

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

共同指導教授：劉榮春博士

基於 Ceph 儲存環境架構建構應用於 NetFlow 日誌資料之深度學習辨識網路攻擊模型實作

The Implementation of Deep Learning Modules for
Cyberattack Identification in NetFlow Data Log with
Ceph Storage

研究生：劉明倫

中華民國一零八年六月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 劉明倫 所提之論文

基於 Ceph 儲存環境架構建置 NetFlow 日誌資料
辨識網路攻擊之深度學習模型實作

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召集人

許慶賢

簽章

委員

劉榮春

詹毓偉

賴廷川

指導教授

楊朝棟

簽章

中華民國 108 年 6 月 12 日

摘要

在現今資訊快速流動的時代，毫無疑問的網路成為人類生活中不可或缺的部分，隨著日漸增長的網路使用量，長久累積下來的日誌資料是非常龐大的，傳統的儲存方式已經漸漸不足以應付龐大的網路日誌資料，而在網路日誌資料中隱藏著一些不正當的網路行為，如何儲存龐大的網路日誌資料並透過資料做即時分析找出可疑的網路行為，將是一項具有挑戰的研究。本篇論文提出一套完整架構的來對收集到的網路日誌資料進行儲存與分析，我們將校園內各台路由器所收集到的網路資料進行處理整合，並將整合完的資料處存到具有開源、高性能、高可靠性與可擴展性的 Ceph 分散式儲存環境，並通過 Python 對原始資料做初步預處理，去除冗餘欄位以及單位統一。將整理後的資料集分為兩個部分分析，一部分是異常分析一部份是攻擊辨識。在分異常分析中，我們透過三倍標準差規則找出流量異常時間段與總流量。另一方面我們透過 Keras 來對取得資料進行網路攻擊辨別。通過遞迴時間網路 (RNN) 建立自動化辨識模型，用來辨識具有固定特徵之攻擊。另外使用 NSL-KDD 數據集作為訓練集，評估各種深度學習模型對無固定特徵之攻擊的辨識能力，並且本論文提出了一個辨識模型，在 NSL-KDD 數據集的辨識準確度可達 99.65%。最後，將即時分析結果透過 MySQL 資料庫進行存取，並將分析結果透過 ECharts 作視覺化呈現，以利於管理者可以快速掌握異常的網路行為，以及即時的攻擊辨識。

關鍵字: 資料儲存、Ceph、深度學習、網路攻擊、網路日誌資料

Abstract

In today's fast-moving information era, there is no doubt that the Internet has become an indispensable part of human life. However, in the world of the Internet, it also hides unusual network behavior. Find the hidden unusual network behavior can reduce the vulnerability in the network. This paper proposes a complete architecture to store and analyze the collected network log data. We process and integrate the network data collected by each router on the campus, and store the integrated data. Ceph distributed storage environment with open source, high performance, high reliability and scalability, and preliminary preprocessing of raw materials through Python, eliminating redundant fields and unit unification. The collated data set is divided into two parts analysis, and part of the abnormal analysis is part of attack identification. In the sub-analysis, we find the abnormal time period and total flow through the standard deviation of three standard deviations. On the other hand, we use Keras to identify the cyber attacks on the data. An automated identification model is built through the Recurring Time Network (RNN) to identify attacks with fixed features. In addition, the NSL-KDD data set is used as a training set to evaluate the ability of various deep learning models to identify attacks without fixed features. In this paper, an identification model is proposed, and the identification accuracy in the NSL-KDD data set can reach 99.65%. Finally, the real-time analysis results are accessed through the MySQL database, and the analysis results are visualized through ECharts, so that managers can quickly grasp abnormal network behavior and instant attack identification.

Keywords: Data Storage, Ceph, Deep Learning, Cyberattack, NetFlow Log

致謝詞

能夠完成這篇論文必須感謝很多人，首先謝謝我的指導教授楊朝棟博士，不管是大三到大四的畢業專題，或是研究所開始進入 HPC 實驗室，老師藉由各種不同的訓練以及工作讓我碩士在學期間能夠快速的累積自己的實力並培養良好的工作態度，除了學術研究外也教會我許多做人處事的態度以及應對，再來要感謝共同指導教授劉榮春博士，沒有老師在大學與研究所期間的支持，老師總會在需要幫助時，不吝的給予幫助與鼓勵，謝謝兩位老師在大學和研究所的指導，相信這段期間的訓練對我往後的人生都有很大的幫助。在我的兩年的研究生涯中過得並不算太順利，從碩一到碩二上的研究方向一直不是非常明確，因此在這期間與其他同儕不同，我多方的學習了許多不同的系統與環境，但也因為所學的東西過於發散使自己開始迷惘，曾經覺得我的研究是不是該有的論文水平，或是需要延畢來重新規劃我的整體論文研究，所幸楊老師能在我茫然無助時，給了我一些新的想法和新的研究方向，讓我有動力繼續我的研究。謝謝三位口試委員許慶賢教授、詹毓偉教授、賴冠州教授在百忙之中抽空參加我的論文口試，每位教授提出的寶貴意見都能夠讓我的碩士論文更加完善。也謝謝 HPC 實驗室的學長姐、學弟們的指教與幫忙，讓我研究上少走了許多的彎路。特別謝謝跟我一起考進研究所的好夥伴們，有你們的陪伴和協助才讓我的研究所生涯更加順利。最後謝謝我的家人，謝謝爸爸、媽媽從小到大對我的栽培，支持我所做的決定，在我迷惘的時候鼓勵我並告訴我許多做人的道理，有你們各方面的陪伴與支持，我才能完成我的碩士學位，由衷感謝一路上幫助我和陪伴我的所有人，謝謝你們。

東海大學資訊工程學系 高效能計算實驗室 劉明倫 二零八年六月

Table of Contents

摘要	i
Abstract	ii
致謝詞	iii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Motivation	2
1.2 Thesis Contributions	3
1.3 Thesis Organization	4
2 Background Review and Related Work	5
2.1 Background Review	5
2.1.1 Ceph	5
2.1.2 Keras	6
2.1.3 Recurrent Neural Networks	7
2.1.4 Long Short Term Memory Network	8
2.1.5 Gate Recurrent Unit	9
2.1.6 NSL-KDD	11
2.1.7 Grafana	11
2.1.8 Bootstrap	12
2.1.9 ECharts	12
2.2 Related Work	12
3 System Design and Implementation	15
3.1 System Architecture	15
3.2 System Services	16
3.2.1 Data Collection	16
3.2.2 Data Storage	17
3.2.3 Data Preprocess	19

3.2.4	Data Analysis	19
3.2.5	Data Visualization	22
3.3	System Implementation	22
4	Experimental Results	26
4.1	Experimental Environment	26
4.2	Historical Flow Changes	27
4.3	Abnormal Analysis Result	28
4.4	Attack Identification with Fixed Features	29
4.5	Attack Identification without Fixed Features	31
4.5.1	RNN Model	31
4.5.2	LSTM Model	36
4.5.3	GRU Model	41
4.5.4	CNN Model	46
4.5.5	CNN and LSTM Model	51
4.6	Visualization of Results	53
5	Conclusions and Future Work	57
5.1	Concluding Remarks	57
5.2	Future Work	58
	References	59
	Appendix	63
A	Ceph Installation	63
B	CUDA 10.0, cuDNN7.4 and tensorflow 1.12 Installation	67
C	Anaconda Installation	69
D	download netflow data	71
E	flow count	72
F	abnormal check	74
G	Attack Identification with Fixed Features RNN Model	77
H	NSL-KDD Training of Rnn Model	79
I	NSL-KDD Training of LSTM Model	82
J	NSL-KDD Training of GRU Model	85
K	NSL-KDD Training of CNN Model	88
L	NSL-KDD Training of 2CNN CuDNNLSTM Model	91

M NSL-KDD Training of 4CNN CuDNNLSTM Model

94



List of Figures

2.1	Ceph architecture	6
2.2	RNN architecture	7
2.3	LSTM architecture	8
2.4	LSTM internal structure	9
2.5	GRU architecture	10
2.6	GRU internal structure	10
3.1	System architecture	16
3.2	NetFlow data format	17
3.3	Ceph storage method	18
3.4	Ceph read speed comparison	18
3.5	Ceph write speed comparison	19
3.6	Data preprocess flow	20
3.7	Data pre-processing step	20
3.8	Remote management interface of ESXi	23
3.9	Virtual machine monitoring interface	24
3.10	Ceph osd monitored	24
3.11	MySQL database	25
4.1	Total flow change graph for 30 consecutive days	27
4.2	Flow difference variation for 30 consecutive days	28
4.3	Abnormal analysis result	29
4.4	Training set for attack identification	30
4.5	RNN model architecture	30
4.6	Attack Classification Result	30
4.7	1-Layer RNN model	32
4.8	1-layer RNN accuracy and loss	32
4.9	2-layer RNN model	33
4.10	2-layer RNN accuracy and loss	33
4.11	3-layer RNN model	34
4.12	3-layer RNN accuracy and loss	34
4.13	4-layer RNN model	35
4.14	4-layer RNN accuracy and loss	35
4.15	1-layer LSTM model	36
4.16	1-layer LSTM accuracy and loss	36
4.17	2-layer LSTM model	37

4.18	2-layer LSTM accuracy and loss	37
4.19	3-layer LSTM model	38
4.20	3-layer LSTM accuracy and loss	38
4.21	4-layer LSTM model	39
4.22	4-layer LSTM accuracy and loss	39
4.23	2-layer CuDNNLSTM model	40
4.24	1-layer GRU model	41
4.25	1-layer GRU accuracy and loss	42
4.26	2-layer GRU model	42
4.27	2-layer GRU accuracy and loss	43
4.28	3-layer GRU model	43
4.29	3-layer GRU accuracy and loss	44
4.30	4-layer GRU model	44
4.31	3-layer GRU accuracy and loss	45
4.32	CNN model	47
4.33	CNN model accuracy	48
4.34	CNN model loss	49
4.35	Comparison of 4 training models	50
4.36	Comparison of 4 training time	50
4.37	CNN and LSTM model	51
4.38	CNN and LSTM model accuracy	52
4.39	CNN and LSTM model loss	52
4.40	Ceph dashbord	54
4.41	Ceph FS dashbord	54
4.42	Ceph grafana dashbord	55
4.43	Total usage of real-time data	55
4.44	Network historical traffic and abnormal traffic	56
4.45	Attack identification	56

List of Tables

3.1	Ceph storage cluster	22
3.2	Software Specifications	23
4.1	Computing environment	26
4.2	Ceph virtual machine environment	27
4.3	Deep learning virtual machine environment	27
4.4	1-layer RNN model training result	31
4.5	1-layer RNN model training result	32
4.6	2-layer RNN model training result	33
4.7	3-layer RNN model training result	34
4.8	4-layer RNN model training result	35
4.9	1-layer LSTM model training result	37
4.10	2-layer LSTM model training result	37
4.11	3-layer LSTM model training result	38
4.12	4-layer LSTM model training result	39
4.13	Comparison of 2-layer LSTM and 2-layer CuDNNLSTM model	40
4.14	Comparison of 3-layer LSTM and 3-layer CuDNNLSTM model	41
4.15	Comparison of 4-layer LSTM and 4-layer CuDNNLSTM model	41
4.16	1-layer GRU model training result	42
4.17	2-layer GRU model training result	43
4.18	3-layer GRU model training result	44
4.19	4-layer GRU model training result	45
4.20	Comparison of 1-layer LSTM and 1-layer GRU model	45
4.21	Comparison of 2-layer LSTM and 2-layer GRU model	45
4.22	Comparison of 3-layer LSTM and 3-layer GRU model	46
4.23	Comparison of 4-layer LSTM and 4-layer GRU model	46
4.24	CNN model training result	49
4.25	Comparison of 2CNN LSTM and 4CNN LSTM model	52
4.26	NetFlow input result	53
4.27	NetFlow and TCP Flags input result	53

Chapter 1

Introduction

In today's fast-moving information era, there is no doubt that the Internet has become an indispensable part of human life. However, in the world of the Internet, it also hides unfair network behavior. Find the hidden unusual network behavior can reduce the vulnerability in the network. In the past, data records of network traffic were stored in the databases, but with the development of technology, a variety of network connections have been generated, and the general database has been difficult to face the growing data. Therefore, it is necessary to build a high-performance, reliable and scalable storage environment to store these materials.

The rise of hardware equipment and deep learning have influenced cyber-attack behavior. Countering this threat needs improvement and new strategy. How to counter the cyber-attack through deep learning technology will be an important issue nowadays.

In this experiment use the NetFlow data. The feature of NetFlow is that it does not contain any packet content and only contains the basic configuration data of the traffic. The advantage of using NetFlow is that the complete data packet is lightweight and fast, and is suitable for abnormal detection in a busy network environment.

In this research, our goal is to implement a Ceph [1] storage environment, store the generated NetFlow data, and use Keras to analyze the stored NetFlow data,

and visualize the NetFlow to monitor and identify the threat. Specific goals are listed below:

1. Establish a decentralized Ceph storage environment and use the CRUSH algorithm to distribute data in a balanced manner.
2. Instantly store the generated NetFlow data into the Ceph environment as a historical database.
3. Use Keras to analyze NetFlow data in real time and evaluate different algorithms and models.
4. Visualize the analysis results of NetFlow data using LAMP.

1.1 Motivation

With the rapid development of the Internet, the connection data generated every minute is enormous. For example, our school light NetFlow data generates nearly 20GB of connection data every day. The amount of data accumulated over a long period of time is tremendous, and it becomes unsuitable to use a general storage environment. Ceph supports the scalability of TeraByte to PetaByte, and the high reliability and high-performance open source distributed storage system using Crush algorithm for distributed storage and self-healing. Therefore, Ceph has become the best choice for our storage environment since our school sometimes has a power outage.

Because today's cyber attacks are emerging, even the combination of deep learning attacks is even more difficult to prevent. The traditional database comparison has not kept up with the ever-changing new attacks [2], and the time cost of machine learning is relatively high [3] [4], and the effect of immediate processing cannot be achieved. Therefore, in order to respond to new types of cyber attacks, would use deep learning to detect and evaluate cyber attacks through different algorithms and models. How to find useful information from the collected data

is the main contribution to the analysis using deep learning. If you can find the characteristics of the attack and the unusual situation in the collected data, then deal with the problems immediately, when the attack or abnormality occurs.

In addition to the above, how to visualize the analysis results is worth studying and discussing. How to help network managers to clearly and quickly find the source of the problem will be the primary value of data visualization. At present, mainstream visual tools on the market are charged. For example, Tableau and Power BI are completely functional. But, if consider development costs and custom applications, use Echarts combined with Responsive Web Design(RWD) to design a friendly monitoring environment.

1.2 Thesis Contributions

In this work, propose a complete integrated system architecture. Use Ceph to create a decentralized historical data storage environment. The original data is classified by Python, and the data is intensely studied and analyzed by the open source neural network library Keras. Finally, the real-time analysis results are stored in MySQL combined with Echarts to analyze the results and make a visual representation of RWD on the webpage. The following are the main contributions:

1. Use open source software to design a complete system architecture to test and implement problem solutions. The architecture completes data capture, data processing, data analysis and analysis visualization.
2. Establish a Ceph decentralized storage environment, through its high fault tolerance and scalability, reduce the risk and cost of data storage, and increase the data storage limit.
3. Use the RNN training model to identify attacks with fixed features and assess the true accuracy of the model.
4. Using different layers of RNN, LSTM, and GRU training models, the attacks with no fixed features are identified and the differences between the models

are evaluated. An identification model with the highest true accuracy is proposed.

5. Use Bootstrap to create a responsive web page and visually present the results of the analysis stored in MySQL by Echarts.

1.3 Thesis Organization

The structure of this paper is as follows. Chapter 2 introduces the background and related work of using tools. Chapter 3 provides an overview of the overall system architecture, explains the design and implementation of the various tools used, and displays the capabilities of each component in the system. Chapter 4 details the experiments and discussions. Finally, the conclusions and future work of this paper are presented in Chapter 5.

Chapter 2

Background Review and Related Work

In this section, review the background knowledge for later use of system design and implementation.

2.1 Background Review

2.1.1 Ceph

Among the areas of software-defined storage, Ceph [5] is open-source software that can be expanded on a large scale. It can store large amounts of data at a lower cost, and can balance the needs of system and data reliability. Supporting three types of services: block storage, object storage, and file system level storage on the same platform to support more types of storage environments in the future. The bottom layer of Ceph is a clustered storage environment. In the future, when expanding capacity, that is, using scale, only more nodes need to be added to the cluster to achieve the goal. Ceph has automatic repair and management functions, and can enhance the placement efficiency of data on storage devices through the

CRUSH [6] algorithm to simultaneously copy data to multiple nodes. When a node in a cluster fails, it does not affect the operation of the entire storage system.

Figure 2.1 is the three-layer basic architecture of Ceph. The first layer provides the storage of objects, blocks, and files. The second layer extracts the underlying data through the RADOS function library. The third layer is a storage space composed of a plurality of RADOS nodes.

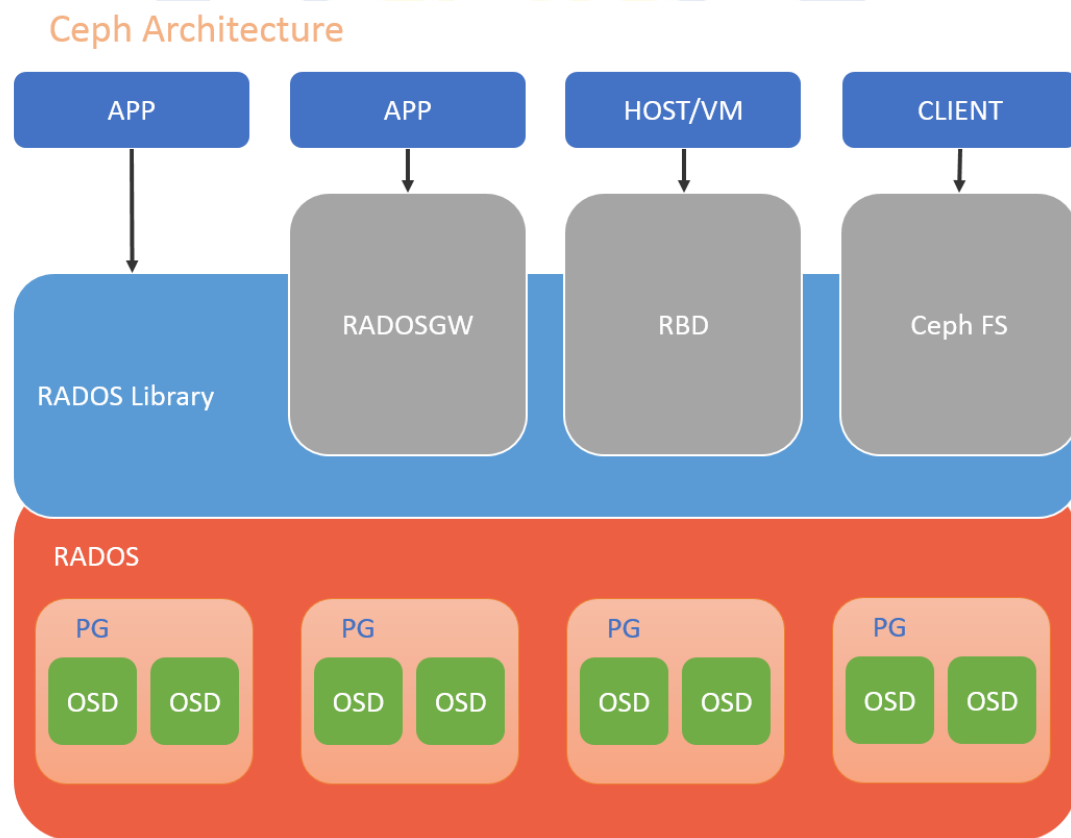


FIGURE 2.1: Ceph architecture

2.1.2 Keras

Keras [7] is an open source, high-level deep learning library written in Python that runs on TensorFlow, CNTK, or Theano. The focus of Keras' development is to support rapid experimentation. It focuses on being user-friendly, modular, and extensible. The main reason why Keras can perform convenient and fast operations is that it has already trained the input layer, hidden layer, and output

layer of the model, and only needs to add the correct parameters. Can be used. Keras supports both convolution and recursive neural networks and can be trained on CPUs and GPUs using deep learning models.

2.1.3 Recurrent Neural Networks

RNN [8] refers to the recurrent neural network, which is a kind of neural network. It is usually used for data that is highly correlated in processing time and spatial sequence. In traditional neural networks, all inputs and outputs are independent of each other. In time series data, this method may not be suitable for all situations. This type of problem can be solved by using RNN in cases where it is necessary to retain previous event information to infer the results. The concept of RNN is to cyclically pass information states through its own network. The output of the network depends on previous calculations, so it can handle a wider range of time series input structures.

Figure 2.2 is Recurrent Neural Networks Architecture.

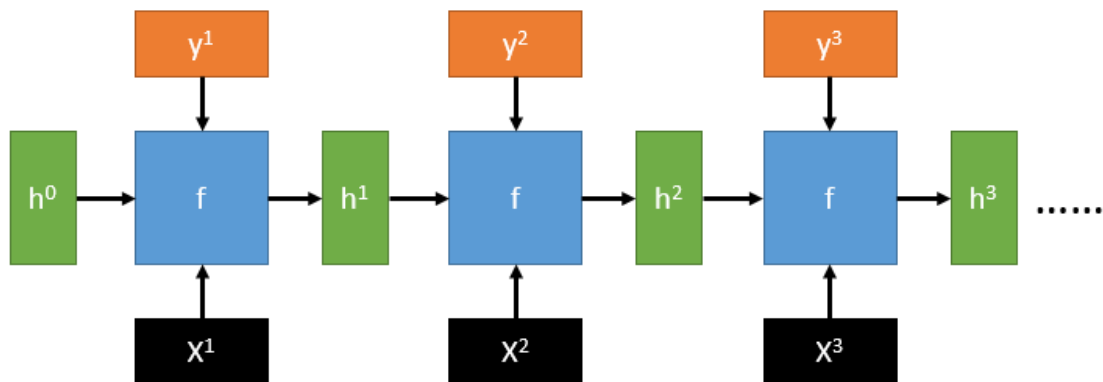


FIGURE 2.2: RNN architecture

2.1.4 Long Short Term Memory Network

Long Short Term is a special type of RNN that adds “long-term dependency” information through memory function [9], mainly to solve the problem of gradient disappearance and gradient explosion in long sequence training. Compared with the general RNN, there is an additional Cell state updated with time. The Forget Gate, Input Gate and Output Gate are used to determine the storage and use of memory. Control the transmission status through three Gates, remember the information that needs to be memorized for a long time, and forget the unimportant information. It can solve the problem of gradient disappearance and explosion in training. However, because of the increased number of introduced parameters, the training difficulty is increased.

