AN ABSTRACT OF THE THESIS OF

Jiaqi Bao for the degree of Master of Science in Electrical and Computer Engineering presented on March 17, 2020.

Title: Network Traffic-Driven Hybrid Learning for Classifying Seen and Unseen IoT Device Types

Abstract approved: ____

Bechir Hamdaoui

IoT networks can be viewed as collections of Internet-enabled physical devices and objects, embedded with sensor, actuator, computation, storage and communication components, that are capable of connecting and exchanging data to one another. In recent years, organizations have allowed more and more IoT devices to be connected to their networks, thereby increasing their risk and exposure to security vulnerabilities and threats. Therefore, it is important for such organizations to be able to identify which devices are connected to their network and which ones are legitimate and pose no risk. Leveraging network traffic to identify devices through supervised learning has recently been gaining popularity, where feature information is first extracted by intercepting device traffic and then exploited to provide device classification. The main limitation of prior works is that they can only identify previously seen types of devices, and any newly added device types are treated as abnormal types. In the real world, hundreds of millions of new IoT devices are produced each year, and the lack of sufficient amount of training data makes a system based solely on supervised learning unrealistic. In this paper, we propose a hybrid supervised and unsupervised learning method that enables secondary classification of unseen device types. Our technique combines deep neural networks with clustering to enable both seen and unseen device classification, and employs autoencoder techniques to reduce dimensionality of datasets, thereby providing a good balance between classification accuracy and overhead.

Network Traffic-Driven Hybrid Learning for Classifying Seen and Unseen IoT Device Types

by

Jiaqi Bao

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 17, 2020
Commencement June 2020

Master of Science thesis of Jiaqi Bao presented on March 17, 2020.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

Head of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

Jiaqi Bao, Author

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

Conventional computer security systems can be divided into two types: network security systems and host security systems. Both systems may contain different integrated security modules, such as firewalls, intrusion detection systems (IDS), and antivirus software, which can monitor the system or network and alert when malicious activity occurs. Among them, IDS plays a vital role in information security technology. It is considered to be a necessary security mechanism to deal with network attacks and identify malicious activities in computer network traffic. Although numerous vendors on the market produce embedded security modules, many attackers are still in a dominant position of power because of the unprecedented amount of daily malware types and production such as worms, viruses, Trojans, botnets, ransomware, etc. Besides, many small and medium-sized Internet networks do not have the ability to update virus databases and patches on time or connected devices do not have defense capabilities at all. Such networks are more vulnerable to attackers. It is, therefore, necessary to design accurate automated systems to detect and classify malware.

Internet of Things (IoT) relates to networks of physical devices and items embedded with electronics, sensors, actuators, software, and connectivity, which enables the communication between these devices and their exchange of data. In recent years, organizations have allowed more and more IoT devices to be connected to their networks, and due to the nature of IoT devices, it may not be able to update security patches promptly. This could give malware a chance to create cybersecurity threats to these networks. More importantly, if an infected IoT device is not isolated and removed from the network in a timely manner, it will cause cross-contamination and inject malware into the entire network. Therefore, it is important for the organization to be able to identify which devices are connected to their network and whether these devices are considered legitimate and pose no risk. Leveraging network traffic to identify devices, in general, has been gaining in popularity in previous works. Specifically, the feature information is extracted by intercepting traffic generated by the general operation of the device after connecting to the network. The effect of classification is achieved through the difference

in feature information.
Previous work mainly focused on the use of supervised learning methods to address these issues. However, the main limitation of these methods is that they require labeled data for training and can only identify previously seen types of devices. Any newly added device types will end up being treated as abnormal device types. In the real world, hundreds of millions of new IoT devices are produced each year, and the lack of a large amount of training data makes a system based solely on supervised learning unrealistic. In order to solve this problem, we propose in this paper a hybrid supervised and unsupervised learning method that enables secondary classification to make the data processing more refined. Clustering is an unsupervised way that divides data into groups so that objects belonging to the same group are similar. There are several types of clustering such as density-based clustering, model-based clustering, fuzzy clustering and hierarchical clustering. For density-based clustering, the more data points contained in a circle of the same size, the denser this area is considered, and a smaller number of points is considered to be sparse. The set of consecutive dense points is a group. Far from other clusters, sparse clusters or clusters of points that do not belong to any group are considered abnormal data. However, the performance of clustering-based methods is very dependent on which algorithm is used, and high-dimensional data is a special challenge for data mining algorithms in density-based clustering. To address this issue caused by the curse of dimensionality [?], methods based on reconstruction errors such as autoencoder techniques are widely used. The data are first reduced in dimension by the autoencoder and then clustered in latent low-dimensional data by the OPTICS.
The remainder of the thesis is structured as follows: Chapter 2 summarizes the related work, specifically the papers this proposed framework is based on. Chapter 3 introduces and reviews the background of the different key concepts used in the thesis. Chapter 4 describes the proposed method in detail. Chapter 5 presents the performance evaluation and result analysis. Chapter 6 presents the future works and conclusion.

Chapter 2: Related Work

Yair Meidan et al. [?] proposed an approach for unauthorized IoT device identification that uses the method of checking whether a device is on the whitelist. They focused exclusively on features extracted from TCP/IP traffic. However, with the increasing number of IoT devices, this method faces the problem of update and excessive retrieval time caused by too many whitelisted devices. Our work is complementary to theirs and can be used along with their approach to identify device type effectively.

B. Bezawada et al. [?] proposed IoTSense, a framework of device fingerprinting that uses machine learning for IoT device identification. Similar work can also be found in [?, ?]. These works specified the IoT device granularity level used for fingerprints—device category, type, and instance. A device type, defined as a specific device model within a general device category, is a classified target.

Yair Meidan et al. [?] developed a multi-stage meta classifier to identify not only IoT and non-IoT devices but also specific IoT device classes. To improve the accuracy of the results, each device type uses its optimal classification threshold (cutoff value) and optimal session sequence size instead of the default values.

Sandhya Aneja et al. [?] designed a device fingerprint using inter arrival time (IAT), which is the time interval between two consecutively received data packets. Since IAT is determined only on hardware or software, it is still reliable even when IP or MAC addresses are spoofed. Device fingerprints are then classified by machine learning. This method is based on the hardware of the IoT device classification. It applies to an IoT network where there are many duplicated device types with a few device types, where the same device with different vendors may be considered as two distinct devices.

M. Yousefi-Azar et al. [?] introduced a feature learning method for network-based anomaly intrusion detection and malware classification using AE (autoencoder). Compared with other algorithms, this method uses fewer features, which makes the model more effective for real-time protection. The experiments used 41 raw byte-features using a portable executable based on protocol, payload and time. Compared with other single classifier models, its advantages are high accuracy and simplicity.

Chapter 3: Background

3.1 Internet Network Traffic

3.1.1 OSI Model

OSI, which stands for Open System Interconnection and was developed by the International Standard Organization, can be used as a reference to help understand how applications communicate through networks. The model was introduced to standardize networks in a way that allowed multi-vendor systems. Prior to this, it was only able to have one vendor network because the devices from one vendor couldn't communicate with others. The OSI model has seven layers, as shown in Table **??**. The data sent from the sender passes from the upper layer to the lower layer interconnected with the lower layer of the receiver and then is transmitted to the higher layer at the receiver. The application layer facilitates communications between end-users and the next layer. The presentation layer is where the operating system is located and it provides a variety of coding and conversion functions to translate data into a format that every different system could understand. The function of this layer includes data conversion, data encryption/decryption, and data compression/decompression. The session layer establishes, manages, and terminates the communication session between two computers. The transport layer chops the data and adds a header. The network layer adds header

Table 3.1: OSI Models and Examples of Protocols

Layer	Name	Protocols
7	Application	HTTP,FTP
6	Presentation	SSL,SSH
5	Session	API's, Sockets
4	Transport	TCP,UDP
3	Network	IP, ICMP, IPsec
2	Data Link	Ethernet, Switch, Bridge
1	Physical	Coax, Fiber, Wireless

information about the destination routers and IP addresses. The data link layer adds more header information, such as source and destination MAC address, to create or read an Ethernet frame. Last but not least, the physical layer transforms data information into either electronic impulses or light, depending on what kind of cable is used.

3.1.2 TCP/IP

Since our model only studies devices that communicate over the Internet, the TCP/IP model (Transmission Control Protocol / Internet Protocol) better represents the data shape of this design. TCP stands for Transmission Control Protocol and IP stands for Internet Protocol. Figure **??** shows a comparison between the OSI model and the TCP/IP model. The main difference is that the TCP/IP model has only three layers, each of which covers the corresponding functions under the OSI model.

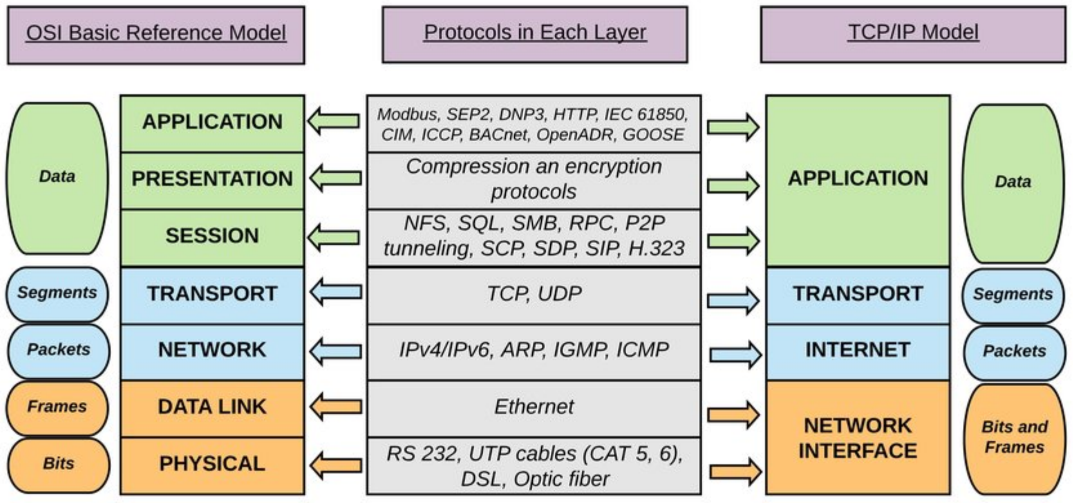

OSI Basic Reference Model
Protocols in Each Layer
TCP/IP Model
Data
APPLICATION
PRESENTATION
SESSION
Segments
TRANSPORT
Packets
NETWORK
Frames
DATA LINK
Bits
PHYSICAL
Modbus, SEP2, DNP3, HTTP, IEC 61850, CIM, ICCP, BACnet, OpenADR, GOOSE
Compression an encryption protocols
NFS, SQL, SMB, RPC, P2P tunneling, SCP, SDP, SIP, H.323
TCP, UDP
IPv4/IPv6, ARP, IGMP, ICMP
Ethernet
RS 232, UTP cables (CAT 5, 6), DSL, Optic fiber
APPLICATION
TRANSPORT
INTERNET
NETWORK INTERFACE
Data
Segments
Packets
Bits and Frames

Figure 3.1: Comparison between OSI model and TCP/IP model [?]

