HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Kiều Công Minh

ỨNG DỤNG REPRESENTATION LEARNING PHÁT HIỆN TẤN CÔNG BOTNET

Chuyên ngành: Hệ thống thông tin.
Mã số: 8.48.01.04

TÓM TẮT LUẬN VĂN THẠC SĨ

TP.HCM - NĂM 2023

Luận văn được hoàn thành tại:

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Người hướng dẫn khoa học: TS Nguyễn Hồng Sơn

(Ghi rõ học hàm, học vị)

Phản biện 1:

Phản biện 2:

Luận văn sẽ được bảo vệ trước Hội đồng chấm luận văn thạc sĩ tại Học viện Công nghệ Bưu chính Viễn thông

Vào lúc: ngày tháng năm 2023

Có thể tìm hiểu luận văn tại:

- Thư viện của Học viện Công nghệ Bưu chính Viễn thông.

PHẦN MỞ ĐẦU

1. Lý do chọn đề tài

Song song với quá trình phát triển mạnh mẽ của Internet là kèm theo những mối đe dọa về tấn công mạng cũng xuất hiện ngày càng nhiều, trong số đó là mối đe dọa về Botnet. Khai thác, phát tán mã độc, phát tán thư rác số lượng lớn, tấn công từ chối dịch vụ DDoS và đặc biệt là tấn công APT là những hành vi nguy hiểm thường thấy của Botnet, nó đã gây ra những thiệt hại không nhỏ về hệ thống mạng và sự mất mát dữ liệu của người dung, dẫn đến thiệt hại về kinh tế, xã hội của các cá nhân, tổ chức, doanh nghiệp và cơ quan hành chính nhà nước.

Với sự phát triển mạnh mẽ của trí tuệ nhân tạo trong những năm gần đây, đặc biệt là các kỹ thuật máy học, đã mở ra như một giải pháp rất có tiềm năng cho việc ứng dụng phát hiện các tấn công mạng Botnet với độ chính xác và đạt hiệu quả cao hơn các phương pháp trước đây. Trong đó mô hình dựa vào phương pháp representatin learning có thể phát huy nhiều ưu điểm cho bài toán này

2. Tổng quan về vấn đề nghiên cứu

Nhìn chung có nhiều nghiên cứu đưa ra mô hình phân tích đánh giá dựa trên các đặc trưng ứng dụng các thuật toán trong máy học để phát hiện các loại botnet. Kết quả nghiên cứu khá đa dạng và đều cho kết quả khả quan. Tuy nhiên, việc nâng cao hiệu quả hơn nữa để phát hiện các

tấn công botnet với mức độ phát triển công nghệ vũ bảo như ngày nay là một nhu cầu thiết yếu. Trong đó việc ứng dụng máy học với kỹ thuật representation learning để phát hiện botnet cũng được xem như là một giải pháp cần thiết.

3. Mục đích nghiên cứu

Mục tiêu chính của nghiên cứu: "Xây dựng mô hình máy học sử dụng phương pháp representation learning để phát hiện tấn công botnet nhằm nâng cao độ chính xác của phát hiện".

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu là hệ thống phát hiện tấn công bot net bằng máy học thông qua phương pháp representation learning, cụ thể là:

Nghiên cứu về representation learning trong học máy.

Nghiên cứu chính là phát hiện tấn công Botnet thông qua máy học sử dụng phương pháp representation learning.

Nghiên cứu các kỹ thuật máy học phổ biến như R, MatLab, Python… để xây dựng mô hình phát hiện tấn công bằng phương pháp máy học.

5. Phương pháp nghiên cứu

Phương pháp luận: Dựa trên cơ sở là các lý thuyết về tấn công mạng Botnet, phương pháp representation learning trong máy học.

Dự kiến ứng dụng dựa trên tập dữ liệu phân loại các dấu vết các lưu lượng truy cập mạng hợp lệ và lưu lượng

truy cập mạng của Botnet hoặc sử dụng tập dữ liệu botnet CTU-13 tại https://mcfp.felk.cvut.cz/publicDatasets/CTU-13-Dataset/ để nghiên cứu, ứng dụng.

Phương pháp đánh giá dựa trên cơ sở toán học: Trên cơ sở lý thuyết của representation learning, đưa ra đề xuất thuật toán thông qua việc xây dựng mô hình detect botnet, có khả năng phát hiện tấn công Botnet với độ chính xác cao. Từ đó đánh giá hiệu quả của mô hình.