Figure 2.3 is Long Short Term Memory Network architecture.

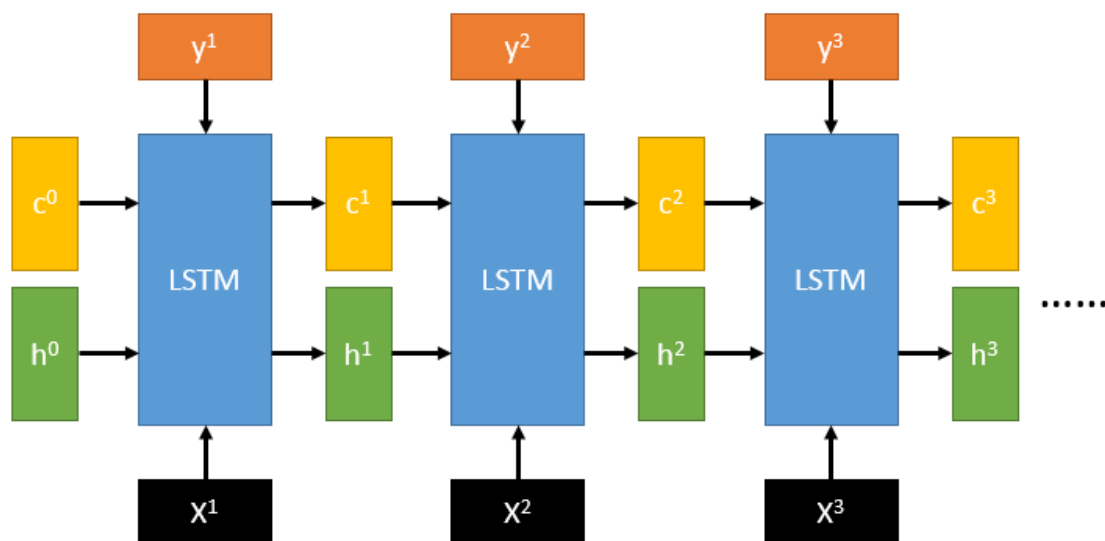


FIGURE 2.3: LSTM architecture

Figure 2.4 is LSTM internal structure.

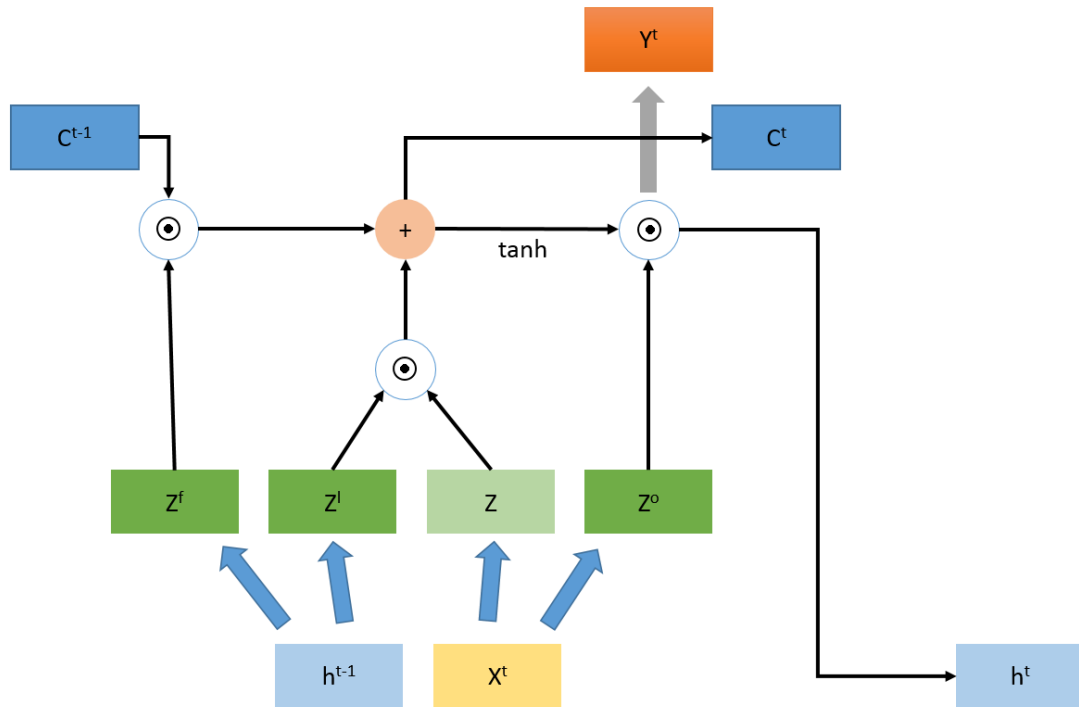


FIGURE 2.4: LSTM internal structure

$$c^t = z^f \odot c^{t-1} + z^i \odot z \quad (2.1)$$

$$h^t = z^o \tanh(c^t) \quad (2.2)$$

$$y^t = \sigma(W' h^t) \quad (2.3)$$

2.1.5 Gate Recurrent Unit

The Gate Recurrent Unit (GRU) [10] is also a type of cyclic neural network. The same as LSTM is to solve the problem of gradient explosion and gradient disappearing during long-term sequence training. The GRU simplifies the LSTM Input Gate and Forget Gate into an Update Gate. Compared with LSTM, one parameter is used, which can speed up execution and reduce memory usage during training. In terms of results, GRU can also achieve results similar to LSTM. The

practicality of the GRU will be higher considering the computing power and time cost of the hardware.

Figure 2.5 is Gate Recurrent Unit Network architecture.

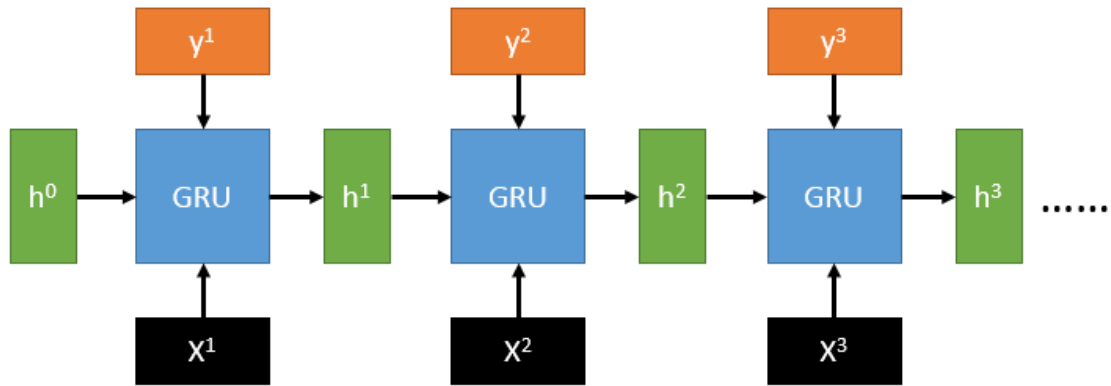


FIGURE 2.5: GRU architecture

Figure 2.6 is GRU internal structure.

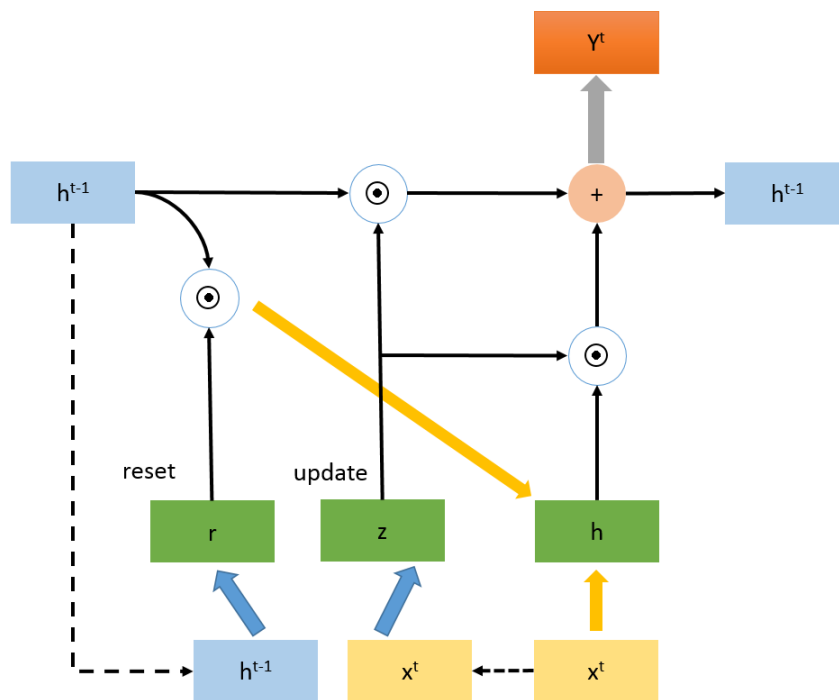


FIGURE 2.6: GRU internal structure

$$z = \sigma(x_t U^z + s_{t-1} W^z) \quad (2.4)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r) \quad (2.5)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \quad (2.6)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1} \quad (2.7)$$

2.1.6 NSL-KDD

The NSL-KDD [11] data set was improved from the KDD99 [12] data set to remove some of the problems in KDD99 [13]. The NSL-KDD data set is mainly made by hand, and there is a lack of public data sets based on the intrusion detection network. Therefore, there are still problems in the NSL-KDD data set, and it is not the perfect representative of the existing real network. But it can still be used as a benchmark for effectively evaluating intrusion detection to help researchers compare different intrusion detection methods. Each flow sample in the data set has forty-one feature indicators, which are mainly divided into three types of functions: basic functions, content-based functions, and flow-based functions. Attacks in the data set are mainly divided into four categories. They are: DoS, R2L, U2R, Probe. Four types of attacks can be subdivided into different types of attacks. Therefore, with the addition of normal data, the data set contains a total of five types.

2.1.7 Grafana

Grafana [14] is a cross-platform open source metric analysis and visualization tool that can be queried and visualized by the collected data and promptly notified. Grafana offers fast and flexible client charts that can be customized by setting up visual indicators and logs in a number of different ways. And Grafana can support a variety of different databases. Grafana can mix different data source displays in the same chart.

2.1.8 Bootstrap

Bootstrap [15] is a set of open source front-end frameworks for web and web application development, including HTML, CSS and JavaScript frameworks, providing typography, forms, buttons, navigation and various other components and Javascript extensions. Through the CSS3+JQuery webpage technology, the graphic content and database of the same website can be displayed on the device or screen of different sizes or resolutions in a layout style.

2.1.9 ECharts

ECharts [16] is a free Javascript chart library that runs smoothly on PCs and mobile devices. ECharts is compatible with most browsers today, and the underlying layer relies on the lightweight Canvas class library ZRender. Provides intuitive, highly customizable data visualization charts.

2.2 Related Work

In a paper published by Knowledge-Base Systems [3], Liu et al. proposed a detection method for instant port-to-port, using PL-CNN (a convolutional neural network-based payload classification method) and PL-RNN (Neural Network-Based Payload Classification Method) performs attack detection. The two methods learn feature representation from the original payload without feature engineering and support end-to-end detection. In this paper, we know that deep learning can be different from the traditional machine learning feature engineering in the complexity and time-consuming, more accurate and rapid detection. Kim et al. proposed a C-LSTM neural network [17] in the paper of Expert Systems with Applications to effectively detect anomalies in network traffic data. This is a method for automatically extracting time and space information from the original data. By extracting the spatial features of the CNN and the temporal characteristics of the LSTM model, it can achieve very good anomaly detection

performance in the network traffic data. In Computers and Security's paper, Ring et al. proposed a new method based on generating anti-neural network (GAN) to generate pseudo-NetFlow data [18], which can achieve good results for detection and generation. The main challenge is GAN can only deal with continuous attributes, and NetFlow usually contains multiple classification attributes. Therefore, this paper proposes three different preprocessing methods and applies this method to the CIDDS-001 data set. Experiments show that two of the three methods can produce high quality data. Tang et al. [19] applied a deep learning method based on flow anomaly detection in an SDN environment. A deep neural network (DNN) model was constructed and the NSL-KDD data set training model was used to obtain 75.75% accuracy. In this work, we propose an identification model with better accuracy. Fu et al. [20] applied the LSTM and GRU methods to traffic flow prediction and evaluated the performance of the two methods. And found that LSTM and GRU NNs have better performance than ARIMA, and GRU NNs perform a little better than LSTM NNs and usually converge faster than LSTM. Since the results of botnet detection methods are usually not compared, García et al. presented two botnet detection methods, BClus, CAMNEP [21], in Computers and Security, and compared three botnets using real data sets. Detection methods (BClus, CAMNEP and BotHunter). Analyze the impact of botnet activity on each method, each method best suited to different botnet phase data sets. This paper provides some of our ideas for testing botnets. Zhang et al. proposed a new method for detecting anomalous behavior in network performance data in a paper by Future Generation Computer Systems [22], which consists of two machine learning algorithms: Boosted Decision Tree (BDT) and simple Feedforward neural network composition. Evaluate and compare the effectiveness of each algorithm. In this experiment we also need to find out the behavior of the data set that does not meet expectations.

Many researchers apply machine learning methods to perform attack detection through payload classification. For example, Wang et al. proposed a payload-based anomaly network intrusion detection method. Kozik et al. used the flexibility of cloud-based architecture [23], as well as the latest advances in the field of

large-scale machine learning, shifting computationally more expensive and more demanding operations. Go to the cloud to efficiently perform traffic classification based on complex extreme learning machine models (ELMs) pre-built on the cloud using edge computing capabilities, reducing performance overhead on edge devices. Anomaly detection is the practice of identifying items or events that do not conform to the expected behavior or are not related to other items in the dataset. Rafał Kozik's paper in *Pattern Recognition Letters* proposes a method to combine NetFlow with an Extreme Learning Machine (ELM) classifier trained in the distributed environment of the Apache Spark framework [24]. The main contribution of this research is to use the Map-Reduce model to extend the training process of the ELM classifier to perform NetFlow-based malware activity detection algorithms. The results reported on the benchmark dataset indicate that the proposed ELM-based NetFlow analysis can be considered a reliable tool for network event detection. Data preprocessing is widely recognized as an important stage in anomaly detection. Davis et al. [25] reviewed the data pre-processing techniques used by the anomaly-based network intrusion detection system (NIDS), focusing on what aspects of network traffic are used and which feature constructs and selection methods are used. Hofstede et al. [26] explain all phases of NetFlow's traffic output and typical traffic monitoring settings, covering the full range from packet capture to data analysis. Terzi et al. [27] proposed a new unsupervised anomaly detection method. Its purpose is to determine the anomaly caused by a UDP flood attack on a specific IP. This approach is implemented on public NetFlow data in case studies.

Chapter 3

System Design and Implementation

3.1 System Architecture

Figure 3.1 is the system architecture diagram. Use the Python language to automate the download and storage of complete campus NetFlows data in the Ceph storage environment. In the virtual machine with GPU environment, NetFlow data is read by Ceph FUSE, then Python language is used for analysis and judgment, and data is visualized through matplotlib so that can analyze and optimize the model. And use sqlalchemy to store the analysis results in MySQL. Finally, use ECharts to present customized analysis results and combine Bootstrap to create responsive web pages.

In the vSphere ESXi Ceph storage environment [28], built four virtual machines as Monitor, OSD, and MDS for the Ceph storage environment. A virtual machine master is used as the master node and the client virtual machine is used to read the NetFlow data through the MDS, and the three virtual machines establish the monitor and the OSD as the nodes of the Ceph distributed storage. The GPU virtual machine node reads the real-time data in Python for data preprocessing, establishes the identification model by using the Keras library, analyzes

the pre-processed data through the identification model, and reads and writes the analysis result into the MySQL database through the sqlalchemy package. Finally, the results are displayed through the webpage.

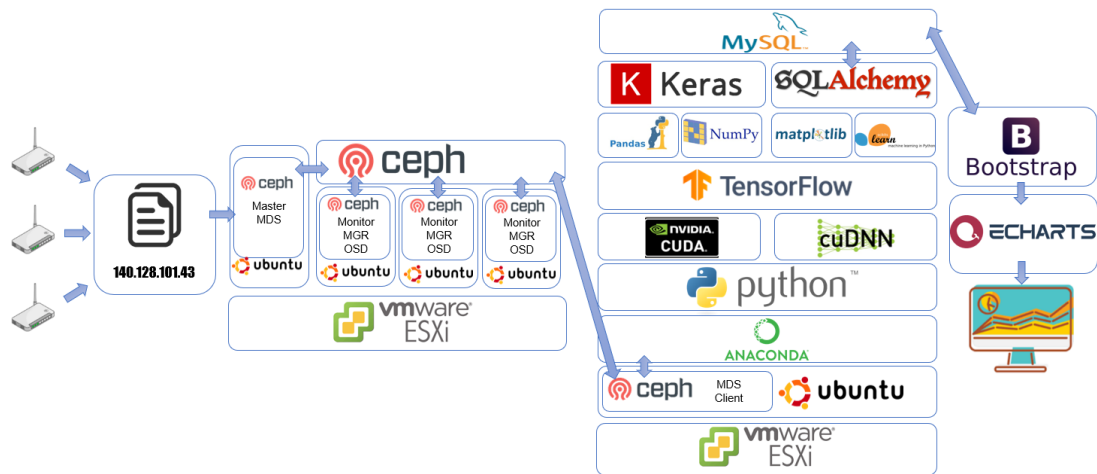


FIGURE 3.1: System architecture

3.2 System Services

This section is introduce the services provided by our system. Including data collection, data storage, data preprocessing, data analysis and data visualization.

3.2.1 Data Collection

The generation of the entire campus NetFlow data is made up of a collection of data collected by routers on the campus. NetFlow data updated every 5 minutes is regularly captured by the Python program and stored in the Ceph storage environment. The Ceph MDS is used to give the GPU virtual machine data analysis using NetFlow data.

The NetFlow data format is shown in Figure 3.2. It should be noted that the NetFlow data recorded by the school is a one-way network connection record. Therefore, the data records of Out Pkt and Out Byte are both zero. In addition, Input, Output records the router number that the network connection occurs.

ID(PG-ID) to be stored in each data and then calculate the stored OSD position. OSDs of different sizes would have different weights, and the data would be allocated in consideration of the weight when stored.

In addition, the test of moving the three OSDs to two and three physical machines for reading and writing was compared. Figure 3.4 is the result of the reading speed, and Figure 3.5 is the result of the writing speed. It can see that the distributed ceph storage environment can improve the read and write speed of files.

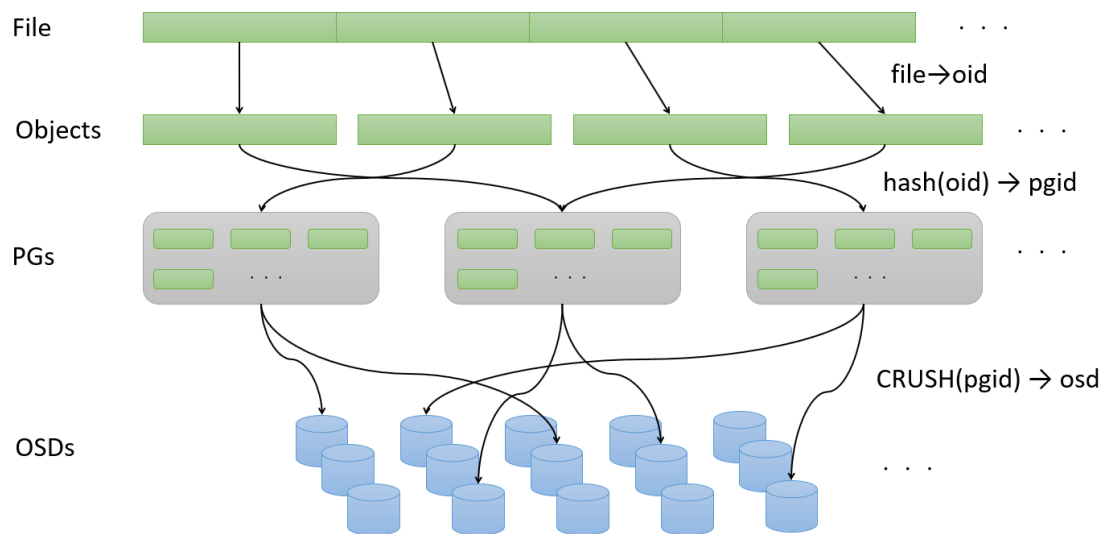


FIGURE 3.3: Ceph storage method

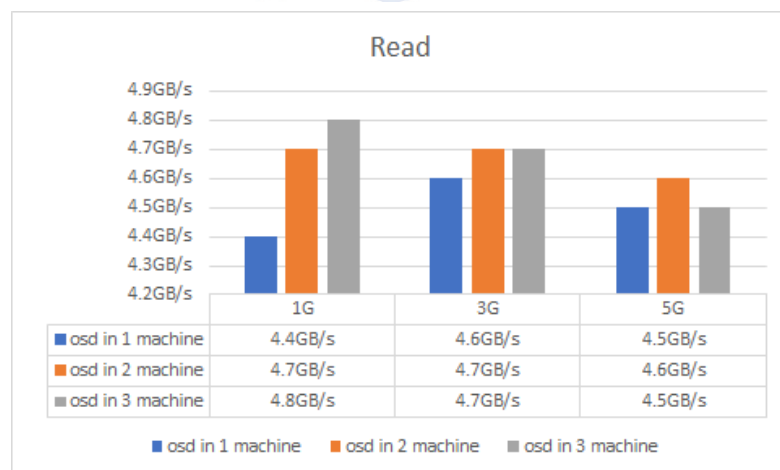


FIGURE 3.4: Ceph read speed comparison

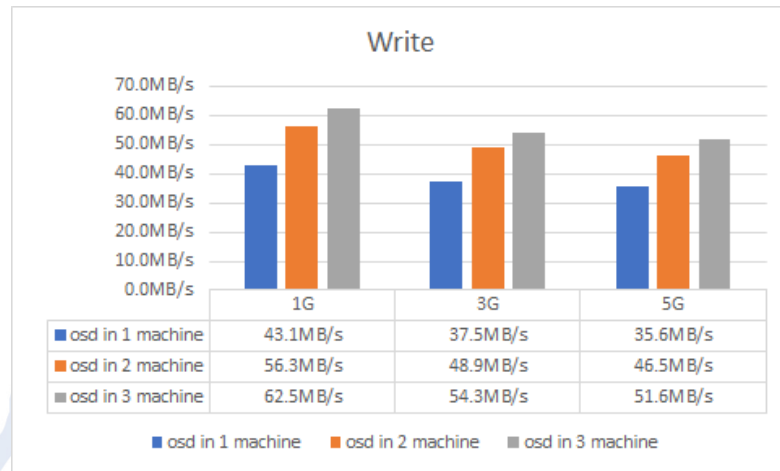


FIGURE 3.5: Ceph write speed comparison

3.2.3 Data Preprocess

When a new piece of real-time data is downloaded, this information cannot be directly used for data analysis. The required about filter out useful data fields and remove noise that may affect the results of the analysis to improve the quality of the data. Figure 3.6 is the flow of data processing. Figure 3.7 is the data preprocessing step. Since the units used by NetFlow data are not identical, have to convert the data into a uniform format. And delete the meaningless data field. Some data formats need to be converted to a format that can be analyzed using LabelEncoder [29] or OneHotEncoder [30]. Finally, the data is sorted into the input type of the in-depth analysis model for analysis.