Important functions are extracted from not only the data but also the header of the TCP data segment. Figure **??** shows the connecting process and the structure of TCP headers. As can be seen in the figure, TCP connection starts with a three-way handshake and ends with a four-way handshake.

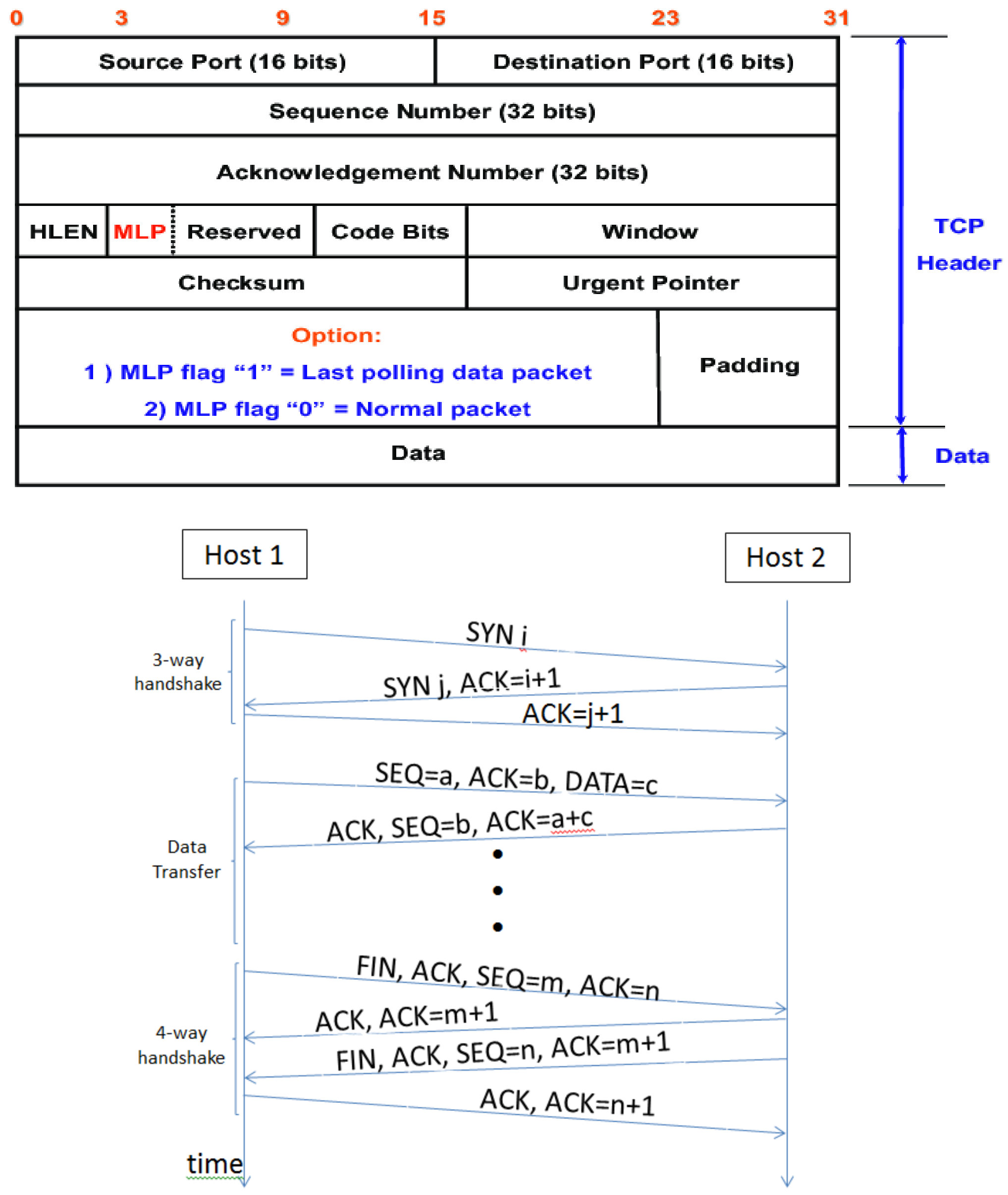


Figure 3.2: TCP header and TCP connection [?]

For the three-way-handshake, the client (Host 1) who initiates the connection first sends a packet to the server (Host 2). The first packet contains a randomly selected initialization sequence number i and a SYN bit to tell it to synchronize the new connection. The server then responds with an acknowledgment with an ACK bit and an acknowledgment number of $i+1$, as well as its own SYN and another random sequence number, j. The connection is established when the client acknowledges with ACK bit and acknowledgment number $j+1$. The four-way handshake is similar to this process. The client decides to complete the connection by sending a FIN bit. The server then responds with an ACK bit and sends another packet with its FIN bit for the server to also close the connection. Eventually, the client sends an acknowledgment and the connection is closed.

3.2 Data Mining Techniques

To discover patterns and knowledge from large amount of collected traffic data, one can use one of 6 common classes of data mining task [?]: anomaly detection, association rule learning, clustering, classification, regression and summarization. Next, we discuss classification and clustering in detail.

3.2.1 Random Forest

Random forest algorithm is an algorithm that uses a large collection of decorrelated decision trees for classification. A decision tree is a structure similar to a flowchart, in which each internal node represents a decision on an attribute, each point is split into two branches, each branch represents the result of the decision, and each leaf node represents a decision result class label (A decision made after calculating all attributes). Generally, there can be many positions for branch split. One of the criterion to measure the quality of a split is Gini Impurity, given as:

$$Gini(t) = 1 - p_{positive}(t)^2 - p_{negative}(t)^2 \tag{3.1}$$

where $p_{positive}(t)$ and $p_{negative}(t)$ are the probabilities of being positive or negative in the test. The smaller the Gini impurity, the better the separation effect. Let S be the matrix of training data feed into the random forest, where n is the number of features, p is the number of data samples, and C is the class label for each data point. S can be represented as

$$S = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} & C_1 \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} & C_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{p,1} & f_{p,2} & \cdots & f_{p,n} & C_p \end{pmatrix} \tag{3.2}$$

From the matrix S, we create a subset of M randomly selected matrices, with each matrix having the same size of the original input matrix S and containing shuffled rows of S (with possibly repeated ones). These subsets are called bootstrapped datasets. Below is an example of such created matrices:

$$S_1 = \begin{pmatrix} f_{2,1} & f_{2,2} & \cdots & f_{2,n} & C_2 \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} & C_2 \\ f_{17,1} & f_{17,2} & \cdots & f_{17,n} & C_{17} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}, \cdots, S_M = \begin{pmatrix} f_{6,1} & f_{6,2} & \cdots & f_{6,n} & C_6 \\ f_{11,1} & f_{11,2} & \cdots & f_{11,n} & C_{11} \\ f_{17,1} & f_{17,2} & \cdots & f_{17,n} & C_{17} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.3}$$

A decision tree is created for each subset, and then the differences between each subset and the original set are used to make predictions to measure the accuracy of the random forest. We can then use this accuracy to fine-tune the parameters.

3.2.2 OPTICS

Ordering points to identify the clustering structure (OPTICS) is unsupervised data clustering algorithm based on data space density. It works by defining a cluster as the maximal set of density-connected points. It was first presented by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander on 1999 [?]. The basic idea comes from the Density-based spatial clustering of applications with noise (DBSCAN) which is their early work. There are three classifications of the points: core, border, and outlier. A core point has at least a minimum number of points within its ϵ-neighborhood radius, including itself. A border point has less than minimum points within its ϵ-neighborhood radius but belongs to the neighborhood of a core point. An outlier point is a point that cannot be reached by any cluster. Different types of points can be visualized in Figure **??**.

