**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**GRADUATION THESIS**

**Developing a solution to the Teaching Assignment Problem using Reinforcement Learning**

**NGUYỄN QUANG MINH**

minh.nq194444@sis.hust.edu.vn

**Major: Data Science  
Specialization: Data Science And Artificial Intelligence**

**Supervisor:** Professor Trần Hải Anh

**School:** School of Information and Communications Technology

**ACKNOWLEDGMENT**

I would like to express my sincere gratitude for my family and friends for their support throughout the thesis process. I am also grateful to Professor Tran Hai Anh for his kind and thoughtful feedback during thesis process.

**ABSTRACT**

The ever-evolving cybersecurity landscape presents significant challenges for Intrusion Detection Systems (IDS), necessitating the ability to continually adapt to novel attacks and shifting attack patterns. Traditional IDS models often rely on complete retraining when encountering new attack types, a resource-intensive and inefficient process. This thesis proposes a novel Class Incremental Learning (CIL) approach to IDS, allowing the model to dynamically learn new attack classes while preserving previously acquired knowledge. Inspired by cutting-edge image classification CIL models, this method leverages a knowledge distillation framework. This framework facilitates the transfer of knowledge from a robust pre-trained model to a dynamically expanding network dedicated to handling new attack classes. Two dynamic networks from image classification CIL (DER and MEMO) are implemented and their performance compared across multiple IDS datasets. The thesis demonstrates the potential of model-centric CIL in constructing adaptive and robust IDS capable of effectively addressing the dynamic threat landscape.

Table of Contents

[Chapter 1. Introduction 8](#_Toc155298765)

[1.1. Problem statement 8](#_Toc155298766)

[1.2. Background and Problems of Research 8](#_Toc155298767)

[1.3. Research Objective and Conceptual Framework 8](#_Toc155298768)

[1.4. Contributions 9](#_Toc155298769)

[1.5. Organization of Thesis 9](#_Toc155298770)

[Chapter 2. Literature Review 9](#_Toc155298771)

[2.1. Scope of Research 9](#_Toc155298772)

[2.2. Related Work 9](#_Toc155298773)

[Chapter 3. Methodology 9](#_Toc155298774)

[3.1 Overview 9](#_Toc155298775)

[3.2 Problem Formulation 10](#_Toc155298776)

[3.2.1. Class Incremental Learning 10](#_Toc155298777)

[3.2.2. Exemplar set 10](#_Toc155298778)

[3.3 DER Net 11](#_Toc155298779)

[3.4 MEMO Net 13](#_Toc155298780)

[Chapter 4: Experimental Evaluation 13](#_Toc155298781)

[4.1. Benchmark Datasets 14](#_Toc155298782)

[4.1.1 KDD99 dataset 14](#_Toc155298783)

[4.1.2 CIC-IDS-2017 Dataset 15](#_Toc155298784)

[4.1.3. TON\_IoT dataset 16](#_Toc155298785)

[4.2. Evaluation Metrics 16](#_Toc155298786)

[4.3. Implementation Details 16](#_Toc155298787)

[Future work 18](#_Toc155298788)

[Conclusion 18](#_Toc155298789)

[Appendix 18](#_Toc155298790)

**LIST OF FIGURES**

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| Abbreviation | Definition |
| NIDS | Network Intrusion Detection System |
| CIL | Class Incremental Learning |
| DER | Dynamic Expandable Representation |
| MEMO | Memory Efficient Expandable Model |
| GAN | Generative Adversarial Network |

# Chapter 1. Introduction

## Problem statement

Traditional Network Intrusion Detection Systems have played a crucial role in safeguarding networks against malicious activities. However, their effectiveness suffers significantly as the threat landscape evolves at an alarming rate. New attack techniques and vulnerabilities emerge constantly, leaving tradition NIDS models struggling to adapt. This vulnerability stems from their reliance on pre-trained data, which cannot capture the ever-changing nature of cyber threats. This Thesis investigates Class Incremental Learning in NIDS, a promising paradigm that empowers detectors to continuously learn and adapt to novel attack patterns without the need for retraining on the entire dataset. By incorporating CIL techniques, NIDS can evolve alongside the threat landscape, maintaining robust detection capabilities in the face of unseen attacks.