3.2.4 Data Analysis

In the part of data analysis, mainly divided into the following two parts:

- **Abnomaly detection**

Since the flow is periodically fluctuating in days, the unit flow of the current time period would be close to the unit flow of the previous day [31]. And the increase the number of consecutively broken traffic would meet a reasonable range. So when doing anomaly detection, it would be divided into two parts.

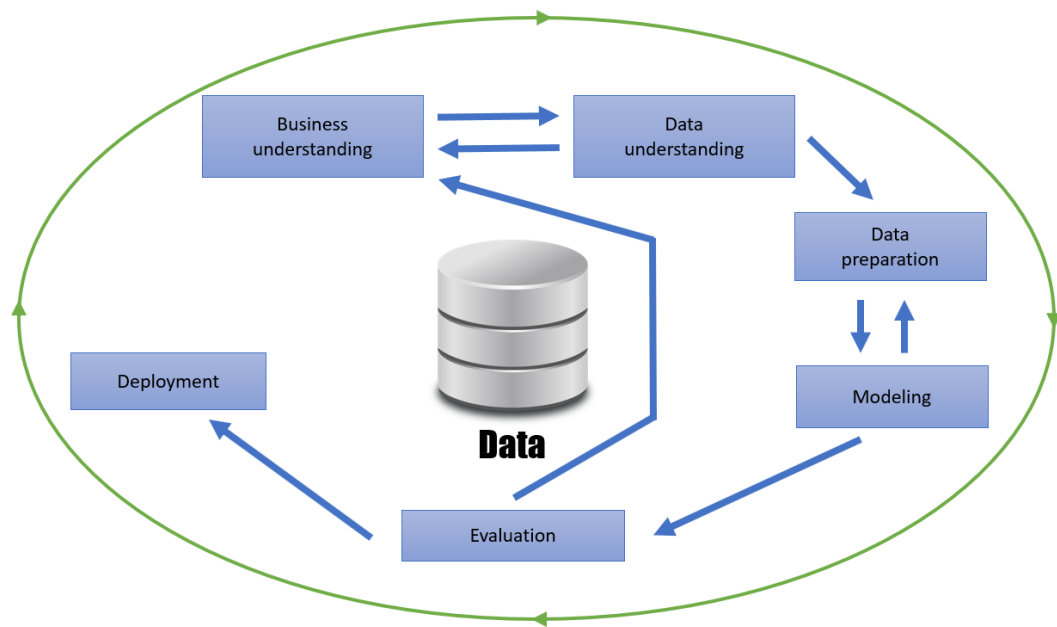


FIGURE 3.6: Data preprocess flow

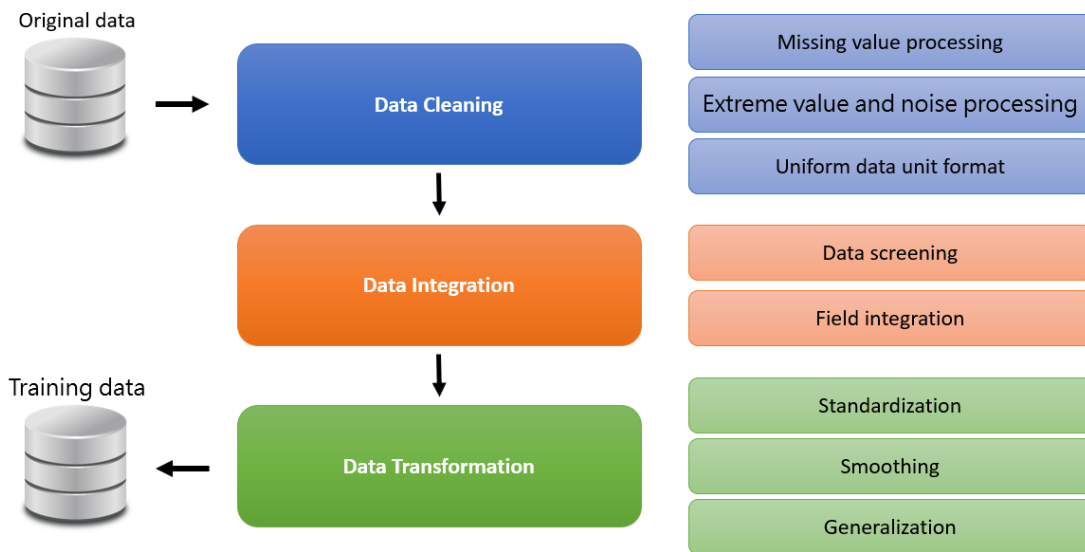


FIGURE 3.7: Data pre-processing step

The first part would compare the unit flow rate of the previous day, and if the increase in the flow rate is greater than 100%, it would be regarded as abnormal. The second part would calculate the difference between the current time and the previous time period, and calculate the average and standard deviation of the flow difference within 30 days. With a three-sigma rule in the case of very general distribution, at least 88.8% of the data would

be included in the range of three standard deviations. Determined the unit flow rate of more than three standard deviations as a moderate abnormal flow rate, and determined that the unit flow rate exceeding five standard deviations by the empirical rule was a high abnormal flow rate. Through the above two methods, can quickly find the time when abnormal traffic occurs.

- **Attack identification**

In the part of the attack identification. For attack methods with fixed features, use the RNN model for identification. This type of attack can achieve a high recognition rate because it has very obvious features that are very different from each other and does not require a very complicated deep learning model. In addition, for the attack mode without fixed attack characteristics, use the NSL_KDD data set and establish the model to train the data set through three different ways of the neural network, which are three different methods: RNN, LSTM and GRU [32]. The ability to identify the attack is evaluated by the loss value, accuracy value and training time of the three methods.

Finally, the results of the anomaly detection and attack identification are stored in MySQL as a visual data display.

Algorithm 1 Attack identification process algorithm

Input: The result of NetFlow data after model prediction , $Result_i$; The numbers of NetFlow data, N

Output: The attack identification result , $Attack_i$;

```

1: for  $i = 0; i \leq N; i++$  do
2:   if  $Result_i = 0$  is true then
3:      $Attack_i = \text{CodeRed};$ 
4:   else if  $Result_i = 1$  is true
5:      $Attack_i = \text{Nimda}$ 
6:   else
7:      $Attack_i = \text{Worm}$ 
8:   end if return  $Result_i$ ;
9: end for

```

3.2.5 Data Visualization

Part of the visualization of the data, connect to the MySQL database through PHP to get the JSON format data, then process and load the data through JQuery and Ajax, and finally the loaded data is presented through the customized chart of ECharts. Through the above process, can easily visualize the front and rear data. Finally, the webpage is combined with the Bootstrap framework to achieve Responsive Web Design that is compatible with various devices.

3.3 System Implementation

In this work, set up four virtual machines under one physical host running ESXi, one as the master node and the other three as the storage nodes of the Ceph environment. Establish a complete Ceph storage environment through this cluster. Create a virtual machine as a data operation node under another physical host running ESXi and having a GPU. NetFlow data is used by mounting the Ceph storage cluster. Figure 3.1 shows the version of the Ceph cluster. Figure 3.2 is the software specification for the GPU machine.

TABLE 3.1: Ceph storage cluster

	Master	ceph-osd1	ceph-osd2	ceph-osd3
Ubuntu	16.04	16.04	16.04	16.04
Ceph version	13.2.5	13.2.5	13.2.5	13.2.5
Python	3.6	3.6	3.6	3.6

TABLE 3.2: Software Specifications

	Version
Ubuntu	16.04
Ceph	13.2.5
Anaconda	conda 4.6.11
Python	3.6
CUDA	10.0.13
CuDNN	7.4.2
TensorFlow	1.13.1
Keras	2.2.4
Apache	2.4.18 (Ubuntu)
MySQL	Ver 14.14 Distrib 5.7.26
PHP	7.0.33-0ubuntu0.16.04.4
phpMyAdmin	4.5.4.1deb2ubuntu2.1

ESXi provides a remote management monitoring interface, such as the CPU, memory, and hard disk usage of the physical host. And the use of virtual machines and events are recorded. Figure 3.8 is the remote management interface of ESXi. As shown in Figure 3.9, the virtual machine monitoring interface can see the performance of the virtual machine.

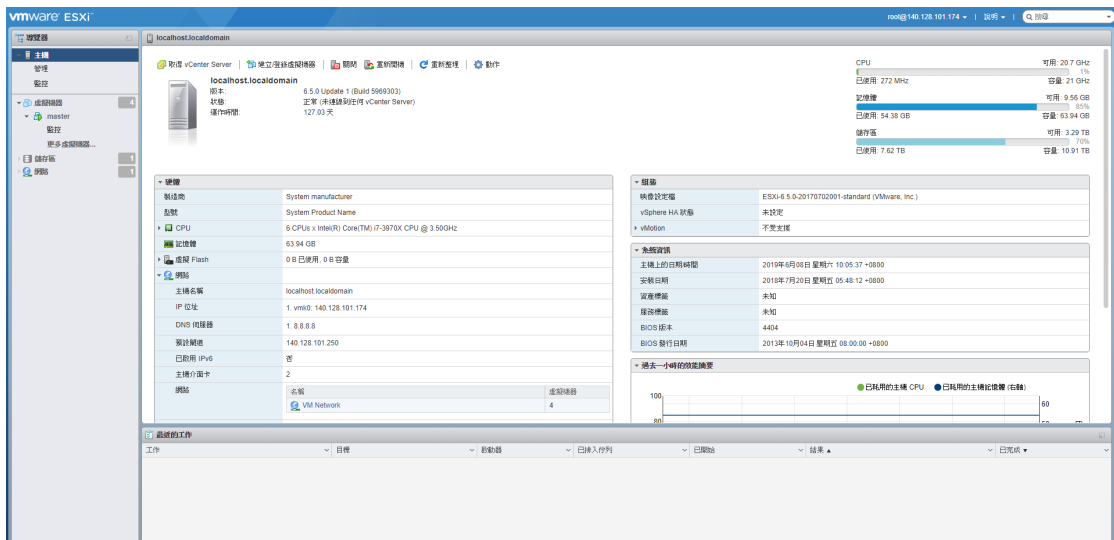


FIGURE 3.8: Remote management interface of ESXi

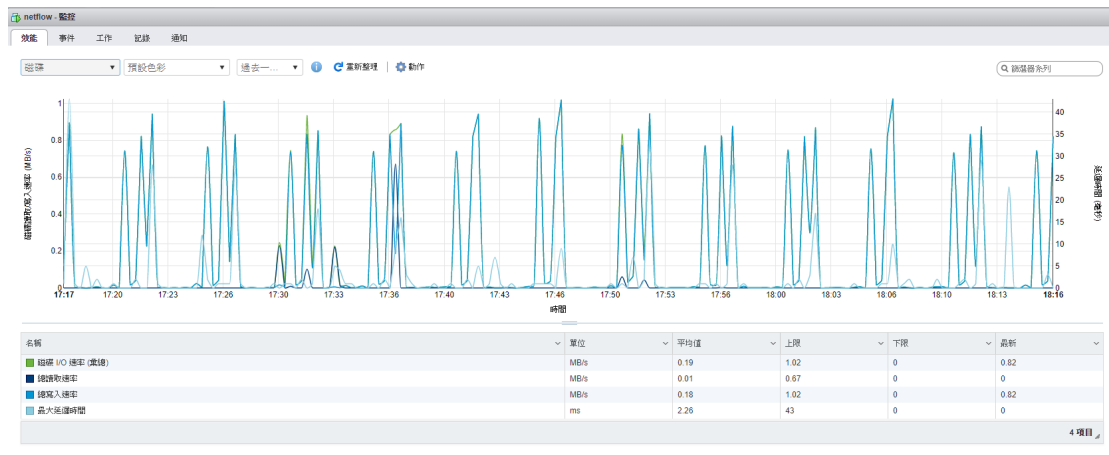


FIGURE 3.9: Virtual machine monitoring interface

Ceph mgr provides monitoring of Ceph clusters, such as cluster health, current usage, OSD usage, and I/O-related information. As shown in Figure 3.10, the usage of the OSD can be monitored through the Ceph dashboard.

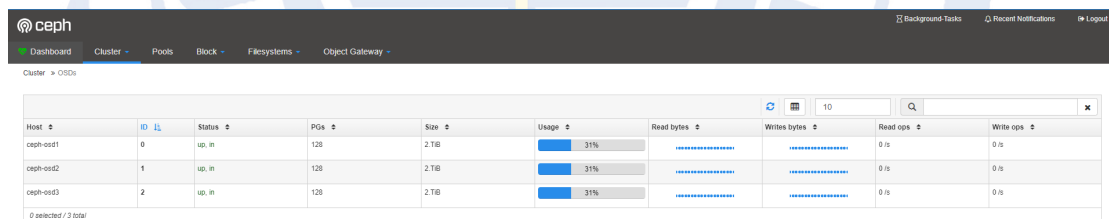
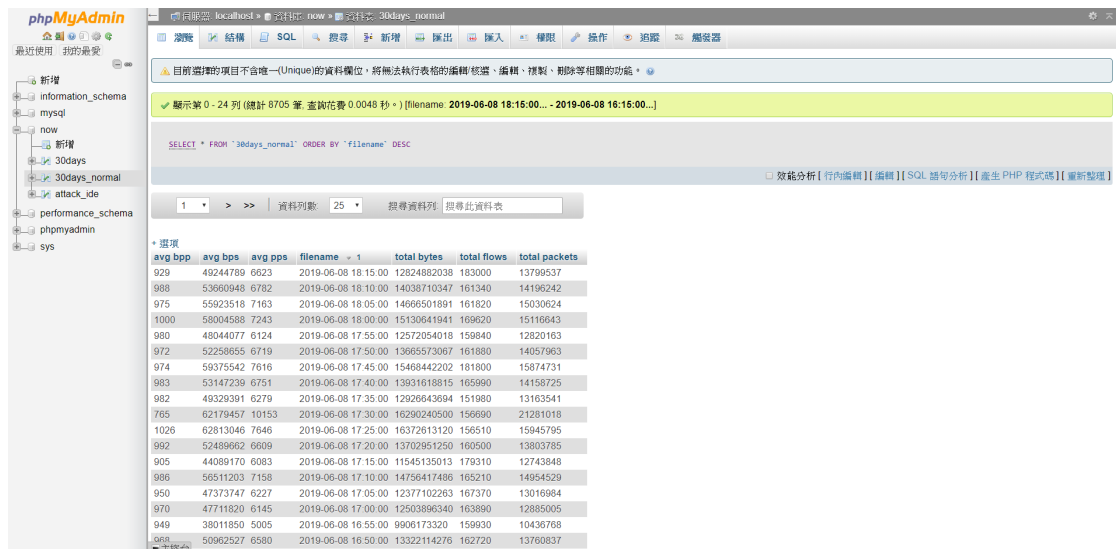


FIGURE 3.10: Ceph osd monitored

Establish a MySQL relational database as a data store for real-time data analysis results. When the new real-time data is entered, the database data would be updated immediately, and the data repository would be stored for only 30 days of data to visualize the data, so as to avoid excessive data delay and storage space exceeding the upper limit. The updated data and historical data are stored in the Figure 3.11 database.



目前選擇的項目不含唯一(Unique)的資料欄位，將無法執行表格的編輯/修護、複製、刪除等相關的功能。

顯示第 0 - 24 列 (總計 8705 筆 查詢花費 0.0048 秒。)| [filename: 2019-06-08 18:15:00... - 2019-06-08 18:15:00...]

```
SELECT * FROM `30days_normal` ORDER BY `filename` DESC
```

效能分析 [行內編輯] [編輯] [SQL 語句分析] [產生 PHP 程式碼] [重新整理]

id	avg bpp	avg bps	avg pps	filename	total bytes	total flows	total packets
929	49244789	6623	2019-06-08 18:15:00	12824882038	183000	13799637	
988	53660948	6782	2019-06-08 18:10:00	14038710347	161340	14196242	
975	55923518	7183	2019-06-08 18:05:00	14666501891	161820	15030624	
1000	58004588	7243	2019-06-08 18:00:00	15130641941	169620	15116643	
980	48044077	6124	2019-06-08 17:55:00	12572054018	159840	12820163	
972	52258655	6719	2019-06-08 17:50:00	13665573067	161180	14057963	
974	59375542	7616	2019-06-08 17:45:00	15468442202	181800	15874731	
983	53147239	6751	2019-06-08 17:40:00	13931618815	165960	14158725	
982	49329391	6279	2019-06-08 17:35:00	12926643694	151980	13163541	
765	62179457	10153	2019-06-08 17:30:00	16290240500	156690	21281018	
1026	62813046	7646	2019-06-08 17:25:00	16372613120	156510	15945795	
992	52488662	6609	2019-06-08 17:20:00	13702951250	160500	13803785	
905	44089170	6083	2019-06-08 17:15:00	11545135013	179310	12743848	
986	56511203	7158	2019-06-08 17:10:00	14756417486	165210	14954529	
950	47373747	6227	2019-06-08 17:05:00	12377102263	167370	13016984	
970	47711820	6145	2019-06-08 17:00:00	12503896340	163890	12885005	
949	38011850	5005	2019-06-08 16:55:00	9906173320	159930	10436768	
068	50962527	6580	2019-06-08 16:50:00	13322114276	162720	13760837	

FIGURE 3.11: MySQL database

Chapter 4

Experimental Results

4.1 Experimental Environment

This experiment uses two physical machines equipped with ESXi, the specifications are shown in Table 4.1. One physical host is used to simulate the Ceph distributed storage environment, and the other physical host is mainly used for deep learning experiments. In the physical host that simulates the Ceph storage environment, it includes a Ceph cluster consisting of four virtual machines, one of which is the Ceph master node, and the other three include Ceph Monitor and Ceph OSD. The composition of the environment is show in Table 4.2 In the physical host used for deep learning experiments, this work built a virtual machine running an RTX 2080 ti display card to accelerate the efficiency of deep learning experiments through the GPU. The composition of the environment is show in Table 4.3

TABLE 4.1: Computing environment

Name	CPU	RAM	Disk	OS
Ceph ESXi	6 CPUs x Intel(R) Core(TM) i7-3970X 3.50GHz	64G	12T	ESXi-6.5.0
DL ESXi	10 CPUs x Intel(R) Core(TM) i9-9900X 3.50GHz	128G	7T	ESXi-6.5.0

TABLE 4.2: Ceph virtual machine environment

Name	CPU	RAM	Disk	OS
master	6 vCPUs	40G	1.5T	Ubuntu 16.04
ceph-osd1	2 vCPUs	4G	2T	Ubuntu 16.04
ceph-osd2	2 vCPUs	4G	2T	Ubuntu 16.04
ceph-osd3	2 vCPUs	4G	2T	Ubuntu 16.04

TABLE 4.3: Deep learning virtual machine environment

Name	CPU	RAM	Disk	OS	GPU
ubuntu	4 vCPUs	20G	1T	Ubuntu 16.04	RTX 2080ti

4.2 Historical Flow Changes

For the application of historical data, this work collect the collected real-time data for processing statistics. Calculate the total number of sent traffic for each time period, the total used traffic size and the total transport packet size, and the average number of items. Then charted the cumulative calculations and found that the data would change periodically under normal conditions. Figure 4.1 below shows the total flow change graph for 30 consecutive days.

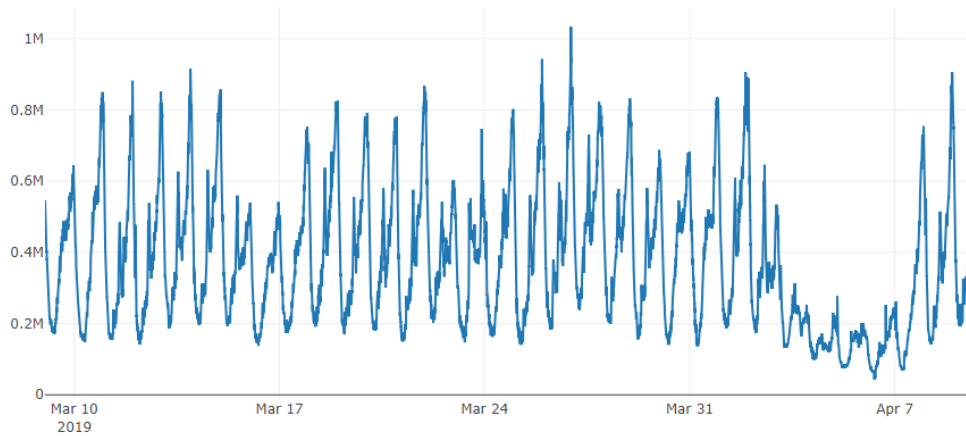


FIGURE 4.1: Total flow change graph for 30 consecutive days

Through the historical data, can find the difference in traffic between each time period and the previous time period. By plotting the difference in flow over time, it can be seen that abnormal flow changes occur at certain times. The point

in time when these traffic differences occur is what should pay attention to. Figure 4.2 below shows the flow difference variation for 30 consecutive days.

