is a required step for training. However, we should use original ratio, which is unbalanced, for testing purposes to validate the IDS can be implemented in the real-world networks.

As mentioned, we explored several common IDSs with a combination of learning and classification as shown in Table 2.3.

Ant Clustering Algorithm (ACA) is one of the most widely used clustering approaches which is originated from swarm intelligence. ACA is an unsupervised learning algorithm that can find near-

Table 2.3: Common IDSs with a Combination of Learning and Classification

Publication	Learning	Classification
AKKK17 [25]	ACA	FIS
HKY14 [26]	ATTA-C	ATTA-C + label
KKK15 [27]	ACA	AIS
KHKY16 [28]	ACA	DT, ANN

optimal clustering solution without a predefined number of clusters needed [22]. However, ACA is rarely used in intrusion detection as the exclusive method for classification. Instead, ACA is combined with other supervised algorithms such as Self Organizing Map (SOM) and Support Vector Machine (SVM) to provide better classification result [13]. In AKKK17 [25], we proposed a novel hybrid IDS scheme based on ACA and Fuzzy Inference System (FIS). We applied ACA for training phase and FIS for classification phase. We chose FIS as classification phase because fuzzy approach can reduce the false alarm with higher reliability in determining intrusion activities [29]. Meanwhile, we also examined the same ACA with different classifiers in KKK15 [27] and KHKY16 [28] by using Artificial Immune System (AIS) and Decision Tree (DT) as well as Artificial Neural Network (ANN), respectively. AIS is designed for the computational system and inspired by Human Inference System (HIS). AIS can differentiate between the "self" (cells that are owned by the system) and "non-self" (foreign entities to the system). We show that ANN can learn more complex structure of certain unknown-attacks due to a characteristic of ANN. Besides, we also investigated an improved ACA which is Adaptive Time-Dependent Transporter Ants Clustering (ATTA-C) in HKY14 [26], which is one of the few algorithms that have been benchmarked on various datasets and is now publicly available under GNU agreement [26]. In addition to above-mentioned common IDSs, we further examined other IDS models taking benefits of Hadoop framework [30] and Software Defined Networking (SDN) environment [31]. In [30], we proposed a method utilizes the advantages of Hadoop as well as behavioral flow analysis. This framework is particularly useful in the case of P2P traffic analysis due to inherent flow characteristics of this type of applications. Meanwhile, we proposed a novel IDS scheme that operates lightweight intrusion detection that keeps a detailed analysis of attacks [31]. In this scheme, a flow-based IDS detects intrusions, but with low operating cost. When an attack is detected, the IDS requests the forwarding of attack traffic to packet-based detection so the detailed results obtained by packet-based detection can be analyzed later by security experts.

2.4 Fuzzy Unknown Attack Detection

2.4.1 ACA

ACA simulates random ant walks on a two-dimensional grid which is all data objects are spread randomly [32]. Unlike the dimension of the input data, each data instance is randomly projected onto a cell of the grid. A grid cell can indicate the relative position of the data instance in the two-dimensional grid. The general idea of ACA is to keep similar items in their original N-dimensional space. Vizine *et al.* [32] assumed that each site or cell on the grid can be resided by at most one object, and one of the two following situations may occur: (i) one ant holds an object i and evaluates the probability of dropping it in its current position; (ii) an unloaded ant assesses the likelihood of picking up an object. An ant is selected randomly and can either pick up or drop an object at its current location [32].

The probability of picking up an object increases by disparity among objects in the surrounding area and *vice versa*. In contrast, the probability of dropping an object increases by high similarity among objects in the surrounding area. Vizine *et al.* [32] defined *d(i,j)* in Eq. (2.1) as the Euclidean distance between objects i and j in their N-dimensional space. The density distribution function for object i, at a particular grid location, is defined by Eq. (2.1) as follows:

$$f(i) = \begin{cases} \frac{1}{s^2} \sum_j (1 - d(i,j)/\alpha) & f(i) > 0 \\ 0 & Otherwise, \end{cases} \tag{2.1}$$

where s^2 is the number of cells in the surrounding area of i, and α is a constant that depicts the disparity among objects. The *f(i)* might reach a maximum value when similar or even equal objects occupy all the sites in the surrounding area. Eqs (2.2) and (2.3), respectively, give the probability of picking up and dropping an object i :

$$P_{pick}(i) = (\frac{k_p}{k_p + f(i)})^2, \tag{2.2}$$

$$P_{drop}(i) = \begin{cases} 2f(i) & f(i) < k_d \\ 1 & Otherwise, \end{cases} \tag{2.3}$$

where the parameters k_p and k_d are threshold constants of the probability of picking up and dropping an object, respectively. A loaded ant considers the first empty cell in its local area to drop the object, since the current position of the ant may be pre-occupied by another object [32].

Tsang *et al.* [22] define two variables: intra-cluster and inter-cluster distance in order to measure ACA performance. High intra-cluster distance means better compactness. Meanwhile, large inter-cluster distance means better separateness. A good ACA should provide minimum intra-cluster distance and maximum inter-cluster distance to presents the internal structures and knowledge from data patterns.

2.4.2 Fuzzy Approach

The fuzzy approach is a method of representing the ambiguity and imprecision of a logic that is usually only 1 and 0 in digital form. This property of the fuzzy set is appropriate to be exploited as anomaly detector for two main reasons [33]:

1. The anomaly detection problem usually includes several numeric attributes in collected data and various derived statistical measurements. Constructing models directly on numeric data might cause many errors in detection.

2. The security term itself involves fuzziness because the boundary between normal and abnormal is not well-defined [29].

Fuzzy logic is usually occupied together with other famous data mining techniques to detect outlier. Malicious behavior is naturally different from normal behavior, and then abnormal behavior might be considered as an outlier. Fuzzy logic can help to construct more abstract and flexible pattern for intrusion detection and thus substantially increase the adaption ability of the detection system [29]. Therefore, the fuzzy approach can achieve reliable intrusive activities detection with a quite low FPR, since we can classify adequately any data instance based on the distance to other clusters. The distance of any data instance to clusters represents a similarity, the nearer the distance means that the data instance is similar to that cluster.

Table 2.4: Packet Distribution of KDD Cup'99 Dataset

Type	# of Packets	Proportion (%)
Normal	972,781	19.86
Probe	41,102	0.84
DoS	3,883,370	79.28
U2R	52	0.00
R2L	1,126	0.02
Total	4,898,431	100

2.4.3 KDD Cup'99 Dataset

KDD Cup'99 dataset has been the most widely used dataset for the evaluation of anomaly detection methods [34]. The dataset is based on the data captured in DARPA'98 IDS evaluation program. KDD Cup'99 dataset consists approximately 4,900,000 single connection instances. Table 2.4 shows the packet distribution of KDD Cup 99 dataset [35]. Each instance contains 41 features and is labeled as either normal or attack instance. The dataset provides four distinct attack types as follows:

1. **Probing Attack**: an attacker attempts to collect information about computer networks to bypass the security controls. An example of probing attack is port scanning.
2. **Denial of Service (DoS) Attack**: an attack in which the attacker prevents legitimate users from accessing authorized data. The attacker made computing resources too exhausted to handle legitimate requests by flooding the network with unnecessary packet requests. An example of DoS attack is syn flood attack.
3. **User to Root (U2R) Attack**: an attacker starts the attack by accessing to a normal user account on the system. Then, the attacker exploits the vulnerability to gain root access to the system. An example of U2R attack is *xterm* exploitation.
4. **Remote to Local (R2L) Attack**: This kind of attack is executed by an attacker who can send packets to a machine over a network but does not have an account on that machine. The attacker exploits some vulnerabilities to gain local access as a user of that machine remotely. An example of R2L attack is *ftp_write* exploitation.

2.4.4 Proposed Approach

This section describes the details of our approach. Our approach consists of two main phases, training and classification. Similar to other approaches, our scheme is illustrated in Fig. 2.6. Each phase is also described as follows:

Training Phase

The training phase implements ACA to clusters the network traffic. ACA incorporates several initialization steps. Thus, it needs several input parameters such as the size of grid area, the number of ants, the size of a local area, and threshold constant. After the clustering phase finished, we refer to Kim *et al. [36] for labeling each data instance according to resulted clusters.* The training phase passes this labeled dataset to the Fuzzy Inference System (FIS) in the classification phase.

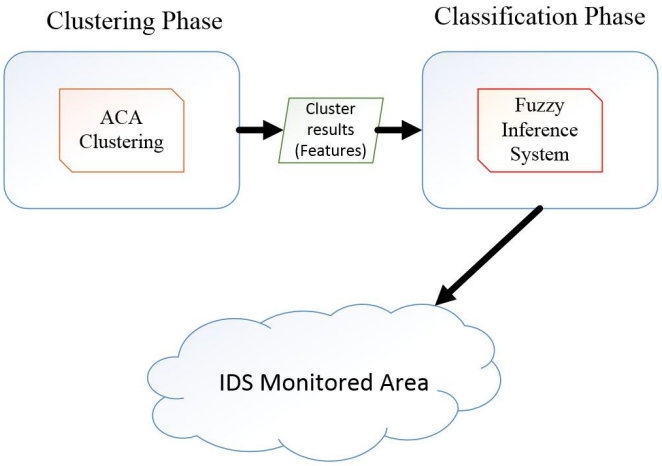


Figure 2.6: Fuzzy unknown attack detection scheme

Classification Phase

The labeled dataset from the training phase is sent to the second phase for anomaly detection when new data arrives. In the classification phase, a fuzzy decision approach is applied to detect attacks. We calculate Euclidean distance of each test data to all clusters as an input to the FIS. Eq. (2.4) shows the Euclidean distance between two points x and y, where x_i and y_i represent features of each test data instance and training data instance within the cluster, respectively. In this case, N represents total features in KDD Cup'99 dataset [37] which has 41 features on each data instances.

$$Distance(x, y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}. \tag{2.4}$$

We deploy a combination of two distance-based [29] methods, *i.e.*, nearest to normal and abnormal:

1. **Nearest to Normal**: The distance between a test data instance and each cluster is calculated using average linkage of Euclidean distance. Average linkage approach considers small variances [29] because the approach considers all members in the cluster rather than just a single member. Also, the average linkage approach tends to be less influenced by the extreme values than other distance methods [38]. A test data instance is classified as nearest to normal when it has minimum average Euclidean distance among clusters labeled as a normal cluster and *vice versa*. This distance-based classification allows us to detect whether normal or abnormal traffic by comparing features similarity that is listed in the training dataset.

2. **Nearest to Abnormal**: Similar as before, we also calculate average linkage of Euclidean distance to find the minimum distance to the abnormal cluster. A test data instance is classified as nearest to abnormal when the data instance has minimum average Euclidean distance among clusters labeled as the abnormal cluster and *vice versa*.

The proposed fuzzy detection method consists of two inputs (nearest to normal and abnormal), one output, and four main parts: fuzzification, rules, inference engine, and defuzzification [29]. In fuzzification step, a crisp set of input data is converted to a fuzzy set using fuzzy linguistic terms and membership functions. Next, we construct rule base. Afterward, an inference is made and combined based on the set of rules. In defuzzification step, the results of fuzzy inference are mapped to a crisp

Table 2.5: Our Training Dataset

Type	# of Packets	Proportion (%)
Normal	78,101	98.00
Probe	398	0.50
DoS	761	0.96
U2R	35	0.04
R2L	398	0.50
Total	79,602	100

(non-fuzzy) output using the output membership functions. Finally, if the crisp output is bigger than a predefined threshold, a test data instance is considered as an abnormal instance; otherwise, it is a normal instance.

2.4.5 Evaluation

Performance Measurement

To evaluate the performance of our proposed approach, we use DR, FPR and False Negative Rate (FNR). We calculate DR by a number of attack instances detected as attacks divided by a total of attack instances included in the test dataset. We have 393 data of attack instances. FPR is legitimate packet identified as a malicious packet. FPR is calculated by a number of legitimate instances detected as attack instances divided by total normal (legitimate) instances included in the data test. We are incorporating 19,268 legitimate instances. Lastly, FNR represents a number of attacks that unable to be detected by our proposed approach. The FNR value can be calculated by one minus DR.

Clustering Phase

We need to extract the KDD Cup'99 dataset to get appropriate traffic data that reflects real network traffic. Also, we need to prepare two sets of data: training and test dataset. Table 2.5 shows the training dataset that we used as an input to ACA in clustering phase. As mentioned in Sec.2.4.4, ACA needs several input parameters, we define the parameters as follows:

- Size of grid area: 600 X 600 size of 2D plane,
- Number of ants: 1000 ants,
- size of local area: 3 X 3 local area,
- Threshold constant: 15.

ACA provides clusters that consolidate similar feature data instances. We label big and small size clusters as normal and attack clusters, respectively. Fig. 2.7 shows the clustering result. The big colony is assumed as benign instances. We prepare the test dataset as shown in Table 2.6. The dataset is processed by measuring the Euclidean distance between each data instance in the test dataset and all data instances in the training dataset. Then, we define two values: closest to normal and abnormal, as an input parameter to the Fuzzy Inference System (FIS).

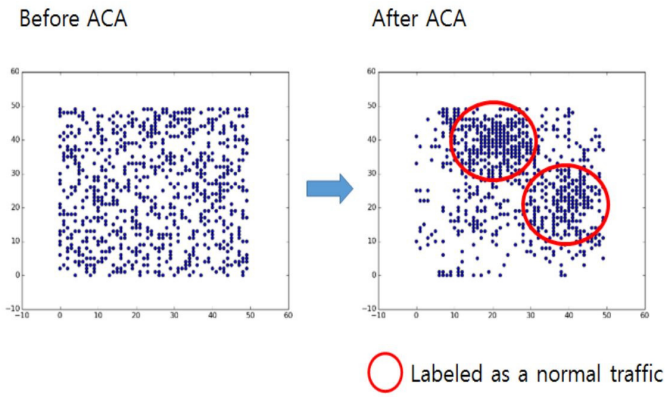


Figure 2.7: Clustering Result

Table 2.6: Our Test Dataset

Type	# of Packets	Proportion (%)
Normal	19,268	98.00
Probe	98	0.50
DoS	277	1.41
U2R	17	0.09
R2L	1	0.00
Total	19,661	100

Classification Phase

We use MATLAB fuzzy logic toolbox for FIS-based intrusion detection. Following components structure the classification phase:

1. Two fuzzy sets of input variables: nearest to normal and abnormal; nearest to normal membership are: Very Close, Close, Average, Far, Very Far; nearest to abnormal membership are: Far, Average, Close.

2. A fuzzy set of output variable: Alarm; alarm membership function: Normal, Less Prone, High Prone, Abnormal.

3. Fuzzy Membership Functions (MF): Figs. 2.8, 2.9 and 2.10 show fuzzy membership function for nearest to normal input, abnormal input and alarm output, respectively.

4. Fuzzy rules: Table 2.7 shows complete fuzzy rules while Table 2.13 shows more detailed fuzzy rules.

5. Inference: We use Mamdani fuzzy inference by the fuzzy set operation as max and min for OR and AND, respectively [29]. Fig. 2.12 shows fuzzy inference rule in 3D form.

6. Defuzzifier: We use Center of Gravity algorithm as shown by Eq.(2.5).

$$CenterOfGravity = \frac{\int_{min}^{max} u * \mu(u) d(u)}{\int_{min}^{max} \mu(u) d(u)}, \tag{2.5}$$

where u represents the output variable, μ denotes the membership function after accumulation, and min and max are lower and upper limits for defuzzification, respectively.

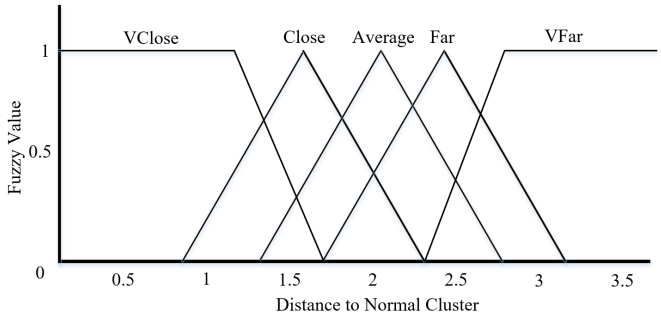


Figure 2.8: Membership Function for Nearest to Normal Input

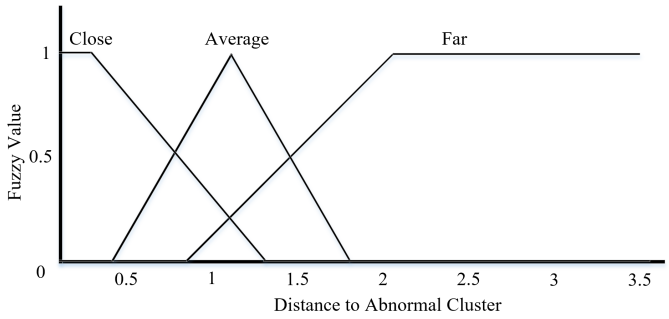


Figure 2.9: Membership Function for Nearest to Abnormal Input

Experiment Result

This section shows our experimental results. To get the best classification phase result, we conduct four different experiments: varying MF positions, MF types, inference rules, and defuzzifier methods.

First, we did experiment with three different MF inputs. Fig. 2.11 shows the three different inputs. Table 2.8 shows that MF input (a,b) is the best choice among three different MF inputs.

Second, we use four different MF types: trapmf, trimmf, gauss2mf, and gbellmf. Trapmf represents trapezoidal distribution function. Trimmf represents triangle distribution function. Meanwhile, both gauss2mf and gbellmf represent gaussian distribution function with different parameters. The selection of those four functions are based on Karami *et al.* [29] experiments. We cannot compare directly with Karami *et al.* [29] as the dataset is different, and thus the metrics are different which forms the core of the experimentation. In Karami *et al.* [29] paper, they use Content Centric Network (CCN) dataset which has a different type with KDD dataset. However, we can compare our works with Karami *et al.* [29] in term of Fuzzy parameter usage. According to Table 2.9, trapezoidal and triangle distributions gave the best result while in Karami *et al.* [29], trapezoidal and Gaussian distribution outperformed other

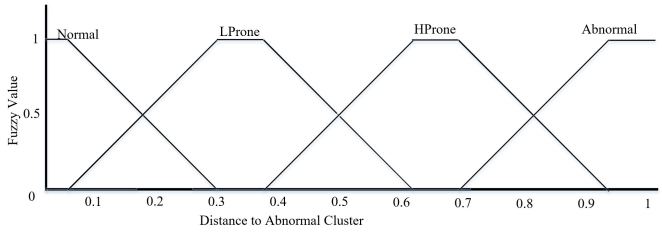


Figure 2.10: Membership Function for Alarm Output

Table 2.7: Fuzzy Rules

Nearest to Abnormal	**Nearest to Normal**				
	VeryClose	Close	Average	Far	VeryFar
Close	HighProne	HighProne	Abnormal	Abnormal	Abnormal
Average	LowProne	LowProne	HighProne	HighProne	HighProne
Far	Normal	Normal	Normal	HighProne	HighProne

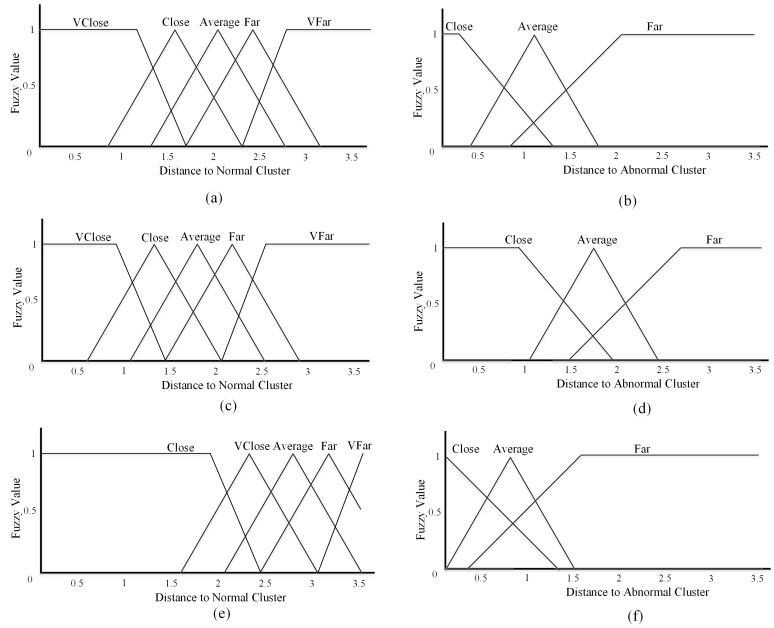


Figure 2.11: Three Different Membership Functions. (a)(c)(e) represent different nearest to normal. Meanwhile, (b)(d)(f) represent different nearest to abnormal.

distributions. Table 2.9 shows the effect of different MF types to FPR and DR. Trapezoidal distribution function is the best function among four functions.

Third, we accommodate different inference rules. The first inference rule is shown in Table 2.7. There are two rules which are not following intuition in Table 2.7, when nearest to normal far and very far to nearest to abnormal far. Both of them are supposed to be low prone as shown in Table 2.10. Table 2.11 shows the effect of different inference rules. Unfortunately, the second rule shown by Table 2.10 results pretty low DR. The result makes sense since once an instance located far or very far from benign instances, the instance has a high probability of being an attack.

Last, we occupy different defuzzifier methods. There are five different defuzzifer methods: Center of Gravity (CoG), bisector, Mean of Max (MoM), Largest of Max (LoM), and Smallest of Max (SoM).

Table 2.8: Performance with Different MF Inputs

Method	FPR (%)	DR (%)
a,b	10.03	92.11
c,d	66.74	95.67
e,f	2.55	0.00

Table 2.9: Performance with Different MF Types

Method	FPR (%)	DR (%)
Trapmf	10.03	92.11
Trimmf	9.37	91.86
Gauss2mf	12.96	92.11
Gbellmf	10.18	93.38

Table 2.10: Another Fuzzy Rules

Nearest to Abnormal	**Nearest to Normal**				
	VeryClose	Close	Average	Far	VeryFar
Close	HighProne	HighProne	Abnormal	Abnormal	Abnormal
Average	LowProne	LowProne	HighProne	HighProne	HighProne
Far	Normal	Normal	Normal	LowProne	LowProne

Table 2.12 shows that CoG is the best defuzzifier method in this case.

Recall in the defuzzification step, the results of fuzzy inference are mapped to a crisp (non-fuzzy) output using the output membership functions. If the crisp output is bigger than a predefined threshold (from now on called fuzzy threshold), a test data instance is considered as an abnormal instance; otherwise, it is a normal instance. Table 2.14 shows the performance of our approach using different fuzzy thresholds. We can see that the bigger the fuzzy threshold, the lower the DR. Unfortunately, we also have bigger FPR as a trade-off. We note that 0.65 as fuzzy threshold provide best performance among others with DR = 92.11% and FPR = 10.03%. It means that there are 1,936 legitimate instances detected as an attack. Also, 31 out of 393 attack data instances aren't identified as attacks. Thus, we conclude that 0.65 is the optimal value for the fuzzy threshold.

To provide the proper measurement, we compare our scheme with other similar schemes as mentioned by Hosseinpour *et al.* [39]. They proposed a hybrid scheme of IS and DBSCAN. Similar to our approach, their approach exploits two phases: clustering and detection phase. Also, they also provide the performance result of another IDS scheme based on AIS and K-means clustering. Table 5.19 shows the comparison of three different schemes. ACA is a proper algorithm for high density and high dimensional data. Also, ACA is insensitive to initialization step. These properties satisfy the needs of real traffic network, which has high density and high dimensional data. Although ACA needs many input parameters, by combining it with FIS, our proposed scheme can achieve significantly higher DR compared to other two schemes. However, our proposed scheme provides quite high FPR which is 10.03%. We can vary the parameters and cut it down a little bit. But, if we want the DR to be high, with our scheme, the FPR remains a bottleneck. This fact remains a trade-off whether using less FPR or higher DR is of greater value to the user.However, we give a comparison with AIS+K-means and AIS+DBSCAN [39] on

Table 2.11: Performance with Different Inference Rules

Method	FPR (%)	DR (%)
Table 2.7	10.03	92.11
Table 2.10	66.74	29.26

Table 2.12: Performance with Different Defuzzifier Methods

Method	FPR (%)	DR (%)
CoG	10.03	92.11
Bisector	15.4	92.37
MoM	55.08	94.91
LoM	0.00	0.00
SoM	0.00	0.00

Table 2.13: Some Fuzzy Rules in Proposed System

IF Normal=*Average* and Abnormal=*Far* **THEN** Alarm=*Normal*
IF Normal=*Close* and Abnormal=*Average* **THEN** Alarm=*LowProne*
IF Normal=*Far* and Abnormal=*Average* **THEN** Alarm=*HighProne*
IF Normal=*VeryFar* and Abnormal=*Close* **THEN** Alarm=*Abnormal*

Table 5.19. Our scheme is efficiently detecting attacks both known and unknown.