## Background and Problems of Research

Traditional NIDS can be broadly categorized into two main approaches:

* Signature-based detection: This method relies on a pre-defined library of attack signatures, representing known malicious patterns. NIDS compares incoming packages against these signatures, flagging any matches as potential intrusions. While effective against well-documented attacks, this approach suffers from inherent limitations. Novel or zero-day attack which does not present in the signature library will evade detection, rendering the system vulnerable to the ever-evolving threat landscape.
* Anomaly-based detection: This approach attempts to identify deviations from normal network behavior, assuming that malicious activity will manifest as anomalous patterns. Statistical model or machine learning model are trained on historical network data to establish baselines of normality. Deviations from these baselines raise suspicion and trigger alerts. However, anomaly-based detection can be prone to false positives, flagging benign activities as suspicious due to their inherent deviations from the norm.

Despite their limitations, traditional NIDS remain valuable tools in network security domain. They offer real-time monitoring capabilities, providing early warnings of potential intrusions. However, their effectiveness hinges on the ability to keep pace with the dynamic threat landscape, a challenge that need the exploration of more advanced and adaptable approaches such as CIL, which forms the focus of this thesis.

## Research Objective and Conceptual Framework

The primary objective of this thesis is to develop and evaluate a novel Class Incremental Learning approach for Network Intrusion Detection System. My approach hinges on 2 modules: a base model pre-trained on known attack classes, and an incremental learning module for efficiently acquiring knowledge about new attacks. The base model initially detects known threats, while the incremental module constantly adapts and expands its ability to handle emerging attack classes without forgetting previously learned knowledge. This interaction continuously improved NIDS detection accuracy and adaptability, safeguarding networks against both familiar and unforeseen malicious activities.

## Contributions

In summary, we make the following contributions in this paper:

1. We apply DER, MEMO model from Image Classification field to Network Intrusion Detection domain.
2. We compare the results of those model in 3 dataset KDD99, TON\_IOT\_NETWORK, CIC-IDS2017.
3. We test the performance of these dataset in custom dataset and analyze the result.

## Organization of Thesis

The remainder of this paper is arranged as follows: Section II presents related work; Section III describes our proposed incremental learning algorithm; Section IV explains the experimental setup of this research, Section V presents a summary of the experimental results. Section VI discusses the challenges of implementing the proposed incremental algorithm in the network intrusion detection problem; and Section VII provides our conclusion and future work.

# Chapter 2. Literature Review

## 2.1. Scope of Research

This section outlines the scope of this thesis, which focuses on applying a specific CIL approach to NIDS. While various NIDS approaches exist, this research prioritizes the dynamic approach, motivated by its state-of-the-art performance on reason image classification datasets. The proposed method leverages backbone expansion, dynamically adding a new backbone for each new task and aggregating features through a large fully connected layer. This focused scope enables a deep exploration of the dynamic approach’sefficiency in the context of NIDS, while acknowledging the broader CIL landscape and the potential for future research on alternative approaches.

I build two model using the idea from two paper DER: Dynamically Expandable Representation for Class Incremental Learning and A MODEL OR 603 EXEMPLARS: TOWARDS MEMORY-

EFFICIENT CLASS-INCREMENTAL LEARNING

## 2.2. Related Work

This section will examine and evaluate recent advancements in CIL and NIDS domains.

### 2.2.1. Class Incremental Learning

There are various approach in Class Incremetal Learning.

Data-centric

Data-centric methods seek help from extra data, e.g., An intuitive way is to utilize former data for rehearsal, which enables the model to review former classes and resist forgetting. On the other hand, some works build regularization terms with the extra data, aiming to control the optimization direction to avoid catastrophic forgetting. Some works argue with a limited size of exemplars, the exemplar should contain the informative one. [] suggests sampling exemplars with high prediction entropy and near the decision boundary. Some works said that since exemplars are raw images, directly saving a set of instances may consume enormous memory costs. To this end, several works are proposed to build a memory-efficient replay buffer.

[46] argues that extracted features are with lower dimension than raw images and proposes to save features in the exemplar set to release the burden. Other work proposes to keep low-fidelity images instead of raw images to save memory. However, since the distributions of extracted features and low-fidelity images may differ from the raw images, an extra adaptation process is needed for these methods, adding to the algorithm’s complexity.

# Chapter 3. Methodology

## Overview

In this section, I will introduce the dynamic model architectures used in the CIL problem. First, I will briefly discuss the architectures I user. However, these architectures are originally from image classification domain of CIL. Therefore, I will present the network architecture I use for the NIDS problem.