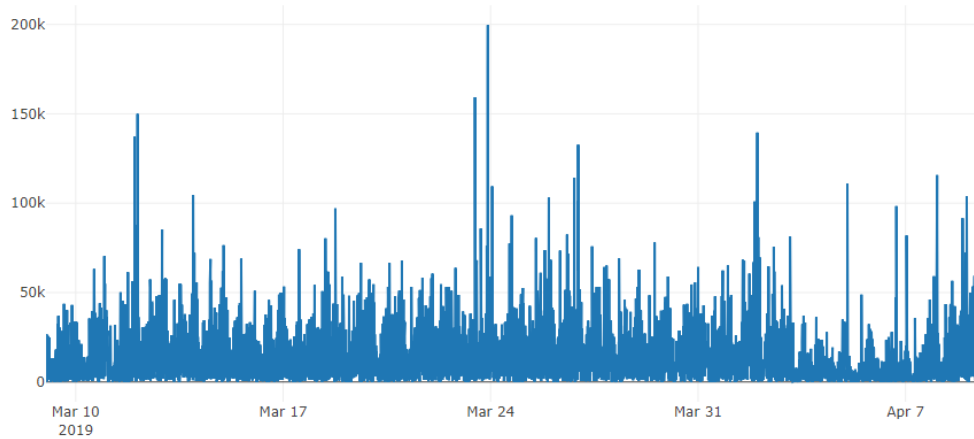


FIGURE 4.2: Flow difference variation for 30 consecutive days

4.3 Abnormal Analysis Result

From the previous experiments, it can be found that under normal conditions, the change in total flow would be consistent with the periodicity, and the change in flow difference would meet the specific interval. Through the above two points can find the time points that do not meet the periodic flow changes and the large changes in the flow difference, and mark these time points for the manager to conduct subsequent investigations. Check whether the total number of traffic at the current time is abnormal by comparing the total number of traffic at the same time as the previous day. If the total traffic volume of the current time is greater than 1.5 times the total traffic volume of the previous day, the current time point and total traffic volume are marked. In the abnormal flow difference part, this work can find the average and standard deviation for 30 consecutive days. According to the standard deviation rule of three times, can quickly find outliers with more than three standard deviations and mark them as medium flow difference changes, greater than five. The standard deviation is marked as a high flow change. Figure 4.3 below is the result of the abnormal analysis.

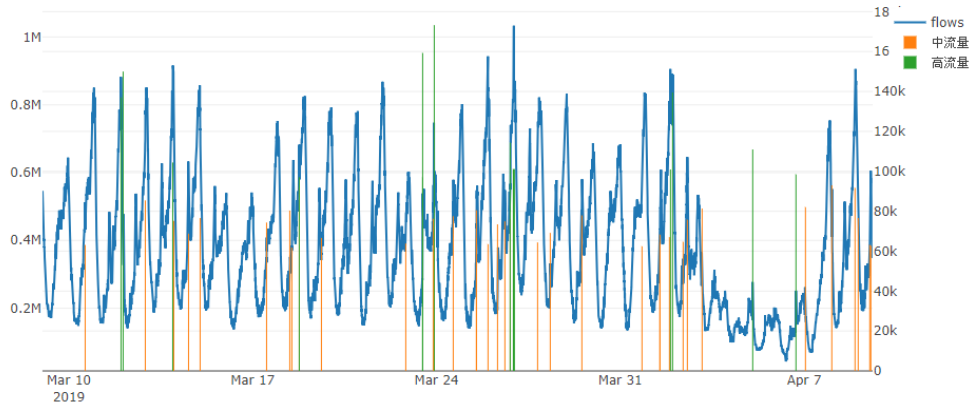


FIGURE 4.3: Abnormal analysis result

4.4 Attack Identification with Fixed Features

Using Python to classify data with fixed features, possibly attacks, and note the types of possible attacks. The accumulated data is integrated as a training set for attack identification as shown in Figure 4.4. And because these types of attacks have fixed features, this work use Keras to build a simple RNN deep learning model as shown in Figure 4.5, and through Mean-Square Error (MSE) [33] as the evaluation criteria for training loss values. The set is trained and verified as shown in Figure 4.6. Table 4.4 can be seen that for such attacks with fixed features, deep learning can easily identify these attacks with fixed features.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_{true}^{(i)} - y_{pred}^{(i)})^2 \quad (4.1)$$

	3	4	5	6	7	8	9	10	11	12	13
0	203.205.138.238	80.0	2.83	144.0	3.0	25.0	42.0	2.0	10.59.2.14	59370.0	CodeRed attack
1	199.93.56.125	80.0	0.00	642.0	8.0	25.0	46.0	2.0	10.30.2.12	55450.0	Nimda
2	5.136.241.233	445.0	0.00	52.0	1.0	25.0	46.0	2.0	10.30.3.119	61678.0	Worm attack
3	13.113.157.96	80.0	3.24	144.0	3.0	25.0	46.0	2.0	10.72.3.48	60774.0	CodeRed attack
4	119.46.206.250	80.0	9.01	144.0	3.0	25.0	45.0	2.0	10.67.2.24	59350.0	CodeRed attack
5	119.46.206.251	80.0	9.00	144.0	3.0	25.0	45.0	2.0	10.67.2.24	59377.0	CodeRed attack
6	213.147.126.18	80.0	1.43	144.0	3.0	25.0	41.0	2.0	10.6.0.14	1989.0	CodeRed attack
7	104.250.139.218	80.0	1.01	144.0	3.0	25.0	46.0	2.0	10.30.5.32	53033.0	CodeRed attack
8	104.250.139.218	80.0	1.01	144.0	3.0	25.0	46.0	2.0	10.30.5.32	53031.0	CodeRed attack
9	104.250.139.218	80.0	1.01	144.0	3.0	25.0	46.0	2.0	10.30.5.32	53022.0	CodeRed attack
10	104.250.139.218	80.0	1.01	144.0	3.0	25.0	46.0	2.0	10.30.5.32	53030.0	CodeRed attack

FIGURE 4.4: Training set for attack identification

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, None, 16)	400
simple_rnn_2 (SimpleRNN)	(None, 16)	528
dense_1 (Dense)	(None, 3)	51
Total params: 979		
Trainable params: 979		
Non-trainable params: 0		

FIGURE 4.5: RNN model architecture

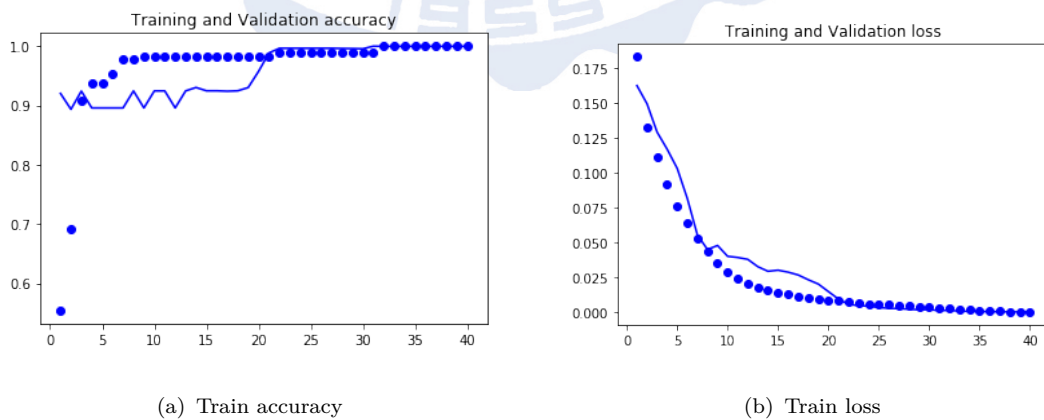


FIGURE 4.6: Attack Classification Result

TABLE 4.4: 1-layer RNN model training result

Validation Loss	Validation Accuracy	Training Time
0.0003	1.0	10.03 sec

4.5 Attack Identification without Fixed Features

Because there is no recognized deep learning model criterion in processing data, the quality of the model needs to be compared through different models to find the most suitable result. In the training attack identification model, using three-quarters of the NSL-KDD data set as the training set of the model, and use a quarter of the NSL-KDD data set as the verification set. Uniformly set `batch_size` to 128, `epochs` to 100, and join `early_stopping` to monitor `val_loss` value to reduce `over_fitting`. Add `ReduceRonPlateau` to monitor `acc` to automatically adjust `learning_rate` to improve learning accuracy. Finally, use different methods to compare differences between layers to evaluate differences between models.

4.5.1 RNN Model

In the experiment of RNN model effectiveness evaluation, tested the training results of the RNN model using the 1st to 4th layers respectively. Figure 4.7 is the training model of the 1-Layer RNN. Figure 4.8 is a graph of the loss value and accuracy of the 1-layer RNN training. It can see that the training has stopped after running 100 epochs, and there is no `over_fitting`. 1-layer RNN results such as Table 4.5

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 16)	912
dropout_1 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 997		
Trainable params: 997		
Non-trainable params: 0		

FIGURE 4.7: 1-Layer RNN model

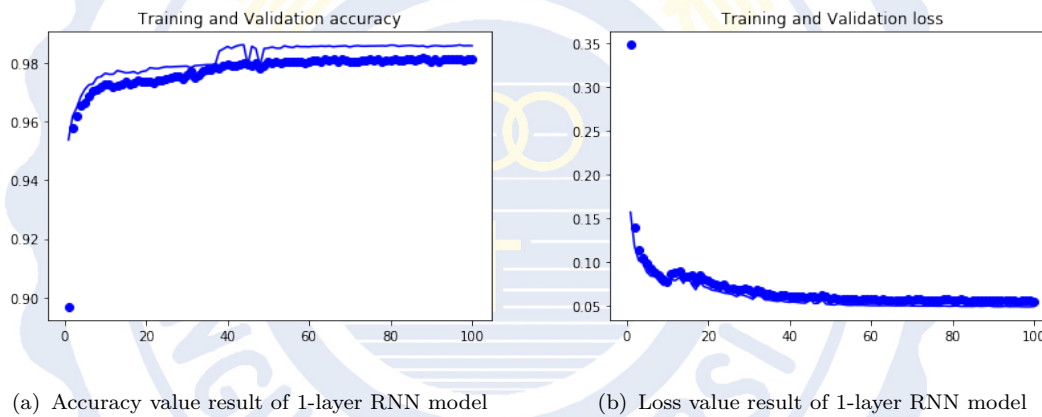


FIGURE 4.8: 1-layer RNN accuracy and loss

TABLE 4.5: 1-layer RNN model training result

Validation Loss	Validation Accuracy	Training Time
0.0485	0.9857	524.50 sec

Figure 4.9 is the training model of the 2-Layer RNN. The changes in Accuracy and Loss of the training shown in Figures 4.10 can be seen to stop after running 100 epochs, without over_fitting. The training result is shown in Table 4.6. Found that adding a layer of RNN can improve the Accuracy of the model and reduce the Loss value for better performance.

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, None, 16)	912
dropout_1 (Dropout)	(None, None, 16)	0
simple_rnn_2 (SimpleRNN)	(None, 16)	528
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 1,525		
Trainable params: 1,525		
Non-trainable params: 0		

FIGURE 4.9: 2-layer RNN model

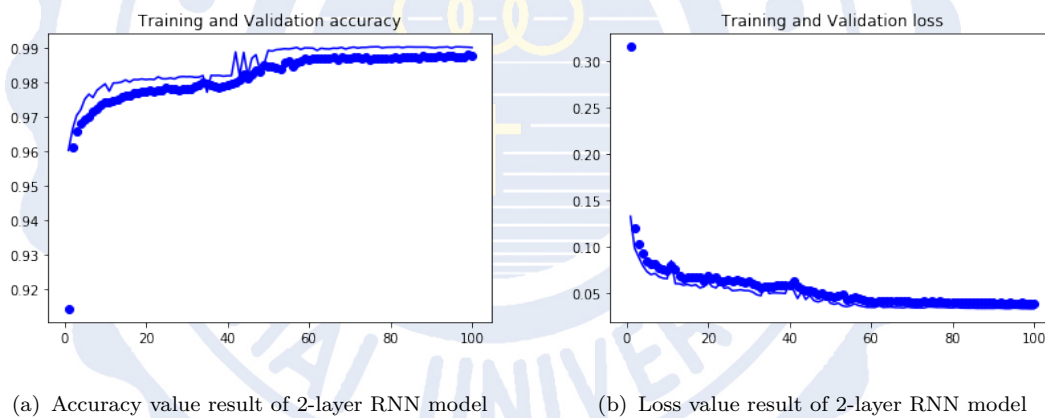


FIGURE 4.10: 2-layer RNN accuracy and loss

TABLE 4.6: 2-layer RNN model training result

Validation Loss	Validation Accuracy	Training Time
0.0333	0.9901	686.55 sec

Train the 2-layer RNN model with the addition of a 1-layer RNN. The architecture of the 3-layer RNN model is shown in Figure 4.11. In the changes of Accuracy and Loss trained in the 3-layer RNN model as shown in Figure 4.12, no over_fitting occurs. The training results are shown in Table 4.7. The 3-layer RNN only increases the value of Accuracy by 0.002, while the loss value is only reduced by 0.009, but it increases by about 1.21 times in the training time than the 2-layer RNN.

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, None, 16)	912
dropout_1 (Dropout)	(None, None, 16)	0
simple_rnn_2 (SimpleRNN)	(None, None, 16)	528
dropout_2 (Dropout)	(None, None, 16)	0
simple_rnn_3 (SimpleRNN)	(None, 16)	528
dropout_3 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 2,053		
Trainable params: 2,053		
Non-trainable params: 0		

FIGURE 4.11: 3-layer RNN model

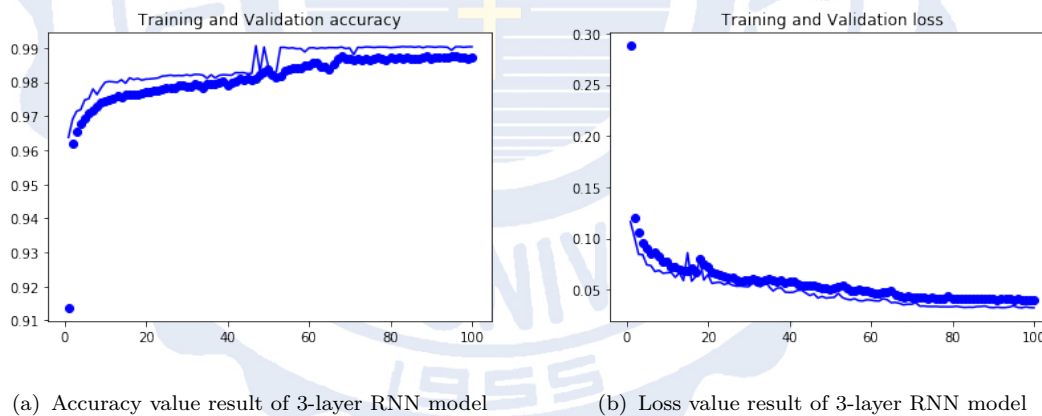


FIGURE 4.12: 3-layer RNN accuracy and loss

TABLE 4.7: 3-layer RNN model training result

Validation Loss	Validation Accuracy	Training Time
0.0324	0.9903	830.77 sec

Finally, use the 4-layer RNN model for training. The 4-layer RNN model architecture is shown in Figure 4.13. The changes in Accuracy and Loss in training are shown in Figure 4.14. The result of the training is shown in Table 4.8. Can find that the 4-layer RNN model takes longer to train than the 2-layer RNN and the 3-layer RNN, but the performance in Accuracy and Loss is worse than that of the 2-layer RNN and the 3-layer RNN.

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, None, 16)	912
dropout_1 (Dropout)	(None, None, 16)	0
simple_rnn_2 (SimpleRNN)	(None, None, 16)	528
dropout_2 (Dropout)	(None, None, 16)	0
simple_rnn_3 (SimpleRNN)	(None, None, 16)	528
dropout_3 (Dropout)	(None, None, 16)	0
simple_rnn_4 (SimpleRNN)	(None, 16)	528
dropout_4 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 2,581		
Trainable params: 2,581		
Non-trainable params: 0		

FIGURE 4.13: 4-layer RNN model

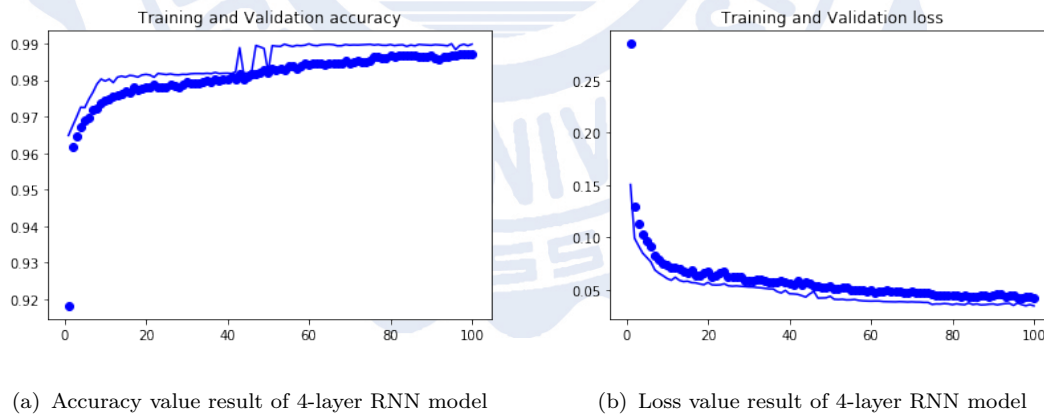


FIGURE 4.14: 4-layer RNN accuracy and loss

TABLE 4.8: 4-layer RNN model training result

Validation Loss	Validation Accuracy	Training Time
0.0347	0.9898	967.11 sec

According to the experimental results of different layers of RNN, can find that the appropriate increase of the number of training layers can improve the performance of the training model, but excessively increasing the number of training

layers would cause the performance of the model to decline. As far as the experimental results are concerned, the 2-layer RNN model has a relatively better performance.

4.5.2 LSTM Model

In the next experiment, use the LSTM training model to perform the experiment. The results of the LSTM training model for different layers are also compared for Accuracy, Loss values, and training time.

Figure 4.15 is an architectural diagram of the 1-layer LSTM model. Figure 4.16 shows the Accuracy and Loss change graph of the training set and the verification set. Training results as shown in Table 4.9

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 16)	3648
dropout_1 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 3,733		
Trainable params: 3,733		
Non-trainable params: 0		

FIGURE 4.15: 1-layer LSTM model

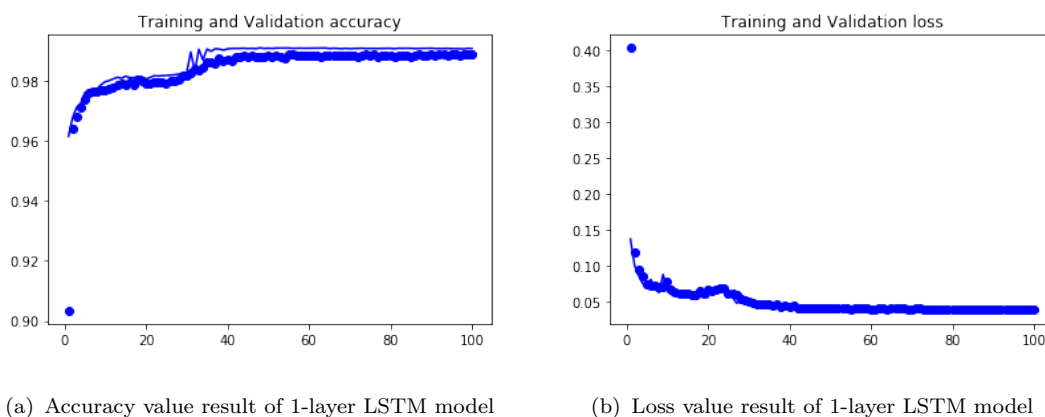


FIGURE 4.16: 1-layer LSTM accuracy and loss

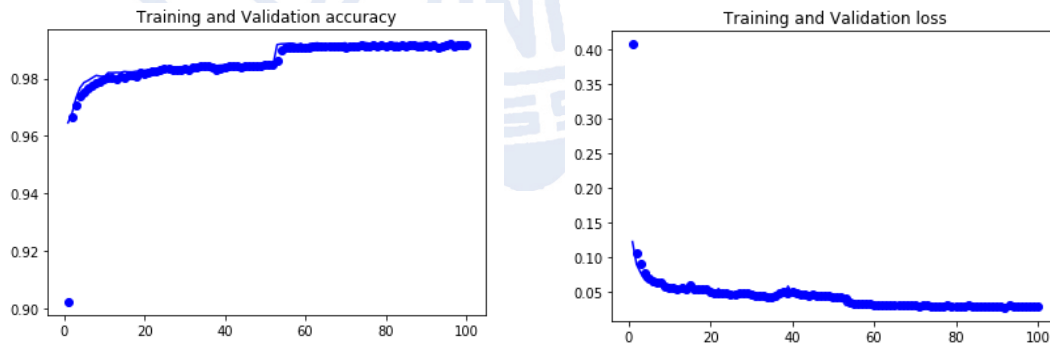
TABLE 4.9: 1-layer LSTM model training result

Validation Loss	Validation Accuracy	Training Time
0.0367	0.9907	761.50 sec

Figure 4.17 is an architectural diagram of the 2-layer LSTM model. Figure 4.18 shows the Accuracy and Loss change graph of the training set and the verification set. Training results as shown in Table 4.10

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 16)	3648
dropout_1 (Dropout)	(None, None, 16)	0
lstm_2 (LSTM)	(None, 16)	2112
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 5,845		
Trainable params: 5,845		
Non-trainable params: 0		

FIGURE 4.17: 2-layer LSTM model



(a) Accuracy value result of 2-layer LSTM model

(b) Loss value result of 2-layer LSTM model

FIGURE 4.18: 2-layer LSTM accuracy and loss

TABLE 4.10: 2-layer LSTM model training result

Validation Loss	Validation Accuracy	Training Time
0.0267	0.9924	1108.47 sec

Figure 4.19 is an architectural diagram of the 3-layer LSTM model. Figure 4.20 shows the Accuracy and Loss change graph of the training set and the verification set. Training results as shown in Table 4.11