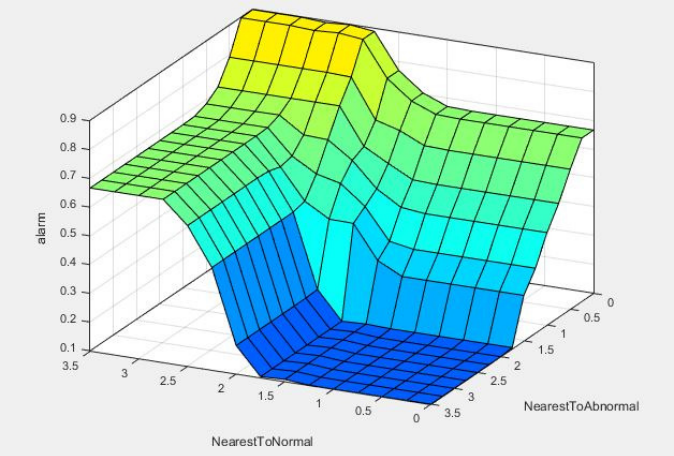


Figure 2.12: Fuzzy inference rule in 3D form

Table 2.14: Performance of Our Proposed Scheme

Fuzzy Threshold	FPR (%)	DR (%)	FNR (%)
0.70	9.40	0.00	100.00
0.65	10.03	92.11	7.89
0.60	20.81	94.91	5.09
0.55	32.35	94.91	5.09
0.30	97.25	98.73	1.27

Table 2.15: Results Comparison

Method	DR (%)	FPR (%)
AIS+K-means [39]	43.1	15.6
AIS+DBSCAN [39]	58.9	0.8
Our Proposed Scheme	92.11	10.03

Chapter 3. Deep Learning

Deep learning originally comes from the advancements of Neural Network (NN) algorithm. Various methods have been applied in order to overcome the limitations of one hidden layer only in NN. Those methods employ consecutive hidden layers which are hierarchically cascaded. Due to vast of methods belong to deep learning, we classify several deep learning methods based on their approach [3]. Deng [40] differentiates deep learning into three sub-groups, generative, discriminative and hybrid. The classification is based on the intention of architectures and techniques, *e.g.*, synthesis/generation or recognition/classification. The classification of the deep learning methods is shown in Fig. 3.1.

3.1 Unsupervised Learning

Unsupervised learning or so-called generative architectures use unlabeled data. The central concept of applying generative designs to pattern recognition is unsupervised learning or pre-training [40]. Since learning the lower levels of subsequent networks are difficult, deep generative structures are needed. Thus, with limited training data, learning each lower layer in layer-by-layer approach without relying on all the layers above is essential.

Some methods classified as unsupervised learning as follows:

3.1.1 Auto Encoder (AE)-Stacked Auto Encoder (SAE)

AE is an ordinary Artificial Neural Network (ANN) with the same neuron number of both input and output layers. Meanwhile, the nodes in the hidden layer are representing new feature set which is low-dimensional. This architecture leads to an ability that can reconstruct the data after complicated computations. AE aims to learn a compact set of data efficiently and can be stacked to build a deep network. Training results of each hidden layer are cascaded. This structure is called Stacked Auto-Encoder (SAE) which can provide new transformed features by different depths. To train more precisely, we can append an additional layer with labels once we have a large amount of tagged samples [41]. Besides, a Denoising Auto Encoder (DAE) is trained to reconstruct a clear correction input from a corrupted by noise input [42]. The DAE may also be stacked to build deep networks as well.

3.1.2 Boltzman Machine (BM)

BM is a network of binary units that symmetrically paired [43]. BM has a structure of neuron units that make stochastic decisions about whether active or not [44]. If one BM result is cascaded into multiple BMs, called Deep BM (DBM). Meanwhile, Restricted Boltzmann machine (RBM) is a customized BM without connections among the hidden units [43]. RBM consists of visible and invisible variables such that their relations can be figured out. If multiple layers are stacked, a layer-by-layer scheme called Deep Belief Network (DBN). DBN could be used as a feature extraction method for dimensionality reduction when unlabeled dataset and back-propagation are used (which means unsupervised training). In contrast, DBN is used for classification when appropriately labeled dataset with feature vectors are used (which means supervised training) [45].

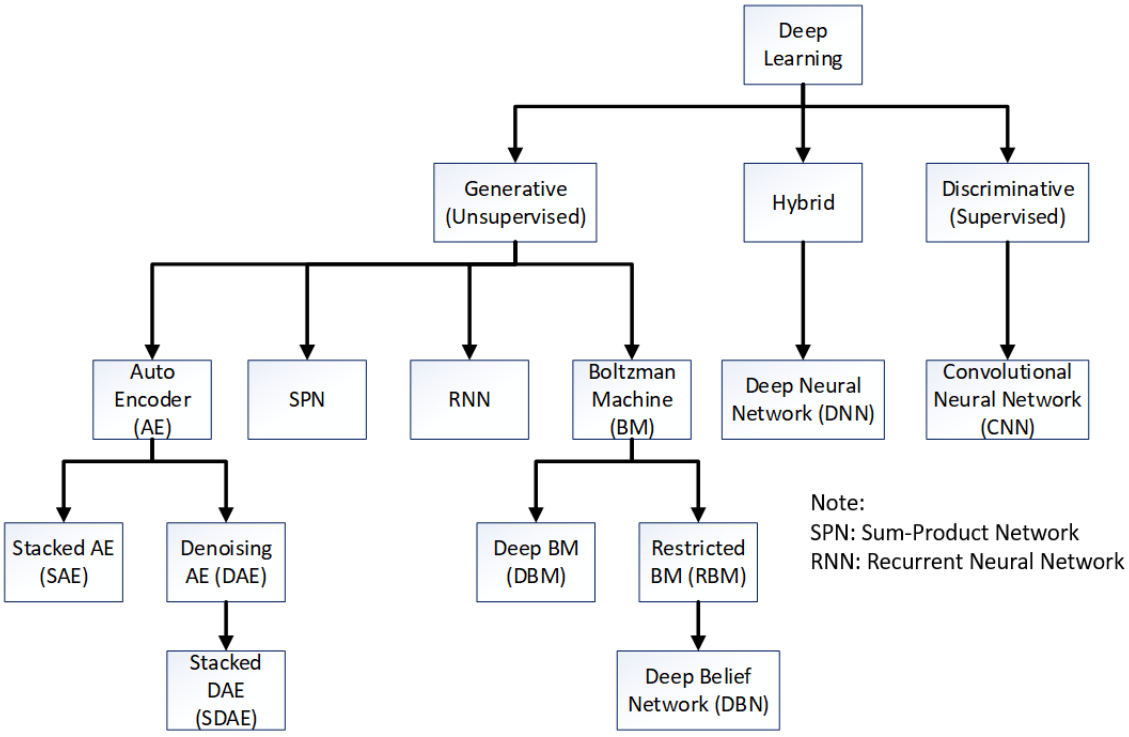


Figure 3.1: Classification of Deep Learning Methods [3]

3.1.3 Sum-Product Networks (SPN)

Other deep generative model is Sum-Product Networks (SPN). SPN is a directed acyclic graph with variables as leaves, sum and product operations as internal nodes, and weighted edges [4]. The sum nodes provide mixture models while the product nodes express the feature hierarchy [44]. Therefore, we can consider SPN as a combination of mixture models and feature hierarchies. Fig. 3.2.

3.1.4 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) is an extension of neural networks with cyclic links to process sequential information. This cyclic links placed between higher and lower layer neurons which enable RNN to propagate data from previous to current events. This property makes RNN having a memory of time series events [46]. Fig. 3.3 shows a single loop of RNN on the left which is comparable to the right topology when the cyclic link unrolled.

One advantage of RNN is the ability connect previous information to present task; however, it cannot reach "far" previous memory. This problem is commonly known as long-term dependencies. Long-Short Term Memory Networks (LSTM) introduced by Hochreiter and Schmidhuber [47] to overcome this problem. LSTMs are an extension of RNN with four neural networks in a single layer, where RNN have one only as shown in Fig. 3.4.

The main advantage of LSTM is the existence of state cell which is the line passing through in the top of every layer. The cell accounts for propagating information from previous to next layer. Then, "gates" in LSTM would manage which information will be passed or dropped. There are three gates to control the information flow, namely input, forget and output gates [48]. These gates are composed of a sigmoid neural network and an operator as shown in Fig. 3.4.

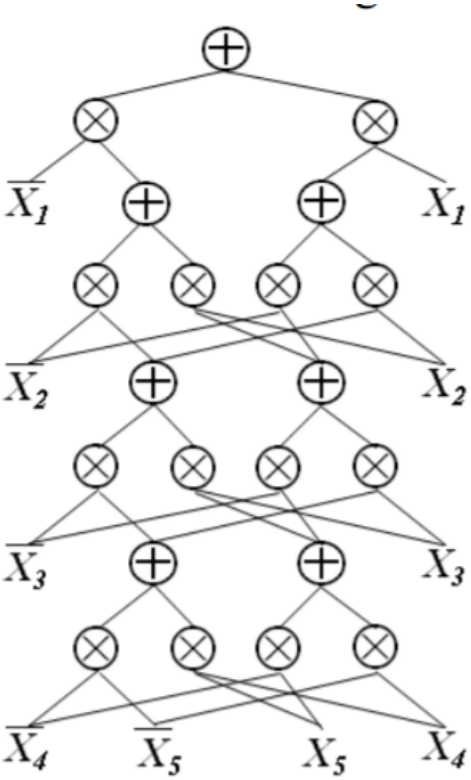


Figure 3.2: One example SPN form with uniform distribution of five variables [4]

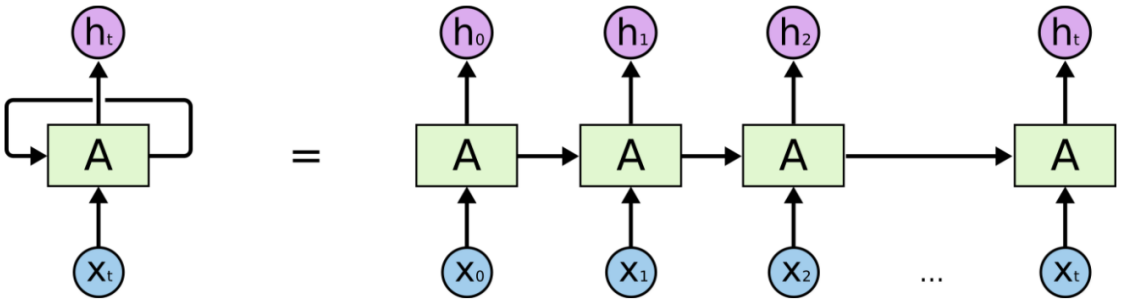


Figure 3.3: RNN with it's unrolled topology [5]

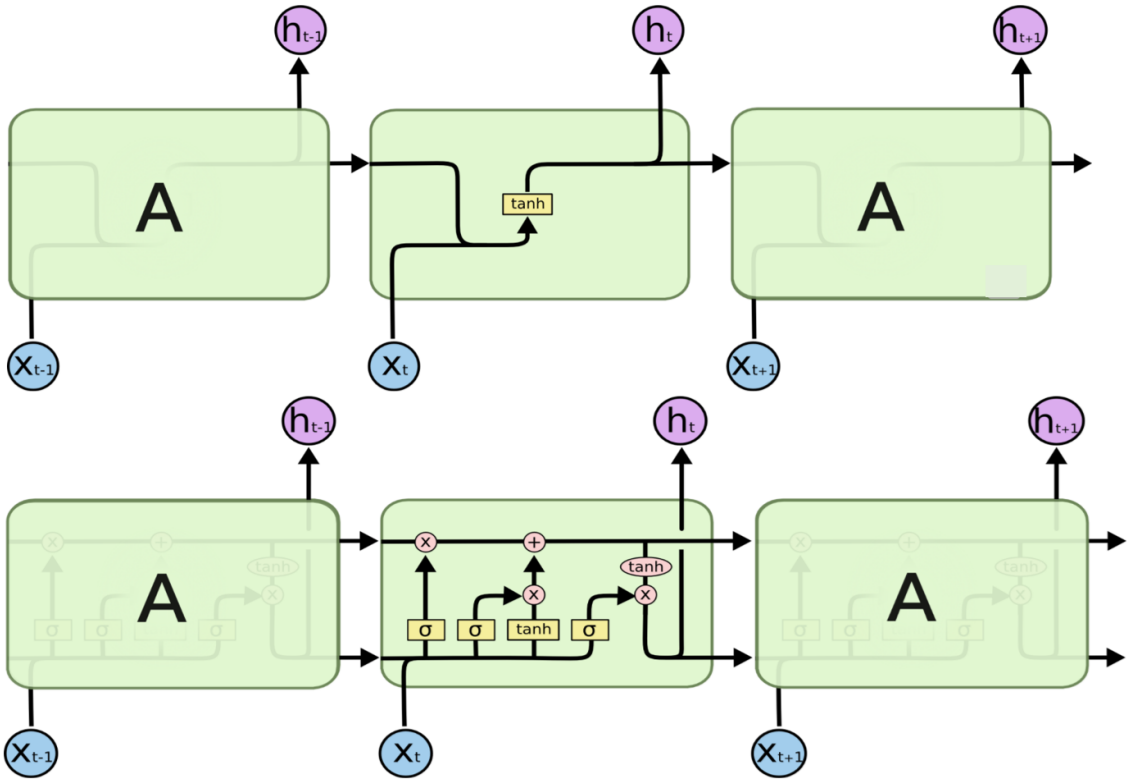


Figure 3.4: RNN topology (Top) vs LSTM topology (Bottom) [5]

3.2 Supervised Learning

Supervised learning or discriminative model is intended to distinguish some parts of data for pattern classification with labeled data [40]. An example of the discriminative architecture is Convolutional Neural Network (CNN) which employs a special architecture particularly suitable for image recognition. The main advantage of CNN is hand-crafted feature extraction is never necessary. CNN can train multilayer networks with gradient descent to learn complex, high-dimensional, nonlinear mappings from large collections of data [6]. CNN uses three basic concepts: local receptive fields, shared weights, and pooling [49]. Fig. 3.5 shows one of typical CNN architectures, LeNet-5 [6]. One extensive research that successfully deployed using CNN is AlphaGo by Google [50]. Other examples of discriminative models are linear and logistic regressions [10].

RNN can also be considered as a discriminative model when the output of RNN used as label

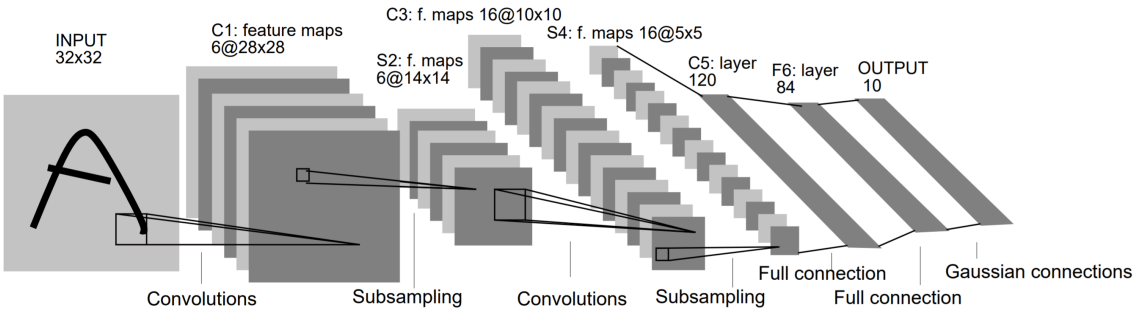


Figure 3.5: Architecture of LeNet-5 [6], one of typical CNN architectures

sequences for the input [44]. One example of this network was proposed by Graves [51]. He leveraged RNNs to build a probabilistic sequence transduction system, which can transform any input sequence into any finite, discrete output sequence.

3.3 Hybrid

The hybrid deep architecture combines both generative and discriminative architectures. The hybrid structure aims to distinguish data as well as discriminative approach. However, in the early step, it has assisted in a significant way with the generative architectures results. An example of hybrid architecture is Deep Neural Network (DNN). However, some confusion terms between DNN and DBN happens. In the open literature, DBN also uses back-propagation discriminative training as a "fine-tuning." This concept of DBN is similar to Deep Neural Network (DNN) [40]. According to Deng [44], DNN is defined as a multilayer network with cascaded fully connected hidden layers and is often use stacked RBM as a pre-training phase. Many other generative models that can be considered as discriminative or hybrid models when classification task added with class labels.

3.3.1 Generative Adversarial Networks (GAN)

Goodfellow [52] introduced a novel framework which trains both generative and discriminative models at the same time, which the generative model G captures the data distribution and the discriminative model D distinguishes the original input data and the data coming from the model G. It is a zero-sum game of G and D models [10] which model G aims to counterfeit the original input data while model D aims to discriminate the original input and output of model G. According to Dimokranitou [10], the advantage of GAN are consistent after equilibrium is achieved, no approximate inference or Markov chains are needed, and can be trained with missing or limited data. On the other hand, the disadvantage of applying GAN is to find the equilibrium between G and D models. A typical architecture of GAN is shown in Fig. 3.6.

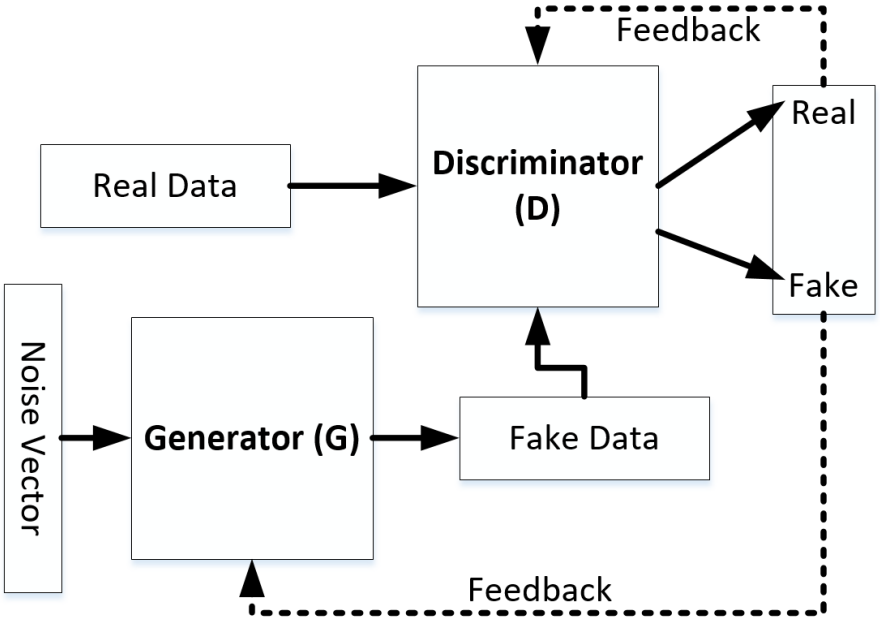


Figure 3.6: Typical architecture of GAN

Chapter 4. Our Methodology

We proposed two methodologies in this book, which are feature extraction and selection and fully unsupervised IDS. Further details are explained as follows.

4.1 Feature Extraction and Selection

Feature learning could be defined as a technique that models the behavior of data from a subset of attributes only. It could also show the correlation between detection performance and traffic model quality effectively [53]. However, feature extraction and feature selection are different. Feature extraction algorithms derive new features from the original features to (i) reduce the cost of feature measurement, (ii) increase classifier efficiency, and (iii) improve classification accuracy, whereas feature selection algorithms select no more than m features from a total of M input features, where m is smaller than M. Thus, the newly generated features are simply selected from the original features without any transformation. However, their goal is to derive or select a characteristic feature vector with a lower dimensionality which is used for the classification task.

We adopt both feature extraction and selection techniques in D-FES. Fig.4.1 shows the stepwise procedure of D-FES with two target classes. A pre-processing process, which comprises the normalization and balancing steps, is necessary. The method is explained in Section 5 in detail. As illustrated in Algorithm 1, we start D-FES by constructing SAE-based feature extractor with two consecutive hidden layers to optimize the learning capability and the execution time [54]. The SAE outputs 50 extracted features, which are then combined with the 154 original features existing in the AWID dataset [2]. Weighted feature selection methods are then utilized using well-referenced machine learners including SVM, ANN, and C4.5 to construct the candidate models, namely D-FES-SVM, D-FES-ANN, and D-FES-C4.5, respectively. SVM separates the classes using a support vector (hyperplane). Then, ANN optimizes the parameters related to hidden layers that minimize the classifying error concerning the training data, whereas C4.5 adopts a hierarchical decision scheme such as a tree to distinguish each feature [55]. The final step of the detection task involves learning an ANN classifier with 12–22 trained features only.

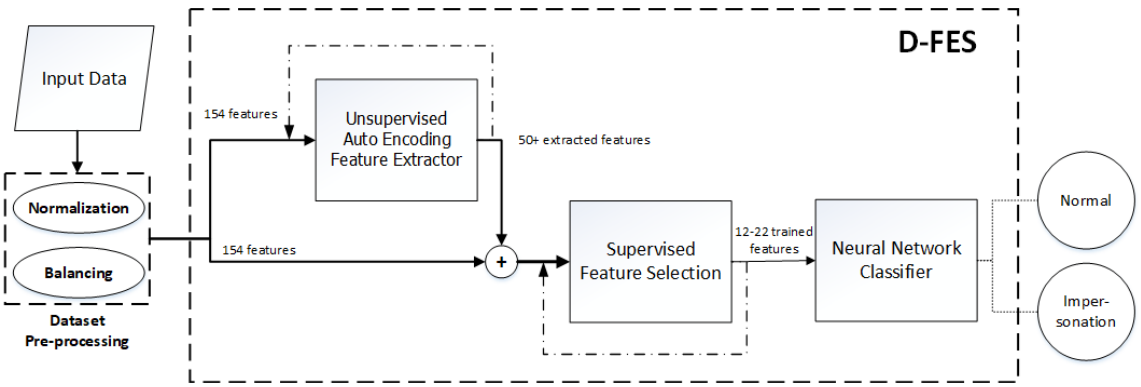


Figure 4.1: Stepwise procedure of D-FES with two target classes: normal and impersonation attack

Algorithm 1 Pseudocode of D-FES

```
procedure D-FES
    function Dataset Pre-processing(Raw Dataset)
        function (Dataset Normalization)Raw Dataset
            return NormalizedDataset
        end function
        function (Dataset Balancing)Normalized Dataset
            return BalancedDataset
        end function
        return InputDataset
    end function
    function Deep Abstraction(InputDataset)
        for i=1 to h do                                  ▷ h=2; number of hidden layers
            for each data instance do
                Compute y_i (Eq. (4.1))
                Compute z_i (Eq. (4.2))
                Minimize E_i (Eq. (4.6))
                θ_i = {W_i, V_i, b_{f_i}, b_{g_i}}
            end for
            W ← W_2                                      ▷ 2nd layer, 50 extracted features
        end for
        InputFeatures ← W + InputDataset
        return InputFeatures
    end function
    function Feature Selection(InputFeatures)
        switch D-FES do
            case D-FES-ANN(InputFeatures)
                return SelectedFeatures
            case D-FES-SVM(InputFeatures)
                return SelectedFeatures
            case D-FES-C4.5(InputFeatures)
                return SelectedFeatures
    end function
    procedure Classification(SelectedFeatures)
        Training ANN
        Minimize E (Eq. (4.7))
    end procedure
end procedure
```

4.1.1 Feature Extraction

(Sparse) Auto Encoder

An Auto Encoder (AE) is a symmetric neural network model, which uses an unsupervised approach to build a model with non-labeled data, as shown in Fig. 4.2. AE extracts new features by using an encoder-decoder paradigm by running from inputs through the hidden layer only. This paradigm enhances its computational performance and validates that the code has captured the relevant information from the data. The encoder is a function that maps an input x to a hidden representation as expressed by Eq. (4.1).

$$y = s_f \left(W \cdot x + b_f \right), \tag{4.1}$$

where s_f is a nonlinear activation function which is a decision-making function to determine the necessity of any feature. Mostly, a logistic sigmoid, $sig(t) = \dfrac{1}{1+e^{-t}}$ is used as an activation function because of its continuity and differentiability properties [56]. The decoder function expressed in Eq. (4.2) maps hidden representation y back to a reconstruction.

$$z = s_g \left(V \cdot y + b_g \right), \tag{4.2}$$

where s_g is the activation function of the decoder which commonly uses either the identity function, $s_g(t) = t$, or a sigmoid function such as an encoder. We use W and V acts as a weight matrix for the features. b_f and b_g acts as a bias vector for encoding and decoding, respectively. Its training phase finds optimal parameters $\theta = \{W, V, b_f, b_g\}$ which minimize the reconstruction error between the input data and its reconstruction output on a training set.