## Problem Formulation

The following formulations are derived from Class Incremental Learning from paper Deep Class-Incremental Learning: A Survey

### 3.2.1. Class Incremental Learning

Class Incremental Learning aims to learn from an evolutive stream with incoming new classes. Assume there is a sequence of B training tasks without overlapping classes, where is the -th incremental step with training instances. is an instance of class . is the label space of task , where for . We can only access data from when training task . The ultimate goal of CIL is to continually build a classification model for all classes. In other words, the model should not only acquire the knowledge from current task but also preserve the knowledge from former tasks. After each task, the trained model is evaluated over all seen classes .

### 3.2.2. Exemplar set

In accordance with the definition provided in 3.2.1, the model’s data accessibility within each incremental task is restricted to the dataset designed for that particular task, represented as Db. This constraint serves a dual purpose: to uphold the integrity of user privacy and to mitigate the demands placed on storage resources. However, under certain circumstances, a degree of flexibility is permitted, allowing the model to retain a carefully curated collection of exemplars – a set of selected instances representative of previous tasks, thereby facilitating a measure of knowledge preservation across tasks.

#### Exemplar set

Exemplar set is an extra collection of instances from former tasks . With the help of the exemplar set, the model can utilize for the update within each task. The model manages the exemplar set after the training process of each task.

#### Exemplar set management

Since the data stream is evolving, there are two main strategies to manage the exemplar set in CIL. The first way is to keep a fix number of exemplars per class, e.g., per class. Under such circumstances, the size of the exemplar set will grow as the data stream evolves – the model keeps R[Yb] after the b-th task. This will result in a linear growing memory budget, which is inapplicable in real-world learning systems. To this end, another strategy advocates saving a fixed number of exemplars, e.g., . The model keeps instances per class, where [.] denotes floor function. It helps to keep a fixed size of exemplars in the memory and release the storage burden. I use the second strategy to organize the exemplar set.

#### Exemplar selection

Exemplars are representative instances of each known class, which needed to be selected from the entire training set. An intuitive way to choose the exemplars is to randomly sample exemplars set for each class, which results in diverse instances. By contrast, a commonly used strategy is called herding, aiming to select the most representative ones of each class. Given the instance set from class , herding first calculates the class center with current embedding.

Afterward, it calculates and rank the distance of each instance to the class center in ascending order. The exemplars are then selected based on ranking, e.g., the top M/yb instances with the least distance. Since the class center can be seen as the most representative pattern of each class, selecting exemplars near the class center also enhance the representativeness of exemplars. Herding is now a commonly used strategy to select exemplars in CIL, and I also use this in my experiments.

## 3.3 DER Net

In the context of CIL, the recently published paper “DER: Dynamic Expandable Representation for Class Incremental Learning” introduces a novel technique known as “expandable representation”. This innovative method has enabled the DER network to achieve breakthrough performance, outperforming all other state-of-the-art networks by a significant margin in a typical CIL setup.

A diagram of a task

Description automatically generated

Figure 1. Illustration of Dynamic Expandable Representation Net

At step , the model is composed of a super feature extractor is build by expanding the feature extractor with a newly created extractor . Specifically, given an image , the feature extracted by is obtained by concatenation as follows

Here reuse the previous and encourage the new extractor to learn only the novel aspect of new classes. The feature is then fed into the classifier to make predictions as follows

= Softmax())

Then the prediction = argmax(). The classifier is designed to match its new input and output dimension for step . The parameters of for the old features are inherited from to retain old knowledge and its newly added parameters are randomly initialized.

To reduce catastrophic forgetting, we freeze the learned function at step t, as it captures the intrinsic structure of previous data. In detail, the parameters of last step super-feature extractor , and the statistics of Batch Normalization are not updated. Besides, we instantiate Ft with Ft-1 as initialization to reuse previous knowledge for fast adaptation and forward transfer.

Detail Flow of DER Net

Train dataset of current task

Exemplar set

Classifier :

Random initilize

Feature extractor :

Random initilize

Train dataset of current task

Exemplar set

Classifier :

Random initilize

Feature extractor

(Freeze)

Feature extractor :

Random initilize

Train dataset of current task

Exemplar set

Classifier :

Random initilize

Feature extractor

(Freeze)

Feature extractor :

Random initilize

Copy Weight Align

In the DER network used for Image Classification CIL, the networks contains three stages, Each stages contains 5 ResnetBasicBlock, which structure be Conv2d layers -> Batch Normalization -> Conv2d layers -> Batch Normalization. The last stages of network will parse to fully connected layers for classification. In my experiments, I use simple ANN net with 4 layers, connected with a fully connected layers for classification. The input layer be depending on the features size of the dataset.