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 16)	3648
dropout_1 (Dropout)	(None, None, 16)	0
lstm_2 (LSTM)	(None, None, 16)	2112
dropout_2 (Dropout)	(None, None, 16)	0
lstm_3 (LSTM)	(None, 16)	2112
dropout_3 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 7,957		
Trainable params: 7,957		
Non-trainable params: 0		

FIGURE 4.19: 3-layer LSTM model

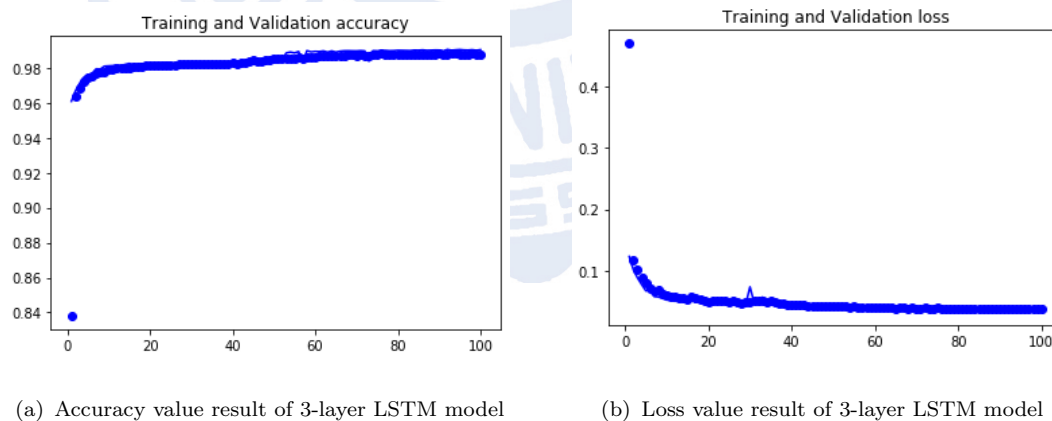


FIGURE 4.20: 3-layer LSTM accuracy and loss

TABLE 4.11: 3-layer LSTM model training result

Validation Loss	Validation Accuracy	Training Time
0.0340	0.9904	1377.52 sec

Figure 4.21 is an architectural diagram of the 3-layer LSTM model. Figure 4.22 shows the Accuracy and Loss change graph of the training set and the verification set. Training results as shown in Table 4.12

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 16)	3648
dropout_1 (Dropout)	(None, None, 16)	0
lstm_2 (LSTM)	(None, None, 16)	2112
dropout_2 (Dropout)	(None, None, 16)	0
lstm_3 (LSTM)	(None, None, 16)	2112
dropout_3 (Dropout)	(None, None, 16)	0
lstm_4 (LSTM)	(None, 16)	2112
dropout_4 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 10,069		
Trainable params: 10,069		
Non-trainable params: 0		

FIGURE 4.21: 4-layer LSTM model

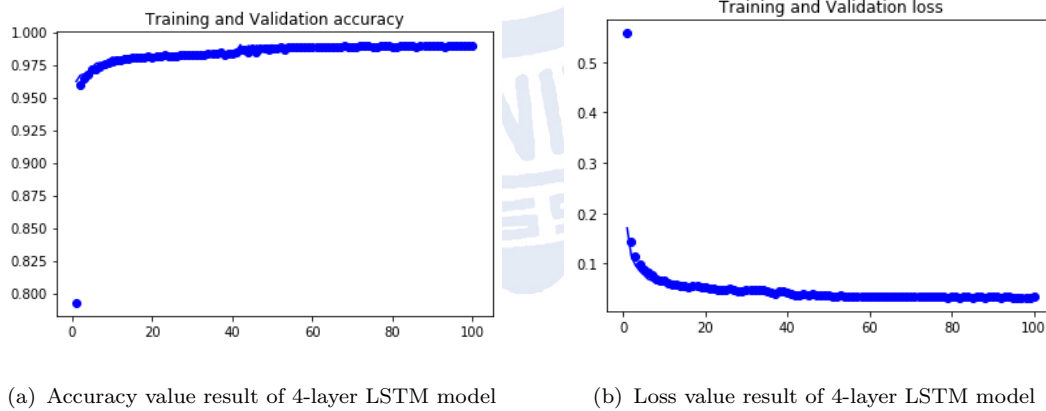


FIGURE 4.22: 4-layer LSTM accuracy and loss

TABLE 4.12: 4-layer LSTM model training result

Validation Loss	Validation Accuracy	Training Time
0.0323	0.9901	1643.36 sec

From the experiments of the LSTM model, it can be found that the best performance in the 2-layer LSTM model architecture is degraded from the 3-layer LSTM model Accuracy.

In addition, this work add CuDNNLSTM specifically for use in the GPU environment to compare with the original LSTM model. Figure 4.23 is an architectural diagram of the 2-layer CuDNNLSTM model. The comparison between the training results and the 2-layer LSTM is shown in Table 4.13. It can be seen that although the performance of Accuracy and Loss values is reduced, it is 0.32 times faster in training time.

Layer (type)	Output Shape	Param #
cu_dnnlstm_1 (CuDNNLSTM)	(None, 1, 16)	3712
dropout_1 (Dropout)	(None, 1, 16)	0
cu_dnnlstm_2 (CuDNNLSTM)	(None, 16)	2176
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 5,973		
Trainable params: 5,973		
Non-trainable params: 0		

FIGURE 4.23: 2-layer CuDNNLSTM model

TABLE 4.13: Comparison of 2-layer LSTM and 2-layer CuDNNLSTM model

Model	Validation Loss	Validation Accuracy	Training Time
LSTM	0.0267	0.9924	1108.47 sec
CuDNNLSTM	0.0332	0.9906	761.66 sec

In the 3-layer and 4-layer environments, also used CuDNNLSTM for experiments. The comparison between the 3-layer LSTM and CuDNNLSTM is shown in Table 4.14. The comparison of the 4-layer LSTM with CuDNNLSTM is shown in Table 4.15. It can be found that the 3-layer and 4-layer training models are better than the 2-layer training models when using CuDNNLSTM.

TABLE 4.14: Comparison of 3-layer LSTM and 3-layer CuDNNLSTM model

Model	Validation Loss	Validation Accuracy	Training Time
LSTM	0.0340	0.9904	1377.52 sec
CuDNNLSTM	0.0318	0.9912	918.11 sec

TABLE 4.15: Comparison of 4-layer LSTM and 4-layer CuDNNLSTM model

Model	Validation Loss	Validation Accuracy	Training Time
LSTM	0.0323	0.9901	1643.36 sec
CuDNNLSTM	0.0306	0.9911	1063.75 sec

4.5.3 GRU Model

In the next experiment, use the LSTM training model to perform the experiment. The results of the LSTM training model for different layers are also compared for Accuracy, Loss values, and training time.

Figure 4.24 is a model architecture diagram of a layer 1 GRU. Figure 4.25 show the Accuracy and Loss changes for the training and validation sets. It can be found that the training stops at epoch 56 because the GRU converges faster than LSTM and triggers `learning_rate_reduction` frequently, which makes it easier to trigger `early_stopping`. The training results of the 1-layer GRU model are shown in Table 4.16.

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 16)	2736
dropout_1 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
Total params: 2,821		
Trainable params: 2,821		
Non-trainable params: 0		

FIGURE 4.24: 1-layer GRU model

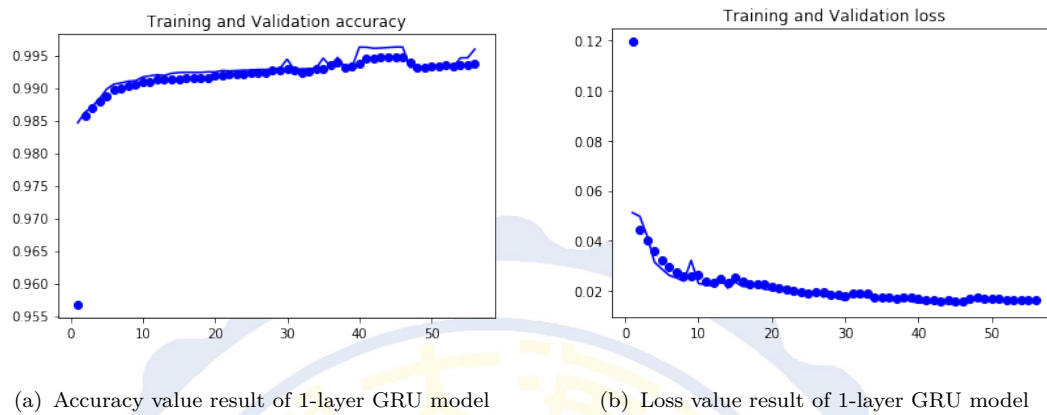


FIGURE 4.25: 1-layer GRU accuracy and loss

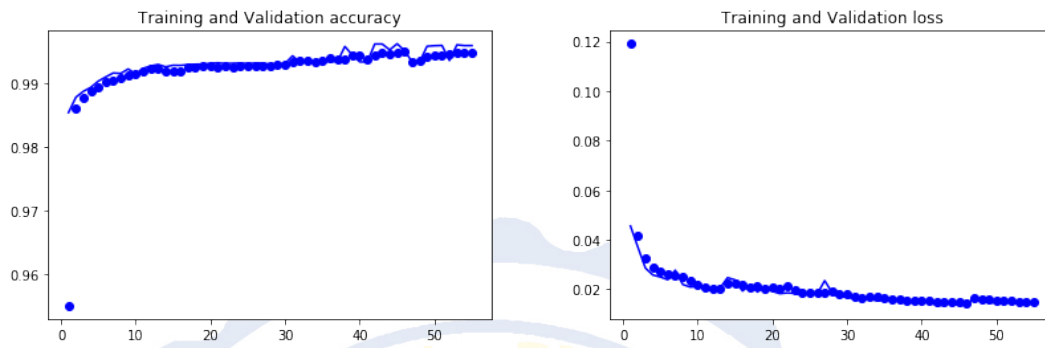
TABLE 4.16: 1-layer GRU model training result

Validation Loss	Validation Accuracy	Training Time
0.0153	0.9898	405.74 sec

Figure 4.26 is a model architecture diagram of a layer 2 GRU. Figure 4.27 show the Accuracy and Loss changes for the training and validation sets. Also trigger early_stopping and stop training at epoch 55. The result of the training is like Table 4.17

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, None, 16)	2736
dropout_1 (Dropout)	(None, None, 16)	0
gru_2 (GRU)	(None, 16)	1584
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
=====		
Total params: 4,405		
Trainable params: 4,405		
Non-trainable params: 0		

FIGURE 4.26: 2-layer GRU model



(a) Accuracy value result of 2-layer GRU model (b) Loss value result of 2-layer GRU model

FIGURE 4.27: 2-layer GRU accuracy and loss

TABLE 4.17: 2-layer GRU model training result

Validation Loss	Validation Accuracy	Training Time
0.0140	0.9896	561.70 sec

Figure 4.28 is a model architecture diagram of a 3-layer GRU. Figure 4.29 show the Accuracy and Loss changes for the training and validation sets. Also trigger early_stopping and stop training at epoch 65. The result of the training is like Table 4.18

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, None, 16)	2736
dropout_1 (Dropout)	(None, None, 16)	0
gru_2 (GRU)	(None, None, 16)	1584
dropout_2 (Dropout)	(None, None, 16)	0
gru_3 (GRU)	(None, 16)	1584
dropout_3 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
=====		
Total params: 5,989		
Trainable params: 5,989		
Non-trainable params: 0		

FIGURE 4.28: 3-layer GRU model

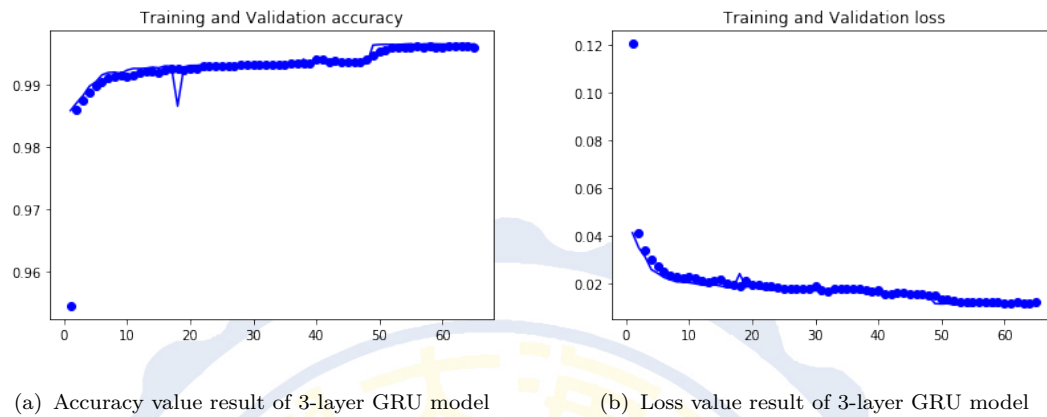


FIGURE 4.29: 3-layer GRU accuracy and loss

TABLE 4.18: 3-layer GRU model training result

Validation Loss	Validation Accuracy	Training Time
0.0105	0.9912	828.30 sec

Figure 4.30 is a model architecture diagram of a 3-layer GRU. Figure 4.31 show the Accuracy and Loss changes for the training and validation sets. And did not trigger early_stopping. The result of the training is like Table 4.19

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, None, 16)	2736
dropout_1 (Dropout)	(None, None, 16)	0
gru_2 (GRU)	(None, None, 16)	1584
dropout_2 (Dropout)	(None, None, 16)	0
gru_3 (GRU)	(None, None, 16)	1584
dropout_3 (Dropout)	(None, None, 16)	0
gru_4 (GRU)	(None, 16)	1584
dropout_4 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 5)	85
=====		
Total params: 7,573		
Trainable params: 7,573		
Non-trainable params: 0		

FIGURE 4.30: 4-layer GRU model

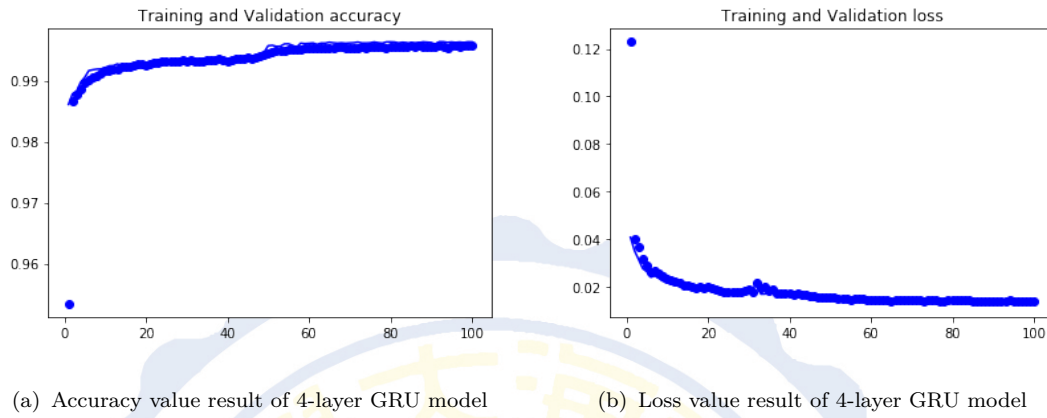


FIGURE 4.31: 3-layer GRU accuracy and loss

TABLE 4.19: 4-layer GRU model training result

Validation Loss	Validation Accuracy	Training Time
0.0126	0.9908	1482.76 sec

In the training results of the GRU model, can find the highest accuracy in the 3-layer GRU training model. The following is a comparison of the results of the LSTM model with different layers and the GRU model. From Table 4.20,4.21,4.22 and 4.23, the training speed of the GRU in each epoch is similar to the speed of the LSTM. And the Accuracy indicator of the verification set differs from the actual verified Accuracy during training. In addition, it is worth mentioning that although the convergence speed of GRU is fast, it is easy to trigger early_stopping and the result of each training is quite different. So this work still choose LSTM in use.

TABLE 4.20: Comparison of 1-layer LSTM and 1-layer GRU model

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
LSTM	0.0367	0.9907	761.50 sec	7.61 sec
GRU	0.0153	0.9898	405.74 sec	7.24 sec

TABLE 4.21: Comparison of 2-layer LSTM and 2-layer GRU model

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
LSTM	0.0267	0.9924	1108.47 sec	11.08 sec
GRU	0.0140	0.9896	561.70 sec	10.21 sec

TABLE 4.22: Comparison of 3-layer LSTM and 3-layer GRU model

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
LSTM	0.0340	0.9904	1377.52 sec	13.77 sec
GRU	0.0105	0.9912	828.30 sec	12.74 sec

TABLE 4.23: Comparison of 4-layer LSTM and 4-layer GRU model

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
LSTM	0.0323	0.9901	1643.36 sec	16.43 sec
GRU	0.0126	0.9908	1482.76 sec	14.82 sec

4.5.4 CNN Model

Next, use CNN to build a model for experimentation. Generally, CNN is mainly used to process image recognition. This work convert the original data into two-dimensional image data, and compare the results of different CNN models through feature extraction of CNN.

Figure 4.32 is a 1 to 4 layer CNN model architecture diagram. Figure 4.33 shows the Accuracy change graph of the training set and the verification set.

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	256
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
flatten_1 (Flatten)	(None, 1280)	0
dense_1 (Dense)	(None, 128)	163968
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645
Total params: 164,869		
Trainable params: 164,869		
Non-trainable params: 0		

(a) 1-layer CNN model

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	256
conv1d_2 (Conv1D)	(None, 40, 64)	12352
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
flatten_1 (Flatten)	(None, 1280)	0
dense_1 (Dense)	(None, 128)	163968
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645
Total params: 177,221		
Trainable params: 177,221		
Non-trainable params: 0		

(b) 2-layer CNN model

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	256
conv1d_2 (Conv1D)	(None, 40, 64)	12352
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
conv1d_3 (Conv1D)	(None, 20, 128)	24704
conv1d_4 (Conv1D)	(None, 20, 128)	49280
max_pooling1d_2 (MaxPooling1D)	(None, 10, 128)	0
flatten_1 (Flatten)	(None, 2560)	0
dense_1 (Dense)	(None, 128)	327808
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645
Total params: 365,765		
Trainable params: 365,765		
Non-trainable params: 0		

(c) 3-layer CNN model

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	256
conv1d_2 (Conv1D)	(None, 40, 64)	12352
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
conv1d_3 (Conv1D)	(None, 20, 128)	24704
conv1d_4 (Conv1D)	(None, 20, 128)	49280
max_pooling1d_2 (MaxPooling1D)	(None, 10, 128)	0
flatten_1 (Flatten)	(None, 1280)	0
dense_1 (Dense)	(None, 128)	163968
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645
Total params: 251,205		
Trainable params: 251,205		
Non-trainable params: 0		

(d) 3-layer CNN model

FIGURE 4.32: CNN model

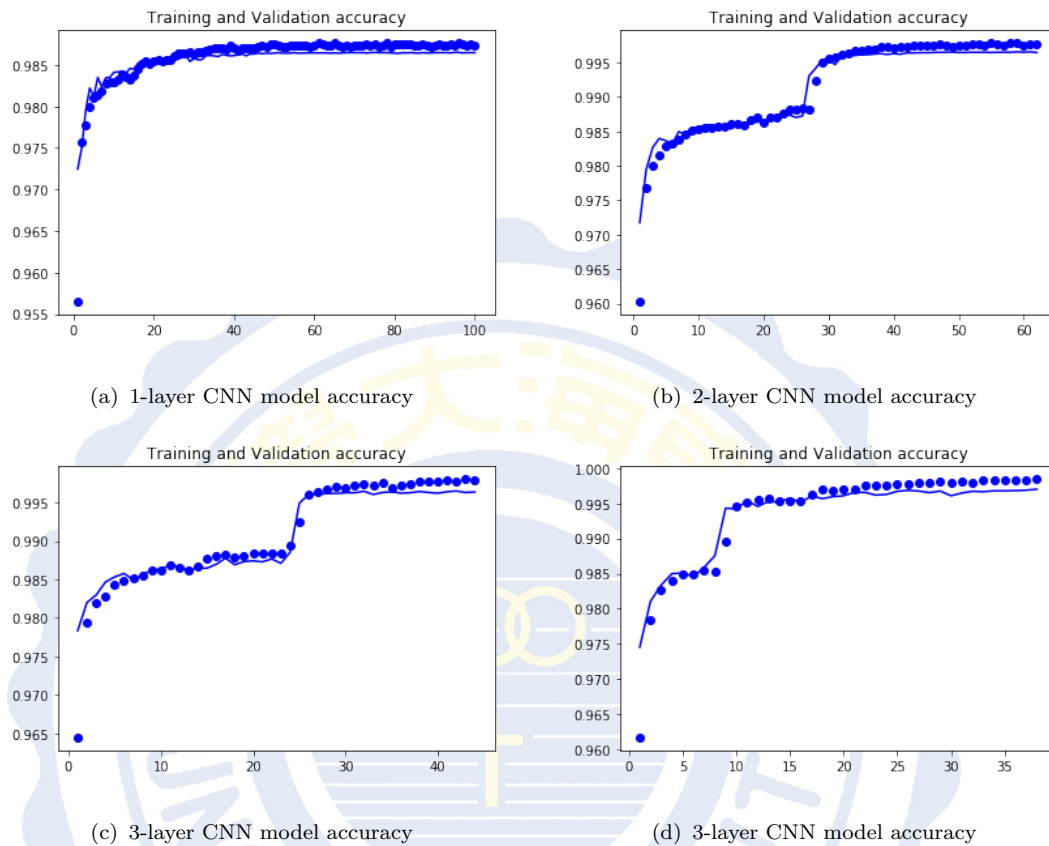


FIGURE 4.33: CNN model accuracy

This experimental result shows that CNN also has the ability to process data type data and has good accuracy. In addition, can find that when the learning_rate is reduced, the feature extraction of the CNN model can be significantly improved, and the accuracy of the model can be improved.

Figure 4.34 shows the Loss change graph of the training set and the verification set. Can see that when the same learning_rate is lowered, the Loss value would also decrease. There is no over_fitting.