This study uses a modified form of AE, *i.e.* sparse AE [57]. This is based on the experiments of Eskin *et al.* [58], in which anomalies usually form small clusters in sparse areas of feature space. Moreover, dense and large clusters usually contain benign data [59]. For the sparsity of AE, we first observe the average output activation value of a neuron i, as expressed by Eq. (4.3).

$$\hat{\rho}_i = \frac{1}{N} \sum_{j=1}^{N} s_f \left(w_i^T x_j + b_{f,i} \right), \tag{4.3}$$

where N is the total number of training data, x_j is the j-th training data, w_i^T is the i-th row of a weight matrix W, and $b_{f,i}$ is the i-th row of a bias vector for encoding b_f. By lowering the value of $\hat{\rho}_i$, a neuron i in the hidden layer shows the specific feature presented in a smaller number of training data.

The task of machine-learning is to fit a model to the given training data. However, the model often fits the particular training data but is incapable of classifying other data, and this is known as the overfitting problem. In this case, we can use a regularization technique to reduce the overfitting problem. The sparsity regularization $\Omega_{sparsity}$ evaluates how close the average output activation value $\hat{\rho}_i$ and the desired value ρ are, typically with *Kullback-Leibler* (KL) divergence to determine the difference between the two distributions, as expressed in Eq. (4.4).

$$\begin{aligned} \Omega_{sparsity} &= \sum_{i=1}^{h} KL(\rho \| \hat{\rho}_i) \\ &= \sum_{i=1}^{h} \left[\rho \log \left(\frac{\rho}{\hat{\rho}_i} \right) + (1-\rho) \log \left(\frac{1-\rho}{1-\hat{\rho}_i} \right) \right], \end{aligned} \tag{4.4}$$

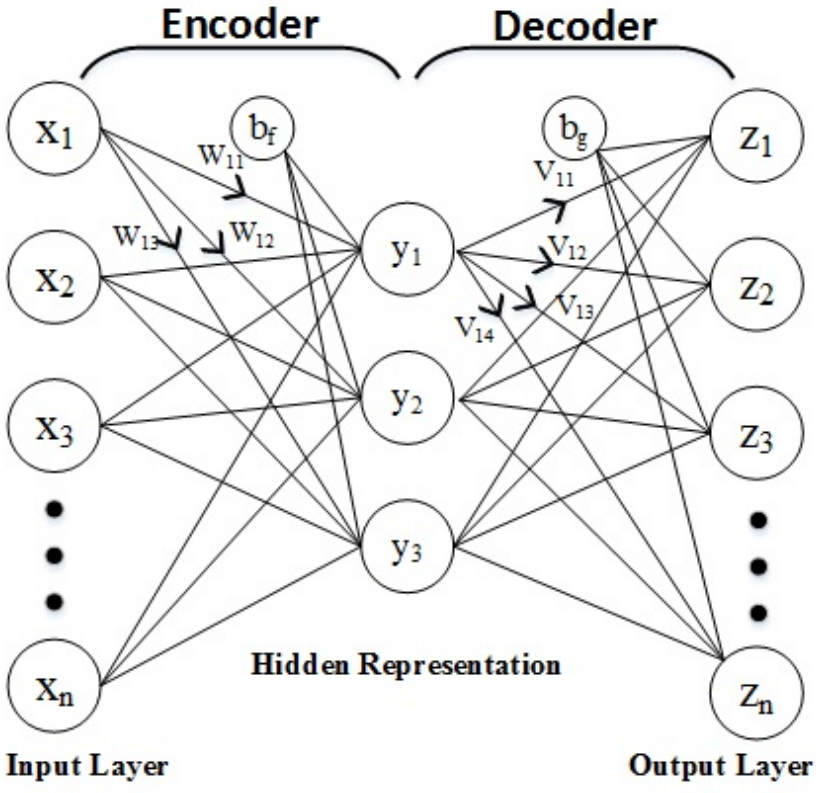


Figure 4.2: AE network with symmetric input-output layers and three neurons in one hidden layer

where h is the number of neurons in the hidden layer.

We may increase the value of the entries of the weight matrix W to reduce the value of sparsity regularization. To avoid this situation, we also add regularization for the weight matrix, known as L_2 regularization as stated in Eq. (4.5).

$$\Omega_{weights} = \frac{1}{2} \sum_{i=1}^{h} \sum_{j=1}^{N} \sum_{k=1}^{K} (w_{ji})^2 , \tag{4.5}$$

where N and K are the number of training data and the number of variables for each data, respectively.

The goal of training sparse AE is to find the optimal parameters, $\theta = \{W, V, b_f, b_g\}$, to minimize the cost function shown in Eq. (4.6).

$$E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} (z_{kn} - x_{kn})^2 + \lambda \cdot \Omega_{weights} + \beta \cdot \Omega_{sparsity}, \tag{4.6}$$

which is a regulated mean square error with L_2 regularization and sparsity regularization. The coefficient of the L_2 regularization term λ and the coefficient of sparsity regularization term β are specified while training the AE.

Stacked (Sparse) AE

(Sparse) AE can be used as deep learning technique by an unsupervised greedy layer-wise pre-training algorithm known as Stacked (Sparse) Auto Encoder (SAE). Here, pre-training refers to the training of a single AE using a single hidden layer. Each AE is trained separately before being cascaded

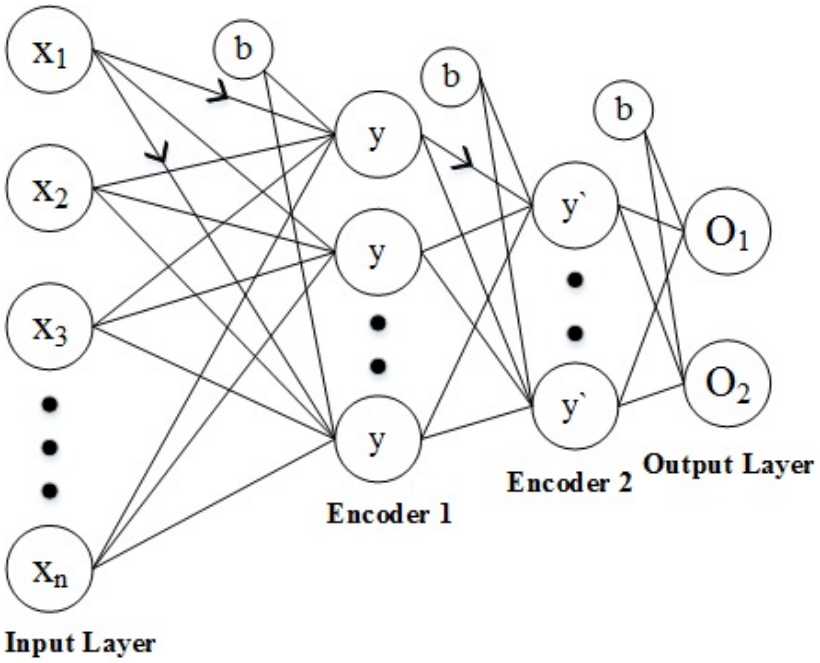


Figure 4.3: SAE network with two hidden layers and two target classes

afterward. This pre-training phase is required to construct a stacked AE. In this algorithm, all layers except the output layer are initialized in a multi-layer neural network. Each layer is then trained in an unsupervised manner as an AE, which constructs new representations of the input.

The performance of the unsupervised greedy layer-wise pre-training algorithm can be significantly more accurate than the supervised one. This is because the greedy supervised procedure may behave too greedy as it extracts less information and considers one layer only [60] [61]. A neural network containing only one hidden layer, it may discard some of the information about the input data as more information could be exploited by composing additional hidden layers.

Fig. 4.3 shows the SAE network with two hidden layers and two target classes. The final layer implements the softmax function for the classification in the deep neural network. Softmax function is a generalized term of the logistic function that suppresses the K-dimensional vector $\mathbf{v} \in \mathbb{R}^K$ into K-dimensional vector $\mathbf{v}^* \in (0,1)^K$, which adds up to 1. In this function, it is defined T and C as the number of training instances and the number of classes, respectively. The softmax layer minimizes the loss function, which is either the cross-entropy function as in Eq. (4.7),

$$E = \frac{1}{T} \sum_{j=1}^{T} \sum_{i=1}^{C} \left[z_{ij} \log y_{ij} + (1 - z_{ij}) \log (1 - y_{ij}) \right], \tag{4.7}$$

or the mean-squared error. However, the cross-entropy function is used in this study.

The features from the pre-training phase, which is greedy layer-wise, can be used either as an input to a standard supervised machine-learning algorithm or as initialization for a deep supervised neural network.

4.1.2 Feature Selection

The supervised feature selection block in Fig.4.1 consists of three different feature selection techniques. These techniques are similar in that they consider their resulting weights to select the subset of

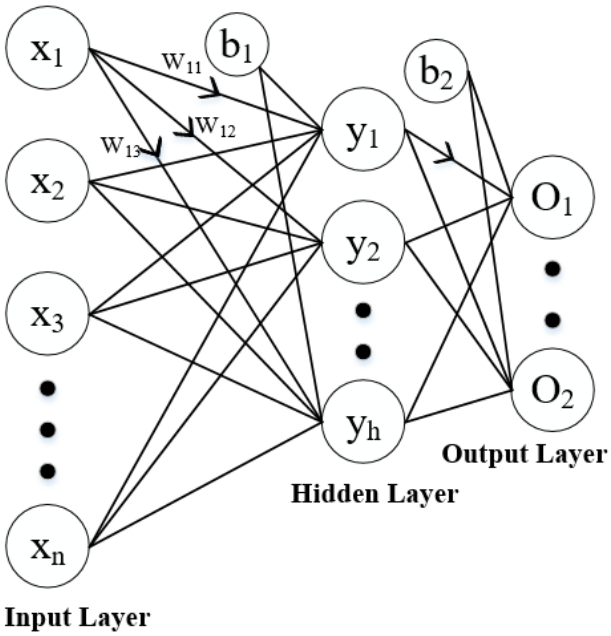


Figure 4.4: ANN network with one hidden layer only

essential features. The following subsections contain further details of each feature selection technique.

D-FES-ANN

ANN is used as a weighted feature selection method. The ANN is trained with two target classes only (normal and impersonation attack classes). Fig. 4.4 shows an ANN network with one hidden layer only where b_1 and b_2 represent the bias values for the corresponding hidden and output layer, respectively.

Algorithm 2 D-FES-ANN Function

```
function D-FES-ANN(InputFeatures)
    Training ANN
    W_ij
    for each input feature do
        Compute V_j (Eq. (4.8))
    end for
    Sort descending
    SelectedFeatures ← V_j > threshold
    return SelectedFeatures
end function
```

To select the critical features, we consider the weight values between the first two layers. The weight represents the contribution from the input features to the first hidden layer. A w_{ij} value close to zero means that the corresponding input feature x_j is meaningless for further propagation, thus having one hidden layer is sufficient for this particular task. We define the important value of each input feature as expressed by Eq. (4.8).

$$V_j = \sum_{i=1}^{h} |w_{ij}|, \tag{4.8}$$

where h is the number of neurons in the first hidden layer. As described in Algorithm 2, the feature selection process involves selecting the features of which the V_j values are higher than the threshold value after the input features are sorted according to their V_j values in descending order.

Following the weighted feature selection, ANN is also used as a classifier. When learning with ANN, a minimum global error function is executed. It has two learning approaches, supervised and unsupervised. This study uses a supervised approach since knowing the class label may increase the classifier performance [62]. Also, a scaled conjugate gradient optimizer, which is suitable for a large scale problem, is used [63].

D-FES-SVM

A supervised SVM is usually used for classification or regression tasks. If n is the number of input features, the SVM plots each feature value as a coordinate point in n-dimensional space. Subsequently, a classification process is executed by finding the hyperplane that distinguishes two classes. Although SVM can handle a nonlinear decision border of arbitrary complexity, we use a linear SVM since the nature of the dataset can be investigated by linear discriminant classifiers. The decision boundary for linear SVM is a straight line in two-dimensional space. The main computational property of SVM is the support vectors which are the data points that lie closest to the decision boundary. The decision function of input vector x as expressed by Eq. (4.9), heavily depends on the support vectors.

$$D(x) = w\vec{x} + b \tag{4.9}$$

$$w = \sum_k \alpha_k y_k x_k \tag{4.10}$$

$$b = (y_k - wx_k) \tag{4.11}$$

Eqs. (4.10) and (4.11) show the corresponding value of w and b, respectively. From Eq. (4.9), we can see that decision function $D(x)$ of input vector $\vec{x}$ is defined as the sum of the multiplication of a weight vector and input vector $\vec{x}$ and a bias value. A weight vector w is a linear combination of training patterns. The training patterns with non-zero weights are support vectors. The bias value is the average of the marginal support vectors.

SVM-Recursive Feature Elimination (SVM-RFE) is an application of RFE using the magnitude of the weight to perform rank clustering [64]. The RFE ranks the feature set and eliminates the low-ranked features which contribute less than the other features for classification task [65]. We use the SVM-RFE by using the linear case [64] described in Algorithm 3. The inputs are training instances and class labels. First, we initialize a feature ranked list that is filled by a subset of essential features that is used for selecting training instances. We then train the classifier and compute the weight vector of the dimension length. After the value of the weight vector is obtained, we compute the ranking criteria and find the feature with the smallest ranking criterion. Using that feature, the feature ranking list is updated and the feature with the lowest ranking criterion is eliminated. A feature ranked list is finally created as its output.

D-FES-DT

C4.5 is robust to noise data and able to learn disjunctive expressions [66]. It has a k-ary tree structure, which can represent a test of attributes from the input data by each node. Every branch of

Algorithm 3 D-FES-SVM Function

function D-FES-SVM($InputFeatures$)
    Training SVM
    Compute w (Eq. (4.10))
    Compute the ranking criteria
    $c_i = {w_i}^2$
    Find the smallest ranking criterion
    $f = argmin(c)$
    Update feature ranked list
    $r = [s(f), r]$
    Eliminate the smallest ranking criterion
    $s = s(1 : f - 1, f + 1 : length(s))$
    $SelectedFeatures \leftarrow s$
    return $SelectedFeatures$
end function

the tree shows potentially selected important features as the values of nodes and different test results. C4.5 uses a greedy algorithm to construct a tree in a top-down recursive divide-and-conquer approach [66]. The algorithm begins by selecting the attributes that yield the best classification result. This is followed by generating a test node for the corresponding attributes. The data are then divided based on the information gain value of the nodes according to the test attributes that reside in the parent node. The algorithm terminates when all data are grouped in the same class, or the process of adding additional separations produces a similar classification result, based on its predefined threshold. The feature selection process begins by selecting the top-three level nodes as explained in Algorithm 4. It then removes the equal nodes and updates the list of selected features.

Algorithm 4 D-FES-C4.5 Function

function D-FES-C4.5($InputFeatures$)
    Training C4.5
    $SelectedFeatures \leftarrow$ top-three level nodes
    for i=1 to n **do** ▷ n = size of $SelectedFeatures$
        for j=1 to n **do**
            if $SelectedFeatures[i]$=$SelectedFeatures[j]$ **then** Remove $SelectedFeatures[j]$
            end if
        end for
    end for
    return $SelectedFeatures$
end function

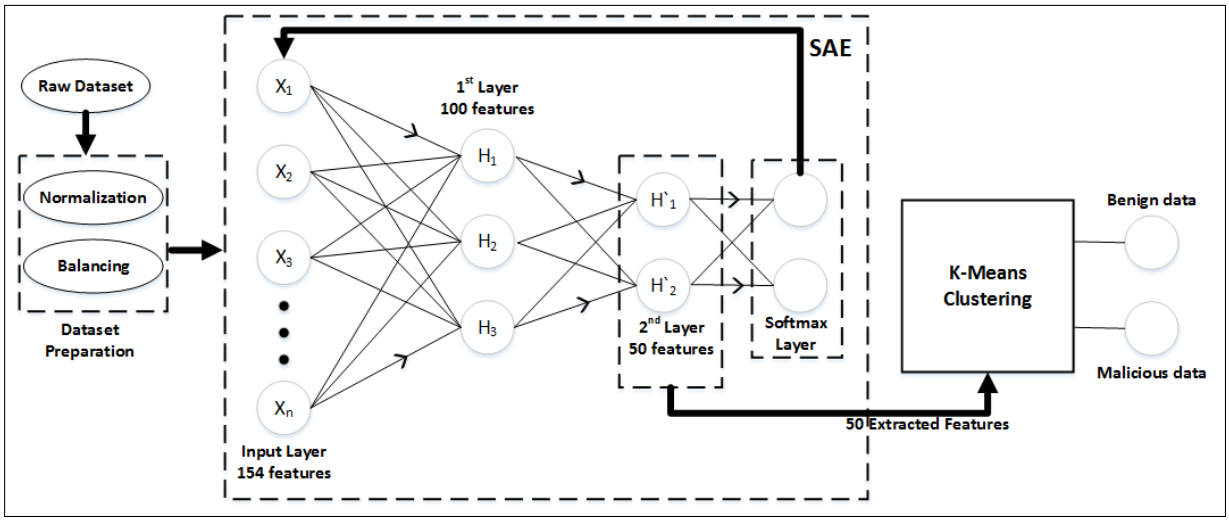


Figure 4.5: Our proposed scheme contains feature extraction and clustering tasks

4.2 Fully Unsupervised IDS

4.2.1 K-means Clustering

K-means clustering algorithm groups all observations data into k clusters iteratively until convergence will be reached. In the end, one cluster contains similar data since each data enters to the nearest cluster. K-means algorithm assigns a mean value of the cluster members as a cluster centroid. In every iteration, it calculates the shortest Euclidean distance from an observation data into any cluster centroid. Besides that, the intra-variances inside the cluster are also minimized by updating the cluster centroid iteratively. The algorithm would terminate when convergence is achieved, which the new clusters are the same as the previous iteration clusters [67].

4.2.2 Fully Unsupervised Method

In this subsection, we describe our novel fully unsupervised deep learning-based IDS for detecting impersonation attacks. There are two main tasks, feature extraction, and clustering tasks. Fig.4.5 shows our proposed scheme which contains two main functions in cascade. We use a real Wi-Fi networks-trace, AWID dataset [2], which contains 154 original features. Before the scheme starts, normalizing and balancing process should be done to achieve best training performance. Algorithm 1 explains the procedure of the proposed scheme in detail.

The scheme starts with two cascading encoders, and the output features from the second layer then forwarded to the clustering algorithm. The first encoder has 100 neurons as the first hidden layer while the second encoder comes with 50 neurons only. We follow a standard rule for choosing the number of neurons in a hidden layer by using 70% to 90% of the previous layer. In this paper, we define k=2 since we consider two classes only. The scheme ends by two clusters formed by k-means clustering algorithm. These clusters represent benign and malicious data.

Algorithm 5 Pseudocode of Fully Unsupervised Deep Learning

procedure START
  function DATASET PREPARATION(Raw Dataset)
    for each data instance **do**
      Convert into integer value
      Normalization $z_i = \frac{x_i - min(x)}{max(x) - min(x)}$
    end for
    Balance the normalized dataset
    return $InputDataset$
  end function
  function SAE($InputDataset$)
    for i=1 to h **do** ▷ h=2; number of hidden layers
      for each data instance **do**
        Compute $H = s_f\left(WX + b_f\right)$
        Compute $X' = s_g\left(VH + b_g\right)$
        Minimize $E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(X'_{kn} - X_{kn}\right)^2 + \lambda \cdot \Omega_{weights} + \beta \cdot \Omega_{sparsity}$
        $\theta_i = \{W_i, V_i, b_{f_i}, b_{g_i}\}$
      end for
      $InputFeatures \leftarrow W_2$ ▷ 2nd layer, 50 extracted features
    end for
    return $InputFeatures$
  end function
  Initialize clusters and k=2 ▷ two clusters: benign and malicious
  function k-MEANS CLUSTERING($InputFeatures$)
    return $Clusters$
  end function
  Plot confusion between $Clusters$ and target classes
end procedure

Chapter 5. Evaluation

A set of experiments was conducted to evaluate the performance of the proposed D-FES method in Wi-Fi impersonation detection. Choosing a proper dataset is an essential step in the IDS research field [68]. We employed the AWID Dataset [2] which comprises of Wi-Fi network data collected from real network environments. We achieved fair model comparison and evaluation by performing the experiments on the same testing sets as in [2]. We then implement the proposed methodology using MATLAB R2016a and Java code extracted and modified from WEKA packages [69] running on an Intel Xeon E-3-1230v3 CPU @3.30 GHz with 32 GB RAM.

5.1 Dataset Pre-processing

There are two types of AWID dataset. The first type named "CLS", has four target classes, whereas the second, named "ATK", has 16 target classes. The 16 classes of the "ATK" dataset belong to the four attack categories in the "CLS" dataset. As an example, the *Caffe-Latte*, *Hirte*, *Honeypot* and *EvilTwin* attack types listed in the "ATK" dataset, are categorized as an impersonation attack in the "CLS" dataset. Based on the size of the data instances included, the AWID dataset comprises both full and reduced versions. In this study, we use the reduced "CLS" for simplicity.

The data contained in the AWID dataset are diverse in value, discrete, continuous, and symbolic, with a flexible value range. These data characteristics could make it difficult for the classifiers to learn the underlying patterns correctly [24]. The pre-processing phase thus includes mapping symbolic valued attributes to numeric values, according to the normalization steps and dataset-balancing process described in Algorithm 5. The target classes are mapped to one of these integer-valued classes: 1 for normal instances, 2 for an impersonation, 3 for flooding, and 4 for an injection attack. Meanwhile, symbolic attributes such as a receiver, destination, transmitter, and source address are mapped to integer values with a minimum value of 1 and a maximum value, which is the number of all symbols. Some dataset attributes such as the WEP Initialization Vector (IV) and Integrity Check Value (ICV) are hexadecimal data, which need to be transformed into integer values as well. The continuous data such as

Table 5.1: Distribution of each class for both balanced and unbalanced dataset

Class		Training	Test
Normal	Unbalanced	1,633,190	530,785
	Balanced	163,319	53,078
Attack	Impersonation	48,522	20,079
	Flooding	48,484	8,097
	Injection	65,379	16,682
	Total	162,385	44,858

AWID dataset mimics the natural unbalanced network distribution between normal and attack instances. "Balanced" means to make equal distribution between the number of normal instances (163,319) and thereof total attack instances (162,385). 15% of training data were withdrawn for validation data.