## 3.4 MEMO Net

MEMO Model: (Memory-Efficient Expandable Model)

The model from Paper A Model or 603 Exemplars: Toward Memory-Efficient Class-Incremental Learning try to solve the problem of expanding memory by answering the question: Given the same memory budget, if we share the generalized block and only extend specialized blocks for new tasks, can we further improve the performance?

Concretely, we redefine the model structure by decomposing the embedding module into specialized and generalized blocks. Specialized blocks correspond to the deep layers in the network, while generalized blocks corresponds to the rest shallow layers. We argue that the features of shallow layers can be shared across different incremental stages, i.e., there is no need to create an extra model.



Figure 2. Illustration of MEMO Net

# Chapter 4: Experimental Evaluation

## 4.1. Benchmark Datasets

The models are experimented on three datasets: KDD’99, CIC-IDS-2017 and ToN\_IoT\_Network dataset. All of them are publicly available and widely used in Network Intrusion Detection Domain.

### 4.1.1 KDD99 dataset

**The KDD99 dataset, a popular benchmark for intrusion detection systems[1], was born in 1999's KDD Cup, a competition to build network intrusion models. Fueling this competition was data from the 1998 DARPA program, where MIT Lincoln Labs mimicked a real Air Force LAN for nine weeks, peppering it with attacks and capturing the raw traffic.This data was then transformed into roughly five million connection records, each representing a sequence of data flowing between two IP addresses under a specific protocol.Importantly, each record is labeled as either normal or a specific type of attack, making it a valuable resource for training and evaluating intrusion detection systems.**

**The KDD99 dataset categorizes network connection features into three distinct groups [4]:**

* **Basic features: Encompasses all directly extractable attributes from a TCP/IP connection.**
* **Traffic features: These features, computed within a designated window interval, further analyze network activity for anomaly detection. They are further subdivided into same host features and same service features.**
* **Content features: Those features were extracted by using domain knowledge for detecting suspicious behaviors in data portions of the packages.**

**Details of all features can be found at [3].**

There are four main categories of attack and 24 attack types:

|  |  |
| --- | --- |
| Category | Attack |
| DOS: Denial-Of-Service | Back, Pod, Land, Teardrop, Smurf, Neptune |
| R2L: unauthorized access from a remote machine | ftp\_write, guess\_passwd, imap, multihop, phf, spy, warezclient, warezmaster |
| U2R: unauthorized access to local superuser privileges | perl, buffer\_overflow, loadmodule, rootkit, |
| Probing: surveillance and other probing | Ipsweep, nmap, portsweep |

In this experiment, I use file kddcup.data\_10\_percent.gz [2] for training and testing purpose. This is a 10% subset of the full dataset. This dataset contains 494021 records in totals, in which 280790 records recognized as smurf attack.

[1] A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015

[2] <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

[3] <https://kdd.ics.uci.edu/databases/kddcup99/task.html>

[4] A detailed analysis of the KDD CUP 99 data set

### 4.1.2 CIC-IDS-2017 Dataset

The Canadian Institute for Cybersecurity recognizes the significant limitations of many networks intrusion detection (IDS) dataset since 1998. These limitations often include outdated attacks models, limited traffic diversity, incomplete attack coverage and lack of feature sets and metadata. To address these shortcomings, CIC has developed the CIC-IDS-2017 dataset. This comprehensive dataset offers several key advantages: Real-World Reflectance, Network Insights, Naturalistic Background Traffic. The network traffic was captured over a five-day period, starting at 9 a.m. on Monday, July 3, 2017, and concluding at 5 p.m. on Friday, July 7, 2017. Monday’s capture exclusively comprised benign traffic, while a spectrum of attacks was executed during working hours on Tuesday, Wednesday, Thursday, and Friday. These attacks encompassed Brute Force FTP, Brute Force SSH, Dos, Heartbleed, Web Attack, Infiltration, Botnet and DDoS.

The CIC-IDS-2017 dataset meticulously analyzes network flows using CICFlowMeter, yielding a comprehensive set of 84 features that provide a granular overview of network activity. These features can be broadly categorized into:

Fundamental Flow Information: This includes essential details such as IP addresses, timestamps, protocols used, and the duration of each flow.