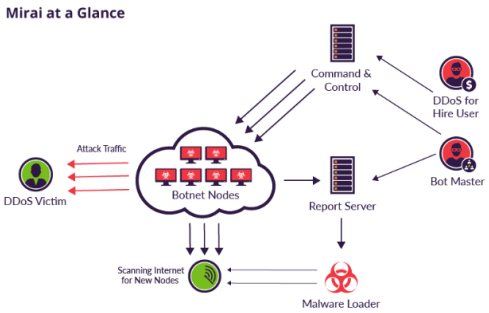
Phương pháp đánh giá bằng mô phỏng thực nghiệm: Xây dựng mô hình mô phỏng và đưa vào thực nghiệm.

CHƯƠNG 1: TỔNG QUAN TẤN CÔNG BOTNET VÀ REPRESENTATION LEARNING

1.1.Tổng quan về tấn công Botnet

1.1.1.Botnet là gì?

Botnet là một mạng lưới các máy tính bị xâm nhập, được điều khiển từ xa, được sử dụng cho các mục đích xấu. Các hệ thống hoặc máy chủ bị nhiễm trong một mạng botnet được gọi là Bots và bộ điều khiển của một mạng botnet được gọi là bot-master. Là mạng lớp phủ khổng lồ, botnet là nền tảng hỗ trợ phức tạp để tấn công người khác bằng cách duy trì quyền kiểm soát một số lượng lớn máy tính và các thiết bị khác. Vòng đời của botnet bao gồm một số bước bắt đầu từ khi bot-master lây nhiễm cho nạn nhân qua phần mềm độc hại, v.v… Bot bị nhiễm kết nối với các C&C bằng cách sử dụng HTTP, IRC hoặc bất kỳ giao thức khác. Bot-master gửi lệnh đến bot bằng máy chủ C&C và từ từ tạo ra một đội quân bot.

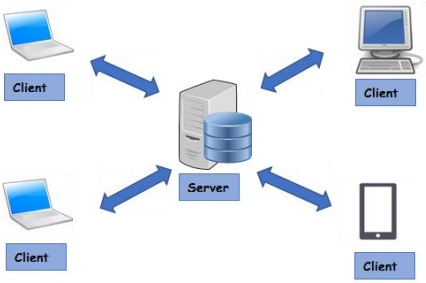


Hình 1.1: Sơ đồ cách thức tấn công của Botnet

1.1.2 Cấu trúc của Botnet

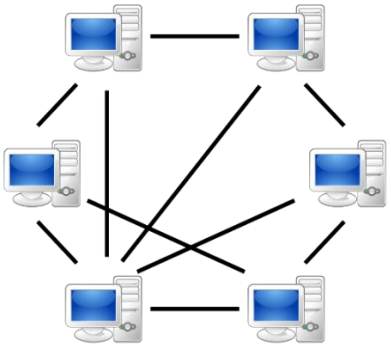
Một Botnet thường có một trong hai dạng như sau:

+ Mô hình client - server (mô hình khách - chủ)



Hình 1.2: Mô hình client – server

+ Mô hình peer-to-peer (ngang hàng)



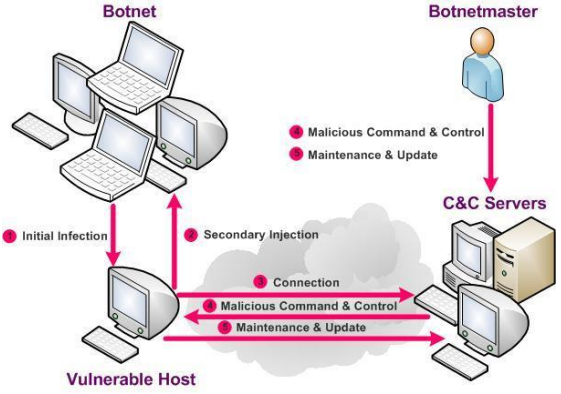
Hình 1.3: Mô hình peer-to-peer

1.1.3. Các loại tấn công Botnet

- Distributed Denial of Operations Service (DDoS)
- Phát tán thư rác và giám sát lưu lượng
- Keylogging
- Đánh cắp danh tính hàng loạt
- Lạm dụng việc trả tiền cho mỗi lần nhấp
- Lây lan botnet
- Phần mềm quảng cáo

1.2 Các đặc trưng của Botnet

Botnet thường được phân loại theo kiến trúc chỉ huy và điều khiển của chúng. Theo kiến trúc lệnh và điều khiển của chúng, các botnet có thể được phân loại thành botnet dựa trên IRC, dựa trên HTTP, dựa trên DNS hoặc ngang hàng (P2P). Các botnet P2P sử dụng giao thức P2P gần đây để tránh một điểm lỗi duy nhất. Hơn nữa, các botnet P2P khó xác định vị trí, tắt máy, giám sát và chiếm quyền điều khiển hơn.