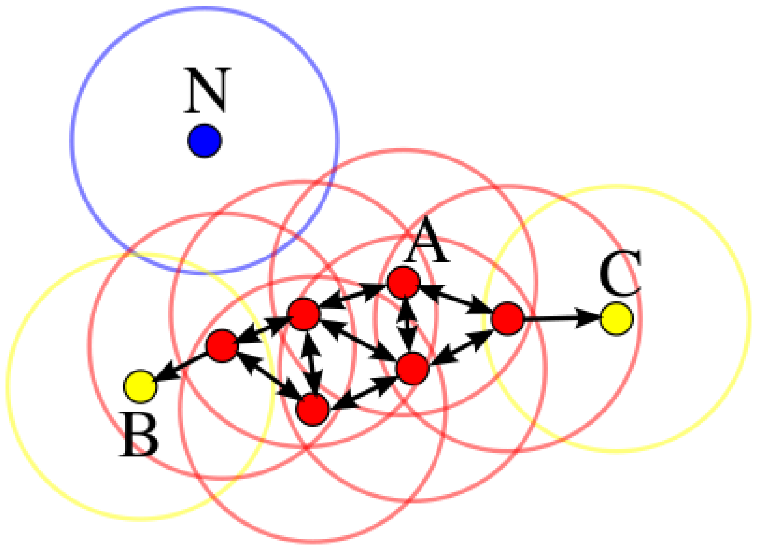


Figure 3.3: Red for core point, yellow for border point, and blue for outlier point, minPts=4

There are two parameters for DBSCAN to define a cluster: ϵ and $minPts$. ϵ is the maximum radius of the neighborhood, and $minPts$ is the minimum number of points in the ϵ-neighborhood to define a cluster. Compared with DBSCAN, OPTICS has the advantage of being less sensitive to parameters because OPTICS only requires one user-specified input parameter, $minPts$. The idea of OPTICS is to create a database in ascending density order based on the distance between data points to represent its density-based clustering structure. It stores such a clustering order using two pieces of information: core distance and reachability distance.
The core distance of a point o, $CD_{\varepsilon,minPts}(o)$, is defined as [**?**]:

$$CD_{\varepsilon,minPts}(o) := \begin{cases} \text{UNDEFINED}, & \text{if } |\{x|d(x,o) \leq \varepsilon_{max}\}| < minPts \\ minPts\text{-}dis(o), & \text{otherwise} \end{cases} \tag{3.4}$$

where $minPts - dist(o)$ is the distance from point o to the nearest neighbor of the given $minPts$. When the point o are sufficiently isolated such that the number of other points within radius ε_{max} is less than $minPts$, the core distance is undefined. Otherwise, the core distance is the $minPts$ distance.
Then the reachability distance from a point p to point o, $RD_{\varepsilon,minPts}(o,p)$ can be defined

as

$$RD_{\varepsilon,minPts}(o,p) := \begin{cases} \text{UNDEFINED}, & \text{if } |N_\varepsilon(p)| < minPts \\ max\{d(o,p), core\text{-}dist(o)\}, & \text{otherwise} \end{cases} \tag{3.5}$$

where $|N_\varepsilon(p)|$ is ε-neighborhood of p. $d(o,p)$ and all distances mentioned above refer to Minkowski distance. Intuitively, the points with the smallest reachable distance (highest density) will be processed first. Therefore, the output of OPTICS will be the data points sorted by processed order along with their reachability distances.

3.3 Autoencoder

Autoencoder (AE) is a symmetrical artificial neural network that trains the output to reconstruct the input x. The network may be viewed as consisting of two parts: an encoder to mapping input x to a given number of features (bottlenecks) and a decoder to mapping the bottlenecks to reconstruction $\hat{x}$. By forcing the reconstructed $\hat{x}$ to be as close as possible to the input, the parameters of the neural network can be learned. The structure of a basic autoencoder is shown in Figure **??**.

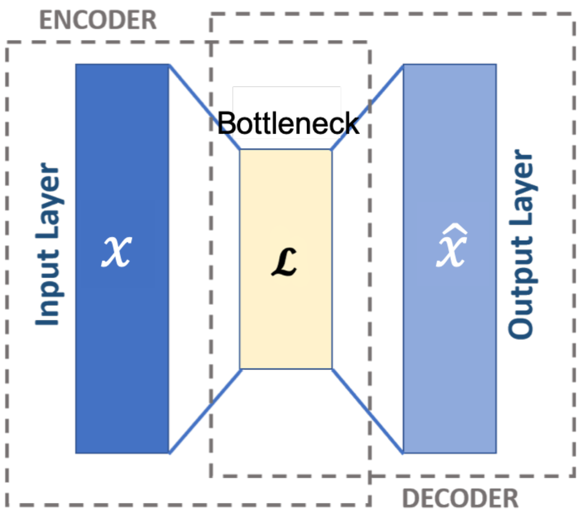


Figure 3.4: The structure of a basic AE

Given a set of p input data vectors, $X = \{x_1, x_2, \ldots, x_p\}$, an input vector x_i feed-forwards to a bottleneck vector $\mathcal{L}_i$ as

$$\mathcal{L}_i = f_\theta(x_i) = \sigma(Wx_i + b) \tag{3.6}$$

where σ is activation function, W is the weight matrix, and b is the bias vector. Weight and bias form the parameter set $\theta = \{W, b\}$. The decoder then feed-forwards the bottlenecks to reconstructed vector $\hat{x_i}$, with same dimension as same the input vector.

$$\hat{x_i} = g_\theta(\mathcal{L}_i) = \sigma(W'\mathcal{L}_i + b') \tag{3.7}$$

Because of AE's symmetrical structure, the weights of the decoder are usually transposed matrices of the encoder weights; i.e., $W' = W^T$. We then back-propagate AE to optimize its parameters by minimizing the loss function

$$J(\theta; x, \hat{x}) = \frac{1}{p}\sum_{i=1}^{p} ||x_i - \hat{x_i}||^2 \tag{3.8}$$

The size of the bottleneck space should have lower dimensionality than the input space. If the hidden layer is larger than the input space, the network is likely to directly copy the input to output without learning. Instead of reducing the number of neurons in the bottleneck, an alternative way is sparse bottleneck space. Sparse AE may include more hidden units than inputs, but only a small number of the hidden units will be active at once [**?**]. Sparsity constraints can be implemented with a regularizer. The loss function after considering sparse space becomes

$$J_{sparse}(\theta; x, \hat{x}) = J(\theta; x, \hat{x}) + \Omega(\mathcal{L}) \tag{3.9}$$

Chapter 4: The Proposed Device Classification Method

4.1 Design Overview

The types of IoT devices that may appear in the network are classified into three categories:

Known: Authorized device types that can be trusted. Here we refer to device types that the system has trained and saved their corresponding device models.

New (Unknown): Unauthorized device but the long-term monitoring has revealed that some observations exhibit very similar behavior. These observations are considered to be of the same device type and may be licensed as known in the future.

Anomaly (Unknown): An observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.

The proposed device identification and classification framework is composed of 6 modules, as shown in Figure **??**. When the device is first connected to the network, the Data Processing module captures the traffic generated by the general operating procedures and extracts the desired features from the observations. The chosen method of the Train module for this research is based on the whitelist method described in [**?**, **?**], where a one-vs-rest binary classifier is created for each authority device type (known). Processed data feeds into the Predict module, stops and labels it when a classifier responds to acceptance. The Prediction module can directly use classifier models from Train to predict the device type and label the feature vector. Then follows a discriminator that determines whether the feature vector is labeled. If yes, the feature vector is sent to the Action module, where the corresponding mitigation strategy is applied. For different phases and categories, the Active module will give different mitigation strategies. If no, the feature vector will be taken to the Clustering module and then continue to feed into the Active module. In the following paragraphs, we will mainly focus on the Clustering module.

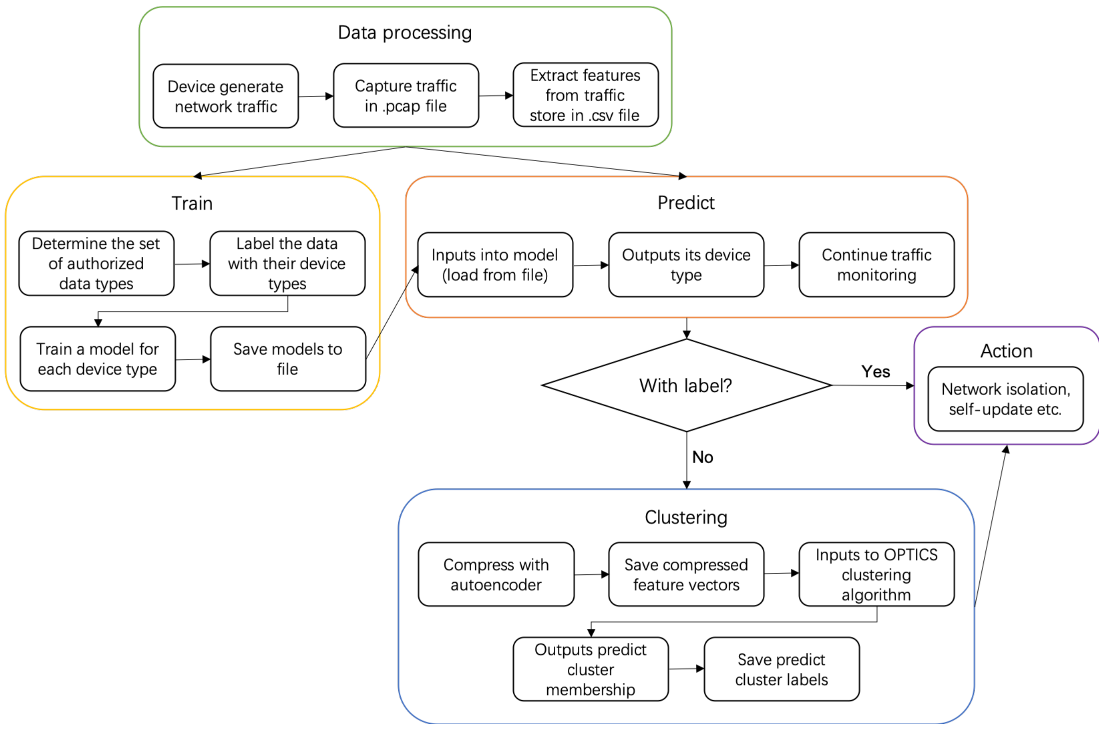

Data processing
Device generate network traffic
Capture traffic in .pcap file
Extract features from traffic store in .csv file
Train
Determine the set of authorized data types
Label the data with their device types
Train a model for each device type
Save models to file
Predict
Inputs into model (load from file)
Outputs its device type
Continue traffic monitoring
With label?
Yes
No
Action
Network isolation, self-update etc.
Clustering
Compress with autoencoder
Save compressed feature vectors
Inputs to OPTICS clustering algorithm
Outputs predict cluster membership
Save predict cluster labels

Figure 4.1: The whole system

4.2 Known Device Classification

The selection of the random forest model and its parameters are obtained by fine-tuning to reach the maximum area under curve (AUC). The reason why the AUC score is used as a criterion is that under the binary classification problem, the data is usually unbalanced. This means that most feature vectors usually belong to 'not device' and few feature vectors belong to 'is device'. Therefore, an accuracy indicator reflecting the distribution of the underlying classes may not be sufficient. The receiver operating characteristic (ROC) curve represents the ratio between the true positive rate (TPR) and the false positive rate (FPR). The closer the area under the curve is to 1, the better the effect. The metric also indicates how good the data imbalance results in fact, and it's not just about accuracy. Besides, when determining the best model, some models may show higher AUC scores in some device types. Here we use the average AUC score to measure the model performance. Algorithm 1 is used to create a model for known device classification.