Packet-Level Insights: These features delve into the number of packets exchanged and their size characteristics, offering insights into the granularity of network communication.

Flow-Specific Statistics: This category encompasses both byte-level statistics, revealing data volume and directionality, as well as time-related statistics, shedding light on flow duration and idle periods.

Inter-Arrival Time Patterns: These features calculate the time gaps between consecutive packets within a flow, potentially unmasking unusual patterns associated with malicious activities.

Flag Distribution: The frequency of specific flags (SYN, FIN, RST, PSH, ACK) within a flow provides valuable insights into the communication protocol and potential deviations from standard patterns.

For a comprehensive exploration of each feature, please refer to reference [3].

### 4.1.3. TON\_IoT dataset

The TON\_IoT dataset presents a valuable resource for evaluating the efficiency of Artificial Intelligent (AI) powered cybersecurity solution in the context of Internet of Things. This rich repository comprises heterogeneous data captured from a meticulously crafted testbed encompassing diverse components like virtual machines, sensor-equipped physical systems, cloud platforms, etc. Its comprehensive coverage of real-world cyberattacks, ranging from denial-of-service attack to ransomware, makes it particularly relevant for developing and refining robust threat detection models. The directories of the TON\_IoT datasets include IoT datasets, Network datasets, Linux datasets and Window datasets. In my experiment, I use Network Dataset, which were collected in the package capture (pcap) formats and processed by network analysis tool ZEEK.

## 4.2. Evaluation Metrics

The accuracy of the CIL model can be measured in several ways. One way is to look at the accuracy after each task. However, the accuracy of the CIL model can decrease as it is updated with more tasks. Therefore, the accuracy after the last stage is the best way to measure the overall accuracy of the model.[1] Consequently, last stage accuracy was employed as the primary evaluation metric within context of the presented experiments.

## 4.3. Implementation Details

4.3.1 Data Preprocessing

One-Hot Encoding was use for categorical features. One-Hot Encoding creates a separate binary dimension for each category within a feature. Each dimension takes a value of 1 for its corresponding category and 0 for all others. Then we use Z-score normalization to normalize data. Z-score is our choice since the method is more outliner-Resistant. Mathematically, Z-score was calculated as followed.

x' = (x - μ) / σ

where μ is mean and σ is standard deviation of data.

By the end, the dataset was divided into train and test data with a ration of 70:30, where 30% was kept aside for testing model’s generalizability. All of data preprocessing step are performs on pandas Dataframe.

Training parameter

Epoch

Setup

KDD99 dataset

Exemplar size: 2000 instances

KDD99 data setup

Finetune setup

Number of tasks: 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training instances / Testing instances | Task 0 | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| normal. | 77931 / 19347 | 0/ 19347 | 0/ 19347 | 0/ 19347 | 0 / 19347 | 0 / 19347 |
| neptune. | 85937 / 21264 | 0 / 21264 | 0 / 21264 | 0 / 21264 | 0 / 21264 | 0 / 21264 |
| satan. | 0 / 0 | 1265 / 324 | 0 / 324 | 0 / 324 | 0 / 324 | 0 / 324 |
| portsweep. | 0 / 0 | 846 / 194 | 0 / 194 | 0 / 194 | 0 / 194 | 0 / 194 |
| back. | 0 / 0 | 0 / 0 | 1773 / 430 | 0 / 430 | 0 / 430 | 0 / 430 |
| nmap. | 0 / 0 | 0 / 0 | 192 / 39 | 0 / 39 | 0 / 39 | 0 / 39 |
| pod. | 0 / 0 | 0 / 0 | 0 / 0 | 211 / 53 | 0 / 53 | 0 / 53 |
| smurf. | 0 / 0 | 0 / 0 | 0 / 0 | 224309 / 56481 | 0 / 56481 | 0 / 56481 |
| warezclient. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 819 / 201 | 0 / 201 |
| teardrop. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 799 / 180 | 0 / 180 |
| ipsweep. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 991 / 256 |