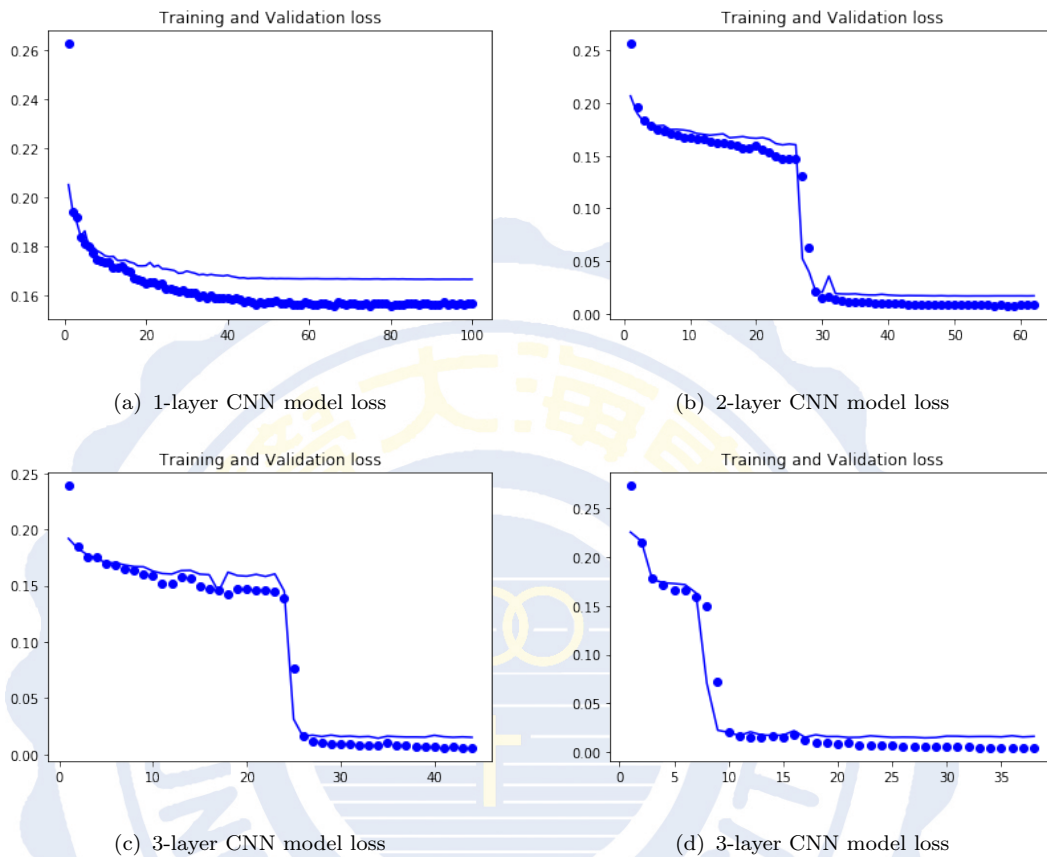


FIGURE 4.34: CNN model loss

The training results of the 1st to 4th CNN models are shown in Figure 4.24. From this experiment, can know that under the 4-layer CNN model, there would be the highest accuracy. And in the case of three layers, the accuracy occurs below the two layers and four layers. The training speed is faster than LSTM and GRU, and the verification set Accuracy is the same as the actual Accuracy, and the same problem as the GRU model does not occur. The results show that CNN also has the ability to process data type data and can effectively reduce training time.

TABLE 4.24: CNN model training result

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
1-Layer	0.0166	0.9864	566.79 sec	5.66 sec
2-Layer	0.0107	0.9954	405.33 sec	6.53 sec
3-Layer	0.0166	0.9953	326.63 sec	7.42 sec
4-Layer	0.0162	0.9959	312.10 sec	8.21 sec

Based on the above experimental results, compare the results of the four experiments and use the average epoch training time to compare the training time. Figure 4.35 below is a comparison of four models of Accuracy. Figure 4.36 below is a comparison of four models of Training time.

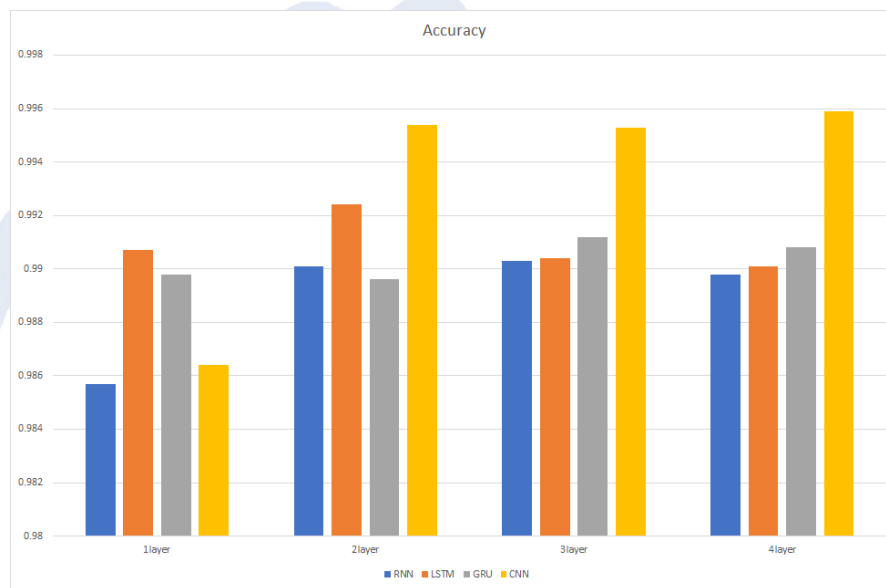


FIGURE 4.35: Comparison of 4 training models

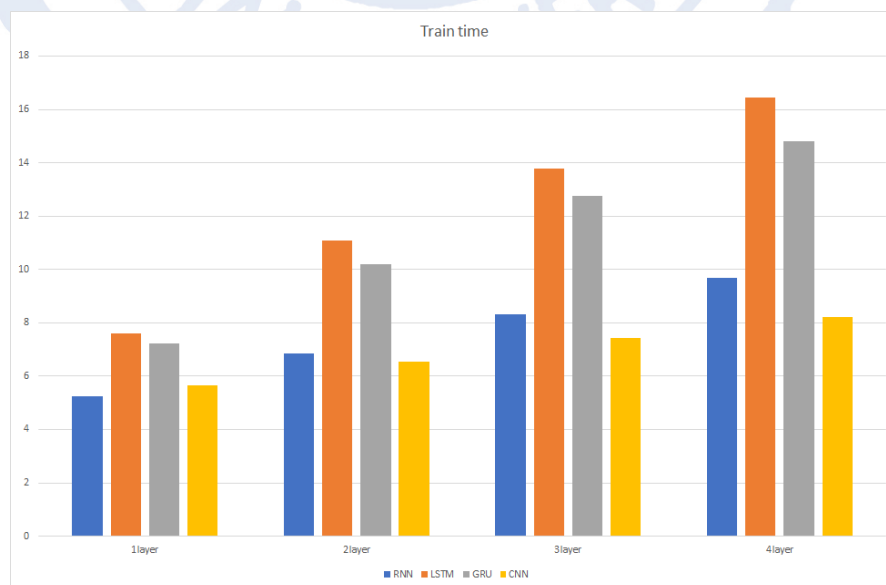


FIGURE 4.36: Comparison of 4 training time

4.5.5 CNN and LSTM Model

CNN has strong performance in extracting features, but CNN cannot handle continuous time issues. In contrast, LSTM is specifically designed to handle continuous time issues. In this paper, propose a model architecture that combines CNN and CuDNNLSTM, with CNN extraction features and rapid training capabilities, and CuDNNLSTM's ability to process continuous time. The experiment was also performed using the NSL-KDD data set.

The architecture of the model is shown in Figure 4.37. tested two layers of CNN combined with LSTM and four layers of CNN in combination with LSTM. Figure 4.38 shows the Accuracy change graph of the training set and the verification set. Figure 4.39 shows the Loss change graph of the training set and the verification set. can see that the changes in Accuracy and Loss are very stable after adding CuDNNLSTM. The result of the experiment is X, and the results prove that our proposed model architecture has the best performance.

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	192
conv1d_2 (Conv1D)	(None, 40, 64)	8256
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
dropout_1 (Dropout)	(None, 20, 64)	0
cu_dnnlstm_1 (CuDNNLSTM)	(None, 70)	38080
dropout_2 (Dropout)	(None, 70)	0
dense_1 (Dense)	(None, 5)	355
Total params: 46,883 Trainable params: 46,883 Non-trainable params: 0		

(a) 2CNN and LSTM model

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 40, 64)	192
conv1d_2 (Conv1D)	(None, 40, 64)	8256
conv1d_3 (Conv1D)	(None, 20, 128)	16512
conv1d_4 (Conv1D)	(None, 20, 128)	32896
max_pooling1d_1 (MaxPooling1D)	(None, 20, 64)	0
max_pooling1d_2 (MaxPooling1D)	(None, 10, 128)	0
dropout_1 (Dropout)	(None, 10, 128)	0
cu_dnnlstm_1 (CuDNNLSTM)	(None, 70)	56080
dropout_2 (Dropout)	(None, 70)	0
dense_1 (Dense)	(None, 5)	355
Total params: 114,211 Trainable params: 114,211 Non-trainable params: 0		

(b) 4CNN and LSTM model

FIGURE 4.37: CNN and LSTM model

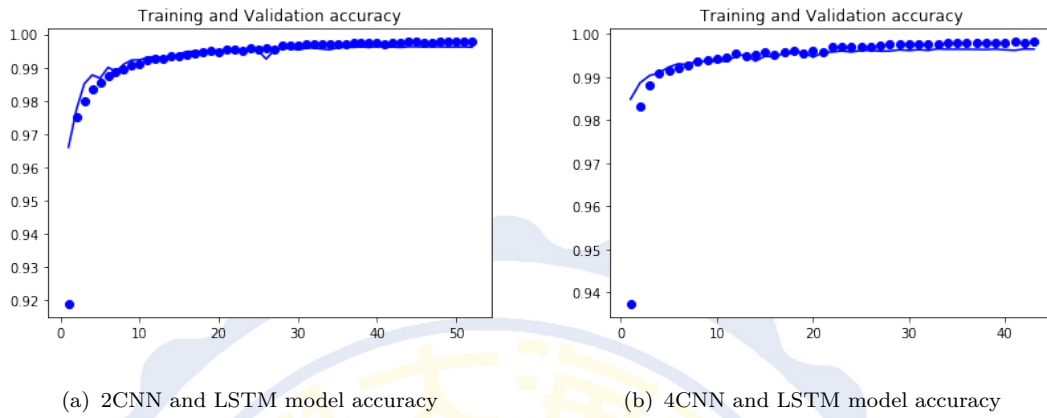


FIGURE 4.38: CNN and LSTM model accuracy

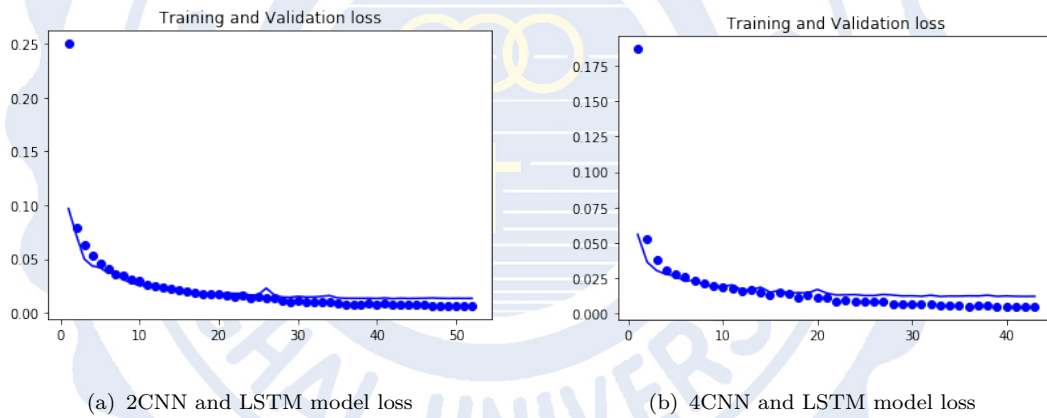


FIGURE 4.39: CNN and LSTM model loss

TABLE 4.25: Comparison of 2CNN LSTM and 4CNN LSTM model

Model	Validation Loss	Validation Accuracy	Training Time	Each Epoch
2CNN LSTM	0.0132	0.9961	477.90 sec	9.19 sec
4CNN LSTM	0.0122	0.9965	437.85 sec	10.18 sec

Finally, reduce the input characteristics to carry out the experiment, because the NetFlow data collected by the school can provide very few features compared with NSL-KDD, and record the one-way network transmission data, only a few features are performed. Fuzzy identification.

It can be seen that if the characteristics data of duration, protocol_type, src_byte and dst_byes which are the same as NetFlow in NSL-KDD are used, the result is very poor, such as Table 4.26, and almost no attack can be recognized.

TABLE 4.26: NetFlow input result

Model	Validation Loss	Validation Accuracy
2CNN LSTM	0.8470	0.5556
4CNN LSTM	0.8486	0.5556

But when add feature TCP Flags, such as Table 4.27, it can see that the accuracy of our proposed training model is significantly improved. This is because a large part of the NSL-KDD data is the DoS attack data, and the characteristic TCP Flags is an important feature of the DoS attack resolution. In the absence of TCP Flags features, the model cannot be well resolved.

TABLE 4.27: NetFlow and TCP Flags input result

Model	Validation Loss	Validation Accuracy
2CNN LSTM	0.453	0.8569
4CNN LSTM	0.440	0.8640

Although the model presented has a good performance on the NSL-KDD dataset. Unfortunately, there are no key features in the NetFlow data obtained. At this stage, it is impossible to carry out good identification. In the future, hope to achieve better performance after obtaining more and more complete data.

4.6 Visualization of Results

In the Ceph section, this work use the Mimic version. The official itself has visual dashboard features. Simply start the Ceph mgr function under the corresponding Ceph machine and you can get information about Ceph and display the information on the dashboard. Figure 4.40 is the home page of Ceph Dashboard. Shows the current health of Ceph, the usage of storage space, the work execution log, and the operation of the Ceph cluster. Figure 4.41 shows the access status of the Ceph File System.

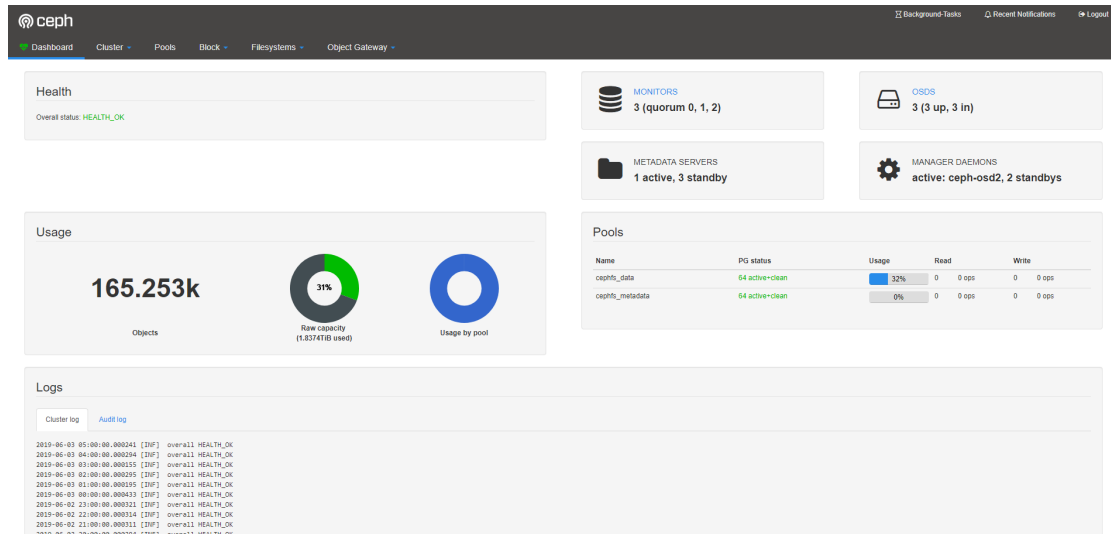


FIGURE 4.40: Ceph dashboard

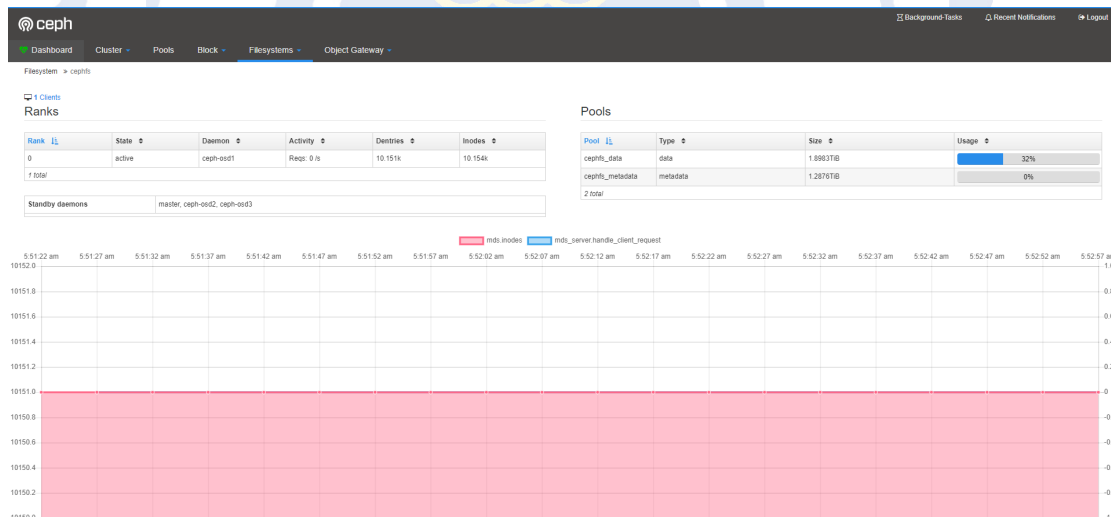


FIGURE 4.41: Ceph FS dashboard

In addition, this paper uses Grafana to build Ceph’s Dashboard. As shown in Figure 4.42, can monitor the operation of Ceph very clearly.

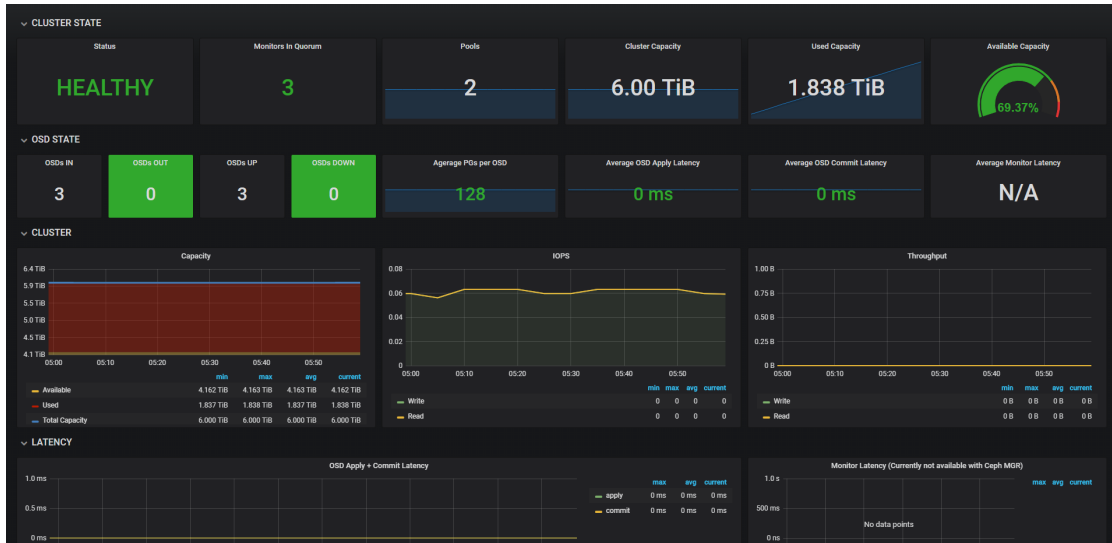


FIGURE 4.42: Ceph grafana dashboard

Store the analysis results and attack identification of the real-time data in the MySQL database. Use PHP to load MySQL data and visualize the results through ECharts, and use the Bootstrap framework to render on web pages of various mobile devices. Through this method can monitor the total usage of real-time data as shown in Figure 4.43. Figure 4.44 shows the instantaneous accumulated network traffic and abnormal traffic occurrence. Figure 4.45 shows the results of the instant attack identification.