Hình 1.4: Vòng đời của Botnet

1.3. Tổng quan các kỹ thuật phát hiện và cơ chế phòng vệ Botnet

1.3.1. Phát hiện dựa trên chữ ký - Signature-based Detection

Kiến thức về các chữ ký hữu ích và hành vi của các mạng botnet hiện có rất hữu ích cho việc phát hiện botnet. Ví dụ, Snort là một hệ thống phát hiện xâm nhập mã nguồn mở (IDS) theo dõi lưu lượng mạng để tìm ra các dấu hiệu

xâm nhập. Giống như hầu hết các hệ thống IDS, Snort được cấu hình với một tập hợp các quy tắc hoặc chữ ký để ghi lại lưu lượng truy cập được cho là đáng ngờ. Tuy nhiên, các kỹ thuật phát hiện dựa trên chữ ký có thể được sử dụng để phát hiện các mạng botnet đã biết. Vì vậy, giải pháp này không hữu ích cho các bot không xác định.

1.3.2. Phát hiện dựa trên điểm bất thường - Aomaly-based Detection

Các kỹ thuật phát hiện dựa trên sự bất thường cố gắng phát hiện các botnet dựa trên một số điểm bất thường về lưu lượng mạng như độ trễ mạng cao, lưu lượng lớn, lưu lượng truy cập trên các cổng bất thường và hành vi hệ thống bất thường có thể cho thấy sự hiện diện của bot độc hại trong mạng

1.4. Tổng quan các ứng dụng học máy về phát hiện tấn công Botnet

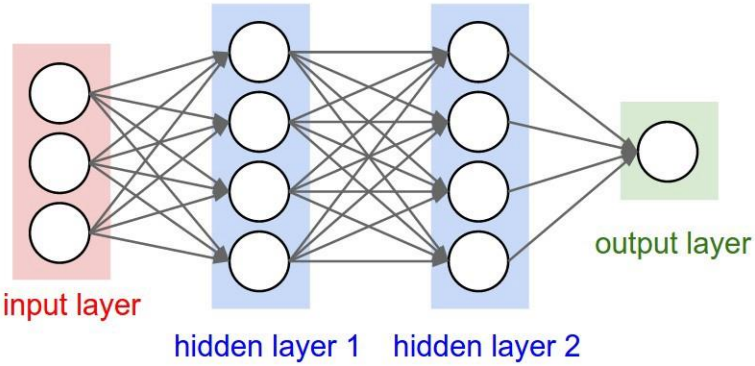
Một số thuật toán machine learning hiện nay thường được sử dụng để phát hiện tấn công botnet là gồm các thuật toán: Naïve Bay. Một số thuật toán machine learning hiện nay thường được sử dụng để phát hiện tấn công botnet là gồm các thuật toán: Naïve Bayes (NB), K-Nearest Neighbor (k-NN), Support Vector Machine (SVM), Decision Tree (DT)

1.5. Mạng nơ-ron và Deep Learning

1.5.1.Mạng nơ-ron

Neural network hay còn gọi là mạng nơ-ron được xây dựng dựa trên mạng nơ-ron sinh học. Nó là một mạng

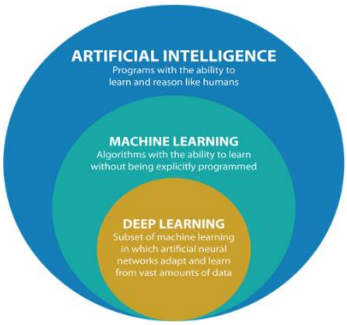
lưới gồm các nút được kết nối với nhau – gọi là nơ-ron và các cạnh nối chúng lại với nhau. Nhiệm vụ chính của mạng nơ-ron là nhận vào một tập đầu vào, thực hiện các phép tính toán phức tạp và sau đó sử dụng đầu ra để giải quyết vấn đề. Mạng nơ-ron là một mạng có cấu trúc và nhiều lớp (layer). Một mạng nơ-ron có 3 lớp chính là: input, hidden và output.