Backbone expansion setup

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training instances / Testing instances | Task 0 | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| normal. | 77931 / 19347 | 1000 / 19347 | 500 / 19347 | 333 / 19347 | 250 / 19347 | 200 / 19347 |
| neptune. | 85937 / 21264 | 1000 / 21264 | 500 / 21264 | 333 / 21264 | 250 / 21264 | 200 / 21264 |
| satan. | 0 / 0 | 1265 / 324 | 500 / 324 | 333 / 324 | 250 / 324 | 200 / 324 |
| portsweep. | 0 / 0 | 846 / 194 | 500 / 194 | 333 / 194 | 250 / 194 | 200 / 194 |
| back. | 0 / 0 | 0 / 0 | 1773 / 430 | 333 / 430 | 250 / 430 | 200 / 430 |
| nmap. | 0 / 0 | 0 / 0 | 192 / 39 | 192 / 39 | 192 / 39 | 192 / 39 |
| pod. | 0 / 0 | 0 / 0 | 0 / 0 | 211 / 53 | 211 / 53 | 200 / 53 |
| smurf. | 0 / 0 | 0 / 0 | 0 / 0 | 224309 / 56481 | 250 / 56481 | 200 / 56481 |
| warezclient. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 819 / 201 | 200 / 201 |
| teardrop. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 799 / 180 | 200 / 180 |
| ipsweep. | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 991 / 256 |

CIC-IDS-2017 data setup

Baseline

|  |  |  |  |
| --- | --- | --- | --- |
| Train Instance / Test Instance | Task 0 | Task 1 | Task 2 |
| BENIGN | 351861 / 88170 | 0 / 88170 | 0 / 88170 |
| DoS Hulk | 184926 / 184926 | 0 / 184926 | 0 / 184926 |
| DoS SlowHttptest | 0 / 0 | 4409 / 1090 | 0 / 1090 |
| DoS slowloris | 0 / 0 | 4682 / 2017 | 0 / 2017 |
| Heartbleed | 0 / 0 | 0 / 0 | 8 / 3 |
| DoS GoldenEye | 0 / 0 | 0 / 0 | 8276 / 2017 |

|  |  |  |  |
| --- | --- | --- | --- |
| Train Instance / Test Instance | Task 0 | Task 1 | Task 2 |
| BENIGN | 351861 / 88170 | 1000 / 88170 | 500 / 88170 |
| DoS Hulk | 184926 / 184926 | 1000 / 184926 | 500 / 184926 |
| DoS SlowHttptest | 0 / 0 | 4409 / 1090 | 500 / 1090 |
| DoS slowloris | 0 / 0 | 4682 / 2017 | 500 / 2017 |
| Heartbleed | 0 / 0 | 0 / 0 | 8 / 3 |
| DoS GoldenEye | 0 / 0 | 0 / 0 | 8276 / 2017 |

ToN\_IoT

4.4. Comparison

Base\_5\_Inc\_5

|  |  |  |  |
| --- | --- | --- | --- |
|  | KDD99 | CIC-IDS-2017 | TON\_IOT\_Network |
| Baseline | 0.76 | 98.82 | 73.74 |
| DER | 94.03 | 97.35 | 65.04 |
| MEMO | 98.0 | 99.21 | 24.61 |

Base\_2\_Inc\_2

|  |  |  |  |
| --- | --- | --- | --- |
|  | KDD99 (after remove class with number of instances < 200) | CIC-IDS-2017 | TON\_IOT\_Network |
| Baseline | 0.39 | 1.85 | 25.99 |
| DER | 90.0 | 67.54 | 68.96 |
| MEMO | 97.55 | 63.25 | 99.84 |

KDD99 dataset accuracy

Setting:

Exemplar: 2000

First epoch: 200

Next epochs: 150

Accuracy curve

Der

Accuracy curve: [100.0, 99.02, 98.65, 96.72, 91.46, 83.3]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training instances / Testing instances | Task 0 | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| normal. | 1.0 | 1.0 | 0.99 | 0.91 | 0.74 | 0.26 |
| neptune. | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| satan. |  | 0.0 | 0.88 | 0.96 | 0.95 | 0.96 |
| portsweep. |  | 0.5 | 0.84 | 0.98 | 0.99 | 1.0 |
| back. |  |  | 0.98 | 0.91 | 1.0 | 0.99 |
| nmap. |  |  | 0.06 | 0.02 | 0.01 | 0.01 |
| pod. |  |  |  | 0.85 | 0.28 | 0.56 |
| smurf. |  |  |  | 1.0 | 1.0 | 1.0 |
| warezclient. |  |  |  |  | 0.17 | 0.05 |
| teardrop. |  |  |  |  |  | 0.99 |
| ipsweep. |  |  |  |  |  | 0.85 |