Algorithm 6 Dataset Pre-processing Function

```
1: function Dataset Pre-processing(Raw Dataset)
2:     function Dataset Normalization(Raw Dataset)
3:         for each data instance do
4:             cast into integer value
5:             normalize (Eq. (5.1))
6:             NormalizedDataset
7:         end for
8:     end function
9:     function Dataset Balancing(NormalizedDataset)
10:        Pick 10% of normal instances randomly
11:        BalancedDataset
12:    end function
13:    InputDataset ← BalancedDataset
14:    return InputDataset
15: end function
```

the timestamps were also left for the normalization step. Some of the attributes have question marks, ?, to indicate unavailable values. We use one alternative in which the question mark is assigned to a constant zero value [70]. After all data are transformed into numerical values, attribute normalization is needed [23]. Data normalization is a process; hence, all value ranges of each attribute are equal. We adopt the mean range method [71] in which each data item is linearly normalized between zero and one to avoid the undue influence of different scales [70]. Eq. (5.1) shows the normalizing formula.

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}, \tag{5.1}$$

where z_i denotes the normalized value, x_i refers to the corresponding attribute value and $min(x)$ and $max(x)$ are the minimum and maximum values of the attribute, respectively.

The reduced "CLS" data are a good representation of a real network, in which normal instances significantly outnumber attack instances. The ratio between the normal and attack instances is 10:1 for both unbalanced training and the test dataset as shown in Table 5.1. This property might be biased to the training model and affect the model performance [59] [72]. To alleviate this, we balance the dataset by selecting 10% of the normal instances randomly. However, we set particular value as the seed of the random number generator for reproducibility purposes. The ratio between normal and attack instances became 1:1, which is an appropriate proportion for the training phase [72]. D-FES is trained using the balanced dataset and then verified on the unbalanced dataset.

5.2 Evaluation Metrics

We ensured that the evaluation of the performance of D-FES was fair by adopting the most well-referenced model-performance measures [73]: accuracy (Acc), Detection Rate (DR), False Alarm Rate (FAR), Mcc, $Precision$, F_1 score, CPU time to build model (TBM), and CPU time to test the model (TT). Acc shows the overall effectiveness of an algorithm [74]. DR, also known as $Recall$, refers to the number of impersonation attacks detected divided by the total number of impersonation attack instances in the test dataset. Unlike $Recall$, $Precision$ counts the number of impersonation attacks detected

among the total number of instances classified as an attack. The F_1 score measures the harmonic mean of *Precision* and *Recall*. FAR is the number of normal instances classified as an attack divided by the total number of normal instances in the test dataset while FNR shows the number of attack instances that are unable to be detected. Mcc represents the correlation coefficient between the detected and observed data [75]. Intuitively, our goal is to achieve a high Acc, DR, $Precision$, Mcc, and F_1 score and at the same time, maintaining low FAR, TBM, and TT. The above measures can be defined by Eqs. (5.2), (5.3), (5.4), (5.5), (5.6), (5.7), and (5.8):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5.2}$$

$$DR(Recall) = \frac{TP}{TP + FN}, \tag{5.3}$$

$$Precision = \frac{TP}{TP + FP}, \tag{5.4}$$

$$FAR = \frac{FP}{TN + FP}, \tag{5.5}$$

$$FNR = \frac{FN}{FN + TP}, \tag{5.6}$$

$$F_1 = \frac{2TP}{2TP + FP + FN}, \tag{5.7}$$

$$Mcc = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}, \tag{5.8}$$

where True Positive (TP) is the number of intrusions correctly classified as an attack, True Negative (TN) is the number of normal instances correctly classified as a benign packet, False Negative (FN) is the number of intrusions incorrectly classified as a benign packet, and False Positive (FP) is the number of normal instances incorrectly classified as an attack.

5.3 Experimental Result

The proposed D-FES is evaluated on a set of experiments. First, we implement and verify different architectures of the feature extractor, SAE. Second, we verify two feature selection approaches: filter-based and wrapper-based methods. We finally validate the usefulness and the utility of D-FES on a realistic unbalanced test dataset.

5.3.1 D-FES

Feature Extraction

We vary the SAE architectures to optimize the SAEs implementation with two hidden layers. The features generated from the first encoder layer are employed as the training data in the second encoder layer. Meanwhile, the size of each hidden layer is decreased accordingly such that the encoder in the second encoder layer learns an even smaller representation of the input data. The regression layer with the softmax activation function is then implemented in the final step. The four schemes are examined to determine the SAE learning characteristics. The first scheme, Imbalance_40, has two hidden layers with 40 and 10 hidden neurons in each layer. The second scheme, Imbalance_100, also has two hidden layers; however, it employs 100 and 50 hidden neurons in each layer. Although there is no strict rule for determining the number of hidden neurons, we consider a universal rule of thumb [76], which ranges from 70% to 90% from inputs. The third and fourth schemes, named Balance_40 and Balance_100, have

Table 5.2: The Evaluation of SAE's Schemes

SAE Scheme	DR (%)	FAR (%)	Acc (%)	F_1 (%)
Imbalance_40 (154:40:10:4)	64.65	**1.03**	96.30	73.13
Imbalance_100 (154:100:50:4)	85.30	1.98	**97.03**	81.75
Balance_40 (154:40:10:4)	72.58	18.87	77.21	74.48
Balance_100 (154:100:50:4)	**95.35**	18.90	87.63	**87.59**

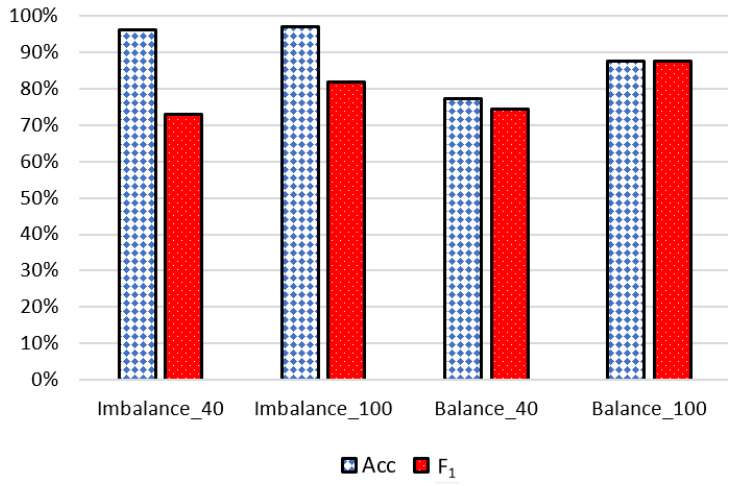


Figure 5.1: Evaluation of SAE's Scheme on Acc and F_1 Score. The red bar represents F_1 score while the blue bar represents Acc rate.

the same hidden layer architecture with the first and second schemes, respectively; however, in this case, we use the balanced dataset, because the common assumption is that a classifier model built by a highly unbalanced data distribution performs poorly on minority class detection [77]. For our testing purpose, we leverage all four classes contained in the AWID dataset.

Table 5.2 shows the evaluation of the SAE schemes. Each model uses either balanced or unbalanced data for the SAE algorithm with following parameters: (input features: number of features in 1^{st} hidden layer: number of features in 2^{nd} hidden layer: target classes. The SAE architectures with 100 hidden neurons have higher DR than those with 40 hidden neurons. On the other hand, the SAE architectures with 40 hidden neurons have lower FAR than those with 100 hidden neurons. To draw a proper conclusion, other performance metrics that consider whole classes are needed as the DR checks for the attack class only and the FAR measures for the normal class only. The Acc metric would be affected by the distribution of data, for which different balanced and unbalanced distributions may result in an incorrect conclusion. If we consider the Acc metric only as in Fig. 5.1, we may incorrectly select the Imbalance_100 with 97.03% accuracy, whereas the Balance_100 only achieved 87.63% accuracy. In fact, the Imbalance_100 achieved the highest accuracy rate because of the unbalanced proportion of normal class to attack class. We obtain the best performance by checking F_1 score, for which the Balance_100 has achieved the highest F_1 score among all schemes with 87.59%. Therefore, we choose the SAE architecture with 154:100:50:4 topology.

Table 5.3: Feature Set Comparisons Between Feature Selection and D-FES

Method	Selected Features	D-FES
CFS	5, 38, 70, 71, 154	38, 71, 154, 197
Corr	47, 50, 51, 67, 68, 71, 73, 82	71, 155, 156, 159, 161, 165, 166, 179, 181, 191, 193, 197
ANN	4, 7, 38, 77, 82, 94, 107, 118	4, 7, 38, 67, 73, 82, 94, 107, 108, 111, 112, 122, 138, 140, 142, 154, 161, 166, 192, 193, 201, 204
SVM	47, 64, 82, 94, 107, 108, 122, 154	4, 7, 47, 64, 68, 70, 73, 78, 82, 90, 94, 98, 107, 108, 111, 112, 122, 130, 141, 154, 159
C4.5	11, 38, 61, 66, 68, 71, 76, 77, 107, 119, 140	61, 76, 77, 82, 107, 108, 109, 111, 112, 119, 158, 160

Feature Selection

To show the effectiveness of D-FES, we compare the following feature selection methods:

- CfsSubsetEval [78] (CFS) considers the predictive ability of each feature individually and the degree of redundancy between them, to evaluate the importance of a subset of features. This approach selects subsets of features that are highly correlated with the class while having low inter-correlation.
- Correlation (Corr) measures the correlation between the feature and the class to evaluate the importance of a subset of features.
- The weight from a trained ANN model mimics the importance of the correspondence input. By selecting the important features only, the training process becomes lighter and faster than before [41].
- SVM measures the importance of each feature based on the weight of the SVM classification results.
- C4.5 is one of the decision tree approaches. It can select a subset of features that are not highly correlated. Correlated features should be in the same split; hence, features that belong to different splits are not highly correlated [66].

A filter-based method usually measures the correlation and redundancy of each attribute without executing a learning algorithm. Therefore, the filter-based method is lightweight and fast. On the other hand, the wrapper-based method examines the results of any learning algorithm that outputs a subset of features [79]. CFS and Corr belong to the filter-based techniques, whereas ANN, SVM, and C4.5 are wrapper-based methods.

We select a subset of features using the wrapper-based method for considering each feature weight. For ANN, we first set a threshold weight value, and if the weight of a feature is higher than the threshold, then the feature is selected. The SVM attribute selection function ranks the features based on their weight values. The subset of features with a higher weight value than the predefined threshold value is then selected. Similarly, C4.5 produces a deep binary tree. We select the features that belong to the top-three levels in the tree. CFS produces a fixed number of selected features and Corr provides a correlated feature list.

During ANN training for both feature selection and classification, we optimize the trained model using a separate validation dataset; that is, we separate the dataset into three parts: training data, validation data and testing data in the following proportions: 70%, 15%, and 15%, respectively. The training data are used as input into the ANN during training, and the weights of neurons are adjusted

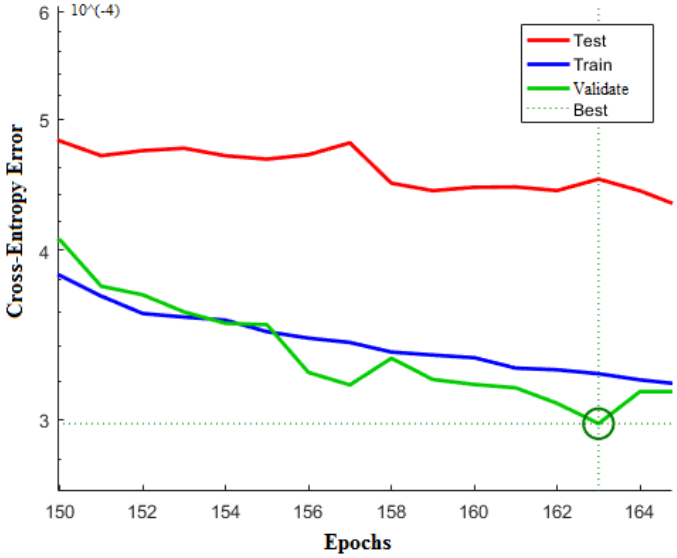
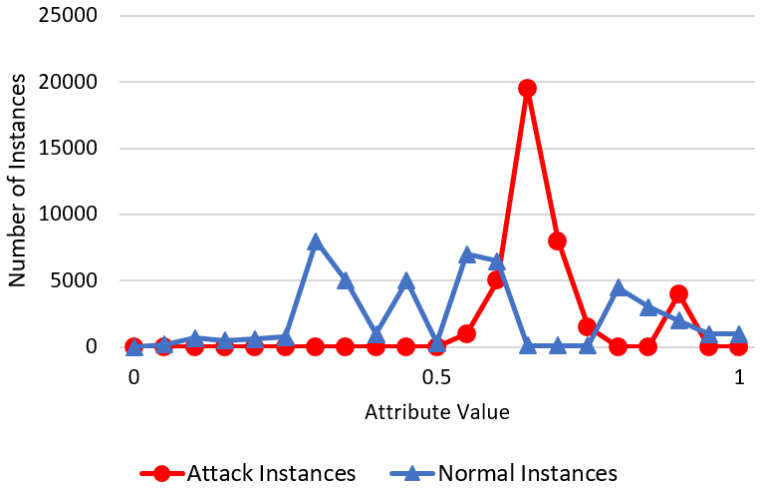


Figure 5.2: Cross entropy error of ANN. The best validation performance is achieved at the epoch of 163.

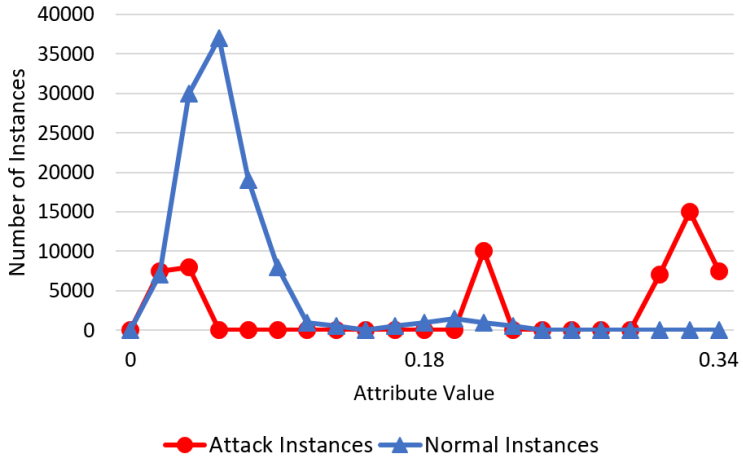
during the training according to its classification error. The validation data are used to measure model generalization providing useful information on when to terminate the training process. The testing data is used for an independent measure of the model performance after training. The model is said to be optimized when it reaches the smallest average square error on the validation dataset. Fig. 5.2 shows an example of ANN performance for the cross-entropy error during ANN training. At the epoch of 163, the cross-entropy error, a logarithmic-based error measurement comparing the output values and desired values, starts increasing, meaning that at the epoch of 163, the model is optimized. Although the training data output decreasing error values after the epoch point of 163, the performance of the model no longer continues to improve, as the decreasing cross-entropy error may indicate the possibility of overfitting.

Table 5.3 contains all the feature lists selected from the various feature selection methods. Different methods output various selected features, which demonstrated the advantage of automatic learning of underlying data done by machine learning models. By leveraging this advantage, we can avoid to rely on expert knowledge to select the most suitable features for detecting an impersonation attack. In Table 5.3, some features are essential for detecting an impersonation attack, which can be the key for classifying impersonation instances. These are the 4^{th} and the 7^{th}, which are selected by the ANN and SVM and the 71^{st}, which is selected by CFS and Corr. However, relying to these features solely might become a vulnerability. Pei *et al.* [80] demonstrated adversaries can add or modify particular features in order to make the trained model of IDS incorrectly classify an attack as benign. D-FES can reduce the probability of this vulnerability to be exploited by incorporating new-transformed features from SAE. These new features contain more high-level abstractions to make classification decision.

The characteristics of sample of selected features are shown in Fig. 5.3. The blue line indicates normal instances, and at the same time, the red line depicts the characteristics of an impersonation attack. We can distinguish between normal and attack instances based on the attribute value of data instances. For example, once a data instance has an attribute value of 0.33 in the 166^{th} feature, the data instance has a high probability of being classified as an attack. This could be applied to the 38^{th} and



(a)



(b)

Figure 5.3: Characteristics of (a) 38^{th} and (b) 166^{th} features. The blue line represents normal instances while the red line represents attack instances.

Table 5.4: Model Comparisons on Selected Features

Model	DR (%)	FAR (%)	Acc (%)	F_1 (%)	Mcc (%)	TBM (s)
CFS	94.85	3.31	96.27	92.04	89.67	80
Corr	92.08	0.39	97.88	95.22	93.96	**2**
ANN	99.79	0.47	97.88	99.10	98.84	150
SVM	**99.86**	0.39	**99.67**	99.28	99.07	10,789
C4.5	99.43	**0.23**	99.61	**99.33**	**99.13**	1,294

Table 5.5: Model Comparisons on D-FES Feature Set

Model	DR (%)	FAR (%)	Acc (%)	F_1 (%)	Mcc (%)	TBM (s)
CFS	96.34	0.46	98.80	97.37	96.61	1,343
Corr	95.91	1.04	98.26	96.17	95.05	**1,264**
ANN	99.88	0.02	99.95	99.90	99.87	1,444
SVM	**99.92**	**0.01**	**99.97**	**99.94**	**99.92**	12,073
C4.5	99.55	0.38	99.60	99.12	98.86	2,595

other features as well.

Table 5.4 lists the performance of each algorithm on the selected feature set only. SVM achieved the highest DR (99.86%) and Mcc (99.07%). However, it requires CPU time of 10,789s to build a model, the longest time among the models observed. As expected, the filter-based methods (CFS and Corr) built their models quickly; however, they attained the lowest Mcc for CFS (89.67%).

Table 5.5 compares the performances of the candidate models on the feature sets that are produced by D-FES. SVM again achieved the highest DR (99.92%) and Mcc (99.92%). It also achieved the highest FAR with a value of only 0.01%. Similarly, the lowest Mcc is achieved by Corr (95.05%). This enables us to conclude that wrapper-based feature selections outperform filter-based feature selections. As SVM showed the best performance, we may consider the properties of selected features by SVM as described in Table 5.6.

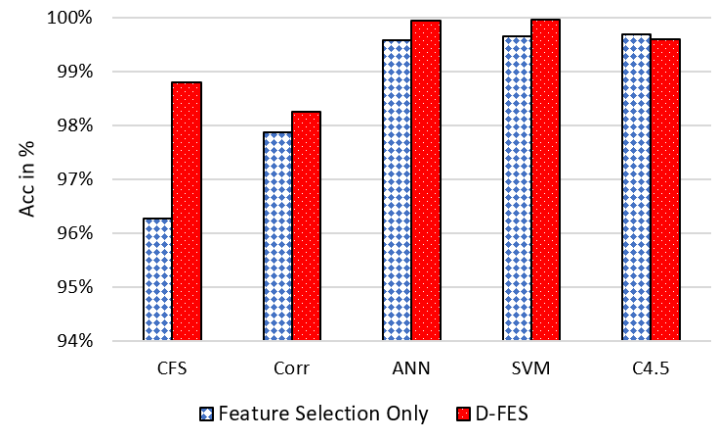
We observe the following patterns from Tables 5.4 and 5.5: Only two out of five methods (Corr and

Table 5.6: Feature Set Selected by D-FES-SVM

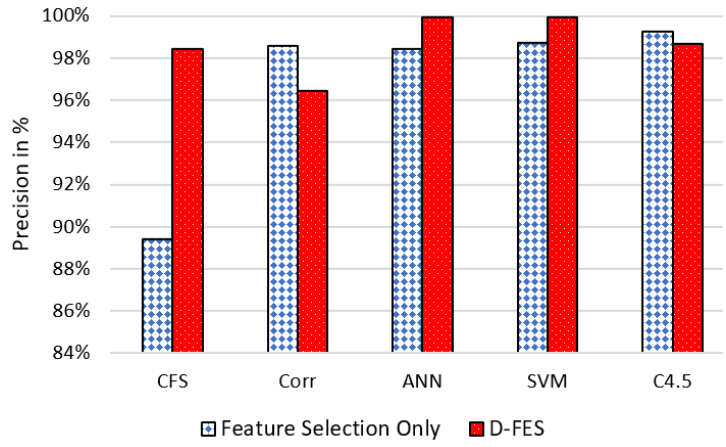
Index	Feature Name	Description
47	radiotap.datarate	Data rate (Mb/s)
64	wlan.fc.type_subtype	Type or Subtype
82	wlan.seq	Sequence number
94	wlan_mgt.fixed.capabilities.preamble	Short Preamble
107	wlan_mgt.fixed.timestamp	Timestamp
108	wlan_mgt.fixed.beacon	Beacon Interval
122	wlan_mgt.tim.dtim_period	DTIM period
154	data.len	Length

C4.5) showed lower FAR without D-FES, which we expect to minimize the FAR value of the proposed IDS. This phenomenon might exist because the original and extracted features are not correlated because Corr and C4.5 measure the correlation between each feature. Filter-based feature selection methods require much shorter CPU time compared to the CPU time taken by D-FES. However, D-FES improves the filter-based feature selections performance significantly.

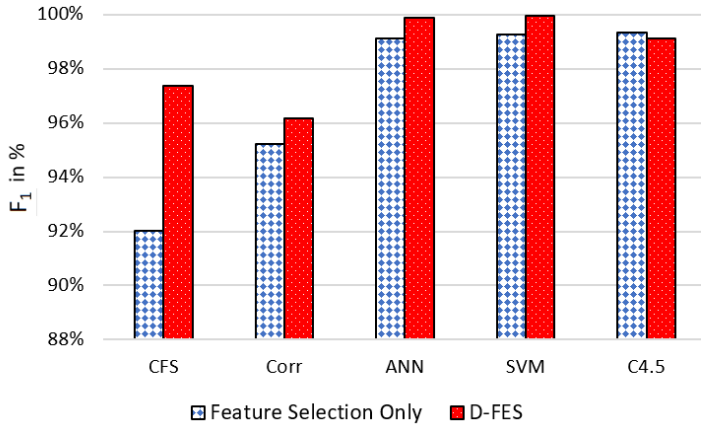
Similar patterns are captured by Fig. **??**, which depicts the performance of different models in terms of *Acc*, *Precision* and F_1 score, respectively. D-FES-SVM achieved the highest *Acc*, *Precision* and F_1 score of 99.97%, 99.96% and 99.94%, respectively. By D-FES, all methods achieve *Precision* of more than 96%, shows that D-FES can reduce the number of incorrect classification of normal instances as an attack. We can also observe that D-FES improves the *Acc* of filter-based feature selections significantly. Except for the C4.5, all feature selection methods are improved both the *Acc* and F_1 score by using D-FES. This makes the proposed D-FES a good candidate for an intrusion detector.