Hình 1.5: Mạng nơ-ron với hai lớp hidden

1.5.2 Deep Learning

Khi mà khả năng tính toán của các máy tính ngày càng được nâng lên một tầm cao mới và lượng dữ liệu ngày càng khổng lồ, Machine Learning (ML) đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning



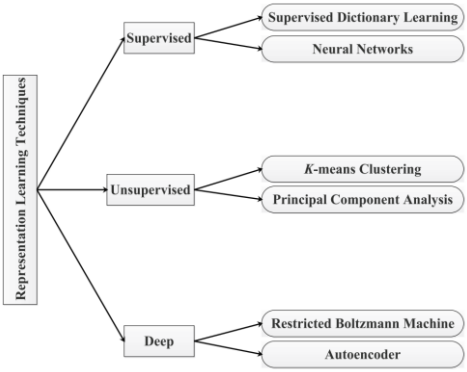
Hình 1.6: Mối liên hệ giữa AI, Machine Learning và Deep Learning

1.6. Tổng quan về Representation Learning

Học đại diện hay học biểu diễn (Representation Learning) là một tập hợp các kỹ thuật cho phép hệ thống tự động khám phá các biểu diễn cần thiết để phát hiện hoặc phân loại tính năng từ dữ liệu thô.

1.7. Các kỹ thuật Representation Learning

Các kỹ thuật này được phân bổ thành ba loại lớn bao gồm: kiến trúc có giám sát, không được giám sát và kiến trúc sâu, như thể hiện trong Hình 1.8. Trong mỗi loại, các kỹ thuật được thảo luận chi tiết với ưu và nhược điểm trong các phần phụ sau:



Hình 1.7: Các kỹ thuật Representation Learning

1.8. Các công trình nghiên cứu liên quan

1.8.1. Các công trình nghiên cứu trong nước

Năm 2018, Nguyễn Trọng Hưng và các cộng sự đã công bố nghiên cứu *"Phát hiện botnet dựa trên phân loại tên miền sử dụng các kỹ thuật học máy"*.

Năm 2019, Vũ Xuân Hạnh và Hoàng Xuân Dậu đã công bố nghiên cứu về *"Phát hiện dga botnet sử dụng kết hợp nhiều nhóm đặc trưng phân loại tên miền"*.

Năm 2019, Hoàng Xuân Dậu và các cộng sự đã công bố nghiên cứu *"Phát hiện botnet dựa trên học máy sử dụng dữ liệu truy vấn DNS: Phân tích ảnh hưởng của các đặc trưng huấn luyện"*.

1.8.1 Các công trình nghiên cứu trên thế giới

Năm 2009, Maryam Feily và các công sự đã công bố nghiên cứu *"A Survey of Botnet and Botnet Detection"*.

Năm 2011, Gregory Fedynyshyn và các cộng sự đã công bố nghiên cứu *"Detection and Classification of Different Botnet C&C Channels"*.

Năm 2019, Riaz Ullah Khan và các cộng sự đã công bố nghiên cứu về *"An Adaptive Multi-Layer Botnet Detection Technique Using Machine Learning Classifiers"*.

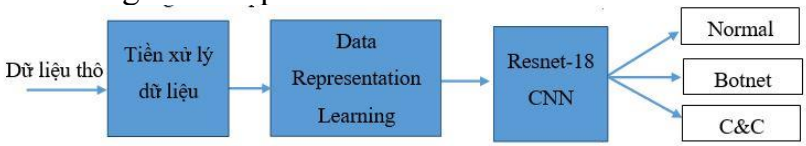
Năm 2020, Ankit Bansal và các cộng sự đã công bố nghiên cứu *"A Comparative Analysis of Machine Learning Techniques for Botnet Detection",*

CHƯƠNG 2: XÂY DỰNG MÔ HÌNH PHÁT HIỆN TẤN CÔNG BOTNET

2.1.Thiết kế mô hình

Mô hình được xây dựng sử dụng công nghệ DL và dùng bộ phân loại CNN học có giám sát dựa vào nhãn để phân loại phát hiện nhận dạng dữ liệu mạng nào là bình thường, dữ liệu nào là tấn công Botnet, dữ liệu nào là tấn công C&C

Mô hình gồm có 3 phần chính:



Hình 2.1: Mô hình phân loại

2.2.Bộ dữ liệu

CTU – 13 là tập dữ liệu về lưu lượng tấn công mạng botnet đã được thu thập tại trường đại học CTU, Cộng Hòa Séc vào năm 2011. Tập dữ liệu bao gồm 13 bản chụp các mẫu botnet khác nhau được chụp lại trong môi trường mạng thực tế và đã được gán nhãn. Trong mỗi bản chụp thu thập bao gồm một lượng lớn lưu lượng bị tấn công botnet, lưu lượng thông thường (normal) và lưu lượng nền (background). Ở mỗi bản chụp, tác giả đã thực thi một phần mềm độc hại cụ thể, các phần mềm này sử dụng một số giao thức và thực hiện các hành động khác nhau để phân tích đặc

điểm của từng bản chụp. Bảng 2.1 cho thấy những đặc điểm của các kịch bản mạng botnet trong bộ dữ liệu.