Memo [100.0, 99.97, 97.29, 97.75, 95.61, 86.14]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training instances / Testing instances | Task 0 | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| normal. | 1.0 | 1.0 | 0.97 | 0.94 | 0.88 | 0.46 |
| neptune. | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| satan. |  | 0.99 | 0.95 | 0.98 | 0.97 | 0.98 |
| portsweep. |  | 0.99 | 0.96 | 0.81 | 0.68 | 0.74 |
| back. |  |  | 0.96 | 0.93 | 1.0 | 0.86 |
| nmap. |  |  | 0.07 | 0.04 | 0.07 | 0.15 |
| pod. |  |  |  | 0.99 | 0.03 | 0.04 |
| smurf. |  |  |  | 1.0 | 1.0 | 1.0 |
| warezclient. |  |  |  |  | 0.41 | 0.04 |
| teardrop. |  |  |  |  | 0.01 | 1.0 |
| ipsweep. |  |  |  |  |  | 0.99 |

The CIC-IDS-2017 dataset is a widely used dataset in the field of network intrusion detection system (NIDS). Current models that run on the CIC-IDS-2017 dataset achieve very good results. These good results are due to spliting of the dataset into a training set and test set. Normally, this division prevents the model from seeing instances of the test set in advance, thereby evaluating the generalization of the model. However, in the context of NIDS (for example KDD99, NSLKDD, CIC-IDS-2017 or most recently TON\_IOT\_Net), these datasets are all formed in a simulated environment, with machines running attack using scripts. I make two assumptions that:

* Models trained in a simulated environment will only classify well in that simulated environment.
* Models learn the properties of the attack script, not the essence of the attack type. In the case of using different attack tool, the model will not be able to identify it.

The dataset was constructed through a multi-stage process involving attack simulation, traffic capture, and flow generation. Initially, A DoS attack was simulated using the wrk tool, a benchmark tool that generates high volumes of HTTP requests to a target server. Concurrently, tcpdump command was employed to capture the network traffic generated during the attack. This captured traffic, stored in a pcap file, encapsulated a comprehensive representation of the attack dynamics. Subsequently, the captured pcap file was processed through CICFlowMeter, a tool renowned for its utilzation in the creation of CIC-IDS-2017 dataset. CICFlowMeter meticulosly parsed the network traffic and generated a comprehensive set of network flows, providing a granular view of the communication patterns within the captured data. To facilitate the subsequent classification tasks, a labeling process was undertaken. Network flows exhibiting Destination IP or Source IP matching the IP address of the attack machine were identified and labed as attack traffic. This labeling strategy enabled a clear distinction between legitmate network flows and those associated with the simulated DoS attack. For a more comprehensive exploration of the dataset creation methodology, refer to [1].

For dataset created with single machine attack, I use a simple ANN model with 3 layers. The whole dataset has 2529 records, in which 1007 records are normal flow.

<https://github.com/NguyenQuangMinh0504/IDS-Dataset>

# Future work

# Conclusion

# Appendix

Features set of each dataset.

CIC-IDS-2017 dataset

|  |  |  |
| --- | --- | --- |
| Flow ID | Bwd IAT Total | Fwd Header Length.1 |
| Source IP | Bwd IAT Mean | Fwd Avg Bytes/Bulk |
| Source Port | Bwd IAT Std | Fwd Avg Packets/Bulk |
| Destination IP | Bwd IAT Max | Fwd Avg Bulk Rate |
| Destination Port | Bwd IAT Min | Bwd Avg Bytes/Bulk |
| Protocol | Fwd PSH Flags | Bwd Avg Packets/Bulk |
| Timestamp | Bwd PSH Flags | Bwd Avg Bulk Rate |
| Flow Duration | Fwd URG Flags | Subflow Fwd Packets |
| Total Fwd Packets | Bwd URG Flags | Subflow Fwd Bytes |
| Total Backward Packets | Fwd Header Length | Subflow Bwd Packets |
| Total Length of Fwd Packets | Bwd Header Length | Subflow Bwd Bytes |
| Total Length of Bwd Packets | Fwd Packets/s | Init\_Win\_bytes\_forward |
| Fwd Packet Length Max | Bwd Packets/s | Init\_Win\_bytes\_backward |
| Fwd Packet Length Min | Min Packet Length | act\_data\_pkt\_fwd |
| Fwd Packet Length Mean | Max Packet Length | min\_seg\_size\_forward |
| Fwd Packet Length Std | Packet Length Mean | Active Mean |
| Bwd Packet Length Max | Packet Length Std | Active Std |
| Bwd Packet Length Min | Packet Length Variance | Active Max |
| Bwd Packet Length Mean | FIN Flag Count | Active Min |
| Bwd Packet Length Std | SYN Flag Count | Idle Mean |
| Flow Bytes/s | RST Flag Count | Idle Std |
| Flow Packets/s | PSH Flag Count | Idle Max |
| Flow IAT Mean | ACK Flag Count | Idle Min |
| Flow IAT Std | URG Flag Count | Label |
| Flow IAT Max | CWE Flag Count |  |
| Flow IAT Min | ECE Flag Count |  |
| Fwd IAT Total | Down/Up Ratio |  |
| Fwd IAT Mean | Average Packet Size |  |
| Fwd IAT Std | Avg Fwd Segment Size |  |
| Fwd IAT Max | Avg Bwd Segment Size |  |
| Fwd IAT Min |  |  |