(a)

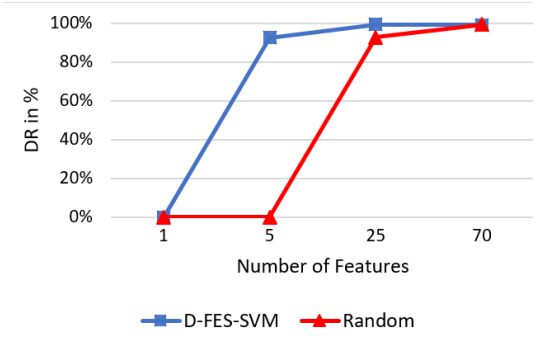


(b)

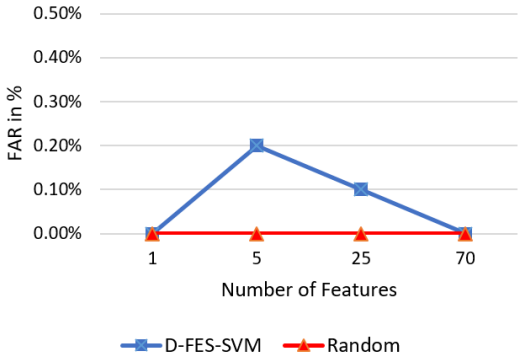


(c)

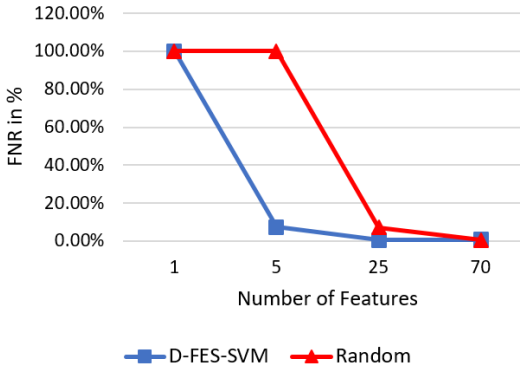
Figure 5.4: Models performance comparisons in terms of (a) *Acc*, (b) *Precision* and (c) F_1 score. The blue bar represents performances by feature selection only while the red bar represents performances by D-FES.



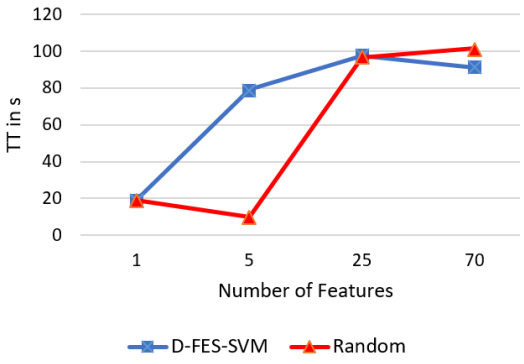
(a)



(b)



(c)



(d)

Figure 5.5: Model performance comparisons between D-FES and random method in terms of: (a) DR (b) FAR (c) FNR (d) TT

Table 5.7:
Single Attack: Impersonation Attack

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	99.911	0.031	99.903	99.955
Validation	99.904	0.033	99.898	99.953
Test	99.932	0.016	99.939	99.972
All	99.913	0.029	99.907	99.958

Table 5.8:
Single Attack: Flooding Attack

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	99.985	0.035	99.934	99.970
Validation	99.918	0.045	99.884	99.946
Test	99.959	0.029	99.932	99.969
All	99.971	0.036	99.926	99.966

D-FES on All Attacks in AWID

In this section, we generalized D-FES to all attacks existed in AWID dataset. By doing so, we show that D-FES achieved good performance not only for impersonation attack but also for other attacks. For experiment purposes, we distinguish train, validation and test dataset which comprises of 75%, 15% and 15% of original dataset and chosen randomly. Test dataset was not shown during training. The last row shows accumulation of all three previous datasets. We experimented with following schemes:

- Binary class: single attack and benign classes for each of injection, impersonation and flooding attacks
- Multiclass: a combination of 2 attacks, and all attacks

Table 5.7 shows experimental results on detecting single attack class only, which is impersonation attack. We can see that D-FES achieved 99.972% of accuracy during the test. While Tables 5.8 and 5.9 show experimental results on detecting flooding and injection attacks, respectively. The accuracy of flooding attack detection is lesser 0.003% of impersonation attack, which can be considerably same. Injection attack achieved the best result among others with 99.991% of accuracy during the test. We can also see in Table 5.9 that D-FES achieved 100% detection rate of injection attacks. However, this 100% of DR might be caused by overfitting due to statistical property of the dataset since the original author also faced a similar problem with injection attack detection.

Tables 5.10, 5.11 and 5.12 show experimental results on combination of attacks existed in AWID dataset which are impersonation and flooding, impersonation and injection and flooding and injection, respectively. Again, detection of injection attacks leads to overfitting shown by 100% of DR in Table 5.12. Table 5.13 shows D-FES performance tested to detect all attacks simultaneously. We can see in Table 5.13 that D-FES achieved 99.908% of detection rate and 0.115% of false alarm rate during the test with all attacks. These results outperform Usha and Kavitha [81], which achieved 99.20% and about 1%

Table 5.9:
Single Attack: Injection Attack

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	100.000	0.009	99.985	99.993
Validation	100.000	0.008	99.986	99.994
Test	100.000	0.012	99.980	99.991
All	100.000	0.009	99.985	99.993

Table 5.10:
Multiclass, Two Attacks:Impersonation and Flooding Attacks

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	99.959	0.365	99.672	99.755
Validation	99.952	0.360	99.676	99.757
Test	99.986	0.339	99.709	99.782
All	99.962	0.361	99.678	99.760

of detection accuracy and false alarm rate, respectively. This shows that D-FES can be generalized to detect any attacks.

We can observe in Table 5.14 each attack detection rate for every previous scheme. As expected, injection attack always causes overfitting. In overall, all attack types can be detected almost 100%.

Figure 5.6 shows DR comparison between each previous scheme and original work by Kolias *et al.* [2]. While Figure 5.7 also shows the DR comparison, however, excluding the Kolias work to see the difference of D-FES performances in more detail.

Figure 5.8 shows FAR comparison between each previous scheme and original work by Kolias *et al.* [2]. While Figure 5.9 also shows the FAR comparison, however, excluding the Kolias work to see the difference of D-FES performances in more detail.

In Table 5.15, we can observed important features for each attack class detection. These subset of features are important for who wants to detect particular attack only.

Table 5.11:
Multiclass, Two Attacks:Impersonation and Injection Attacks

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	99.951	0.024	99.958	99.965
Validate	99.924	0.020	99.948	99.957
Test	99.941	0.016	99.959	99.966
All	99.946	0.023	99.957	99.964

Table 5.12:
Multiclass, Two Attacks:Flooding and Injection Attacks

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	100.000	0.329	99.764	99.806
Validate	100.000	0.332	99.765	99.805
Test	100.000	0.396	99.717	99.767
All	100.000	0.340	99.757	99.800

Table 5.13:
Multiclass, Three Attacks:Impersonation, Flooding and Injection Attacks

Dataset	DR (%)	FAR (%)	F_1 (%)	Acc (%)
Train	99.943	0.109	99.917	99.917
Validate	99.897	0.145	99.829	99.871
Test	99.945	0.115	99.877	99.908
All	99.939	0.115	99.904	99.910

Table 5.14:
Detection Rate as per Scheme (in %)

Dataset	Impersonation	Flooding	Impersonation
Impersonation and Flooding	99.972	100.000	-
Impersonation and Injection	99.863	-	100.000
Injection and Flooding	-	100.000	100.000
All	99.850	99.808	100.000

Figure 5.6: DR with Kolias

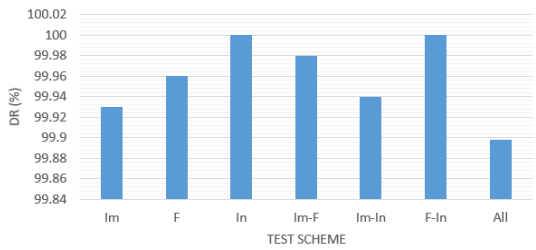


Figure 5.7: DR without Kolias

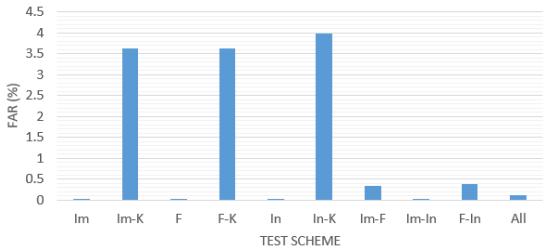


Figure 5.8: FAR with Kolias

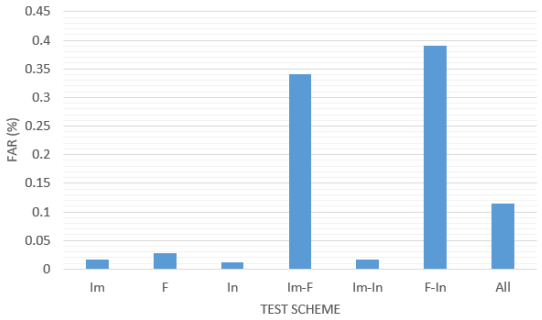


Figure 5.9: FAR without Kolias

Table 5.15:
Feature Sets Among All Schemes

Method	Feature Sets
Single Attack: Impersonation	4,7,38,61,66,67,71,73,77,82,93,94,107,108,118,12,142,154,181,193
Single Attack: Flooding	64,90,77,67,66,70,75,38,118,4,82,7,112,73,108,176,128,177,98
Single Attack: Injection	191, 199, 190, 7, 131,73,140,67,68
Two Attacks: Im-F	67,73,130,90,64,78,70,111,98,75,66,118,38,82,112,107,7,4
Two Attacks: Im-In	93,108,122,203,94,66,140,142,82,38,67,68,107,47
Two Attacks: F-In	191, 199, 190, 7, 131,73,140,67,68
Three Attacks	183,142,73,130,90,140,64,98,75,77,94,70,66,38,82,67,118,68,112,107,4,7

Table 5.16: The evaluation of our proposed scheme

Input	$DR(\%)$	$FAR(\%)$	$Acc(\%)$	$Precision(\%)$	$F_1(\%)$
Original data	100.00	57.17	55.93	34.20	50.97
1^{st} hidden layer	100.00	57.48	55.68	34.08	50.83
2^{nd} hidden layer	92.18	4.40	94.81	86.15	89.06

5.3.2 Fully Unsupervised IDS

There are two hidden layers in the SAE network with 100 and 50 neurons accordingly. The encoder in the second layer fed with features formed by the first layer of the encoder. The softmax activation function is implemented in the final stage of the SAE to optimize the SAE training. The 50 features obtained from the SAE are then forwarded to k-means clustering algorithm as input. We use random initialization for k-means clustering algorithm. However, we set particular value as a random number seed for reproducibility purpose. We compare clustering results from three inputs: original data, features from the first hidden layer of the SAE and features from the second hidden layer of the SAE as shown in Table 5.16.

We observe the limitation of a traditional k-means algorithm, which unable to clusters complex and high dimensional data of AWID dataset, as expressed by 55.93% of accuracy only. Although 100 features coming from the 1^{st} hidden layer achieved 100% of detection rate, the false alarm rate is still unacceptable with 57.48%. The k-means algorithm fed by 50 features from the 2^{nd} hidden layer achieved the best performance among all as shown by the highest F_1 score (89.06%) and Acc (94.81%), also the lowest FAR (4.40%). Despite a bit lower detection rate, our proposed scheme improves the traditional k-means algorithm in overall by almost twice F_1 score and accuracy.

Fig. 7.5 shows cluster assignment result in Euclidean space, by our proposed scheme. Black dots represent attack instances, while gray dots represent benign instances. The location of cluster centroid for each cluster is expressed by X mark.

We also compare the performance of our proposed scheme against two previous related work by Kolias *et al.*[2] and Aminanto and Kim [82] as shown in Table 5.17. Our proposed scheme can classify impersonation attack instances with a detection rate of 92.18% while maintaining low FAR, 4.40%. Kolias *et al.* [2] tested various classification algorithms such as Random Tree, Random Forest, J48, Naive Bayes, *etc.*, on AWID dataset. Among all methods, Naive Bayes algorithm showed the best performance by correctly classifying 4,419 out of 20,079 impersonation instances. It achieved approximately 22% DR only, which is unsatisfactory. Aminanto and Kim [83] proposed another impersonation detector by combining Artificial Neural Network (ANN) with SAE. They successfully improved the IDS model for impersonation attack detection task by achieving a DR of 65.18% and a FAR of 0.14%. In this study, we leverage SAE for assisting traditional k-means clustering with extracted features. We still have a high false alarm rate, which leads to a severe impact of IDS [15]. However, we can accept false alarm rate value about 4% since we use fully unsupervised approach here. We can adjust the parameters and cut the FAR down, but, less FAR or high DR remains a tradeoff for users and will be discussed in further work. We observe the advantage of SAE for abstracting a complex and high dimensional data to assist traditional clustering algorithm which is shown by reliable DR and F_1 score achieved by our proposed scheme.

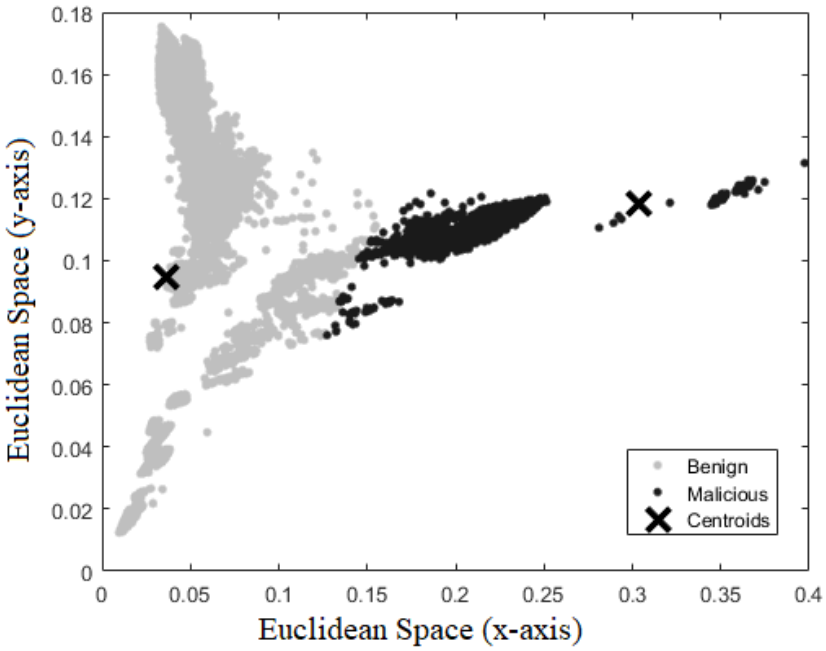


Figure 5.10: Cluster assignment result in Euclidean space by our proposed scheme

Table 5.17: Comparison with previous work

Method	$DR(\%)$	$FAR(\%)$	$Acc(\%)$	$Precision(\%)$	$F_1(\%)$
Kolias *et al.*[2]	22.01	0.02	97.14	97.57	35.92
Aminanto and Kim [82]	65.18	0.14	98.59	94.53	77.16
Our proposed scheme [84]	92.18	4.40	94.81	86.15	89.06

Table 5.18: IDSs Leveraging SAE

Publication	Role of SAE	Combined with
AK16a [82]	Classifier	ANN
AK16b [86]	Feature Extractor	Softmax Regression
AK17 [84]	Clustering	K-means Clustering
ACTYK17 [87]	Feature Extractor	SVM, DT, ANN

5.4 Comparison

The goal of deep learning method is learning feature hierarchies from the lower level to higher level features [85]. The technique can learn features independently at multiple levels of abstraction, and thus discover complicated functions mapping between the input to the output directly from raw data without depending on customized features by the experts. In higher-level abstractions, humans often have no idea to see the relation and connection from the raw sensory input. Therefore, the ability to learn sophisticated features, also called as feature extraction, will become necessarily needed as the amount of data increased sharply [85]. SAE is one good instance of feature extractors. Therefore, we discuss several previous works which implement SAE as the feature extractor and other roles as well, in the IDS module as shown in Table 5.18.

Feature extraction by SAE can reduce the complexity of original features of the dataset. However, besides as a feature extractor, we validated that SAE can also be used for classifying and clustering tasks as shown in Table 5.18. In AK16b [86], we used semi-supervised approach for our IDS which contains feature extractor (unsupervised learning) and classifier (supervised learning). We leveraged SAE for feature extraction and regression layer with softmax activation function for a classifier. We implemented SAE as feature extractor as well in ACTYK17 [87], but we leveraged ANN, DT, and SVM as a feature selection. In other words, we combine stacked feature extraction and weighted feature selections. By our experiments [87], we improved our feature learning process by combining stacked feature extraction with weighted feature selection. The feature extraction of SAE is capable of transforming the original features into a more meaningful representation by reconstructing its input and providing a way to check that the relevant information in the data has been captured. SAE can be efficiently used for unsupervised learning on a complex dataset.

Unlike two previous approaches, we use SAE for other roles than a feature extractor, namely classifying and clustering methods in AK16a [82] and AK17 [84], respectively. We adopted ANN as a feature selection since the weight from trained models mimics the significance of the corresponding input [82]. By selecting the essential features only, the training process becomes lighter and faster than before. In AK16a [82], we exploited SAE as a classifier, since this employs consecutive layers of processing stages in hierarchical manners for pattern classification and feature or representation learning. On the other hand, we proposed a novel fully unsupervised method [84] which can detect attacks without prior information on data label. An unsupervised SAE equips our method for extracting features and a K-means clustering algorithm for clustering task.

Kolias *et al.* [2] tested some existing machine learning models on the dataset in a heuristic manner. The lowest detection rate is observed particularly on impersonation attack reaching an accuracy of 22% only. Therefore, we focus on improving impersonation detection and hence comparing our approaches on impersonation detection as summarized in Table 5.19. Detection Rate (DR) refers to the number of

Table 5.19: Comparison on Impersonation Detection

Method	Detection Rate (%)	False Alarm Rate (%)
AK16a [82]	65.178	0.143
AK16b [86]	92.674	2.500
AK17 [84]	92.180	4.400
ACTYK17 [87]	99.918	0.012
KKSG15 [2]	22.008	0.021

attacks detected divided by the total number of attack instances in the test dataset while False Alarm Rate (FAR) is the number of normal instances classified as an attack divided by the total number of normal instances in the test dataset.

From Table 5.19, we can observe that SAE can improve the performance of our IDS compared to KKSG15 [2]. We verified that SAE achieved high-level abstraction of complex and massive Wi-Fi network data. The SAE's model free properties and learnability on complex and large-scale data fit into the open nature of Wi-Fi networks. Among all IDSs, the one using SAE as a classifier achieved the lowest impersonation attack detection rate with 65.178% only. It shows that SAE can be a classifier but not excellent as the original role of SAE is a feature extractor. The usability of SAE as a feature extractor validated by AK16b [86] and ACTYK17 [87] which achieved highest DR. Even more, by a combination of SAE extractor and weighted selection [87], we delivered the best performance of DR and FAR among other. Besides that, we found an interesting fact that SAE can assist K-means clustering algorithm to achieve better performance with DR of 92.180% [84]. However, we need to analyze further to reduce the FAR since it reached the highest FAR which is undesirable in IDS.

Chapter 6. Related Work

An IDS has been studied for decades especially on anomaly-based IDSs. Fragkiadakis *et al.* [14] proposed an anomaly-based IDS using *Dempster-Shafer's rule* for measurement. Shah *et al.* [88] also developed an evidence theory for combining anomaly-based and misuse-based IDSs. Bostani *et al.* [23] proposed an anomaly-based IDS by modifying an optimum path forest combined with k-means clustering. A distributed anomaly-based IDS, called $TermID$, proposed by Kolias *et al.* [89] incorporating ant colony optimization and rule induction in a reduced data operation to achieve data parallelism and reduce privacy risks.

Feature selection techniques are useful for reducing model complexity, which leads to faster learning and real-time processes. Kayacik *et al.* [90] investigated the relevance of each feature in the KDD'99 Dataset with a list of the most relevant features for each class label and provided useful discussions on the roles of information gain theories. Their work confirmed the importance and the role of feature selection for building an accurate IDS model. Puthran *et al.* [91] also worked on relevant features in the KDD'99 Dataset and improved the decision tree by using binary and quad splits. Almusallam *et al.* [59] leveraged a filter-based feature selection method. Zaman and Karray [92] categorized IDSs based on the Transmission Control Protocol/Internet Protocol (TCP/IP) network model using a feature selection method known as the Enhanced Support Vector Decision Function (ESVDF). Louvieris *et al.* [93] proposed an effect-based feature identification IDS using naïve Bayes as a feature selection method. Zhu *et al.* [94] also proposed a feature selection method using a multi-objective approach.

On the other hand, Manekar and Waghmare [95] leveraged Particle Swarm Optimization (PSO) and SVM. PSO performs feature optimization to obtain an optimized feature, after which SVM conducts the classification task. A similar approach was introduced by Saxena and Richariya [96], although Schaffernicht and Gross [97] introduced the concept of weighted feature selection. Exploiting SVM-based algorithms as a feature selection method was proposed by Guyon *et al.* [64]. This method leveraged the weights adjusted during support vector learning and resulted in ranking the importance of input features. Another related approach was proposed by Wang [41] who ranked input features based on weights learned by an ANN. This method showed the ability of deep neural networks to find useful features among the raw data. Aljawarneh *et al.* [98] proposed a hybrid model of feature selection and an ensemble of classifiers which tend to be computationally demanding.

We have examined several feature selection methods for IDS. Huseynov *et al.* [26] inspected ant colony clustering method to find feature clusters of botnet traffic. The selected features in [26] are independent of traffic payload and represent the communication patterns of botnet traffic. However, this botnet detection does not scale for large and noisy dataset due to the absence of control mechanism for clustering threshold. Kim *et al.* [27] tested artificial immune system and swarm intelligence-based clustering to detect unknown attacks. Furthermore, Aminanto *et al.* [99] discussed the utility of ant clustering algorithm and fuzzy inference system for IDS. We can claim that their bio-inspired clustering methods need to be scrutinized further.

Not only feature selection but also feature extraction has been proposed to improve classification performance. Shin *et al.* [100] leveraged SAE for unsupervised feature learning in the field of medical imaging. This method showed that SAE, which is a type of deep learning techniques, can be efficiently used for unsupervised feature learning on a complex dataset. Unsupervised learning by SAE can be used

to learn hierarchical features that are useful for limited instances on a dataset.

Roy *et al.* [101] proposed an IDS leveraging deep learning models. They validated that a deep learning approach can improve IDS performance. Deep Neural Network (DNN) is selected comprising of multilayer feedforward neural network with 400 hidden layers. Shallow models, rectifier and softmax activation functions, were used in the output layer. The advantage of the feedforward neural network is the ability to provide a precise approximation for complex multivariate nonlinear function directly from input values and strong modeling capabilities for large classes. Besides that, the authors claimed that DNN is better than DBN since the discriminating ability for pattern classification by characterizing the posterior distributions of classes conditioned on the data [101].

For validation purposes, KDD Cup'99 dataset was used. This dataset has 41 features that become the input to the network. The authors divided training data into 75% of training data and 25% of validation data. They also compared the performance of a shallow classifier, SVM. Based on their experimental result, DNN outperforms SVM by the accuracy of 99.994%, while SVM achieved 84.635% only. This result showed the effectiveness of DNN for IDS purposes.

Another DNN but the different architecture was proposed by Potluri and Diedrich [102] in 2016. This paper mainly focuses on improving DNN implementation for IDS by using multi-core CPUs and GPUs. This is important since DNN requires large computation for training [103]. In the beginning, the authors [102] review some IDSs using a hardware enhancement: GPU, multicores CPU, memory management, and FPGA. Also, load balancing and splitting or parallel processing were discussed. For the proposed approach, a deep learning model, SAE was chosen to construct the DNN in this work. The architecture of this network has 41 input features from NSL-KDD dataset, 20 neurons in the first hidden layer by first AE, ten neurons in the second hidden layer by second AE and five neurons in the output layer containing softmax activation function. In the training phase, each AE is trained separately but in sequence since the hidden layer of first AE becomes the input of second AE. There are two times of fine-tuning processes, first done by softmax activation function and second done by backpropagation through the entire network.

NSL KDD dataset was selected for testing the approach. This dataset is a revised version of KDD Cup'99 dataset. It has the same number of features which is 41 but with more rational distributions and without redundant instances as exist in KDD Cup'99 dataset. The authors firstly tested the network with different attack class combinations from 2 classes to 4 classes. The lesser number of attack classes performs better than the higher number of attack classes as expected since imbalance class distribution leads to a good result for fewer attack types. For the accelerated purpose, the authors used two different CPUs and a GPU. They also experimented using serial and parallel CPUs. Their experimental result shows that training using parallel CPU achieved three times faster than serial CPU. Training using GPU delivered similar performance to parallel CPU as well. An interesting point here is the training using parallel of the second CPU was more rapid than GPU. The authors explained that this case happened due to the clock speed of modern CPU is too high. Unfortunately, the authors do not provide performance comparison regarding detection accuracy or false alarm rate.