We enter a list with m device types and training data to get m classifier models.

Algorithm 1 Train Model For Known Device

```
1:  Inputs:
        $X = \{x_1, x_2, \cdots, x_p\}$                ▷ Input data
        $C = \{c_1, c_2, \cdots, c_q\}$                ▷ Device type list
2:  Initialize:
        $X' = \{x'_1, x'_2, \cdots, x'_p\}$             ▷ Feature data
        $Y = \{y_1, y_2, \cdots, y_p\}$                ▷ Label data
        $x_i = x'_i + y_i$
3:  $X = shuffle(X)$
4:  $X = MinMaxScaler().fit(X)$
5:  for $c \in C$ do
6:      $X = X' + Y$
7:      for $y \in Y$ do
8:          if $y = c$ then
9:              $y = 1$
10:         else
11:             y=0
12:         end if
13:     end for
14:     $Model = RandomForest()$
15:     $Model.fit(X', Y)$
16:     Save Model
17: end for
```

The training data for a device type is processed separately: the features and labels are separated, and the data points of 'is device' are marked with 1 and 'not device' are marked with 0. The processed data is used to train the model and the trained model is named after its device type. When predicting, take one model from the saved models each time. If a test data returns 'is device' through the model, the process stops and marks this piece of data as its model name. Otherwise, it continues to take the next one from the saved model for testing.

4.3 Unknown Device Classification

4.3.1 Using AE for Data Compression

Besides the AE architecture discussed in Section **??**, other different architectures can be used for deep AEs. In our proposed framework, a 4-layer deep AE architecture, shown in Figure **??**, is used, and the reason for such a choice is because deep AEs are known to outperform shallow AEs [**?**, **?**, **?**]. As described in [**?**], the backward propagation gradient of the underlying error will be small for deep AEs. Instead of manually choosing the size of the hidden layer, we sparse the autoencoder [**?**] on two hidden layers. The idea is that when sparse constraints are imposed on the hidden units, only some of the units will be active, and the remaining cells will be set to zero. Therefore, sparse AE will neither directly copy the input (because the size of the hidden layer is larger than the input size), nor cause under-learning (because the size of the hidden layer is too small). To achieve this, we will add an L2 activity regularizer. The size of hidden layers h_1 and h_2 can be set equal to the size of input and output layers as n. The size of the bottleneck is m.
For this, we can then write

$$h_1 = f_{\theta 1}(x) = R(W_1 x + b_1) \tag{4.1}$$

where $R(z) = max(0, z)$ is ReLu function and the parameter W_1 is an $n \times n$ weight matrix and b_1 is an n-length bias vector. The expression relating h_1 to $\mathcal{L}$ is

$$\mathcal{L} = f_{\theta 2}(h_1) = R(W_2 h_1 + b_2) \tag{4.2}$$

with the parameter W_2 being an $m \times n$ weight matrix and b_2 being an m-length bias vector. The expression relating $\mathcal{L}$ to h_2 is

$$h_2 = f_{\theta 3}(\mathcal{L}) = \sigma(W_3 \mathcal{L} + b_3) \tag{4.3}$$

where $\sigma(z) = \frac{1}{1+e^{-x}}$ is Sigmoid function, and W_3 is an $n \times m$ weight matrix and b_3 is an n-length bias vector. The expression relating h_2 to $\hat{x}$ is

$$\hat{x} = f_{\theta 4}(h_2) = \sigma(W_4 h_2 + b_4) \tag{4.4}$$

with W_4 being an $n \times n$ weight matrix and b_4 being an n-length bias vector. Finally, the loss function then can be written as

$$J_{sparse}(\theta; x, \hat{x}) = \frac{1}{p} \sum_{i=1}^{p} ||x_i - \hat{x_i}||^2 + \lambda \sum_{i=1}^{p} (Wx_i + b)^2 \tag{4.5}$$

where λ is the regularization parameter.

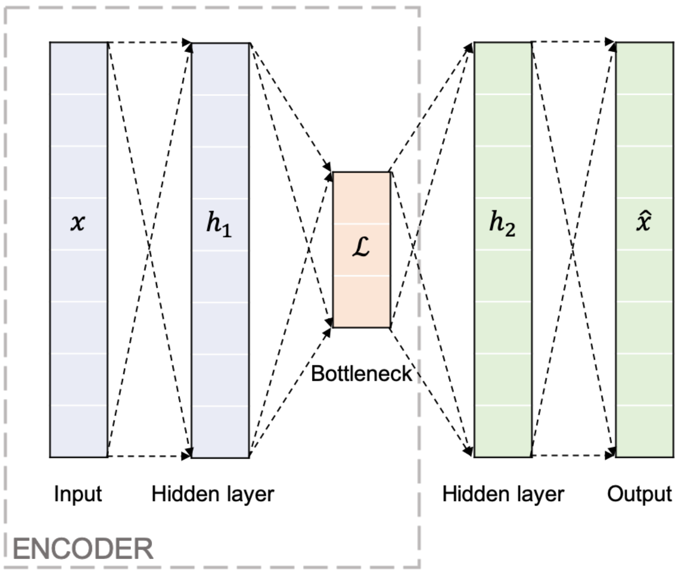


Figure 4.2: The architecture of the autoencoder

4.3.2 Density-Based Clustering as Unknown Classification and Anomaly Detection

Density-based clustering locates regions of high density that are separated from one another by regions of low density. Density in this context is defined as the number of points within a specified radius [?]. In this work, we use Order Points to Identity Clustering Structure (OPTICS) [?]. OPTICS can order the data points in an ascending order according to the reachability distance. Through ranking, we can visualize the pattern of clustering, called reachability plot. Figure **??** shows the data points displayed in two dimensions and the reachability plot corresponding to each point. We find that the reachable graph presents a valley-shaped wave. Each 'valley' is a cluster, and some individual points (outliers) on the peak do not remain in any clusters. In the proposed method, we consider the outlier as an *anomaly*. We use the ξ method [?] to extract clusters, so that no reachability threshold needs to be specified. OPTICS label the points in the same cluster as the same number and all outliers are marked as negative ones.

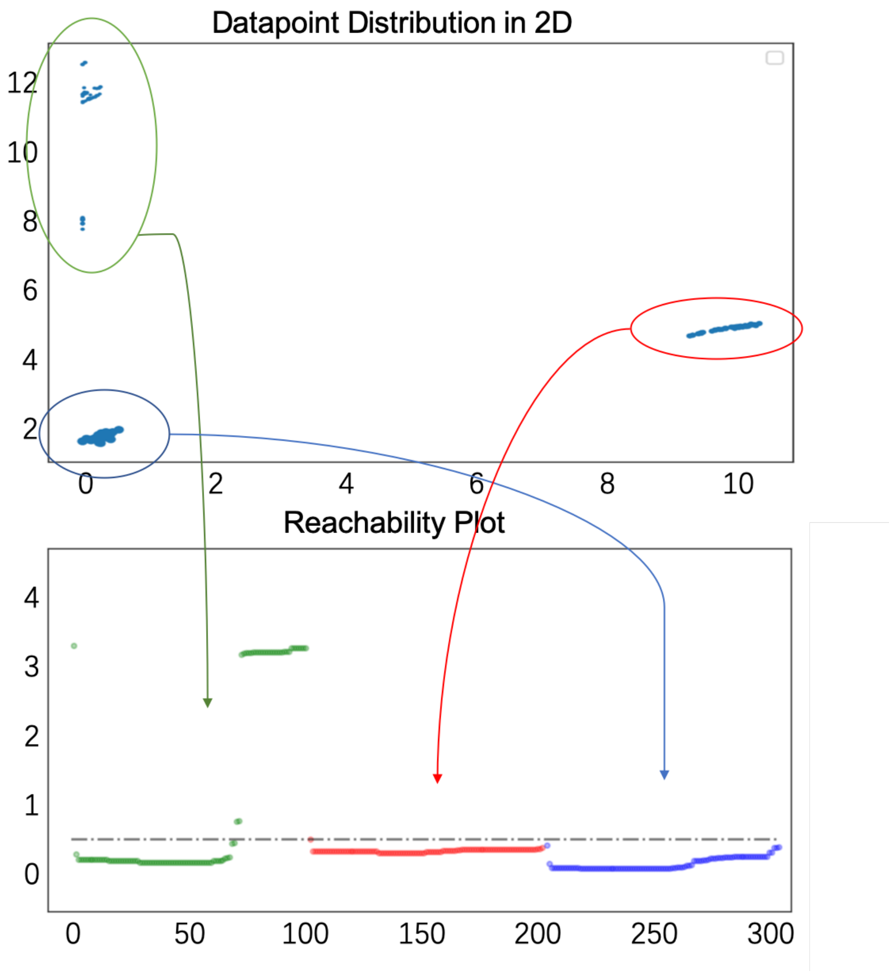


Figure 4.3: The reachability plot

Chapter 5: Performance Evaluation and Result Analysis

In this section, we describe the experimental setup, and analyze the performance of our proposed framework.

5.1 Datasets

The datasets we used in this thesis are comprised of real-world data that was originally collected by Yair Meidan et al. [?] from 10 different IoT device types out of 17 IoT devices, including baby monitor, lights, motion sensor, security camera, smoke detector, socket, thermostat, TV, watch, and water sensor. Some of these types are also produced by different vendors or same vendor with different models. Wireshark is used to record traffic data passed through a switch that connects all devices in a pcap file. Then, the recorded data is converted into a feature vector with $n = 297$ features by the feature extractor. We assume that the devices we studied are all Internet-enabled devices. Examples of features are the destination port number, packet size, number of packets with the PUSH bit set, number of out-of-order TCP segments of higher-level protocols running on top of it like SSL and HTTP. The full list can be found in [?]. A TCP session refers to a successful connection that starts with a 3-way-handshake and stops when it times out or receives a pair of FIN and ACK segments.
The dataset for training consists of 1400 feature vectors from 7 device types with their device-type labeled. The dataset for testing has 1005 feature vectors from 10 device types with 3 device types out the 10 are provided without label and considered as 'anomaly' device types. That is, 3 of the 10 device types are considered *new* and 7 are considered *known*. Detailed information about these experimental IoT devices are given in Table **??**.