Fwd Header Length.1 is a duplicate feature of Fwd Header Length.

|  |  |
| --- | --- |
| BENIGN | 440031 |
| DoS Hulk | 231073 |
| DoS GoldenEye | 10293 |
| DoS slowloris | 5796 |
| DoS Slowhttptest | 5499 |
| Heartbleed | 11 |

Missing data 1008 / 692703

Ton\_Iot\_Network Dataset

|  |  |  |
| --- | --- | --- |
| ts | dst\_ip\_bytes | http\_trans\_depth |
| src\_ip | dns\_query | http\_method |
| src\_port | dns\_qclass | http\_uri |
| dst\_ip | dns\_qtype | http\_version |
| dst\_port | dns\_rcode | http\_request\_body\_len |
| proto | dns\_AA | http\_response\_body\_len |
| service | dns\_RD | http\_status\_code |
| duration | dns\_RA | http\_user\_agent |
| src\_bytes | dns\_rejected | http\_orig\_mime\_types |
| dst\_bytes | ssl\_version | http\_resp\_mime\_types |
| conn\_state | ssl\_cipher | weird\_name |
| missed\_bytes | ssl\_resumed | weird\_addl |
| src\_pkts | ssl\_established | weird\_notice |
| src\_ip\_bytes | ssl\_subject | label |
| dst\_pkts | ssl\_issuer | type |

|  |  |
| --- | --- |
| normal | 300000 |
| scanning | 20000 |
| dos | 20000 |
| injection | 20000 |
| ddos | 20000 |
| password | 20000 |
| xss | 20000 |
| ransomware | 20000 |
| backdoor | 20000 |
| mitm | 1043 |

Missing data: 0/461043

KDD99

|  |  |  |
| --- | --- | --- |
| duration | su\_attempted | same\_srv\_rate |
| protocol\_type | num\_root | diff\_srv\_rate |
| service | num\_file\_creations | srv\_diff\_host\_rate |
| flag | num\_shells | dst\_host\_count |
| src\_bytes | num\_access\_files | dst\_host\_srv\_count |
| dst\_bytes | num\_outbound\_cmds | dst\_host\_same\_srv\_rate |
| land | is\_host\_login | dst\_host\_diff\_srv\_rate |
| wrong\_fragment | is\_guest\_login | dst\_host\_same\_src\_port\_rate |
| urgent | count | dst\_host\_srv\_diff\_host\_rate |
| hot | srv\_count | dst\_host\_serror\_rate |
| num\_failed\_logins | serror\_rate | dst\_host\_srv\_serror\_rate |
| logged\_in | srv\_serror\_rate | dst\_host\_rerror\_rate |
| num\_compromised | rerror\_rate | dst\_host\_srv\_rerror\_rate |
| root\_shell | srv\_rerror\_rate | outcome |

|  |  |
| --- | --- |
| smurf | 280790 |
| neptune | 107201 |
| normal | 97278 |
| back | 2203 |
| satan | 1589 |
| ipsweep | 1247 |
| portsweep | 1040 |
| warezclient | 1020 |
| teardrop | 979 |
| pod | 264 |
| nmap | 231 |
| guess\_passwd | 53 |
| buffer\_overflow | 30 |
| land | 21 |
| warezmaster | 20 |
| imap | 12 |
| rootkit | 10 |
| loadmodule | 9 |
| ftp\_write | 8 |
| multihop | 7 |
| phf | 4 |
| perl | 3 |
| spy | 2 |

Missing data: 0/494021