Self Taught Learning (STL) was proposed as a deep learning model for IDS by Niyaz *et al.* [7]. The authors mentioned two challenges to develop an efficient IDS. The first challenge is feature selection task since selected features for a particular attack might different for other attack types. The second challenge is limited amounts of a labeled dataset for training purpose. Therefore, a generative deep learning model was chosen to deal with this unlabeled dataset. The proposed approach is STL which comprises of two stages, Unsupervised Feature Learning (UFL) and Supervised Feature Learning (SFL). For UFL, the

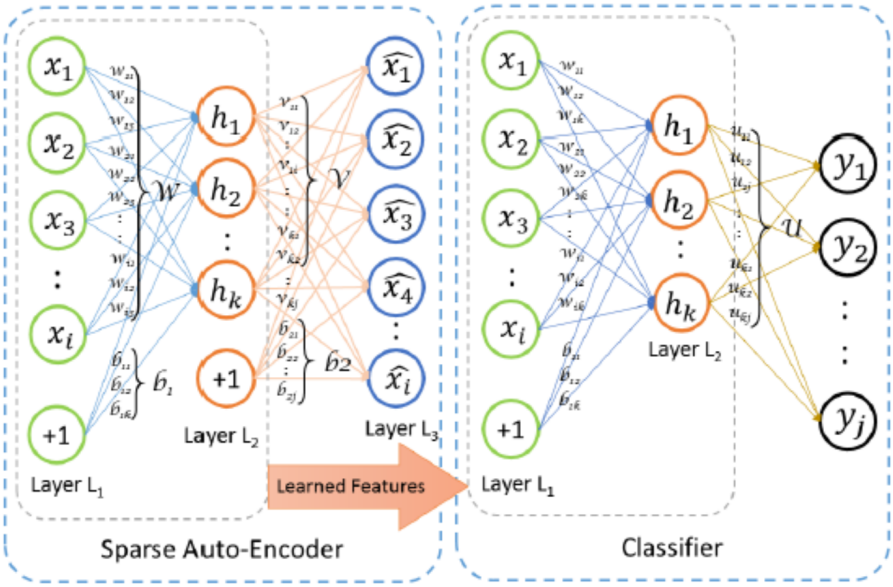


Figure 6.1: The two stage of STL [7]

authors leveraged sparse AE while softmax regression for SFL. Fig. 6.1 shows the two-stage process of STL used in this paper. The UFL accounts for feature extraction with unlabeled dataset while the SFL accounts for classification task with labeled data.

The authors verified their approach using NSL KDD dataset. Before the training process, the authors defined a pre-processing step for the dataset which contains 1-to-N encoding and min-max normalization. After 1-to-N encoding process, 121 features were ready for normalization step and input features for the UFL. For testing purposes, 10-fold cross validation selected for training data and test dataset from NSL KDD dataset chosen for test data. The authors also evaluated the STL for three different attack combinations, 2-class, 5-class, and 23-class. In general, their STL achieved higher than 98% of classification accuracy for all combinations during the training phase. In the testing phase, the STL attained an accuracy of 88.39% and 79.10% for 2-class and 5-class classifications, respectively. At the end of this paper, the authors mentioned the future work are to develop real-time IDS using deep learning models and an IDS with on-the-go feature learning on raw network traffic.

Yu *et al.* [104] introduced a session based IDS using a deep learning architecture. They came up with common IDS shortcomings: high false positive and false negative, most attack features in common dataset are heavily structured and have special semantics involved in specific expert knowledge, and the heavily hand-crafted dataset is closely related to particular attack classes. Therefore, a deep learning model was leveraged since unsupervised deep learning can learn essential features automatically from large data. The proposed approach is comprised of extracting features from raw data and applying unsupervised Stacked Denoising AE (SDAE). Session-based data were extracted from raw network data, which drawn from UNB ISCX 2012 and CTU-13 for benign and botnet instances, respectively. Since the data are extracted from raw data, a pre-processing step is necessary. Data pre-processing procedure consists of session construction, record construction, and normalization. Session construction distinguishes three different sessions namely TCP, UDP, and ICMP. Record construction draws first 17 features from packet headers and the rest 983 features from the payload. While normalization uses the min-max function. The SDAE itself contains two hidden layers and a softmax regression layer for the classification task. For the denoising purpose, the authors randomly set input features using zero value for 10%, 20% and

30% of input features. The authors mentioned that the advantage of using SDAE is three-fold: able to learn important features from unlabeled instances automatically, denoising strategy makes it robust from missing and noisy input, and good dimensionality reduction when the hidden layer is non-linear.

As mentioned earlier, the authors extracted data from raw network data from UNB ISCX 2012 and CTU-13. They also measured accuracy, precision, recall, F-score and ROC curve as the performance metrics. Binary and multi-class classifications were used along with 43% of the dataset and whole dataset combinations to verify the SDAE performance. The SDAE also compared to other deep learning models, namely SAE, DBN and AE-CNN models. In overall, the SDA achieved the best performance with the highest accuracy rate of 98.11% of multi-class classification using the whole dataset.

Kim *et al.* [48] adopted the generative approach of LSTM-RNN for an IDS purpose. They leveraged softmax regression layer as the output layer. While other hyper-parameters are 100, 50 and 500 of the time step, batch size, and epoch, respectively. Also, Stochastic Gradient Descent (SGD) and Mean Square Error (MSE) were used as the optimizer and loss function, respectively. 41 input features drawn from KDD Cup'99 dataset. Experimental results show the best learning rate is 0.01 and hidden layer size is 80 with 98.88% of detection rate and 10.04% of false alarm rate. Similar network topology also proposed by Liu *et al.* [105] with different hyper-parameters: time step, batch size, and age are 50, 100 and 500, respectively. Tested using KDD Cup'99 dataset, 98.3% of detection rate and 5.58% of false alarm rate were achieved.

Software Defined Networks (SDN) is an emerging network technology of today's applications since it has a unique property that builds by controller plane and data plane. The controller plane decouples the network control and forwarding functions. The centralized approach of controller plane makes SDN controller suitable for IDS function due to complete network overview captured by the controller. Unfortunately, due to the separation of control and data plane, it leads to some critical threats. Therefore, Tang *et al.* [8] proposed a DNN approach for IDS in SDN context. The DNN architecture is 6-12-6-3-2 which means six input features, three hidden layers with 12, 6 and three neurons for each layer and two classes output as shown in Fig. 6.2.

In this paper, the authors used NSL KDD dataset to check their approach. Since the dataset has 41 features, the authors selected six features that are fundamental features in SDN based on their expertise. They measured accuracy, precision, recall, F-score and ROC curve as the performance metrics. Based on the experimental result, the learning rate of 0.001 is the best hyper-parameter since the learning rate of 0.0001 already over-fitted. The proposed approach then compared to previous work that leverages various machine learning models. The DNN achieved 75.75% of accuracy, which is lower than other methods using whole 41 features but higher than other methods using six features only. From this fact, the authors claimed that the proposed DNN could generalize and abstract the characteristics of network traffic with limited of features alone.

Yin *et al.* [106] highlighted the shortcoming of traditional machine learning methodologies that are unable to solve the massive intrusion data classification problem efficiently. They took advantages of RNN implementation in IDS context. RNN contains forward and backward propagation, where the latter is the same neural network which computes the residual of forward propagation. The proposed RNN-IDS begins with data pre-processing step which comprises of numericalization and normalization. Feature-ready data are propagated to training step of RNN. The output model from the training is used to a testing phase using test dataset.

For experimental purposes, the authors used NSL KDD dataset, both training, and test dataset. The original features are 41 features but became 122 features after numericalization which maps string

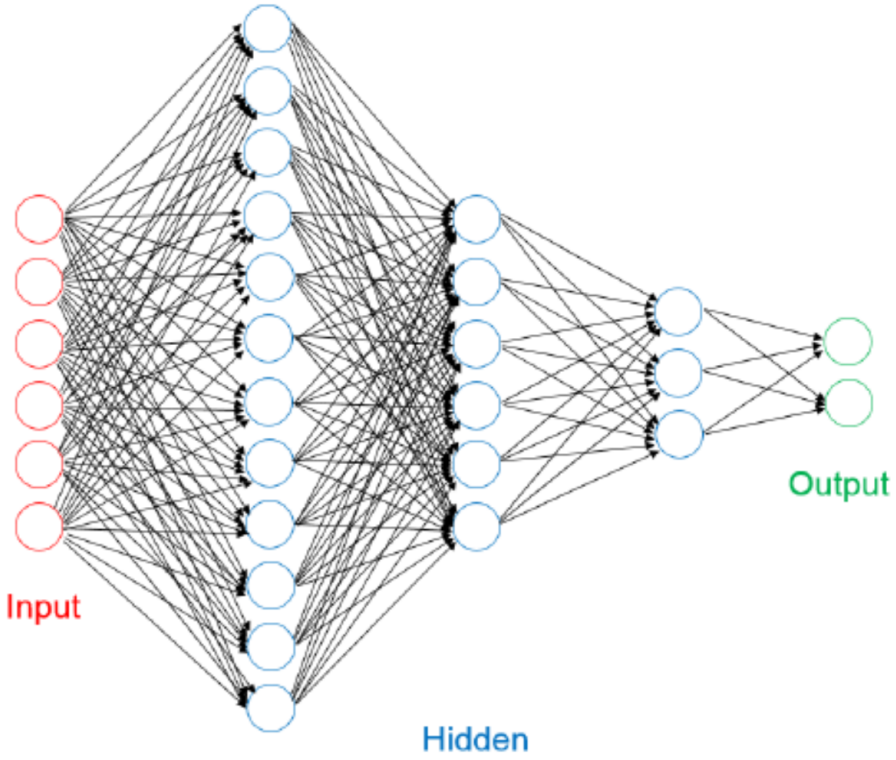


Figure 6.2: Architecture of DNN used in [8]

to binary. Two types of classification were tested namely binary and multi-class classification. Based on the experimental results, the best hyper-parameter during binary classification is learning rate of 0.1, the epoch of 100 and hidden nodes of 80 with an accuracy of 83.28% (using KDDTest$^+$). Meanwhile, during multi-class classification, the best hyper-parameter is learning rate of 0.5 and hidden nodes of 80 with an accuracy of 81.29%. The RNN-IDS outperformed other machine learning methodologies tested by the authors for both binary and multi-class classification.

Li *et al.* [107] experimented using CNN as the IDS's feature extractor and classifier. CNN achieved many successful implementations in image-related classification tasks, however, still a big challenge for text classification. Therefore, the main challenge of implementing CNN in IDS context is the image conversion step, which is proposed by Li *et al.* [107]. NSL KDD dataset was used for experimental purposes. The image conversion step begins by mapping of 41 original features into 464 binary vectors. The mapping step comprises of two types mapping, one hot encoding and one hot encoder with ten binary vectors for symbolic and continuous features, respectively. The image conversion step continues with converting 464 vectors into 8×8 pixel images. These images are ready for training input of CNN. The authors decided to experiment with learned CNN models, ResNet 50 and GoogLeNet. Experimental results on KDDTest$^+$ show the accuracy of 79.14% and 77.14% using ResNet 50 and GoogLeNet, respectively. Although this result does not improve the state of the art of IDS, this work demonstrated how to apply CNN with image conversion in IDS context.

LSTM-RNN became more popular due to its successful applications in various research areas. The ability to consider previous events can be applied for IDS' objective which is learning from previous attack behaviors. Some IDSs that were implementing LSTM-RNN described as follow. Staudemeyer [46] experimented various network topologies of LSTM-RNN for network traffic modeling as a time series.

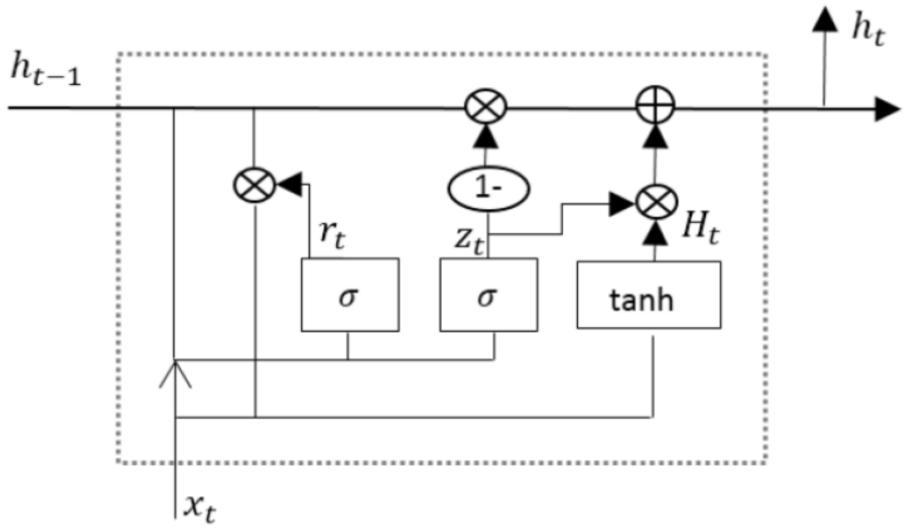


Figure 6.3: GRU cell [9]

Training data were extracted from KDD Cup'99 dataset. The author also selected subset of salient features by using decision tree algorithm and compared the whole and subset features performance in the experiment. Experiments were run with different parameters and structures of an LSTM-RNN, such as the number of memory blocks and the cells per memory block, the learning rate, and the number of passes through the data. Besides that, experiments were also run with a layer of hidden neurons, with peephole connections, with forget gates, and with LSTM shortcut connections. Based on the experimental results, the best performance was achieved by four memory blocks containing two cells, with forget gates and shortcut connections, 0.1 of learning rate and up to 1,000 epochs. The overall accuracy is 93.82%. The author also mentioned in the conclusion that LSTM-RNN is suitable for classifying attacks with a big number of records and poor for a limited number of attack instances.

Bontemps *et al.* [108] leveraged LSTM-RNN in IDS for two objectives: a time series anomaly detector and collective anomaly detector by proposing a circular array. Collective anomaly itself is a collection of related anomalous data instances concerning the whole dataset [108]. The authors used KDD Cup'99 dataset for their experiment and explained the pre-processing steps needed to build a time series dataset from KDD Cup'99 dataset. Putchala [9] implemented a simplified form LSTM, called Gated Recurrent Unit (GRU) in IoT environments. GRU is suitable for IoT due to its simplicity which caused by reducing some gates in the network. GRU merges both forget and input gate to an update gate and combines the hidden and cell state to become a simple structure as shown in Fig. 6.3.

The author then adopted a multi-layer GRU, which is GRU cells used in each hidden layer of RNN and feature selection also done by using random forest algorithm. Experiments were conducted using KDD Cup'99 dataset and achieved 98.91% and 0.76% of accuracy and false alarm rate, respectively.

Bediako [109] proposed a Distributed Denial of Service (DDoS) detector using LSTM-RNN. The author experimented LSTM-RNN using both CPU and GPU. NSL KDD dataset was used for experiments. The notable detection accuracy is 99.968%.

Dimokranitou *et al.* [10] proposed an abnormal events detector in images using an adversarial networks. Although the detector, not an IDS, it has the same objective to detect anomalies. The authors implemented an adversarial autoencoder which combines autoencoders and GAN as shown in Fig. 6.4.

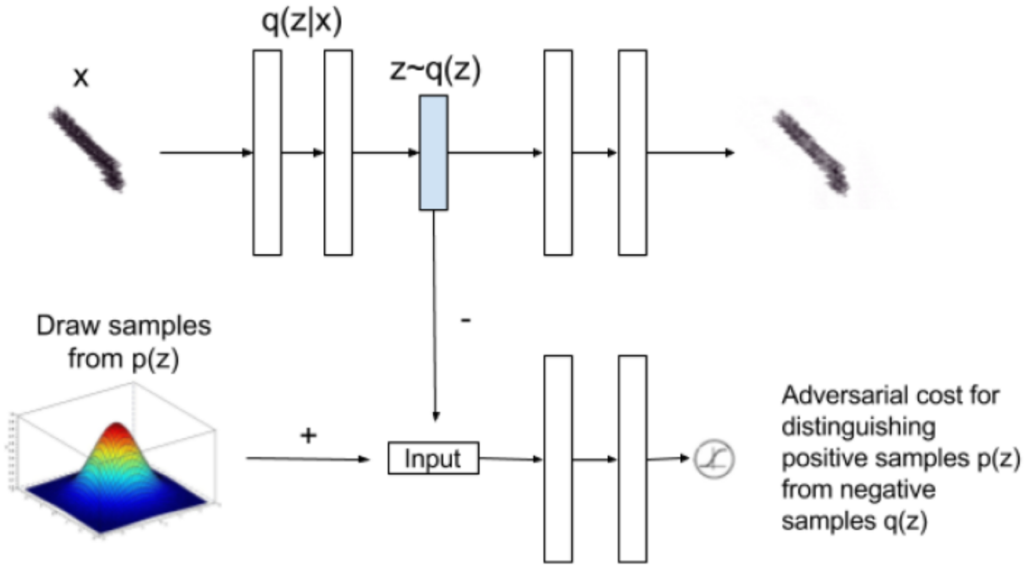


Figure 6.4: Architecture of adversarial autoencoders [10]

Table 6.1: Model Comparisons on KDD Cup'99 Dataset

Model	*Feature Extractor*	*Classifier*	*Accuracy* (%)
DNN [101]	FF-NN	Softmax	99.994
LSTM-RNN-K [48]	LSTM-RNN	Softmax	96.930
LSTM-RNN-L [105]	LSTM-RNN	Softmax	98.110
LSTM-RNN-S [46]	LSTM-RNN	LSTM-RNN	93.820
GRU [9]	GRU	GRU	98.920

The network attempts to match the aggregated posterior of the hidden code vector of AE, with an arbitrary prior distribution. The reconstruction error of learned AE is low for normal events and high for irregular events.

We compare and summarize all previous work mentioned earlier in this Chapter. We compare all work based on the dataset used for experiments, which are KDD Cup'99 and NSL KDD datasets as summarized in Table 6.1 and Table 6.2, respectively.

Performances of IDSs on KDD Cup'99 are promising, as expected, more than 90% of accuracy. Four IDSs in Table 6.1 are using LSTM-RNN approach which means that a time series analysis is suitable for distinguishing benign and anomalies in network traffic. Even more, GRU [9] demonstrated that a lightweight deep learning model is possible to be implemented in IoT environments which is crucial these days.

There is still a space for improvement when we are using NSL KDD dataset as shown in Table 6.2. The most accurate model is RNN [106] with 81.29% of accuracy. Again, this fact infers that a time series analysis may improve IDS performance. Although IDS using CNN achieved not the best performance, it is noticed that by applying a proper text-to-image conversion, we may benefit full potential of CNN as already shown in image recognition researches.

Owing to the scale and complexity of recent data, building a machine-learning-based IDS has become a daunting task. As we aim to detect impersonation attacks in large-scale Wi-Fi networks, a large AWID

Table 6.2: Model Comparisons on NSL KDD Dataset

Model	*Feature Extractor*	*Classifier*	*Accuracy* (%)
STL [7]	AE	Softmax	79.10
DNN-SDN [8]	NN	NN	75.75
RNN [106]	RNN	RNN	81.29
CNN [107]	CNN	CNN	79.14

dataset was chosen for this study. Kolias *et al.* [2] created a comprehensive 802.11 networks dataset that is publicly available and evaluated various machine-learning algorithms to validate their dataset in a heuristic manner. The performance of the IDS was unsatisfactory, indicating that conventional machine-learning methods are incapable of detecting attacks on large-scale Wi-Fi networks. Moreover, among all the classification results obtained, an impersonation attack detection was the most unsatisfactory. One of the primary goals of our study thus is to improve the detection of impersonation attacks by leveraging the advancement of modern machine-learning techniques. Recently, Usha and Kavitha [81] proposed a combination of modified normalized gain and PSO to select optimal features and successfully improved the attack detection rate tested on the AWID dataset. However, they focus on detecting all attack classes rather than impersonation attacks only, which is the problem raised by Kolias *et al.* in [2]. We leveraged SAE for classification purposes and then improved our impersonation detector using weighted feature learning from shallow machine learners [83][110]. Thus, this study extends our previous work [110] to a novel IDS, which combines deep learning abstraction and a weighted feature selection technique.

Chapter 7. Concluding Remark

7.1 Conclusion

We investigated various algorithms especially bio-inspired algorithms to bring significance in the field of IDS research. We believe that by adopting what nature does, we can improve current methods. We started with observing ant behavior and taking Ant Clustering Algorithm as our clustering algorithm as shown in Table 2.3. However, we need other methods for improving the performance of our IDSs. We believe that ACA is still limited to distinguish between benign and attack instances. Therefore, we shifted to more recent bio-inspired algorithms, deep learning, which is the advance of a neural network. Incorporating deep learning methods as a real-time classifier will be a challenging task. Majority of previous work that leveraging deep learning methods in their IDS environment, they perform the feature extraction or reducing feature dimensionalities only. However, we show that deep learning methods can do clustering task as well.

In summary, we can conclude that SAE is very useful for following tasks:

- Feature extraction
- Clustering
- Classification

We presented two novel methods, D-FES and fully unsupervised IDS. Both methods incorporate SAE as a deep abstraction to improve data learnability. The former combines stacked feature extraction and weighted feature selection techniques to detect impersonation and other attacks in Wi-Fi networks. While the latter improves traditional k-means clustering algorithm by proposing a novel fully unsupervised-based intrusion detection system incorporating deep learning technique. SAE is implemented to achieve high-level abstraction of complex and copious amounts of Wi-Fi network data. The model-free properties in SAE and its learnability on complex and large-scale data take into account the open nature of Wi-Fi networks, where an adversary can easily inject false data or modify data forwarded in the network. The proposed methodologies achieved impressive results that the best results on impersonation attacks reported in the literature.

7.2 Further Directions

Further challenges are left for improving IDS in the future. Based on our previous work, we recommend the followings for future directions in IDS researches, but are not limited to:

1. Training load in deep learning methods are usually huge. One should combine deep neural network with an asynchronous multi-threaded search that executes simulations on CPU, and compute policy and value networks in parallel on GPUs. Therefore, how to apply these deep learning models in a constrained-computation device is a challenging task. We should make it lighter to be suitable for IoT environments such as CAN (Controller Area Network) used by Unmanned Vehicle.

2. Incorporating deep learning models as a real-time classifier will be challenging. In the most previous works that leverage deep learning methods in their IDS environment, they perform the feature extraction or reducing feature dimensionalities. Even more, a complete dataset with class labels is not easy to get. However, deep learning models still a suitable method to analyze massive data.

3. Improving unsupervised approach since huge labeled data are difficult to obtain. Therefore an IDS leveraging unsupervised method is desirable.

4. Build an IDS that can detect zero-day attacks with high detection rate and low false alarm rate.

5. A comprehensive measure not only detection but also prevention is needed in the future. Therefore, building an IDS with both detection and prevention capabilities (*e.g.* Intrusion Prevention System (IPS)) is expected.

6. A time series analysis by using LSTM-networks promise a good anomaly detector. However, again, the training workload still high for real-time analysis. Therefore, lightweight models of this network are desirable as shown in [9].

7. CNN achieved outstanding results in many research areas, especially in image recognition fields. However, in IDS researches, not many works benefited by using CNN. We expect that by applying a proper text-to-image conversion, we may help full potential of CNN as already shown in image recognition researches.

Bibliography

[1] M. Alvarez, N. Bradley, P. Cobb, S. Craig, R. Iffert, L. Kessem, J. Kravitz, D. McMilen, and S. Moore, "IBM X-force threat intelligence index 2017," *IBM Corporation*, pp. 1–30, 2017.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 184–208, 2015.

[3] M. E. Aminanto and K. Kim, "Deep learning in intrusion detection system: An overview," *International Research Conference on Engineering and Technology 2016*, 2016.