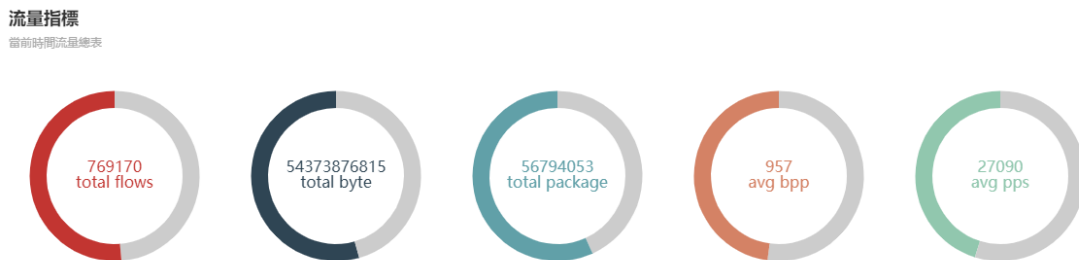


FIGURE 4.43: Total usage of real-time data

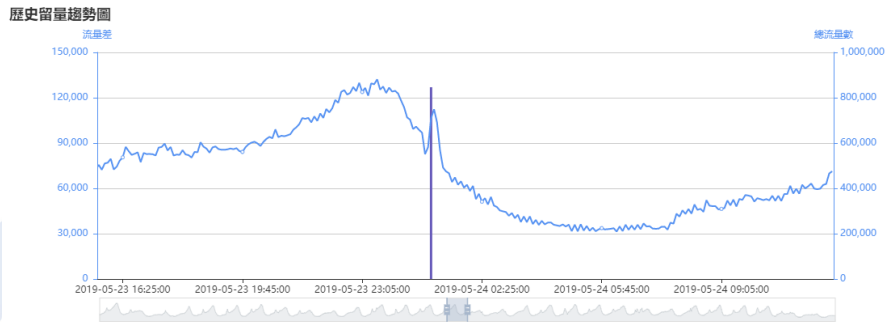


FIGURE 4.44: Network historical traffic and abnormal traffic

Attack:

Show 10 entries

attack	Date first seen	Date last seen	Duration	Dst IP Addr	Dst Pt	In Byte	In Pkt	Proto	Src IP Addr	Src Pt
CodeRed	2019-06-10 21:24:30.266	2019-06-10 21:24:33.266	3	116.255.234.236	80	144	3	1	10.20.0.66	41275
CodeRed	2019-06-10 21:24:15.396	2019-06-10 21:24:29.846	14.45	96.7.252.184	80	144	3	1	10.21.2.27	58540
CodeRed	2019-06-10 21:24:12.286	2019-06-10 21:24:13.626	1.34	140.128.97.232	80	144	3	1	10.65.2.19	62222
CodeRed	2019-06-10 21:24:04.696	2019-06-10 21:24:07.696	3	106.11.129.144	80	144	3	1	10.20.0.66	41181
CodeRed	2019-06-10 21:23:52.616	2019-06-10 21:23:53.836	1.22	140.128.97.216	80	144	3	1	10.30.2.49	57791
CodeRed	2019-06-10 21:23:52.506	2019-06-10 21:23:53.826	1.32	140.128.97.232	80	144	3	1	10.30.2.49	57786
CodeRed	2019-06-10 21:23:41.086	2019-06-10 21:23:55.386	14.3	96.7.252.184	80	144	3	1	10.21.2.27	58505
CodeRed	2019-06-10 21:23:37.076	2019-06-10 21:23:38.536	1.46	140.128.97.232	80	144	3	1	10.73.7.30	3311
CodeRed	2019-06-10 21:23:29.006	2019-06-10 21:23:30.236	1.23	140.128.97.216	80	144	3	1	10.22.3.70	49876
CodeRed	2019-06-10 21:23:28.967	2019-06-10 21:23:30.337	1.37	140.128.97.232	80	144	3	1	10.22.3.70	49871

Showing 1 to 10 of 57 entries Previous Next

FIGURE 4.45: Attack identification

Chapter 5

Conclusions and Future Work

Due to the frequent occurrence of cyber attacks in recent years, campus network monitoring software is generally expensive. These software typically include a database, attack detection, and a fixed front-end web page. In order to solve this problem, we propose an architecture that uses Ceph to do data storage environment, deep learning to identify attacks, and design monitoring web pages according to requirements.

5.1 Concluding Remarks

This work provides a fully open source solution architecture that combines storage, analysis and monitoring. By using the Ceph architecture to store ever-increasing historical data, it can achieve the ability to achieve massive storage at relatively low cost. Designed an anomaly detection method to find out when the network traffic increased significantly. In terms of attack identification, designed experiments to compare the recognition rate and training time of multiple deep learning models for cyber attacks. Finally, designing a graphical web page with ECharts can help managers quickly identify network problems.

5.2 Future Work

Limited by the limitations of the original NetFlow data, except for specific attacks with fixed features. The proposed model cannot be accurately identified due to the lack of many key features in the face of unfixed feature attacks, and it is impossible to achieve effective identification. In the future, hope to be able to obtain more complete data and use the ever-increasing historical data to find key features to accurately identify multiple cyber attacks. In addition, in the historical data, due to the influence of hardware facilities and network speed, the reading analysis is slow. In the future, hope to be able to apply the system to a hardware environment with Solid-state drive, and use a distributed computing architecture under the system to the physical host, and finally achieve faster performance with high-speed network.

References

- [1] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.
- [2] Abebe Abeshu Diro and Naveen Chilamkurti. Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761 – 768, 2018.
- [3] Hongyu Liu, Bo Lang, Ming Liu, and Hanbing Yan. Cnn and rnn based payload classification methods for attack detection. *Knowledge-Based Systems*, 163:332 – 341, 2019.
- [4] B. Dong and X. Wang. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pages 581–585, June 2016.
- [5] Ceph. <https://ceph.com/>, 2019.
- [6] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 31–31, Nov 2006.
- [7] Keras. <https://keras.io/>, 2019.

- [8] Wikipedia contributors. Recurrent neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=889817533, 2019. [Online; accessed 16-April-2019].
- [9] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=892473357, 2019. [Online; accessed 16-April-2019].
- [10] Wikipedia contributors. Gated recurrent unit — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gated_recurrent_unit&oldid=883963113, 2019. [Online; accessed 16-April-2019].
- [11] NSL-KDD. Nsl-kdd. <https://www.unb.ca/cic/datasets/nsl.html>, 2019. [Online; 2019].
- [12] KDD99. Kdd99. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 2019. [Online; 1999].
- [13] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, July 2009.
- [14] Grafana. Grafana. <https://grafana.com/grafana>, 2019. [Online; 2019].
- [15] Bootstrap. <https://getbootstrap.com/>. <https://getbootstrap.com/>, 2019.
- [16] ECharts. <https://echarts.baidu.com/>, 2019.
- [17] Tae-Young Kim and Sung-Bae Cho. Web traffic anomaly detection using c-lstm neural networks. *Expert Systems with Applications*, 106:66 – 76, 2018.
- [18] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156 – 172, 2019.
- [19] 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 *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 258–263, Oct 2016.
- [20] R. Fu, Z. Zhang, and L. Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328, Nov 2016.
- [21] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100 – 123, 2014.
- [22] James Zhang, Robert Gardner, and Ilija Vukotic. Anomaly detection in wide area network meshes using two machine learning algorithms. *Future Generation Computer Systems*, 93:418 – 426, 2019.
- [23] Rafał Kozik, Michał Choraś, Massimo Ficco, and Francesco Palmieri. A scalable distributed machine learning approach for attack detection in edge computing environments. *Journal of Parallel and Distributed Computing*, 119:18 – 26, 2018.
- [24] Rafał Kozik. Distributing extreme learning machines with apache spark for netflow-based malware activity detection. *Pattern Recognition Letters*, 101:14 – 20, 2018.
- [25] Jonathan J. Davis and Andrew J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6): 353 – 375, 2011.
- [26] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, Fourthquarter 2014.
- [27] D. S. Terzi, R. Terzi, and S. Sagiroglu. Big data analytics for network anomaly detection from netflow data. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 592–597, Oct 2017.

- [28] Wikipedia contributors. Vmware esxi — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=VMware_ESXi&oldid=891722497, 2019. [Online; accessed 16-April-2019].
- [29] Scikit Learn. Labelencoder. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>, 2019. [Online; accessed 16-April-2019].
- [30] Scikit Learn. Onehotencoder. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>, 2019. [Online; accessed 16-April-2019].
- [31] Juan M. Estevez-Tapiador, Pedro Garcia-Teodoro, and Jesus E. Diaz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, 27(16):1569 – 1584, 2004.
- [32] N. Ramakrishnan and T. Soni. Network traffic prediction using recurrent neural networks. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 187–193, Dec 2018.
- [33] Wikipedia contributors. Mean squared error — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=890166297, 2019. [Online; accessed 17-April-2019].

Appendix A

Ceph Installation

Set host

```
$ sudo vim /etc/hostname  
$ sudo vim /etc/hosts
```

Create cephuser

```
$ useradd -m -s /bin/bash cephuser  
$ passwd cephuser  
$ echo "cephuser ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/cephuser  
$ chmod 0440 /etc/sudoers.d/cephuser  
$ sed -i s'/Defaults requiretty/#Defaults requiretty'/g /etc/sudoers
```

Install python and parted

```
$ sudo apt-get install -y python python-pip parted
```

Create ssh-key and set config

```
$ ssh-keygen  
$ vim ~/.ssh/config  
$ chmod 644 ~/.ssh/config  
$ ssh-keyscan ceph-osd1 ceph-osd2 ceph-osd3 ceph-client mon1 >> ~/.ssh/known_hosts  
$ ssh-copy-id ceph-osd1  
$ ssh-copy-id ceph-osd2  
$ ssh-copy-id ceph-osd3  
$ ssh-copy-id mon1
```

Install and set ntp

```
$ sudo apt-get install ntp
$ sudo ntpdate -s ntp.ubuntu.com pool.ntp.org
```

install ceph-deploy

```
$ sudo pip install ceph-deploy
```

Create cluster

```
$ mkdir cluster
$ cd cluster/
$ ceph-deploy new ceph-osd1 ceph-osd2 ceph-osd3
$ vim ceph.conf
```

ceph-deploy install ceph

```
$ ceph-deploy install master ceph-osd1 ceph-osd2 ceph-osd3
$ ceph-deploy mon create-initial
```

add OSD

```
$ ceph-deploy osd create --data /dev/sdb ceph-osd1
$ ceph-deploy osd create --data /dev/sdb ceph-osd2
$ ceph-deploy osd create --data /dev/sdb ceph-osd3
$ ceph-deploy admin ceph-admin mon1 ceph-osd1 ceph-osd2 ceph-osd3
$ sudo chmod 644 /etc/ceph/ceph.client.admin.keyring
```

create mgr

```
$ ceph-deploy mgr create ceph-osd{1,2,3}
```

install mds

```
$ ceph-deploy mds create master
```

create pool and file system

```
$ ceph osd pool create cephfs_data 128
$ ceph osd pool create cephfs_metadata 128
$ ceph fs new cephfs cephfs_metadata cephfs_data
```

add client and mount FUSE

```
$ ceph-deploy install client
$ sudo apt-get install -y ceph-fuse
$ sudo mkdir netflow
$ sudo ceph-fuse -m 140.128.101.175:6789 netflow/
```

open dashboard

```
$ ceph mgr module enable dashboard
$ ceph dashboard create-self-signed-cert
$ ceph dashboard set-login-credentials admin admin
$ ceph mgr services
{
"dashboard": "https://140.128.101.176:8080/",
}
```

open Prometheus

```
$ ceph mgr module enable prometheus
$ ss -tlnp |grep 9283
$ ceph mgr services
$ tar -zxvf prometheus-*.tar.gz
$ cd prometheus-*
$ cp prometheus-promtool /usr/local/bin/
$ mkdir /etc/prometheus && mkdir /var/lib/prometheus
$ vim /usr/lib/systemd/system/prometheus.service
[Unit]
Description=Prometheus
Documentation=https://prometheus.io
[Service]
Type=simple
WorkingDirectory=/var/lib/prometheus
EnvironmentFile=/etc/prometheus/prometheus.yml
ExecStart=/usr/local/bin/prometheus \
--config.file /etc/prometheus/prometheus.yml \
--storage.tsdb.path /var/lib/prometheus/
[Install]
WantedBy=multi-user.target
$ vim /etc/prometheus/prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
- job_name: 'prometheus'
static_configs:
- targets: ['140.128.101.175:9090']
- job_name: 'ceph'
```

```
static_configs:
- targets:
    - 140.128.101.176:9283
      - 140.128.101.172:9283
      - 140.128.98.54:9283
$ systemctl daemon-reload
$ systemctl start prometheus
$ systemctl status prometheus
```

install grafana

```
$ wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana-5.2.2-1.x86_64.rpm
$ yum -y localinstall grafana-5.2.2-1.x86_64.rpm
$ systemctl start grafana-server
$ systemctl status grafana-server
```

Appendix B

CUDA 10.0, cuDNN7.4 and tensorflow 1.12 Installation

GPU set

```
$ sudo vim /etc/modprobe.d/blacklist.conf
$ # for nvidia display device install
blacklist vga16fb
blacklist nouveau
blacklist rivafb
blacklist rivatv
blacklist nvidiafb
$ sudo update-initramfs -u
$ sudo reboot
$ wget -c https://cn.download.nvidia.com/XFree86/Linux-x86_64/430.14/NVIDIA-Linux-x86_64-430.14.run
$ sudo chmod +x NVIDIA-Linux-x86_64-430.14.run
$ sudo ./NVIDIA-Linux-x86_64-430.14.run
```

update check

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa libxi-dev libxmu-dev -y
$ sudo apt-get --yes install build-essential
$ sudo apt-get install python-pip python-dev -y
$ sudo apt-get install python-numpy python-scipy -y
```

download, install and set CUDA

```
$ wget https://developer.nvidia.com/compute/cuda/10.0/Prod/local_installers/cuda_10.0.130_410.48_linux
$ sudo chmod +x cuda_10.0.130_410.48_linux.run
```

```
$ sudo ./cuda_10.0.130_410.48_linux --driver --silent
$ sudo ./cuda_10.0.130_410.48_linux --toolkit --silent
$ sudo ./cuda_10.0.130_410.48_linux --samples --silent
$ sudo apt-get install libglu1-mesa libxi-dev libxmu-dev libglu1-mesa-dev
$ sudo vim ~/.bashrc
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/64:$LD_LIBRARY_PATH
$ source ~/.bashrc
```

download, install and set CuDNN

```
$ wget -c https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.4.2/prod/10.0_20181213
$ tar -xvf cudnn-10.0-linux-x64-v7.3.1.20.tgz
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include/
$ sudo cp cuda/lib64/lib* /usr/local/cuda/lib64/
$ cd /usr/local/cuda/lib64/
$ sudo chmod +r libcudnn.so.7.4.2
$ sudo ln -sf libcudnn.so.7.4.2 libcudnn.so.7
$ sudo ln -sf libcudnn.so.7 libcudnn.so
$ sudo ldconfig
```

tensorflow download

```
$ wget -c https://github.com/fo40225/tensorflow-windows-wheel/raw/master/1.12.0/py36/GPU/cuda100cudnn7
$ pip install tensorflow_gpu-1.12.0-cp36-cp36m-win_amd64.whl
```

Appendix C

Anaconda Installation

download anaconda

```
$ wget https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
```

install anaconda

```
$ bash Anaconda3-2019.03-Linux-x86_64.sh
```

bashrc activate

```
$ source ~/.bashrc
```

conda create and activate

```
$ conda create --name tf python=3.6  
$ conda activate tf  
$ conda install jupyter
```

set jupyter passwd

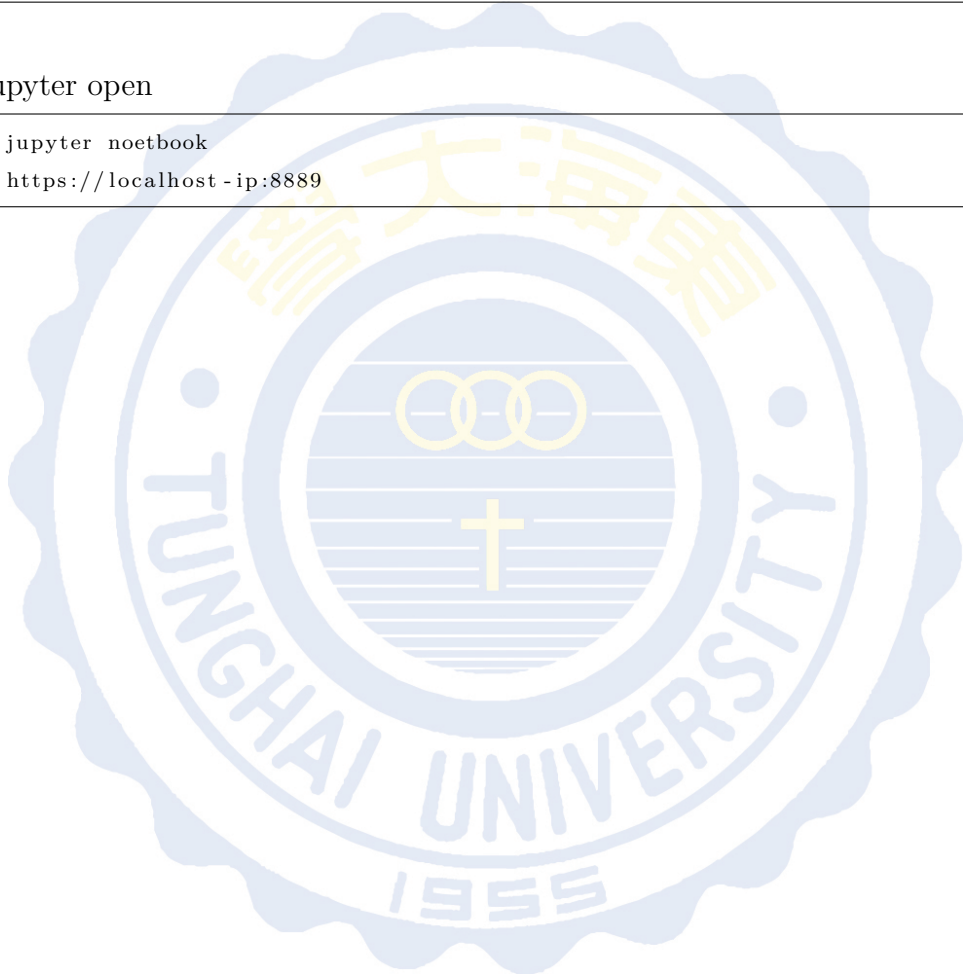
```
$ jupyter notebook --generate-config  
$ ipython  
$ from notebook.auth import passwd  
$ passwd()
```

connect set


```
$ sudo vim ~/.jupyter/jupyter_notebook_config.py
$ c.NotebookApp.ip='*'
$ c.NotebookApp.password = u'#passwd'
$ c.NotebookApp.open_browser = False
$ c.NotebookApp.port = 8889
```

jupyter open

```
$ jupyter noetbook
$ https://localhost -ip:8889
```



Appendix D

download netflow data

```
#!/usr/bin/python3

import urllib
import urllib.request
from urllib.request import urlopen
import datetime
import time

times = datetime.datetime.now() - datetime.timedelta(seconds=300)
times = times.strftime("%Y%m%d%H%M")
time.sleep(30)
url = 'http://140.128.197.43/nfcapd.' + times + '.txt'
filename = 'netflow2/nfcapd.' + times + '.txt'
urlopen(url, filename)
```

Appendix E

flow count

```
import re
import os
import pandas as pd
import numpy as np
from pandas import read_csv
path = 'netflow'
files = os.listdir(path)
files_csv = list(filter(lambda x: x[-4:] == '.txt', files))

def get_last_line(inputfile):
    filesize = os.path.getsize(inputfile)
    blocksize = 1024
    dat_file = open(inputfile, 'rb')
    last_line = b""
    lines = []
    if filesize > blocksize:
        maxseekpoint = (filesize // blocksize)
        maxseekpoint -= 1
        dat_file.seek(maxseekpoint * blocksize)
        lines = dat_file.readlines()
        while ((len(lines) < 2) | ((len(lines) >= 2) & (lines[1] == b'\r\n'))):
            maxseekpoint -= 1
            dat_file.seek(maxseekpoint * blocksize)
            lines = dat_file.readlines()
    elif filesize:
        dat_file.seek(0, 0)
        lines = dat_file.readlines()
    if lines:
        for i in range(len(lines) - 4, -1, -1):
            last_line = lines[i].strip()
            if (last_line != b''):
```

```
break
dat_file.close()
return last_line

data_list = []
num_filter = re.compile(r'\d+')
for file in files_csv:
tmp = get_last_line(path + '/' + file)
tmp = tmp.decode()
list = re.findall(r"\d+\.\d*",tmp)
arr1 = np.array(list)
data={'total flows':[arr1[0]],'total bytes':[arr1[1]],'total packets':[arr1[2]],'avg bps':[arr1[3]],'a
df=pd.DataFrame(data)
data_list.append(df)
all_data = pd.concat(data_list)
all_data.to_csv("all_sum.csv",index=False)

all_sum = read_csv('all_sum.csv', engine='python')
files_csv1 = pd.DataFrame(files_csv)
splitfile = files_csv1[0].str.strip().str.split('.',0,True) #split 檔案名為時間序列
all_sum['filename'] = splitfile[1]
all_sum = all_sum.set_index('filename')
all_sum.to_csv("all_sum.csv",index=True)
```

Appendix F

abnormal check

```
from pandas import read_csv
from matplotlib import pyplot
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
dataset = read_csv('all_sum.csv', header=0)
dataset['filename'] = pd.to_datetime(dataset['filename'], format="%Y%m%d%H%M")
dataset.set_index("filename", inplace=True)
dataset = dataset.sort_index()
#dataset = dataset['2019-01-14 12' : '2019-01-18 15:05:00']
del dataset['total bytes'], dataset['total packets'], dataset['avg bps'], dataset['avg pps'], dataset['avg
df=dataset.copy()
for i in range(289, len(dataset[:])-1):
if dataset['total flows'][i]-dataset['total flows'][i-288]>dataset['total flows'][i-288]*0.5:
ty=dataset['total flows'][i]-dataset['total flows'][i+1]
td=dataset['total flows'][i]-dataset['total flows'][i-1]
if (20000>ty>10000)&(20000>td>10000):
print(dataset.index[i], dataset['total flows'][i-1], dataset['total flows'][i], dataset['total flows'][
if (ty>20000)&(td>20000):
print(dataset.index[i], dataset['total flows'][i-1], dataset['total flows'][i], dataset['total flows'][
dataset['1']=None
dataset['2']=None
dataset['3']=None
for i in range(1, len(dataset[:])):
dt1=abs(dataset['total flows'][i]-dataset['total flows'][i-1])
dataset['1'][i]=dt1
dataset[:, '3']=None
dataset[:, '2']=None
mean = dataset[:, '1']
```

```

mu = np.mean(mean)
std3 = np.std(mean)*3
std5 = np.std(mean)*5
for i in range(1, len(dataset[:-1])):
if (mu+std5>dataset['1'][i]>mu+std3)&(dataset['total flows'][i-1]<dataset['total flows'][i]):
dt2=dataset['1'][i]
dataset['2'][i]=dt2
for i in range(1, len(dataset[:-1])):
if (dataset['1'][i]>mu+std5)&(dataset['total flows'][i-1]<dataset['total flows'][i]):
dt3=dataset['1'][i]
dataset['3'][i]=dt3
import plotly
from plotly.graph_objs import Scatter, Layout, Bar
import plotly.graph_objs as go
plotly.offline.init_notebook_mode(connected=True)
trace0 = Scatter(
x = dataset[:].index,
y = dataset[:]['total flows'],
mode = 'lines',
name = 'flows',
xaxis='x',
yaxis='y'
)
trace1 = Bar(
x = dataset[:].index,
y = dataset[:]['2'],
name = '中流量',
yaxis='y2'
)
trace2 = Bar(
x = dataset[:].index,
y = dataset[:]['3'],
name = '高流量',
yaxis='y2'
)
data = [trace0, trace1, trace2]
layout = go.Layout(
yaxis2=dict(anchor='x', overlaying='y', side='right')#設定座標軸的格式
)
fig = go.Figure(data=data, layout=layout)
plotly.offline.iplot(fig, filename='scatter-mode')
plotly.offline.init_notebook_mode(connected=True)
x1 = dataset[:].index
y1 = dataset[:]['1']
plotly.offline.iplot({
"data": [Scatter(x=x1, y=y1)],
"layout": Layout(title="flow")
})

```



Appendix G

Attack Identification with Fixed Features RNN Model