Table 5.1: Instances Model and Distribution

Device Type	Make and Model	Number of Instances	
		train	test
baby monitor	Beseye Baby_Monitor,_Pro	200	100
lights	Samsung	200	100
motion sensor	D_Link DCH_S150	200	100
security camera	Simple_Home XCS7_1001, Withings WBP02_WT9510	200	100
smoke detector	Nest Nest_Protect	200	100
TV	Samsung UA40H6300AR, UA55J5500AKXXS	200	100
watch	LG G_Watch_R, Urban,Sony _SWR50	200	100
unknown1	N/A	0	100
unknown2	N/A	0	100
unknown3	N/A	0	100

5.2 Experimental Setup

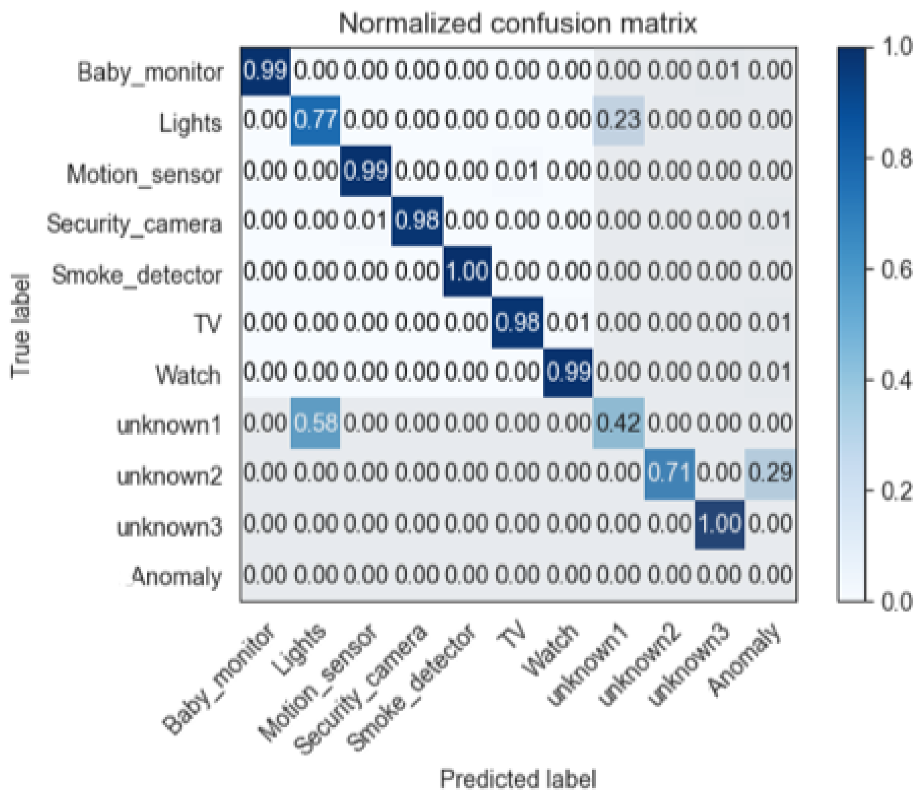
Our proposed method was implemented using Python3 on the macOS Catalina operating system. Because different features in the raw dataset have different ranges of values, we used the built-in MinMaxScaler of the Sklearn library in Python to perform min-max scaling on the data, making all feature values falling within the [0,1] range. (It is well-known that highly different values in the features' ranges might lead to less accurate results and problems with the training.) We use the same experiment setups as in [?] for training and predicting known devices. The classifier uses Random Forest from Sklearn and sets its tool-kit specific parameters as follows: The number of trees in the forest is $n\_estimators = 200$, the function to measure the quality of a split $criterion =' entropy'$, the maximum depth of the tree $max\_depth = 15$, and the minimum number of samples required to be at a leaf node $min\_samples\_leaf = 5$.
As mentioned in Section **??**, an L2 regularizer is chosen to sparse the hidden layer h_1. The regularizer parameter in loss function (**??**) is $\lambda = 10^{-5}$. The optimization algorithm uses Adam, with a constant learning rate of 10^{-3}. The dimension of hidden layer and bottleneck layer is set to $n = 297$ and $m = 20$. For the parameters used in OPTICS, the number of samples in a neighborhood for a point to be considered as a core point

is $minPt = 50$, and the other parameters are set to the default values provided from Sklearn library.

5.3 Results

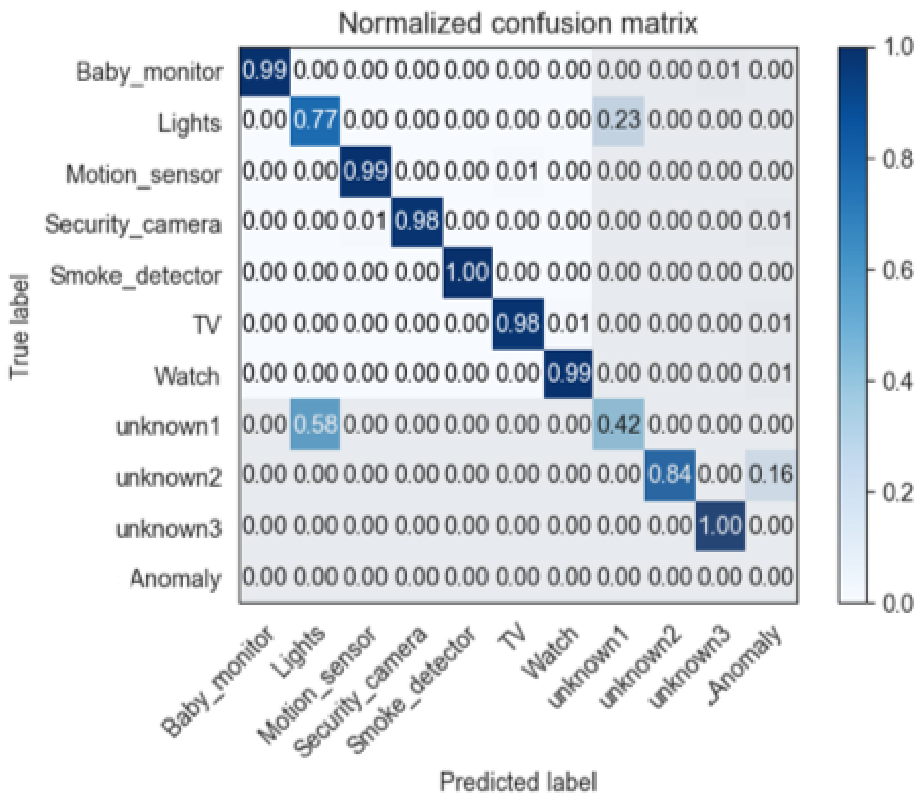
5.3.1 Accuracy

Our experimental results were performed using Python's Matplotlib library on the macOS Catalina operating system. Under the parameter settings mentioned above, **??**–**??** shows the prediction accuracy of classifying known and unknown device types in the test dataset with the size of bottleneck layer sizes equaling 2, 3, 5, 20, and 200. Our method achieved an average of 95% accuracy for *known* device classification and an average of 71%, 75.3%, 80.3%, 80.6%, and 80.6% for *new* devices under different bottleneck layer sizes. It can be seen from the figure that as the bottleneck size increases, the accuracy of predictions for unknown devices increases. When the bottleneck size increases to a certain value, the accuracy stabilized.



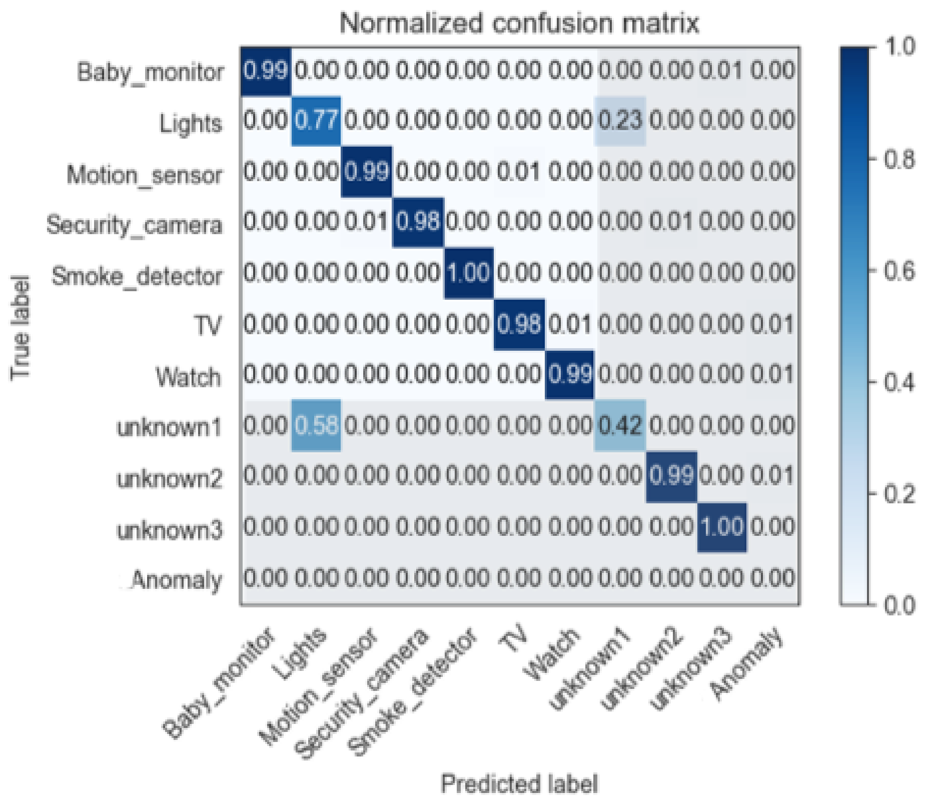
	Baby_monitor	Lights	Motion_sensor	Security_camera	Smoke_detector	TV	Watch	unknown1	unknown2	unknown3	Anomaly
Baby_monitor	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Lights	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.23	0.00	0.00	0.00
Motion_sensor	0.00	0.00	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
Security_camera	0.00	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Smoke_detector	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
TV	0.00	0.00	0.00	0.00	0.00	0.98	0.01	0.00	0.00	0.00	0.01
Watch	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.01
unknown1	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00
unknown2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.00	0.29
unknown3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Anomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 5.1: The confusion matrix with $\mathcal{L}$=2