[4] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on.* IEEE, 2011, pp. 689–690.

[5] C. Olah, "Understanding LSTM networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015, [Online; accessed 20-February-2018].

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS).* ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26.

[8] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on.* IEEE, 2016, pp. 258–263.

[9] M. K. Putchala, "Deep learning approach for intrusion detection system (ids) in the internet of things (iot) network using gated recurrent neural networks (gru)," Ph.D. dissertation, Wright State University, 2017.

[10] A. Dimokranitou, "Adversarial autoencoders for anomalous event detection in images," Ph.D. dissertation, Purdue University, 2017.

[11] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, H. Tullberg, M. A. Uusitalo, B. Timus, and M. Fallgren, "Scenarios for 5G mobile and wireless communications: The vision of the metis project," *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 26–35, May 2014.

[12] C. Kolias, A. Stavrou, J. Voas, I. Bojanova, and R. Kuhn, "Learning internet-of-things security" hands-on"," *IEEE Security Privacy*, vol. 14, no. 1, pp. 37–46, 2016.

[13] C. Kolias, G. Kambourakis, and M. Maragoudakis, "Swarm intelligence in intrusion detection: A survey," *Computers & Security*, vol. 30, no. 8, pp. 625–642, 2011.

[14] A. G. Fragkiadakis, V. A. Siris, N. E. Petroulakis, and A. P. Traganitis, "Anomaly-based intrusion detection of jamming attacks, local versus collaborative detection," *Wireless Communications and Mobile Computing*, vol. 15, no. 2, pp. 276–294, 2015.

[15] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. Symp. Security and Privacy, Berkeley, California*. IEEE, 2010, pp. 305–316.

[16] G. Anthes, "Deep learning comes of age," *Communications of the ACM*, vol. 56, no. 6, pp. 13–15, 2013.

[17] A. H. Farooqi and F. A. Khan, "Intrusion detection systems for wireless sensor networks: A survey," in *Proc. Future Generation Information Technology Conference, Jeju Island, Korea*. Springer, 2009, pp. 234–241.

[18] H. Motoda and H. Liu, "Feature selection, extraction and construction," *Communication of IICM (Institute of Information and Computing Machinery), Taiwan*, vol. 5, pp. 67–72, 2002.

[19] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, no. 2007, 2007.

[20] R. Mitchell and I. R. Chen, "Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 1, pp. 16–30, Jan 2015.

[21] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 266–282, 2014.

[22] C.-H. Tsang and S. Kwong, "Ant colony clustering and feature extraction for anomaly intrusion detection," *Swarm Intelligence in Data Mining*, pp. 101–123, 2006.

[23] H. Bostani and M. Sheikhan, "Modification of supervised OPF-based intrusion detection systems using unsupervised learning and social network concept," *Pattern Recognition*, vol. 62, pp. 56–72, 2017.

[24] M. Sabhnani and G. Serpen, "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context." in *Proc. Int. Conf. Machine Learning; Models, Technologies and Applications (MLMTA), Lax Vegas, USA*, 2003, pp. 209–215.

[25] M. E. Aminanto, H. Kim, K. M. Kim, and K. Kim, "Another fuzzy anomaly detection system based on ant clustering algorithm," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 100, no. 1, pp. 176–183, 2017.

[26] K. Huseynov, K. Kim, and P. Yoo, "Semi-supervised botnet detection using ant colony clustering," in *Symp. Cryptography and Information Security (SCIS), Kagoshima, Japan*, 2014.

[27] K. M. Kim, H. Kim, and K. Kim, "Design of an intrusion detection system for unknown-attacks based on bio-inspired algorithms," in *Computer Security Symposium (CSS), Nagasaki, Japan*, 2015.

[28] K. M. Kim, J. Hong, K. Kim, and P. Yoo, "Evaluation of aca-based intrusion detection systems for unknown-attacks," *The 33th Symposium on Cryptography and Information Security (SCIS)*, 2016.

[29] A. Karami and M. Guerrero-Zapata, "A fuzzy anomaly detection system based on hybrid pso-kmeans algorithm in content-centric networks," *Neurocomputing*, vol. 149, pp. 1253–1269, 2015.

[30] K. Huseynov, P. D. Yoo, and K. Kim, "Scalable p2p botnet detection with threshold setting in hadoop framework," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 25, no. 4, pp. 807–816, 2015.

[31] D. S. Lee and K. Kim, "Improving detection capability of flow-based ids in sdn," *KAIST, Department of Computer Science, Thesis Book*, 2015.

[32] A. L. Vizine, L. N. de Castro, and E. Hrusch, "Towards improving clustering ants: an adaptive ant clustering algorithm," *Journal of Informatica*, vol. 29, no. 2, pp. 143–154, 2005.

[33] H. Izakian and W. Pedrycz, "Agreement-based fuzzy c-means for clustering data with blocks of features," *Neurocomputing*, vol. 127, pp. 266–280, 2014.

[34] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, pp. 53–58, 2009.

[35] H. M. Shirazi, "An intelligent intrusion detection system using genetic algorithms and features selection," *Majlesi Journal of Electrical Engineering*, vol. 4, no. 1, 2010.

[36] K. Kim, H. Kim, and K. Kim, "Design of an intrusion detection system for unknown-attacks based on bio-inspired algorithms," *Proceeding of Computer Security Symposium 2015 (CSS 2015)*, 2015.

[37] D. Pelleg and A. W. Moore, "Accelerating exact k-means algorithms with geometric reasoning," *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 277–281, 1999.

[38] J. P. Verma, "Cluster analysis: for segmenting the population," *Data Analysis in Management with SPSS Software*, pp. 317–354, 2012.

[39] F. Hosseinpour, V. P. Amoli, F. Frahnakian, J. Plosila, and T. Hämäläinen, "Artificial immune system based intrusion detection: Innate immunity using an unsupervised learning approach," *International Journal of Digital Content Technology and its Applications*, vol. 8, no. 5, pp. 1–12, 2014.

[40] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[41] Z. Wang, "The applications of deep learning on traffic identification," in *Conf. BlackHat, Las Vegas, USA*. UBM, 2015.

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

[43] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," *Artificial Intelligence and Statistics*, pp. 448–455, 2009.

[44] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[45] M. Salama, H. Eid, R. Ramadan, A. Darwish, and A. Hassanien, "Hybrid intelligent intrusion detection scheme," *Soft computing in industrial applications*, pp. 293–303, 2011.

[46] R. C. Staudemeyer, "Applying long short-term memory recurrent neural networks to intrusion detection," *South African Computer Journal*, vol. 56, no. 1, pp. 136–154, 2015.

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[48] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Platform Technology and Service (PlatCon), 2016 International Conference on*. IEEE, 2016, pp. 1–5.

[49] M. A. Nielsen, "Neural networks and deep learning," 2015.

[50] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[51] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

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

[53] F. Palmieri, U. Fiore, and A. Castiglione, "A distributed approach to network anomaly detection based on independent component analysis," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1113–1129, 2014.

[54] Q. Xu, C. Zhang, L. Zhang, and Y. Song, "The learning effect of different hidden layers stacked autoencoder," in *Proc. Int. Con. Intelligent Human-Machine Systems and Cybernetics (IHMSC), Zhejiang, China*, vol. 02. IEEE, Aug 2016, pp. 148–151.

[55] H. Shafri and F. Ramle, "A comparison of support vector machine and decision tree classifications using satellite data of langkawi island," *Information Technology Journal*, vol. 8, no. 1, pp. 64–70, 2009.

[56] R. Rojas, "The backpropagation algorithm," in *Neural networks*. Berlin, Springer, 1996, pp. 149–182.

[57] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[58] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," *Applications of Data Mining in Computer Security*, vol. 6, pp. 77–101, 2002.

[59] N. Y. Almusallam, Z. Tari, P. Bertok, and A. Y. Zomaya, "Dimensionality reduction for intrusion detection systems in multi-data streams—a review and proposal of unsupervised feature selection scheme," *Emergent Computation*, vol. 24, pp. 467–487, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-46376-6_22

[60] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, pp. 153–160, Sep 2007.

[61] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards AI," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[62] L. Guerra, L. M. McGarry, V. Robles, C. Bielza, P. Larrañaga, and R. Yuste, "Comparison between supervised and unsupervised classifications of neuronal cell types: a case study," *Developmental neurobiology*, vol. 71, no. 1, pp. 71–82, 2011.

[63] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[64] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[65] X. Zeng, Y.-W. Chen, C. Tao, and D. van Alphen, "Feature selection using recursive feature elimination for handwritten digit recognition," in *Proc. Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), Kyoto, Japan.* IEEE, 2009, pp. 1205–1208.

[66] C. A. Ratanamahatana and D. Gunopulos, "Scaling up the naive Bayesian classifier: Using decision trees for feature selection," in *Workshop on Data Cleaning and Preprocessing (DCAP) at IEEE Int. Conf. Data Mining (ICDM), Maebashi, Japan.* IEEE, Dec 2002.

[67] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine learning paradigms for next-generation wireless networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, 2017.

[68] A. Özgür and H. Erdem, "A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015," *PeerJ PrePrints*, vol. 4, p. e1954v1, 2016.

[69] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[70] D. T. Larose, *Discovering knowledge in data: an introduction to data mining.* John Wiley & Sons, 2014.

[71] W. Wang, X. Zhang, S. Gombault, and S. J. Knapskog, "Attribute normalization in network intrusion detection," in *Proc. Int. Symp. Pervasive Systems, Algorithms, and Networks (ISPAN), Kaohsiung, Taiwan.* IEEE, Dec. 2009, pp. 448–453.

[72] Q. Wei and R. L. Dunbrack Jr, "The role of balanced training and testing data sets for binary classifiers in bioinformatics," *Public Library of Science (PloS) one*, vol. 8, no. 7, pp. 1–12, 2013.

[73] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, "Data randomization and cluster-based partitioning for botnet intrusion detection," vol. 46, no. 8, pp. 1796–1806, 2015.

[74] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation," in *Australasian Joint Conference on Artificial Intelligence, Hobart, Australia.* Springer, 2006, pp. 1015–1021.

[75] P. D. Schloss and S. L. Westcott, "Assessing and improving methods used in operational taxonomic unit-based approaches for 16s rRNA gene sequence analysis," *Applied and environmental microbiology*, vol. 77, no. 10, pp. 3219–3226, 2011.

[76] Z. Boger and H. Guterman, "Knowledge extraction from artificial neural network models," in *Proc. Int. Conf. Systems, Man, and Cybernetics, Orlando, USA*, vol. 4. IEEE, 1997, pp. 3030–3035.

[77] G. M. Weiss and F. Provost, *The effect of class distribution on classifier learning: an empirical study.* Dept. of Computer Science, Rutgers University, New Jersey, Tech. Rep. ML-TR-44, 2001.

[78] M. A. Hall and L. A. Smith, "Practical feature subset selection for machine learning," in *Proc. Australasian Computer Science Conference (ACSC), Perth, Australia.* Springer, 1998, pp. 181–191.

[79] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[80] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles.* ACM, 2017, pp. 1–18.

[81] M. Usha and P. Kavitha, "Anomaly based intrusion detection for 802.11 networks with optimal features using SVM classifier," *Wireless Networks*, vol. 22, pp. 1–16, 2016.

[82] M. E. Aminanto and K. Kim, "Detecting impersonation attack in Wi-Fi networks using deep learning approach," *Information Security Applications: 17th International Workshop, WISA 2016*, 2016.

[83] ——, "Detecting impersonation attack in Wi-Fi networks using deep learning approach," in *Proc. Workshop of Information Security Applications (WISA), Jeju Island, Korea.* Springer, 2016, pp. 136–147.

[84] ——, "Improving detection of Wi-Fi impersonation by fully unsupervised deep learning," *Information Security Applications: 18th International Workshop, WISA 2017*, 2017.

[85] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[86] M. E. Aminanto and K. Kim, "Detecting active attacks in Wi-Fi network by semi-supervised deep learning," *Conference on Information Security and Cryptography 2017 Winter*, 2016.

[87] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection for Wi-Fi impersonation detection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 621–636, 2018.

[88] V. Shah and A. Aggarwal, "Enhancing performance of intrusion detection system against kdd99 dataset using evidence theory," *Int. Journal of Cyber-Security and Digital Forensics*, vol. 5(2), pp. 106–114, 2016.

[89] C. Kolias, V. Kolias, and G. Kambourakis, "Termid: a distributed swarm intelligence-based approach for wireless intrusion detection," *International Journal of Information Security*, vol. 16, no. 4, pp. 401–416, 2017.

[90] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets," in *Proc. Privacy, Security and Trust, New Brunswick, Canada.* Citeseer, 2005.

[91] S. Puthran and K. Shah, "Intrusion detection using improved decision tree algorithm with binary and quad split," in *Proc. Security in Computing and Communication.* Springer, 2016, pp. 427–438.

[92] S. Zaman and F. Karray, "Lightweight IDS based on features selection and IDS classification scheme," in *Proc. Computational Science and Engineering (CSE).* IEEE, 2009, pp. 365–370.

[93] P. Louvieris, N. Clewley, and X. Liu, "Effects-based feature identification for network intrusion detection," *Neurocomputing*, vol. 121, pp. 265–273, 2013.

[94] Y. Zhu, J. Liang, J. Chen, and Z. Ming, "An improved NSGA-iii algorithm for feature selection used in intrusion detection," *Knowledge-Based Systems*, vol. 116, pp. 74–85, 2017.

[95] V. Manekar and K. Waghmare, "Intrusion detection system using support vector machine (SVM) and particle swarm optimization (PSO)," *Int. Journal of Advanced Computer Research*, vol. 4, no. 3, pp. 808–812, 2014.

[96] H. Saxena and V. Richariya, "Intrusion detection in KDD99 dataset using SVM-PSO and feature reduction with information gain," *Int. Journal of Computer Applications*, vol. 98, no. 6, 2014.

[97] E. Schaffernicht and H.-M. Gross, "Weighted mutual information for feature selection," in *Proc. Artificial Neural Networks, Espoo, Finland.* Springer, 2011, pp. 181–188.

[98] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model," *Journal of Computational Science*, Mar 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jocs.2017.03.006

[99] M. E. Aminanto, H. Kim, K.-M. Kim, and K. Kim, "Another fuzzy anomaly detection system based on ant clustering algorithm," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 100, no. 1, pp. 176–183, 2017.

[100] H.-C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach, "Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4D patient data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1930–1943, 2013.

[101] S. S. Roy, A. Mallik, R. Gulati, M. S. Obaidat, and P. Krishna, "A deep learning based artificial neural network approach for intrusion detection," in *International Conference on Mathematics and Computing.* Springer, 2017, pp. 44–53.

[102] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system," in *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on.* IEEE, 2016, pp. 1–8.

[103] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of machine learning research*, vol. 10, no. Jan, pp. 1–40, 2009.

[104] Y. Yu, J. Long, and Z. Cai, "Session-based network intrusion detection using a deep learning architecture," in *Modeling Decisions for Artificial Intelligence*. Springer, 2017, pp. 144–155.

[105] Y. LIU, S. LIU, and Y. WANG, "Route intrusion detection based on long short term memory recurrent neural network," *DEStech Transactions on Computer Science and Engineering*, no. cii, 2017.

[106] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

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

[108] L. Bontemps, J. McDermott, N.-A. Le-Khac *et al.*, "Collective anomaly detection based on long short-term memory recurrent neural networks," in *International Conference on Future Data and Security Engineering*. Springer, 2016, pp. 141–152.

[109] P. K. Bediako, "Long short-term memory recurrent neural network for detecting ddos flooding attacks within tensorflow implementation framework." 2017.

[110] M. E. Aminanto, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Weighted feature selection techniques for detecting impersonation attack in Wi-Fi networks," in *Symp. Cryptography and Information Security (SCIS), Naha, Japan*, 2017.

Acknowledgments in Korean

이 논문을 작성하기 위해 많은 분들의 도움을 받았습니다. 먼저 부족한 저를 잘 이끌어주시고 많은 지식뿐만 아니라 인생의 지혜도 알려주시며 항상 저를 믿어주신 김광조 교수님께 감사드립니다. 또한 바쁘신 와중 에도 저의 학위논문심사에 참여하셔서 진심어린 조언을 해주신 김명철 교수님, 현순주 교수님, 최호진 교수님과 신승원 교수님께도 감사의 말씀을 드립니다.

연구실 친구 최락용, 이지은, Harry Chandra Tanuwidjaja, Edwin Opare, 안형철, 김성숙, 최낙준, 한성호, 홍진아, 김학주, 이동수, Khalid Huseynov, 안수현, 김경민, 정제성, 박준정 모두 감사합니다.

I am also very grateful for my parents – Erni Yuniati and Muhamad Amin Mustafa, my beloved wife – Nazlatan Ukhra Kasuba, my parents in law – Faonia Djauhar and Abdul Gani Kasuba and my families for their strong support and endless love.

Curriculum Vitae in Korean

이 름: Muhamad Erza Aminanto

생 년 월 일: 1993년 02월 05일

출 생 지: Jakarta, Indonesia

주 소: H. Awal 33-C, Jakarta, Indonesia

전 자 주 소: aminanto@kaist.ac.kr

학 력

2009. 8. – 2013. 7. Institute of Technology Bandung (ITB)(Bachelor)

2013. 8. – 2014. 7. Institute of Technology Bandung (ITB)(Master)

2014. 8. – 2018. Korea Advanced Institute of Science and Technology (Ph.D)

연 구 업 적

1. Aminanto, M.E. and Kim, K.J., "Security Challenges of Sensor Network Query Processor in Wireless Sensor Network", 2015 Conference on Information Security and Cryptology, Jun. 25-26, 2015, KAIST, Daejeon, Republic of Korea.
2. Aminanto, M.E., Kim, H.J., Kim, K-M., Kim, K.J., "Another Fuzzy Anomaly Detection System Based on Ant Clustering Algorithm", 2016 Symposium on Cryptography and Information Security, The Institute of Electronics, Information and Communication Engineers, Jan. 19 - 22, 2016, Kumamoto, Japan.
3. Aminanto, M.E. and Kim, K.J., "Deep Learning-based Feature Selection for Intrusion Detection System in Transport Layer", 2016 Conference on Information Security and Cryptology, Jun. 23-24, 2016, Pukyong National University, Busan, Republic of Korea.
4. Aminanto, M.E. and Kim, K.J., "Deep Learning in Intrusion Detection System: An Overview", International Research Conference on Engineering and Technology-IRCET 2016, Jun. 28-30, 2016, Bali, Indonesia.
5. Aminanto, M.E., and Kim, K.J., "Higher Detection of Impersonation Attack in 802.11 Networks using Deep Learning", World Conference on Information Security Applications-WISA 2016, LNCS Springer, vol. 10144, pp. 136-147, Aug. 25-27, 2016, Jeju, Republic of Korea. [SCIE]
6. Aminanto, M.E. and Kim, K.J., "Detecting Active Attacks in WiFi Network by Semi-supervised Deep Learning", 2016 Conference on Information Security and Cryptology, December 2016, Yonsei University, Seoul, Republic of Korea.
7. Aminanto, Muhamad Erza, et al. "Another Fuzzy Anomaly Detection System Based on Ant Clustering Algorithm." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 100.1 (2017): 176-183. [SCIE, IF: 0.40]

8. Aminanto, M.E. and Kim, K.J., "Improving Detection of Wi-Fi Impersonation by Fully Unsupervised Deep Learning", World Conference on Information Security Applications-WISA 2017, Springer, Aug., 24-26, 2017, Jeju, Korea.

9. Aminanto, M.E., Tanuwidjaja, H.C., Yoo Paul D., and Kim, K.J., "Wi-Fi Intrusion Detection Using Weighted-Feature Selection for Neural Networks Classifier." IEEE International Workshop on Big Data and Information Security 2017 IWBIS 2017, Sept 23-24, 2017, Jakarta, Indonesia. [Best Student Paper Award]

10. Kim, K.J., and Aminanto, M.E., "Deep Learning in Intrusion Detection Perspective: Overview and Further Challenges." Keynote Speech at IEEE International Workshop on Big Data and Information Security 2017 IWBIS 2017, Sept 23-24, 2017, Jakarta, Indonesia.

11. Aminanto, M.E., Choi, R.Y., Tanuwidjaja, H.C., Yoo, P.D., and Kim, K.J., "Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection." IEEE Transactions on Information Forensics and Security, 13(3), 621-636. 2018. [SCI, IF: 4.332]

12. Kim, K.J., Aminanto, M.E., and Tanuwidjaja, H.C. "Network Intrusion Detection from Deep Learning - Feature Learning Approach." Springer Book Series: SpringerBriefs on Cyber Security Systems and Networks [In Publication]