```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
import numpy as np
from pandas import read_csv
np.random.seed(7)
dataset = read_csv("attack_data.csv",header=None, skiprows=[0])
del dataset[0]
del dataset[1],dataset[2]
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
dataset[3] = dataset[3].str.lstrip()
dataset[11] = dataset[11].str.rstrip()
le = LabelEncoder()
le.fit(dataset[11])
dataset[11] = le.transform(dataset[11])
le.fit(dataset[3])
dataset[3] = le.transform(dataset[3])
le.fit(dataset[13])
dataset[13] = le.transform(dataset[13])
Y_train = OneHotEncoder(sparse = False).fit_transform( dataset[[13]] )
dataset = dataset.values
dataset2 = read_csv("attack_data4.csv",header=None, skiprows=[0])
del dataset2[0]
dataset2[3] = dataset2[3].str.lstrip()
dataset2[11] = dataset2[11].str.rstrip()
le.fit(dataset2[11])
dataset2[11] = le.transform(dataset2[11])
le.fit(dataset2[3])
```



```
dataset2[3] = le.transform(dataset2[3])
del dataset2[1], dataset2[2]
le.fit(dataset2[13])
dataset2[13] = le.transform(dataset2[13])
Y_val = OneHotEncoder(sparse = False).fit_transform( dataset2[[13]] )
dataset2 = dataset2.values
X_val = dataset2[3:,1:9]
Y_val = Y_val[3:]
X = dataset[3:,1:9]
Y = Y_train[3:]
X = np.reshape(X, (X.shape[0], 1, X.shape[1]))
X_val = np.reshape(X_val, (X_val.shape[0], 1, X_val.shape[1]))
# create model
model = Sequential()
model.add(SimpleRNN(16, input_dim=8, return_sequences=True))
model.add(SimpleRNN(16, return_sequences=False))
model.add(Dense(3, activation="softmax"))
# Compile model
model.summary()
import time
start = time.time()
sgd = SGD(lr=0.001)
model.compile(loss='mse', optimizer='RMSprop', metrics=['accuracy'])
history = model.fit(X, Y, epochs=40, batch_size=10, shuffle=True, verbose=1, validation_data=(X_val,
score = model.evaluate(X, Y, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label='Validation_acc')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label='Validation_loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

Appendix H

NSL-KDD Training of Rnn Model

```
import pandas as pd
import numpy as np
import h5py
np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLRonPlateau, CSVLogger
from pandas.core.frame import DataFrame
def get_total_data():
data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)
data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'disc
data[3]=data[3].map({'OTH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S
data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan'
data[2] = (data[2]-data[2].min())/(data[2].max() - data[2].min())
data[4] = (data[4]-data[4].min())/(data[4].max() - data[4].min())
data[5] = (data[5]-data[5].min())/(data[5].max() - data[5].min())
data[22] = (data[22]-data[22].min())/(data[22].max() - data[22].min())
data[23] = (data[23]-data[23].min())/(data[23].max() - data[23].min())
data[31] = (data[31]-data[31].min())/(data[31].max() - data[31].min())
data[32] = (data[32]-data[32].min())/(data[32].max() - data[32].min())
return data
def get_target_data():
data = get_total_data()
enc = OneHotEncoder(sparse = False)
enc.fit([[0], [1], [2], [3], [4]])
```

```
result = enc.transform(data[[41]])

return DataFrame(result)

def get_input_data():
data = get_total_data()
del data[41]
data = data.iloc[:,0:4]
return data

if __name__ == '__main__':
data_input = get_input_data()
data_target = get_target_data()

x_train, x_test, y_train, y_test = train_test_split(
data_input, data_target, test_size=0.25, random_state=42)
x_train = x_train.values
y_train = y_train.values
x_test = x_test.values
y_test = y_test.values

X_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
X_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
batch_size = 128

# RNN
model = Sequential()
model.add(SimpleRNN(16,input_dim=4, return_sequences=True))
model.add(Dropout(0.1))
model.add(SimpleRNN(16,input_dim=40, return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(5, activation="softmax"))
model.summary()

learning_rate_reduction = ReduceLRonPlateau(monitor='acc',
patience=3,
verbose=1,
factor=0.5,
min_lr=0.00001)

from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)

import time
start = time.time()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=batch_size, nb_epoch=100, validation_data=(X_test, y_test))
model.save("RNN2layer_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
```

```
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label='Validation_acc')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label='Validation_loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

Appendix I

NSL-KDD Training of LSTM

Model

```
import pandas as pd
import numpy as np
import h5py

np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
from pandas.core.frame import DataFrame

def get_total_data():
    data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)
    data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
    data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'disc
    data[3]=data[3].map({'OTH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S
    data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan'
    data[2] = (data[2]-data[2].min())/(data[2].max() - data[2].min())
    data[4] = (data[4]-data[4].min())/(data[4].max() - data[4].min())
    data[5] = (data[5]-data[5].min())/(data[5].max() - data[5].min())
    data[22] = (data[22]-data[22].min())/(data[22].max() - data[22].min())
    data[23] = (data[23]-data[23].min())/(data[23].max() - data[23].min())
    data[31] = (data[31]-data[31].min())/(data[31].max() - data[31].min())
    data[32] = (data[32]-data[32].min())/(data[32].max() - data[32].min())
    return data
```

```
def get_target_data():

    data = get_total_data()

    enc = OneHotEncoder(sparse = False)
    enc.fit([[0], [1], [2], [3], [4]])
    result = enc.transform(data[[41]])

    return DataFrame(result)

def get_input_data():
    data = get_total_data()
    del data[41]
    data = data.iloc[:,0:4]
    return data

if __name__ == '__main__':
    data_input = get_input_data()
    data_target = get_target_data()

    x_train, x_test, y_train, y_test = train_test_split(
        data_input, data_target, test_size=0.25, random_state=42)
    x_train = x_train.values
    y_train = y_train.values
    x_test = x_test.values
    y_test = y_test.values

    X_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
    X_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
    batch_size = 128

    # LSTM
    model = Sequential()
    model.add(LSTM(16, input_dim=40, return_sequences=True))
    model.add(Dropout(0.1))
    model.add(LSTM(16, input_dim=40, return_sequences=False))
    model.add(Dropout(0.1))
    model.add(Dense(5, activation="softmax"))
    model.summary()
    learning_rate_reduction = ReduceLROnPlateau(monitor='acc',
        patience=3,
        verbose=1,
        factor=0.5,
        min_lr=0.00001)
    from keras.callbacks import EarlyStopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)
    import time
    start = time.time()
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=batch_size, nb_epoch=100, validation_data=(X_test, y_test))
model.save("LSTM2layer_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label="Validation acc")
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label="Validation loss")
plt.title('Training and Validation loss')
plt.legend()
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

Appendix J

NSL-KDD Training of GRU

Model

```
import pandas as pd
import numpy as np
import h5py
np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
from pandas.core.frame import DataFrame
def get_total_data():
data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)
data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'disc
data[3]=data[3].map({'OTH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S
data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan'
data[2] = (data[2]-data[2].min())/(data[2].max() - data[2].min())
data[4] = (data[4]-data[4].min())/(data[4].max() - data[4].min())
data[5] = (data[5]-data[5].min())/(data[5].max() - data[5].min())
data[22] = (data[22]-data[22].min())/(data[22].max() - data[22].min())
data[23] = (data[23]-data[23].min())/(data[23].max() - data[23].min())
data[31] = (data[31]-data[31].min())/(data[31].max() - data[31].min())
data[32] = (data[32]-data[32].min())/(data[32].max() - data[32].min())
return data
```



```
def get_target_data():

    data = get_total_data()

    enc = OneHotEncoder(sparse = False)
    enc.fit([[0], [1], [2], [3], [4]])
    result = enc.transform(data[[41]])

    return DataFrame(result)

def get_input_data():
    data = get_total_data()
    del data[41]
    data = data.iloc[:,0:4]
    return data

if __name__ == '__main__':
    data_input = get_input_data()
    data_target = get_target_data()

    x_train, x_test, y_train, y_test = train_test_split(
        data_input, data_target, test_size=0.25, random_state=42)
    x_train = x_train.values
    y_train = y_train.values
    x_test = x_test.values
    y_test = y_test.values

    X_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
    X_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
    batch_size = 128

    # GRU
    model = Sequential()
    model.add(GRU(16, input_dim=40, return_sequences=True))
    model.add(Dropout(0.1))
    model.add(GRU(16, input_dim=40, return_sequences=False))
    model.add(Dropout(0.1))
    model.add(Dense(5, activation="softmax"))
    model.summary()
    learning_rate_reduction = ReduceLRonPlateau(monitor='acc',
        patience=3,
        verbose=1,
        factor=0.5,
        min_lr=0.00001)
    from keras.callbacks import EarlyStopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)
    import time
    start = time.time()
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=batch_size, nb_epoch=100, validation_data=(X_test, y_test))
model.save("GRU2layer_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label="Validation acc")
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label="Validation loss")
plt.title('Training and Validation loss')
plt.legend()
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

Appendix K

NSL-KDD Training of CNN Model

```
import pandas as pd
import numpy as np
import h5py

np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
from pandas.core.frame import DataFrame

def get_total_data():
    data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)
    data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
    data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'discuss':7, 'ftp':8, 'http':9, 'irc':10, 'kermitech':11, 'koreaserver':12, 'koreaserver':13, 'koreaserver':14, 'koreaserver':15, 'koreaserver':16, 'koreaserver':17, 'koreaserver':18, 'koreaserver':19, 'koreaserver':20, 'koreaserver':21, 'koreaserver':22, 'koreaserver':23, 'koreaserver':24, 'koreaserver':25, 'koreaserver':26, 'koreaserver':27, 'koreaserver':28, 'koreaserver':29, 'koreaserver':30, 'koreaserver':31, 'koreaserver':32, 'koreaserver':33, 'koreaserver':34, 'koreaserver':35, 'koreaserver':36, 'koreaserver':37, 'koreaserver':38, 'koreaserver':39, 'koreaserver':40, 'koreaserver':41, 'koreaserver':42, 'koreaserver':43, 'koreaserver':44, 'koreaserver':45, 'koreaserver':46, 'koreaserver':47, 'koreaserver':48, 'koreaserver':49, 'koreaserver':50, 'koreaserver':51, 'koreaserver':52, 'koreaserver':53, 'koreaserver':54, 'koreaserver':55, 'koreaserver':56, 'koreaserver':57, 'koreaserver':58, 'koreaserver':59, 'koreaserver':60, 'koreaserver':61, 'koreaserver':62, 'koreaserver':63, 'koreaserver':64, 'koreaserver':65, 'koreaserver':66, 'koreaserver':67, 'koreaserver':68, 'koreaserver':69, 'koreaserver':70, 'koreaserver':71, 'koreaserver':72, 'koreaserver':73, 'koreaserver':74, 'koreaserver':75, 'koreaserver':76, 'koreaserver':77, 'koreaserver':78, 'koreaserver':79, 'koreaserver':80, 'koreaserver':81, 'koreaserver':82, 'koreaserver':83, 'koreaserver':84, 'koreaserver':85, 'koreaserver':86, 'koreaserver':87, 'koreaserver':88, 'koreaserver':89, 'koreaserver':90, 'koreaserver':91, 'koreaserver':92, 'koreaserver':93, 'koreaserver':94, 'koreaserver':95, 'koreaserver':96, 'koreaserver':97, 'koreaserver':98, 'koreaserver':99})
    data[3]=data[3].map({'OTH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S4':0, 'S5':0, 'S6':0, 'S7':0, 'S8':0, 'S9':0, 'S10':0, 'S11':0, 'S12':0, 'S13':0, 'S14':0, 'S15':0, 'S16':0, 'S17':0, 'S18':0, 'S19':0, 'S20':0, 'S21':0, 'S22':0, 'S23':0, 'S24':0, 'S25':0, 'S26':0, 'S27':0, 'S28':0, 'S29':0, 'S30':0, 'S31':0, 'S32':0, 'S33':0, 'S34':0, 'S35':0, 'S36':0, 'S37':0, 'S38':0, 'S39':0, 'S40':0, 'S41':0, 'S42':0, 'S43':0, 'S44':0, 'S45':0, 'S46':0, 'S47':0, 'S48':0, 'S49':0, 'S50':0, 'S51':0, 'S52':0, 'S53':0, 'S54':0, 'S55':0, 'S56':0, 'S57':0, 'S58':0, 'S59':0, 'S60':0, 'S61':0, 'S62':0, 'S63':0, 'S64':0, 'S65':0, 'S66':0, 'S67':0, 'S68':0, 'S69':0, 'S70':0, 'S71':0, 'S72':0, 'S73':0, 'S74':0, 'S75':0, 'S76':0, 'S77':0, 'S78':0, 'S79':0, 'S80':0, 'S81':0, 'S82':0, 'S83':0, 'S84':0, 'S85':0, 'S86':0, 'S87':0, 'S88':0, 'S89':0, 'S90':0, 'S91':0, 'S92':0, 'S93':0, 'S94':0, 'S95':0, 'S96':0, 'S97':0, 'S98':0, 'S99':0})
    data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan':4})
    data[2] = (data[2] - data[2].min()) / (data[2].max() - data[2].min())
    data[4] = (data[4] - data[4].min()) / (data[4].max() - data[4].min())
    data[5] = (data[5] - data[5].min()) / (data[5].max() - data[5].min())
    data[22] = (data[22] - data[22].min()) / (data[22].max() - data[22].min())
    data[23] = (data[23] - data[23].min()) / (data[23].max() - data[23].min())
    data[31] = (data[31] - data[31].min()) / (data[31].max() - data[31].min())
    data[32] = (data[32] - data[32].min()) / (data[32].max() - data[32].min())
    return data
```

```
def get_target_data():

data = get_total_data()

enc = OneHotEncoder(sparse = False)
enc.fit([[0], [1], [2], [3], [4]])
result = enc.transform(data[[41]])

return DataFrame(result)

def get_input_data():
data = get_total_data()
del data[41]
data = data.iloc[:,0:4]
return data

if __name__ == '__main__':
data_input = get_input_data()
data_target = get_target_data()

x_train, x_test, y_train, y_test = train_test_split(
data_input, data_target, test_size=0.25, random_state=42)
x_train = x_train.values
y_train = y_train.values
x_test = x_test.values
y_test = y_test.values

X_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
X_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
model = Sequential()
model.add(Convolution1D(64, 3, border_mode="same", activation="relu", input_shape=(40, 1)))
model.add(Convolution1D(64, 3, border_mode="same", activation="relu"))
model.add(MaxPooling1D(pool_length=(2)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(5, activation="softmax"))
model.summary()
learning_rate_reduction = ReduceLRonPlateau(monitor='acc',
patience=3,
verbose=1,
factor=0.5,
min_lr=0.00001)
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)
import time
start = time.time()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, batch_size=batch_size, nb_epoch=100, validation_data=(X_test, y_test))
model.save("CNN2layer_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label="Validation acc")
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label="Validation loss")
plt.title('Training and Validation loss')
plt.legend()
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

Appendix L

NSL-KDD Training of 2CNN CuDNNLSTM Model

```
import pandas as pd
import numpy as np
import h5py
np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
from pandas.core.frame import DataFrame
def get_total_data():

data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)

data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'disc
data[3]=data[3].map({'OIH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S
data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan'
'apache2':1, 'back':1, 'land':1, 'mailbomb':1, 'neptune':1, 'pod':1, 'processtable':1, 'smurf':1, 'teard
'buffer_overflow':3, 'httptunnel':2, 'loadmodule':3, 'perl':3, 'ps':3, 'rootkit':3, 'sqlattack':3, 'xter
'ftp_write':2, 'guess_passwd':2, 'imap':2, 'multihop':2, 'named':2, 'phf':2, 'sendmail':2, 'snmpgetattac

data[2] = (data[2]-data[2].min())/(data[2].max() - data[2].min())
data[4] = (data[4]-data[4].min())/(data[4].max() - data[4].min())
```

```
data[5] = (data[5]-data[5].min())/(data[5].max() - data[5].min())
data[22] = (data[22]-data[22].min())/(data[22].max() - data[22].min())
data[23] = (data[23]-data[23].min())/(data[23].max() - data[23].min())
data[31] = (data[31]-data[31].min())/(data[31].max() - data[31].min())
data[32] = (data[32]-data[32].min())/(data[32].max() - data[32].min())

return data

def get_target_data():

data = get_total_data()

enc = OneHotEncoder(sparse = False)
enc.fit([[0], [1], [2], [3], [4]])
result = enc.transform(data[[41]])

return DataFrame(result)

def get_input_data():

data = get_total_data()
del data[41]
data = data.iloc[:,0:40]
return data

data_input = get_input_data()
data_target = get_target_data()

x_train, x_test, y_train, y_test = train_test_split(
data_input, data_target, test_size=0.25, random_state=42)
x_train = x_train.values
y_train = y_train.values
x_test = x_test.values
y_test = y_test.values

X_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
X_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
model = Sequential()

#clstm
model.add(Convolution1D(64, 2, border_mode="same", activation="relu",input_shape=(40, 1)))
model.add(Convolution1D(64, 2, border_mode="same", activation="relu"))
model.add(MaxPooling1D(pool_length=(2)))
model.add(Dropout(0.25))
model.add(CuDNNLSTM(70))
model.add(Dropout(0.25))
model.add(Dense(5, activation="softmax"))
```

```
model.summary()
learning_rate_reduction = ReduceLROnPlateau(monitor='acc',
patience=3,
verbose=1,
factor=0.5,
min_lr=0.00001)
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)

import time
start = time.time()
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
# train
history = model.fit(X_train, y_train, batch_size=128, nb_epoch=100, validation_data=(X_test, y_test), cal
model.save("2cnnlstm_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label="Validation acc")
plt.title('Training and Validation accuracy')
plt.legend
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label="Validation loss")
plt.title('Training and Validation loss')
plt.legend
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```


Appendix M

NSL-KDD Training of 4CNN CuDNNLSTM Model

```
import pandas as pd
import numpy as np
import h5py
np.random.seed(1337) # for reproducibility
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from keras.layers import Convolution1D, Dropout, Flatten, MaxPooling1D
from keras.layers import LSTM, GRU, SimpleRNN, CuDNNLSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
from pandas.core.frame import DataFrame
def get_total_data():

data = pd.read_csv('/home/hpcuser/KDD/KDDTrain+.csv', header=None)

data[1]=data[1].map({'tcp':0, 'udp':1, 'icmp':2})
data[2]=data[2].map({'aol':0, 'auth':1, 'bgp':2, 'courier':3, 'csnet_ns':4, 'ctf':5, 'daytime':6, 'disc
data[3]=data[3].map({'OIH':0, 'REJ':0, 'RSTO':0, 'RSTOS0':0, 'RSTR':0, 'S0':0, 'S1':0, 'S2':0, 'S3':0, 'S
data[41]=data[41].map({'normal':0, 'ipsweep':4, 'mscan':4, 'nmap':4, 'portsweep':4, 'saint':4, 'satan'
'apache2':1, 'back':1, 'land':1, 'mailbomb':1, 'neptune':1, 'pod':1, 'processtable':1, 'smurf':1, 'teard
'buffer_overflow':3, 'httptunnel':2, 'loadmodule':3, 'perl':3, 'ps':3, 'rootkit':3, 'sqlattack':3, 'xter
'ftp_write':2, 'guess_passwd':2, 'imap':2, 'multihop':2, 'named':2, 'phf':2, 'sendmail':2, 'snmpgetattac

data[2] = (data[2]-data[2].min())/(data[2].max() - data[2].min())
data[4] = (data[4]-data[4].min())/(data[4].max() - data[4].min())
```

```

data[5] = (data[5]-data[5].min())/(data[5].max() - data[5].min())
data[22] = (data[22]-data[22].min())/(data[22].max() - data[22].min())
data[23] = (data[23]-data[23].min())/(data[23].max() - data[23].min())
data[31] = (data[31]-data[31].min())/(data[31].max() - data[31].min())
data[32] = (data[32]-data[32].min())/(data[32].max() - data[32].min())

return data

def get_target_data():

data = get_total_data()

enc = OneHotEncoder(sparse = False)
enc.fit([[0], [1], [2], [3], [4]])
result = enc.transform(data[[41]])

return DataFrame(result)

def get_input_data():

data = get_total_data()
del data[41]
data = data.iloc[:,0:40]
return data

data_input = get_input_data()
data_target = get_target_data()

x_train, x_test, y_train, y_test = train_test_split(
data_input, data_target, test_size=0.25, random_state=42)
x_train = x_train.values
y_train = y_train.values
x_test = x_test.values
y_test = y_test.values

X_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
X_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
model = Sequential()

#clstm
model.add(Convolution1D(64, 2, border_mode="same", activation="relu",input_shape=(40, 1)))
model.add(Convolution1D(64, 2, border_mode="same", activation="relu"))
model.add(MaxPooling1D(pool_length=(2)))
model.add(Convolution1D(128, 2, border_mode="same", activation="relu"))
model.add(Convolution1D(128, 2, border_mode="same", activation="relu"))
model.add(MaxPooling1D(pool_length=(2)))
model.add(Dropout(0.25))
model.add(CuDNNLSTM(70))

```

```
model.add(Dropout(0.25))
model.add(Dense(5, activation="softmax"))

model.summary()
learning_rate_reduction = ReduceLRonPlateau(monitor='acc',
patience=3,
verbose=1,
factor=0.5,
min_lr=0.00001)
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=2)

import time
start = time.time()
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
# train
history = model.fit(X_train, y_train, batch_size=128, nb_epoch=100, validation_data=(X_test, y_test), cal
model.save("2cnnlstm_model.hdf5")
score = model.evaluate(X_test, y_test, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
stop = time.time()
print(str(stop-start) + "秒")
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label="Training acc")
plt.plot(epochs, val_acc, 'b', label='Validation_acc')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label="Training loss")
plt.plot(epochs, val_loss, 'b', label='Validation_loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
from sklearn import metrics
# Measure accuracy
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)
y_eval = np.argmax(y_test, axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```