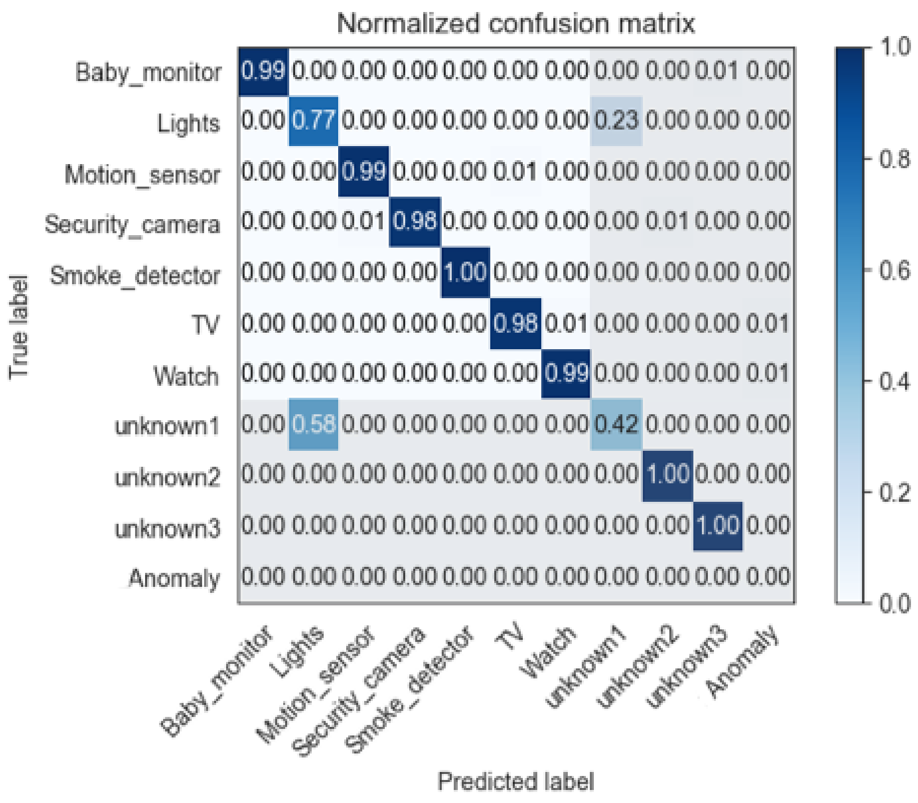
	Baby_monitor	Lights	Motion_sensor	Security_camera	Smoke_detector	TV	Watch	unknown1	unknown2	unknown3	Anomaly
Baby_monitor	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Lights	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.23	0.00	0.00	0.00
Motion_sensor	0.00	0.00	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
Security_camera	0.00	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Smoke_detector	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
TV	0.00	0.00	0.00	0.00	0.00	0.98	0.01	0.00	0.00	0.00	0.01
Watch	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.01
unknown1	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00
unknown2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.84	0.00	0.16
unknown3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Anomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 5.2: The confusion matrix with $\mathcal{L}$=3



	Baby_monitor	Lights	Motion_sensor	Security_camera	Smoke_detector	TV	Watch	unknown1	unknown2	unknown3	Anomaly
Baby_monitor	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Lights	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.23	0.00	0.00	0.00
Motion_sensor	0.00	0.00	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
Security_camera	0.00	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.01	0.00	0.00
Smoke_detector	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
TV	0.00	0.00	0.00	0.00	0.00	0.98	0.01	0.00	0.00	0.00	0.01
Watch	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.01
unknown1	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00
unknown2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.01
unknown3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Anomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 5.3: The confusion matrix with $\mathcal{L}$=5



	Baby_monitor	Lights	Motion_sensor	Security_camera	Smoke_detector	TV	Watch	unknown1	unknown2	unknown3	Anomaly
Baby_monitor	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Lights	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.23	0.00	0.00	0.00
Motion_sensor	0.00	0.00	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
Security_camera	0.00	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.01	0.00	0.00
Smoke_detector	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
TV	0.00	0.00	0.00	0.00	0.00	0.98	0.01	0.00	0.00	0.00	0.01
Watch	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.01
unknown1	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00
unknown2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
unknown3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Anomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 5.4: The confusion matrix with $\mathcal{L}$=20

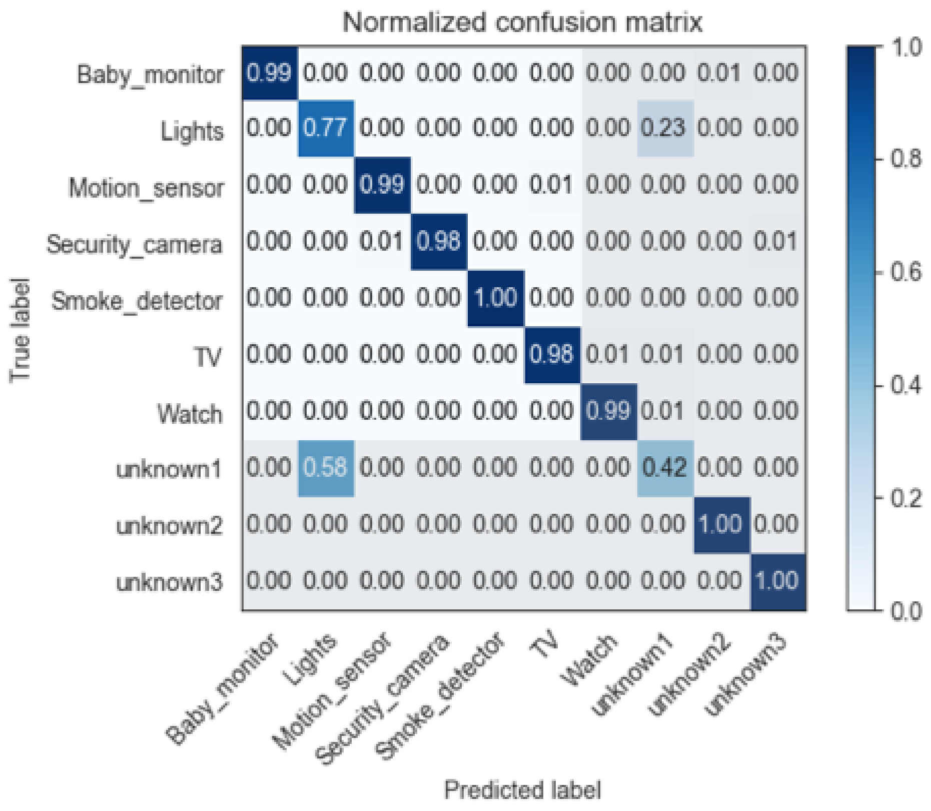


Figure 5.5: The confusion matrix with $\mathcal{L}$=200

5.3.2 Optimal Bottleneck Size

To determine the optimal bottleneck size, we tested different compression ratios to get Figure **??** and **??**. The calculation formula for the compression ratio is

$$Compression\ Ratio = \frac{Uncompressed\ Size}{Compressed\ Size} = \frac{Feature\ Size}{Bottleneck\ Size} \tag{5.1}$$

We compress the bottleneck layer of the autoencoder to determine whether there is a dimensional region in which the behavior of the reconstruction loss, SPI, and SAI metrics change significantly, which is also called the 'elbow region.' We trained an autoencoder to reduce the size of the bottleneck layer from 297 to 2 in steps of 1.

Figure **??** shows how system runtime and memory usage changes at different autoen-

coder compression ratios, compared to the performance without the autoencoder. As can be seen from the figure, our proposed autoencoder can significantly reduce system running time and save memory compared with no autoencoder. At the same time, as the compression ratio increases, time and memory both decrease. Therefore, we conclude from these two restrictions that the smaller the bottleneck size, the better, as long as the reconstruction loss is kept small at the same time. Figure **??** shows three criteria used in [**?**] to evaluate the reconstruction capability of the autoencoder. Multiple criteria can better evaluate the quality and performance of the network training process. The calculation of reconstruction loss is given in function (**??**).
The Structure Preservation Index (SPI) captures the input structure distortion caused by the autoencoder process. The calculation is as follow:

$$SPI = \frac{1}{p^2}\sum_{i=1}^{p} ||D_{ij} - \hat{D_{ij}}||^2 \tag{5.2}$$

where D_{ij} is the cosine similarity score between ith and jth data vector in the input dataset. Similarly $\hat{D_{ij}}$ is the cosine similarity score between ith and jth data vector in the reconstructed dataset. The closer the SPI value is to zero, the better.
The Similarity Accumulation Index (SAI) illustrates the level of similarity between the input vector and its reconstructed version by looking at the angle instead of the magnitude. This validates the preservation of the relative strength of the vector dimension in the reconstruction. The calculation is as follows:

$$SAI = \frac{1}{p}\sum_{i=1}^{p} cosine(x_i - \hat{x_i}) \tag{5.3}$$

As we can see from Figure **??**, SPI shows the clearest 'elbow region.' It clearly captures the fact that after $m = 20$, the value of SPI will not increase significantly much if the value of m is greatly increased. Therefore, the size of the bottleneck should not be chosen less than $m = 20$. Combined with the conclusion of Figure **??**, the optimal bottleneck size is $m = 20$.