Flowcharts of D-FES

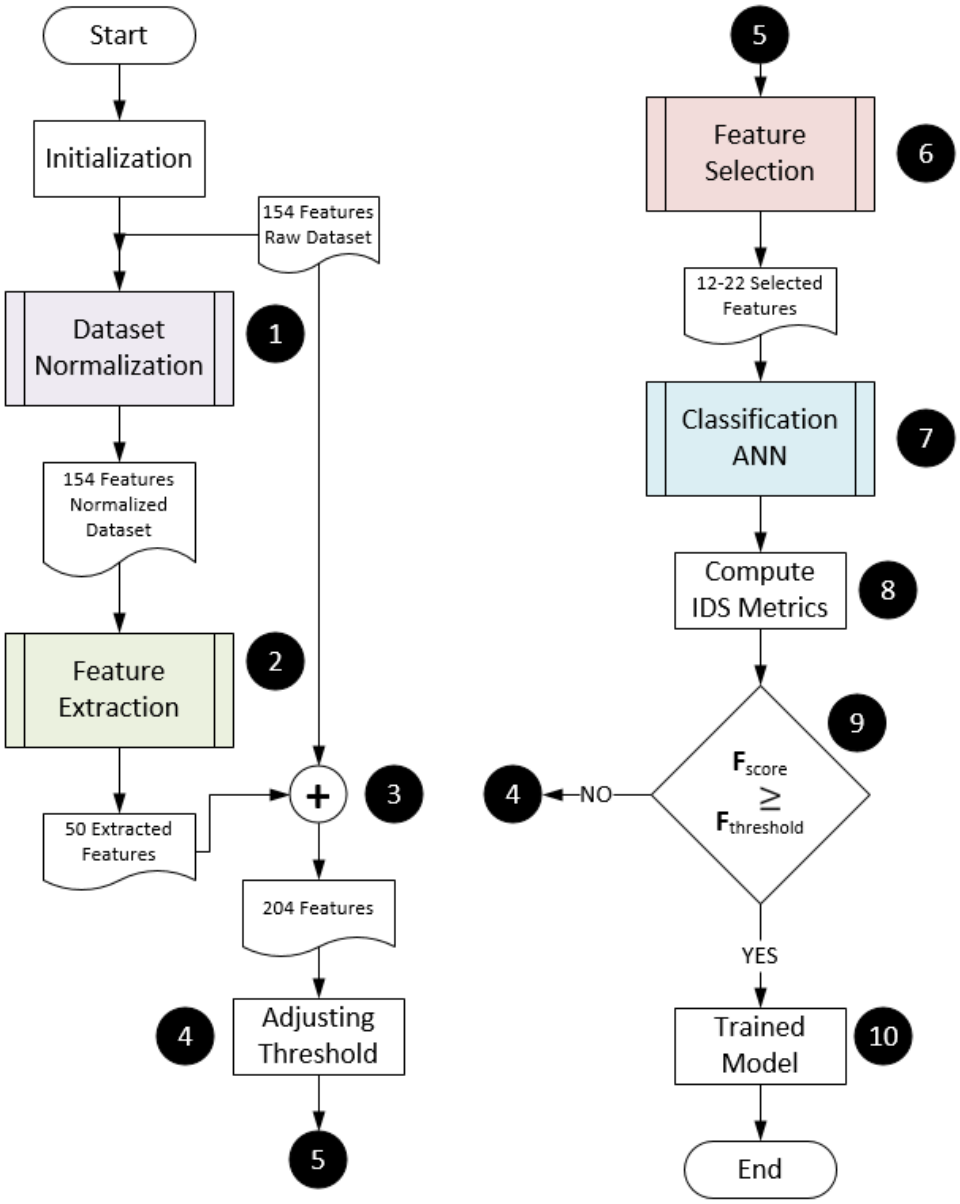


Figure 7.1: Flowchart of D-FES

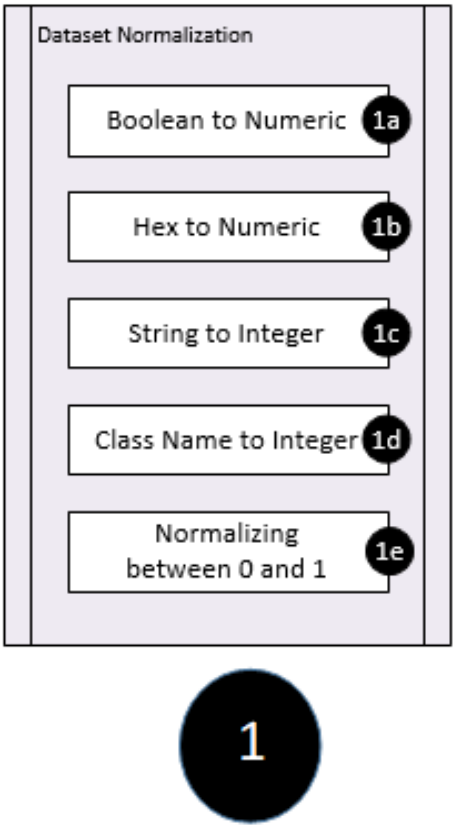


Figure 7.2: Flowchart of D-FES: sub-process 1

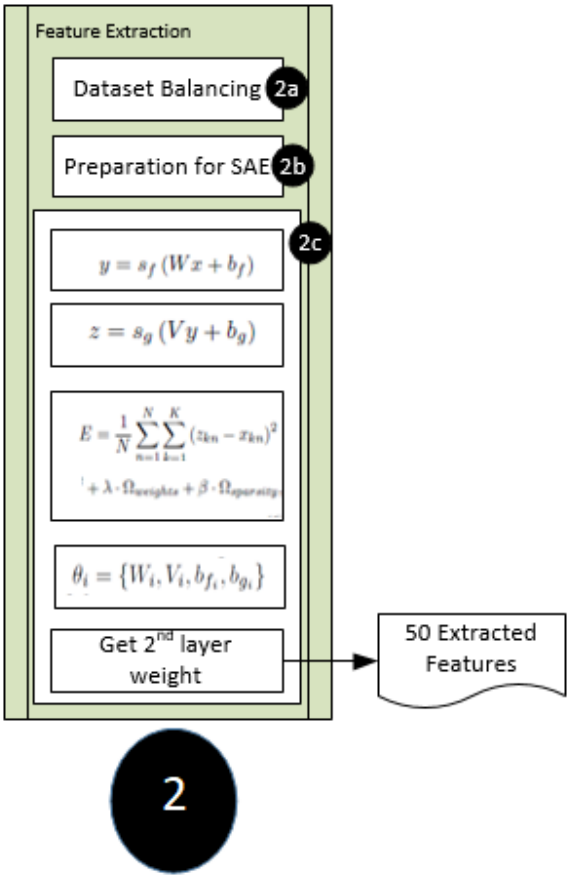


Figure 7.3: Flowchart of D-FES: sub-process 2

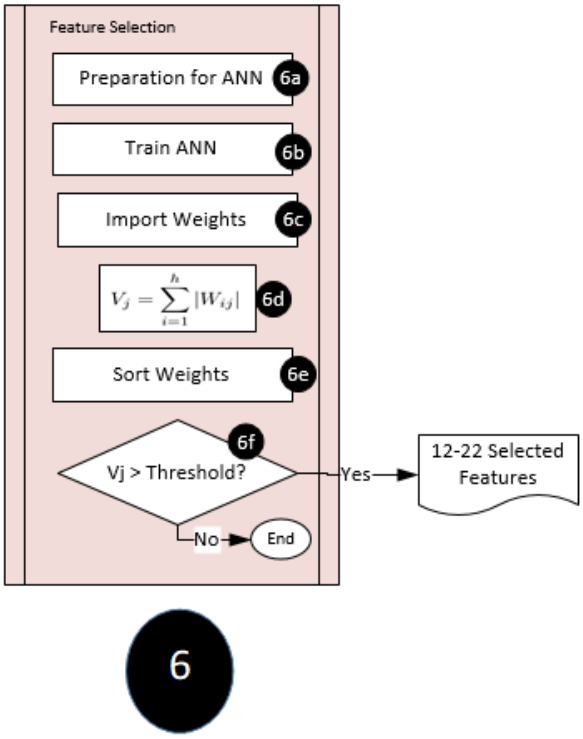


Figure 7.4: Flowchart of D-FES: sub-process 6

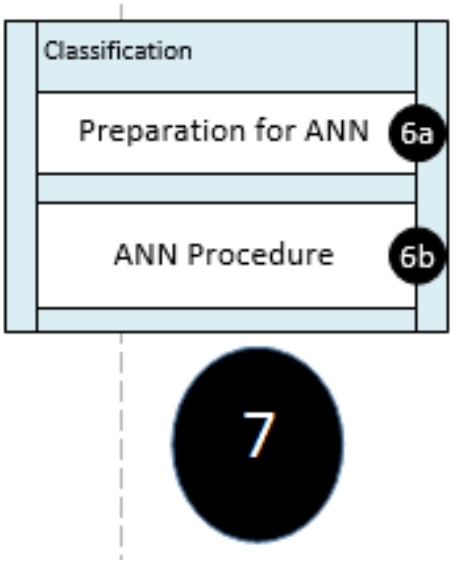


Figure 7.5: Flowchart of D-FES: sub-process 7

Source Code of D-FES

```
%============================================================%
%================ Beginning of Flowchart #1 ================%
%====== Import the dataset and normalization process =======%
%============================================================%
fid = fopen ('1','r'); %import AWID dataset (both training and test dataset named "1")
data=textscan(fid,repmat('%s ',1,155), 'delimiter',','); %read the dataset using delimiter

%=========== Beginning of Flowchart 1a covert String Boolean into Numeric type ==========%
i_att=[10,11,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,35,36,37,39,40,41,42,
43,44,45,46,49,50,51,52,53,54,55,56,57,58,59,60,63,69,70,71,72,73,74,84,85,86,89,90,91,93,
94,95,96,97,98,99,100,101,102,103,123,131,132,135,136,137,147,149,150,151,153];
size_att=size(i_att,2); %get the size of nuber of attributes
newM=zeros(i_end,155); %allocate new matrix
for a=1:size_att
    index_att=i_att(a)
e=data{index_att};%grab target attribute
i_end=size(e,1);%get the last row index
    for i=1:i_end
if e{i,1}=='?'%set ? as 0
newM(i,index_att)=0;
%cek=[i,index_att]
else
tmp9=e{i,1};
tmp10=num2cell(tmp9);%get each char
tmp7=cell2mat(tmp10);
tmp8=str2double(tmp7);%string to numeric
newM(i,index_att)=tmp8;
end %end if
    end %end for
end %end for
%================ End of Flowchart 1a covert String Boolean into Numeric type ==========%

%================ Beginning of Flowchart 1b covert String Hex into Numeric type =========%
i_att=[1,2,3,4,5,6,7,8,9,12,13,14,34,38,47,48,61,62,64,65,66,67,68,75,81,82,83,87,92,104,
106,107,108,109,110,111,112,113,114,115,116,117,120,121,122,124,125,126,127,128,129,130,
133,134,138,139,140,141,142,145,146,148,152,154];
size_att=size(i_att,2);
e=data{1};%just for get #rows
i_end=size(e,1);%just for get #rows
%newM=zeros(i_end,155);%allocate new matrix
```

```
for a=1:size_att
    index_att=i_att(a)
e=data{index_att};%grab target attribute
i_end=size(e,1);%get the last row index
    for i=1:i_end
if e{i,1}=='?'%set ? as 0
newM(i,index_att)=0;
else
tmp9=e{i,1};
tmp10=num2cell(tmp9);%get each char
if size(tmp10,2)>1
if strcmp(tmp10(2),'x')%case hex
tmp3=tmp10(:,3:end);
tmp4=cell2mat(tmp3);
dec_val=hex2dec(tmp4);
newM(i,index_att)=dec_val;
else%case more than one digit int/float
                    tmp7=cell2mat(tmp10);
                    tmp8=str2double(tmp7);%string to numeric
                    newM(i,index_att)=tmp8;
end %end if
else %case one digit integer
tmp7=cell2mat(tmp10);
tmp8=str2double(tmp7);%string to numeric
newM(i,index_att)=tmp8;
end %end if
end %end if
    end %end for
end %end for
%================ End of Flowchart 1b covert String Hex into Numeric type ==========%

%================ Beginning of Flowchart 1c covert String into Integer type ========%
i_att=[76,77,78,79,80,88,105,118,119,143,144];%
size_att=size(i_att,2);
%newM=zeros(i_end,155);%allocate new matrix
for a=1:size_att
    uniqueM={'',0};
    index_att=i_att(a)
c=data{index_att};%grab target attribute
i_end=size(c,1);%get the last row index
incr=1;%initial increment var
for i=1:i_end%for loop all data instances
flag='1';
if c{i,1}=='?'%set ? as 0
```

```
newM(i,index_att)=0;%
%flag='0';
else
%if i>1
size_uniqueM=size(uniqueM,1);
for j=1:size_uniqueM %for loop all values in unique matrix
if strcmp(c(i,1),uniqueM(j,1)) %check whether same values on unique matrix
newM(i,index_att)=uniqueM{j,2};%if same, assign same integer value
flag='0';
break; %get out of the loop immediately
end %end if
end %end for
if flag=='1'
newM(i,index_att)=incr; %if new, assign integer value with incr var
newR=[c(i,1),incr];%store unique string with associated assigned value
uniqueM=[uniqueM;newR];%append new row in unique Matrix
incr=incr+1; %incr++
end %end if
end %end if
end %end for
end %end for
%================ End of Flowchart 1c covert String into Integer type ===============%

%===== Beginning of Flowchart 1d covert String (class name) into Integer type =======%
target=data{155};%grab target class attribute
i_end=size(target,1);%get the last row index
%newM1=zeros(i_end,1);%allocate new matrix
for i=1:i_end
    if strcmp(target{i,1},'normal')
        newM(i,155)=1;%set normal as 1
else if strcmp(target{i,1},'impersonation')
newM(i,155)=2;%set impersonation as 2
else if strcmp(target{i,1},'flooding')
newM(i,155)=3;%set flooding as 3
else if strcmp(target{i,1},'injection')
newM(i,155)=4;%set injection as 4
else  newM5(i,155)=5;%set undefined as 5, supposed to be not exist
end %end if
end %end if
end %end if
end %end if
end %end for
%======== End of Flowchart 1d covert String (class name) into Integer type ==========%
```

```
%=============== Beginning of Flowchart 1e Normalization Procedure ==================%
i_end=size(newM,1);%get the last row index
normM=newM(:,1:154);%allocate new matrix
i_att=[1,2,3,4,5,6,7,8,9,12,13,14,34,38,47,48,61,62,64,65,66,67,68,75,81,82,83,87,92,104,
106,107,108,109,110,111,112,113,114,115,116,117,120,121,122,124,125,126,127,128,129,130,
133,134,138,139,140,141,142,145,146,148,152,154,76,77,78,79,80,88,105,118,119,143,144];%
size_att=size(i_att,2);
%newM=zeros(i_end,155);%allocate new matrix
for j=1:size_att
    index_att=i_att(j)
    maxV=max(newM(:,index_att));%max value
minV=min(newM(:,index_att));%min value
tmp=maxV-minV;%store max-min value
if tmp==0
normM(:,index_att)=0;%assign zero values for same coulmn value
else
for i=1:i_end
normM(i,index_att)=(newM(i,index_att)-minV)/tmp;%normalized value
end %end for
end %end if
end %end for
finalM=[normM, newM(:,155)]; % concat the normalized data with corresponding target class
csvwrite('norm_AWID_train.txt',finalM);%change to norm_AWID_test.txt for test dataset
%%%% norm_AWID_train.txt is the 154 Features normalized dataset %%%%%%%%%%%%%%%%%%%%
%================== End of Flowchart 1e Normalization Procedure ==================%

%============================================================%
%=================== End of Flowchart #1 ====================%
%====== Import the dataset and normalization process =======%
%============================================================%

%============================================================%
%=============== Beginning of Flowchart #2 ==================%
%=============== Feature Extraction process ================%
%============================================================%

%================== Beginning of Flowchart 2a Dataset Balancing ==================%
raw_train=importdata('norm_AWID_train.txt');%import raw train data
raw_test=importdata('norm_AWID_test.txt');%import raw test data
% pick two class only : normal and impersonation
tmpt1=raw_train(:,155)==1;%normal
tmpt3=raw_train(:,155)==2;%Impersonation Attack
tmpt2=raw_test(:,155)==1;%normal
tmpt4=raw_test(:,155)==2;%Impersonation Attack
```

```
train_IA=raw_train(tmpt3,:); %get the impersonation data only
test_IA=raw_test(tmpt4,:);%get the impersonation data only
train_N=raw_train(tmpt1,:);%get the normal data only
test_N=raw_test(tmpt2,:);%get the normal data only
rng(1);%force the same seed value anytime
n_train = size(train_N,1);%get #rows
c_train = cvpartition(n_train,'Holdout',1/10);   % hold out 1/3 of the dataset
train_N_reduced=train_N(test(c_train),:); %training data
rng(1);
n_test = size(test_N,1);%get #rows
c_test = cvpartition(n_test,'Holdout',1/10);   % hold out 1/3 of the dataset
test_N_reduced=test_N(test(c_test),:); %test data
d_IA_train=cat(1,train_N_reduced,train_IA);%train dataset for IA
d_IA_test=cat(1,test_N_reduced,test_IA);%test for IA
csvwrite('data_train_IA_reduced.txt',d_IA_train);%write train dataset for IA
csvwrite('data_test_IA_reduced.txt',d_IA_test);%write test dataset for IA
%=================== End of Flowchart 2a Dataset Balancing ====================%

%================ Beginning of Flowchart 2b Preparation for SAE ===============%
raw_train=importdata('data_train_IA_reduced.txt');%import balanced data for training
raw_test=importdata('data_test_IA_reduced.txt'); %import balanced data for test
x_train_all=raw_train(:,1:154); %pick attributes only without class
x_train_all=x_train_all'; %transpose
y_train_all_int=raw_train(:,155); %pick target class only
y_train_all_int=y_train_all_int';
y_train_all=dummyvar(y_train_all_int); %create as one hot vector
y_train_all=y_train_all';
x_test_all=raw_test(:,1:154); %pick attributes only without class
x_test_all=x_test_all';
y_test_all_int=raw_test(:,155); %pick target class only
y_test_all_int=y_test_all_int';
y_test_all=dummyvar(y_test_all_int); %create as one hot vector
y_test_all=y_test_all';
%=================== End of Flowchart 2b Preparation for SAE ==================%

%=================== Beginning of Flowchart 2c SAE procedure ==================%
rng('default');
hiddenSize1 = 100; %should be smaller than the input size = 155
%First AutoEncoder
autoenc_saya = trainAutoencoder(x_train_all,hiddenSize1, ...
    'MaxEpochs',200, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
```

```
    'ScaleData', false);%
%view(autoenc1); %view ae diagram
%plotWeights(autoenc_saya);%only for image input
feat1_saya = encode(autoenc_saya,x_train_all);%100 features
hiddenSize2 = 50;
%Second AutoEncoder
autoenc2_saya = trainAutoencoder(feat1_saya,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);
feat2_saya = encode(autoenc2_saya,feat1_saya);%%%% results 50 extracted Features
softnet_saya = trainSoftmaxLayer(feat2_saya,y_train_all,'MaxEpochs',200);%train supervised
%view(autoenc1)
%view(autoenc2)
%view(softnet)
deepnet_saya = stack(autoenc_saya,autoenc2_saya,softnet_saya);%stacking AutoEncoder
%view(deepnet)
y = deepnet_saya(x_test_all); %training with input
plotconfusion(y_test_all,y); %cross check the results with corresponding target class
deepnet_fine_tune = train(deepnet_saya,x_train_all,y_train_all); %fine tuning the training
y_ft = deepnet_fine_tune(x_test_all);
plotconfusion(y_test_all,y_ft); %cross check the results with corresponding target class
%====================== End of Flowchart 2c SAE procedure ====================%

%============================================================%
%================== End of Flowchart #2 ====================%
%=============== Feature Extraction process ================%
%============================================================%

%============================================================%
%============== Beginning of Flowchart #6 ==================%
%=============== Feature Selection process =================%
%============================================================%

%====================== Beginning of Flowchart 6a Preparation for ANN ============%
d_IA_train=importdata('data_train_IA_reduced.txt');%import training data for IA
d_IA_test=importdata('data_test_IA_reduced.txt');%import test data for IA
%%%%%%data for training%%%%%%%
d_IA_train_WoT=d_IA_train(:,1:154);%get rid target class att
d_IA_train_T=d_IA_train(:,155);%get target class att only
d_IA_train_T_dummy=dummyvar(d_IA_train_T);%get the hot 1 vector of target class
```

```
%============================================================%
%============== Beginning of Flowchart #3 ==================%
%============================================================%
d_IA_train_WoT = cat(1,d_IA_train_WoT, feat2_saya); %concatenation -> 204 Features
%============================================================%
%================= End of Flowchart #3 ======================%
%============================================================%

Xtrain=d_IA_train_WoT;%assign Xtrain
Xtrain=Xtrain';%transpose input
Ytrain=d_IA_train_T_dummy;%assign Ytrain
Ytrain=Ytrain';%transpose input target
%%%%%%%%data for test%%%%%%%%%
d_IA_test_WoT=d_IA_test(:,1:154);%get rid target class att
d_IA_test_T=d_IA_test(:,155);%get target class att only
d_IA_test_T_dummy=dummyvar(d_IA_test_T);%get the hot 1 vector of target class
Xtest=d_IA_test_WoT;%assign Xtest
Xtest=Xtest';%transpose input test
Ytest=d_IA_test_T;%assign target class for testing
Ytest=Ytest';%transpose target class for testing
Ytestd=d_IA_test_T_dummy;%assign Ytest with dummy
Ytestd=Ytestd';%transpose target with dummy
%======================== End of Flowchart 6a Preparation for ANN =================%

%====================== Beginning of Flowchart 6b ANN Procedure ===================%
% >> nnstart
% >> Select Pattern Recognition App
% >> Select Xtrain for input and Ytrain for target data
% >> Set Hidden Neuron as 100
% >> TRAIN
% >> Save the output model
%======================== End of Flowchart 6b ANN Procedure ======================%

%============================ Beginning of Flowchart 6c Import Weight =============%
berat=net.IW{1,1};%import output weight from the hidden layer
indeks=net.inputs{1}.range;%range yg tidak penting bernilai 0, kolom ke 1 kosong
for l=1:154 %check range (min and max) on each input attribute
    range_indeks=indeks(l,2)-indeks(l,1);
    if range_indeks==0
        tmp(l,1)=0;%if range=0, it means this att is not important
    else tmp(l,1)=1;
    end %end if
end %end for
tmp=logical(tmp);
```

```
indeks(:,1)=[1:154]; %fill first column with increment 1-41
%tmp=indeks(:,2)~=0;%get the index that != 0
indeks2=indeks(tmp,:);%select row  != 0
indeks_aja=indeks2(:,1);%get the index only
indeks_aja=indeks_aja';
berat=abs(berat);%absolute value of weight
%============================== End of Flowchart 6c Import Weight ==================%

%======================= Beginning of Flowchart 6d Summation of Weight ============%
sizenya = size(berat);%size of weight
col=sizenya(1,2);%column index
row=sizenya(1,1);%row index
berat_per_input=zeros(1,col);
berat_per_input(1,:)=indeks_aja;
for i=1:col %iteration column
    berat_per_input(2,i)=0;
    for j=1:row %iteration rows
        berat_per_input(2,i)=berat_per_input(2,i)+berat(j,i);
    end %end for
end %end for
%========================== End of Flowchart 6d Summation of Weight ===============%

%======================= Beginning of Flowchart 6e Sorting Weight =================%
berat_per_input=berat_per_input'; %transpose
berat_sort=sortrows(berat_per_input, 2); %sort descending
%========================== End of Flowchart 6e Sorting Weight ====================%

%============================================================%
%============== Beginning of Flowchart #4 ==================%
%============================================================%
%======================= Beginning of Flowchart 6f Compare Weight =================%
tmp2=berat_sort(:,2)>=15;%adjust the trehsold value
%============================================================%
%======================= Beginning of Flowchart 6f Compare Weight ===============%
%================== End of Flowchart #4 ====================%
%============================================================%

selected_features=berat_sort(tmp2,1); %%% 12-22 Selected Features %%%%

%============================================================%
%================== End of Flowchart #6 ====================%
%============== Feature Selection process =================%
%============================================================%
```

```
%============================================================%
%=============== Beginning of Flowchart #7 =================%
%================= Classification process ==================%
%============================================================%

%================== Beginning of Flowchart 7a Preparation for Classification ========%
d_IA_train_trans=d_IA_train';
d_IA_test_trans=d_IA_test';
ukuran_mat=size(selected_features);%get # features
jumlah_fitur=ukuran_mat(1,1);%get # row (# features)
indeks_fitur=[1:154];
for j=1:154%increment at feature index
    for i=1:jumlah_fitur%increment at selected features
        if indeks_fitur(1,j)==selected_features(i,1)%if equal, assign 1
            indeks_fitur(2,j)=1;
            break;%get out of if
        else indeks_fitur(2,j)=0;
        end %end if
    end %end for
end %end for
%recover selected features within all features
indeks_fitur_aja=indeks_fitur(2,:);%get the boolean only
indeks_fitur_aja=indeks_fitur_aja';
indeks_fitur_aja_kelas=indeks_fitur_aja;%add 155th index as class
indeks_fitur_aja_kelas(155,1)=1;%add 155th
indeks_fitur_aja_logical=logical(indeks_fitur_aja_kelas);%covert into logical
%training dataset with selected features
train_data=d_IA_train_trans;
train_data=train_data(indeks_fitur_aja_logical,:);
input_fitur=train_data(1:jumlah_fitur,:);%training without target class
target_train=train_data((jumlah_fitur+1),:);%training target class
target_train_final=dummyvar(target_train);
target_train_final=target_train_final';
%test dataset with selected features
test_data=d_IA_test_trans;
test_data=test_data(indeks_fitur_aja_logical,:);
input_fitur_test=test_data(1:jumlah_fitur,:);%test without target class
target_test=test_data((jumlah_fitur+1),:);%test target class
target_test_final=dummyvar(target_test);
target_test_final=target_test_final';
%final assignment
x_train = input_fitur;    % the input for training
y_train = target_train_final;  % the target for training
x_test = input_fitur_test;  % the input for testing
```

```
y_test = target_test_final;   % the target for testing
%====================== End of Flowchart 7a Preparation for Classification =========%

%========================== Beginning of Flowchart 7b ANN Procedure =============%
% >> nnstart
% >> Select Pattern Recognition App
% >> Select Xtrain for input and Ytrain for target data
% >> Set Hidden Neuron as 50
% >> TRAIN
% >> Get the confusion matrix ==> Flowchart #8 Compute IDS Metrics
% >> Save the output model
%========================= End of Flowchart 7b ANN Procedure ==================%

%=============================================================%
%================= End of Flowchart #7 ====================%
%================ Classification process ==================%
%=============================================================%

%====================== Beginning of Flowchart #9 Comparison ==============%
% Compare F score and F treshold
%========================= End of Flowchart #9 Comparison ================%

%====================== Beginning of Flowchart #10 Model ===============%
% Trained model complete
%========================= End of Flowchart #10 Model ==================%

%============ END of File ====================%
```