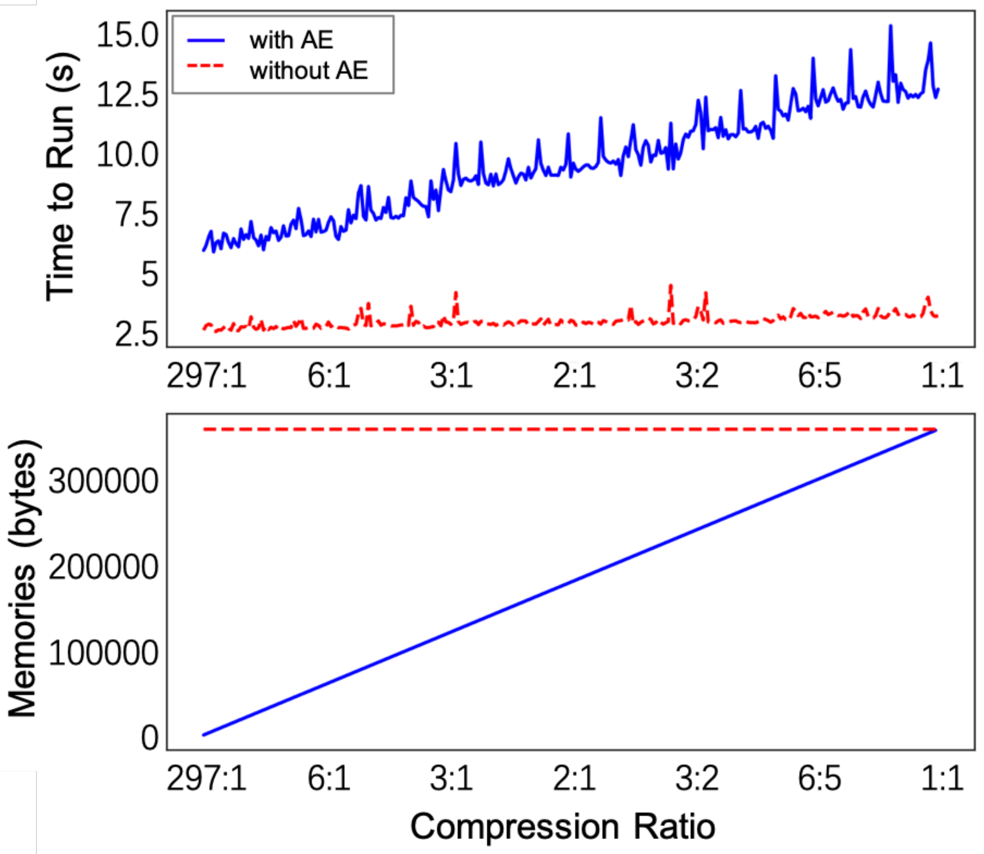


Figure 5.6: System runtime and memory usage at different AE compression ratios and without AE

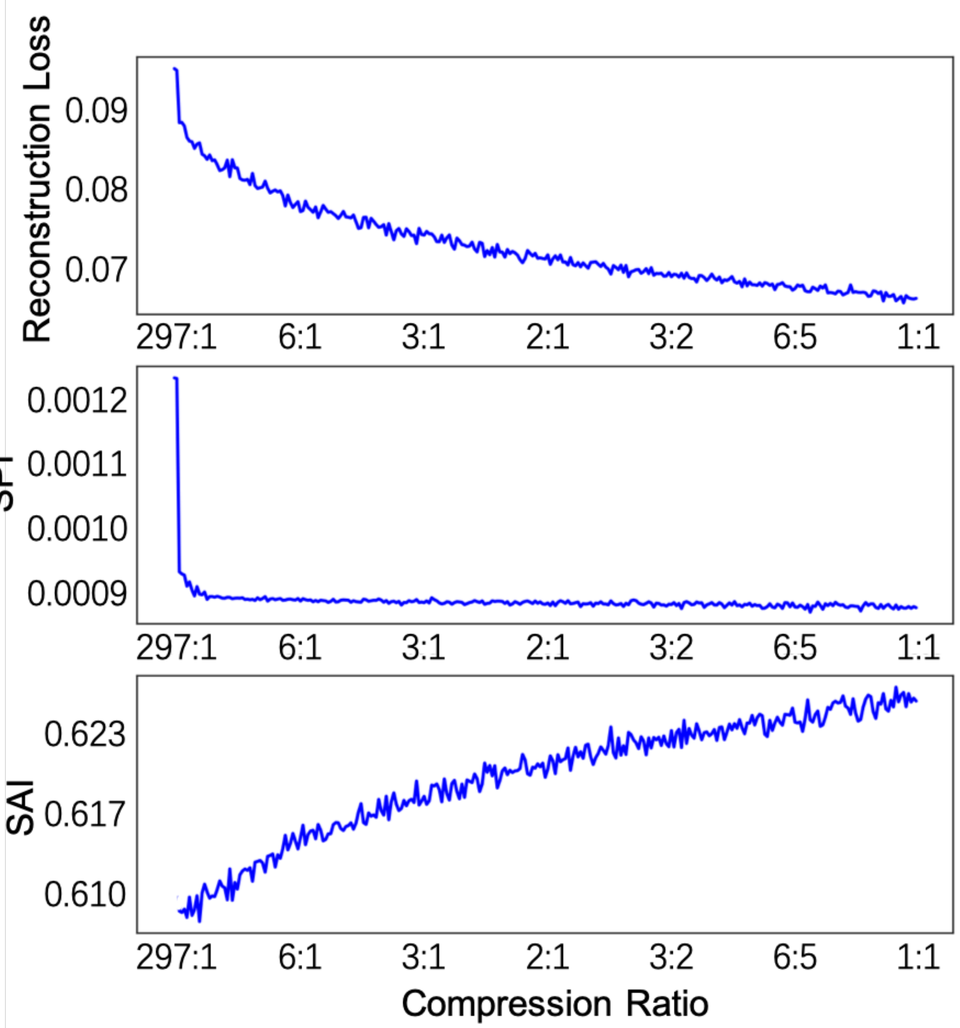


Figure 5.7: Reconstruction loss, SPI and SAI metrics as compression ratio decreases

5.3.3 Dataset Revisited: Feature Selection Consideration

When classifying the device type, there seems to be some feature vectors that dominate the discrimination. Therefore, we assess in this section the dominance of each feature by calculating its mutual information (MI) with the different device types. Let C denote the set of the device types, F denote the set of all features used in the dataset, and for each feature $f \in F$, V_f denote the set of all possible values that feature f can take on. For each feature $f \in F$, the mutual information (MI) between feature f and device type set C is given by:

$$MI(C, f) = \sum_{c \in C} \sum_{a \in V_f} p_{(C,f)}(c, a) \log \frac{p_{(C,f)}(c, a)}{p_C(c) p_f(a)} \tag{5.4}$$

where $p_C(c)$ for each $c \in C$ represents the probability that device type c is occurred in the dataset and can be expressed as

$$p_C(c) = \frac{\text{Number of data points belonging to device type } c}{\text{Total number of data points in dataset}}, \tag{5.5}$$

for each $f \in F$, $p_f(a)$ for each $a \in V_f$ represents the probability that value a of feature f is occurred in the dataset and can be expressed as

$$p_f(a) = \frac{\text{Number of data points in which feature value } a \text{ occurred}}{\text{Total number of data points in dataset}} \tag{5.6}$$

and for each $f \in F$, $p_{(C,f)}(c, a)$ for each device type $c \in C$ and each feature value $a \in V_f$ represents the joint probability for device type c and feature value a of feature f and can be expressed as

$$p_{(C,f)}(c, a) = \frac{\text{Number of data points in which value } a \text{ of feature } f \text{ occurred for device type } c}{\text{Total number of data points in dataset}} \tag{5.7}$$

Figure **??** shows the MI of each feature across all device types. We observe that a significant number of features used in the dataset have high MI values, dictating that these features are too device-type specific, and hence, they provide high device separability. For instance, if a MAC address is used as a feature, then this feature will have very high MI, as each device will have a different MAC address, and hence, if used for separability, such a feature will suffice to ensure 100% classification accuracy.

We simulate a potential network security hazard that supposes an attacker learns to mimic normal device communication behavior by observing the IDS classification method

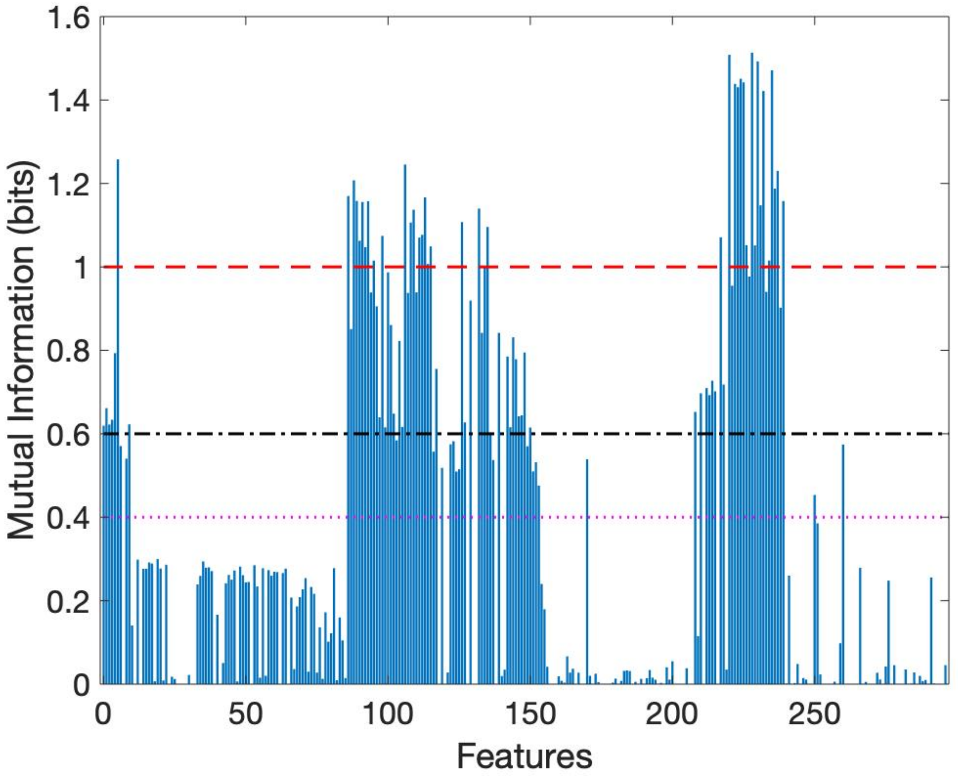


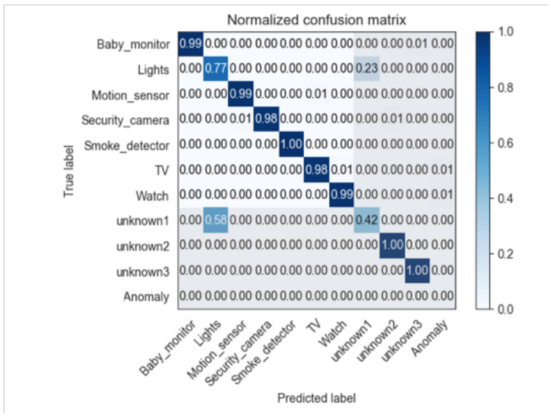
Figure 5.8: Mutual information of all 297 features of the dataset

Table 5.2: Features Found to be Important

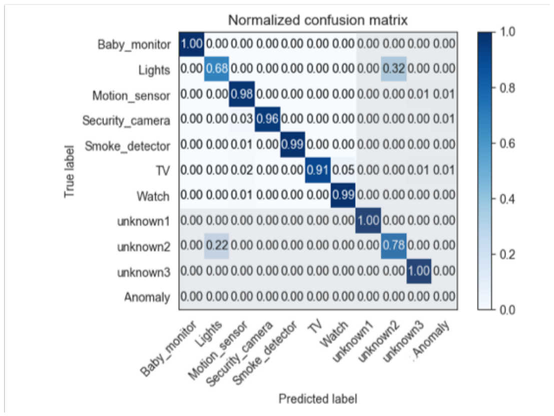
Features description	Average mutual information (bits)
TCP packets time-to-live by server	1.28
TCP packets time-to-live	1.19
Packets inter arrival time by client	1.1
Packets inter arrival time	1.07
Packets inter arrival time by sever	0.8
Packets size	0.7
Number of bytes send and receive	0.66
TCP packets time-to-live by client	0.61
Packets size by client	0.54
Total packets number	0.54

so that a malicious device can generate some features similar to normal devices to fool IDS. We need to verify whether our proposed method can still accurately identify the device type in this case. To this end, we set three thresholds for MI of 1 bit, 0.6 bits, and 0.4 bits, and remove features with MI values higher than the threshold, leaving us with features that we refer to as 'important'. The description of features that considered to be important and their mutual information are shown in Table **??**, and their corresponding classification accuracy results are shown in the Figure **??**. The average accuracy is 91.2%, 92.9%, 89.8%, and 81.8% when the number of input features is 297 (initial dataset), 234, 203 and 179, respectively. Therefore, it can be said that in terms of accuracy, the system can remain stable even if attackers spoof 40% of the features and retains only 60% of the original features.

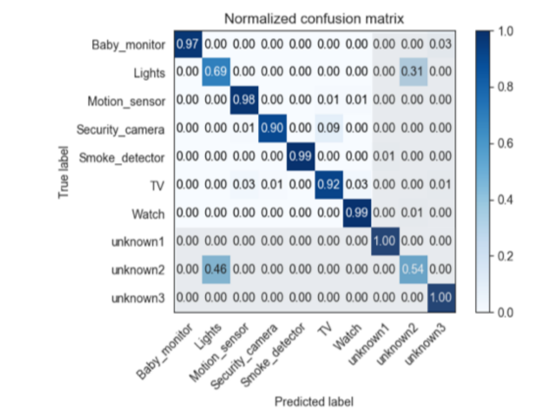
The main assumption that must be validated to verify the experimental results is that the inter-class variance is closer and stronger than the intra-class variance. Devices in the same type behave sufficiently similarly, while devices in separate types behave differently. However, the results show that intra-class differences between certain device types ("lights" and "unknown1") are not negligible and will have an impact during the analysis. This assumption applies well in some types and less in others. The problem of differences in specific types is a limitation of this design. One solution is to use more devices in each device type and continuously capture more data for the training section. The similarity between different types of certain devices reduces the impact of similar characteristics by increasing the diversity of devices. In regard to the most appropriate



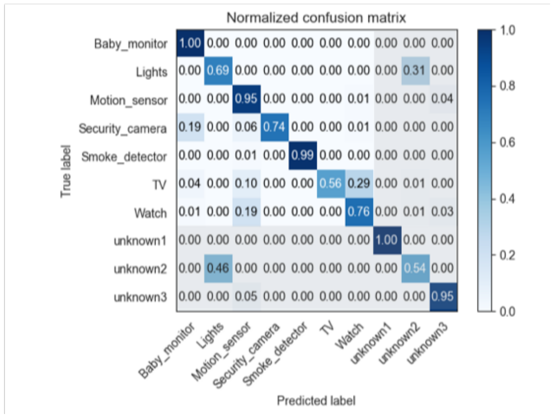
(a) $n = 297$



(b) $n = 234$



(c) $n = 203$



(d) $n = 179$

Figure 5.9: The confusion matrix with different numbers of input features n

bottleneck layer size, the reconstruction capability of the autoencoder is highly related to the size of the bottleneck layer, because the smaller the bottleneck layer is, the more information is lost. The simplified steps of the autoencoder are also called hashing, and because similar sentences in the projection space are close to each other, this technique is also called semantic hashing. Choosing the right bottleneck layer size is important for two reasons: (i) too large a size may lead to higher computing and storage costs, and (ii) too small a size may cause high information loss [**?**]. The method of calculating the most suitable bottleneck dimension has rarely been mentioned in previous studies. We can only weigh the optimal size based on actual experiment run time, memory consumed, and reconstruction loss.

Chapter 6: Future Work and Conclusion

6.1 Future Work

There is still much work to be done in the future to create a powerful self-contained anomaly detection system for IoT devices. One of them is to formulate a specific network restriction rule for a classified equipment system. For example, there can be three isolation levels: enabling communication with other devices and all Internet connections for known devices, disabling communication with other devices for new devices, opening only a limited set of remote targets, and disabling all communication for abnormal devices.

Another possibility worth considering is increasing the number of device types to evaluate the system accuracy. Because in actual applications, known devices cannot be identified 100%, this will cause the data of some unidentified known devices to fall into the unknown set and become the noise of the unknown classifier. This reduces the accuracy of the entire system. Therefore, measures need to be taken to mitigate this loss of accuracy.

Future research could also explore developing applications and adapting our designs to other scenarios [**?**]. These may include different network protocols and various application environments to better understand how our approach can be extended and generalized.

6.2 Conclusion

In this paper, we propose a system that uses device behavior to classify IoT device types. The ultimate goal is to detect anomalies in order to protect network security, using unsupervised learning to classify device types to make the system self-updating. The whole system consists of two parts: the classification of the known device category and the classification of the unknown category (including new and anomaly). The raw data is first processed by the data processor into a feature vector represented by 297 features. For known devices, the system then trains a corresponding classifier model for each device type and passes the data through the model in turn. If any model gives a positive response, the data is verified as the corresponding equipment type of its model. For unknown devices, the data is compressed into low dimensions by the encoder in the autoencoder. These data are stored and clustered to label the nearest data points as a class. Individual data points are labeled as anomalies. We successfully identified unknown devices using our proposed method with accuracy as high as 100%. Therefore, our system shows both high accuracy and good practicability.

Bibliography

[1] A. Sundararajan, A. Chavan, D. Saleem, and A. Sarwat, "A survey of protocol-level challenges and solutions for distributed energy resource cyber-physical security," *Energies*, vol. 11, p. 2360, 09 2018.

[2] B. Park, Y.-H. Han, and H. Latchman, "An approach to reliable and efficient routing scheme for tcp performance enhancement in mobile ipv6 networks," vol. 4412, 01 2006, pp. 160–169.

[3] R. E. Bellman and R. E. Kalaba, "Dynamic programming and feedback control," 1959.

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

[5] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Iotsense: Behavioral fingerprinting of iot devices," *arXiv preprint arXiv:1804.03852*, 2018.

[6] S. Siby, R. R. Maiti, and N. Tippenhauer, "Iotscanner: Detecting and classifying privacy threats in iot neighborhoods," *arXiv preprint arXiv:1701.05007*, 2017.

[7] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[8] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "Profiliot: a machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the symposium on applied computing*. ACM, 2017, pp. 506–509.

[9] S. Aneja, N. Aneja, and M. S. Islam, "Iot device fingerprint using deep learning," in *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*. IEEE, 2018, pp. 174–179.

[10] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3854–3861.

[11] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From data mining to knowledge discovery in databases,” *AI magazine*, vol. 17, no. 3, pp. 37–37, 1996.

[12] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: ordering points to identify the clustering structure,” in *ACM Sigmod record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[13] P. Domingos, *The master algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books, 2015.

[14] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] H. Mhaskar, Q. Liao, and T. Poggio, “When and why are deep networks better than shallow ones?” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[16] A. Ng *et al.*, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[17] M. Steinbach, L. Ertöz, and V. Kumar, “The challenges of clustering high dimensional data,” in *New directions in statistical physics.* Springer, 2004, pp. 273–309.

[18] E. Schubert and M. Gertz, “Improving the cluster structure extracted from optics plots.” in *LWDA*, 2018, pp. 318–329.

[19] “Full network traffic features set.”

[20] P. Gupta, R. E. Banchs, and P. Rosso, “Squeezing bottlenecks: exploring the limits of autoencoder semantic representation capabilities,” *Neurocomputing*, vol. 175, pp. 1001–1008, 2016.