Bảng 2.1: Đặc điểm các kịch bản mạng Botnet trong bộ dữ liệu CTU-13

Id	IRC	SPAM	CF	PS	DdoS	FF	P2P	US	HTTP	Ghi chú
1	✓	✓	✓							
2	✓	✓	✓							
3	✓			✓				✓		
4	✓				✓			✓		UDP và ICMP DdoS
5		✓		✓					✓	Scan web proxies.
6				✓						Proprietary C&C. RDP.
7									✓	Chinese hosts
8				✓						Proprietary C&C. Net-BIOS, STUN.
9	✓	✓	✓	✓						
10	✓				✓			✓		UDP DdoS
11	✓				✓			✓		ICMP DdoS
12							✓			Synchronization

13		✓		✓					✓	Captcha. Web mail.

2.3.Hiện thực mô hình

2.3.1. Chuẩn bị và xử lý dữ liệu

Tạo nơi để lưu trữ các bản chụp của bộ dữ liệu CTU-13, với tổng số 13 bản chụp ta khởi tạo các thư mục với đường dẫn từ Google Drive

Cài đặt các thư viện cần thiết, cài đặt GPU cho notebook để tăng tốc độ xử lý (Google Colab mặc định chạy trên CPU)

Tiếp theo ta sẽ chuyển các bản chụp (binetflow) sang tệp csv để dễ dàng đọc và xử lý dữ liệu.

Ở bước xử lý dữ liệu này, ta sẽ chọn 1 trong 13 bản chụp để làm dataset cho bài toán, vì số lượng dữ liệu rất lớn nên sẽ tốn kém nhiều thời gian để huấn luyện. Cụ thể, ta sẽ chọn bản chụp thứ 8.

Bản chụp thứ 8 này có 2954230 dòng dữ liệu với 15 thuộc tính, trong đó có 3 thuộc tính với kiểu dữ liệu số thực, 3 thuộc tính kiểu số nguyên và 9 thuộc tính kiểu object.

Bước tiếp theo là chuẩn hóa cột Label – chứa nhãn của dữ liệu, cột Label có 4 giá trị là Normal, Botnet, C&C và Background. Mục đích của ta là phân loại lưu lượng có phải là Botnet, C&C hay không, vì thế ta sẽ bỏ qua lưu lượng có label Background và tính nó như một lưu lượng bình thường.

Lúc này ta thấy rõ sự chênh lệch dữ liệu giữa 3 nhãn, điều này là không tốt vì có quá ít dữ liệu cho nhãn Botnet và C&C dẫn đến mô hình sẽ khó có thể học được chính xác khi bộ dữ liệu mất cân bằng. Chính vì thế ta sẽ giới hạn lại bộ dữ liệu sao cho tỉ lệ giữa 3 nhãn là cân đối. Ta thực hiện giảm dòng dữ liệu cho nhãn Normal.

Ta sẽ tiến hành biến đổi và mã hóa dữ liệu của các thuộc tính có kiểu dữ liệu object. Cụ thể ở đây là các thuộc tính ((StartTime, SrcAddr, Sport, DstAddr, Dport). Nhóm có thể phân loại bao gồm Proto, Dir và State. Vì đặc trưng của dữ liệu nên ta sẽ chia 2 cách mã hóa khác nhau. Cụ thể sử dụng fit_transform của LabelEncoder để biến đổi nhóm đầu tiên gồm 5 thuộc tính và OneHotEncoder để biến đổi nhóm còn lại

Kết quả cho thấy sau khi thực hiện quá trình biến đổi các thuộc tính trên, ta thấy kích thước của bộ dữ liệu đã tăng lên từ 15 lên 85 cột dữ liệu.

	StartTime	Dur	SrcAddr	Sport	DstAddr	Dport	sTos	dTos	TotPkts	TotBytes	...	State_51	State_52	State_53	State_54	State_55	State_56	State_57	State_58	State_59	State_60
0	1798	0.0	180	3197	221	56	0.0	NaN	1	62	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	3644	0.0	180	1787	327	56	0.0	NaN	1	62	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	395	6.758171	180	2041	599	348	0.0	0.0	18	6592	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	3321	0.0	180	4620	551	56	0.0	NaN	1	62	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	261	0.0	180	4179	318	56	0.0	NaN	1	62	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12122	4619	0.000286	178	4464	139	279	0.0	0.0	2	214	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12123	6540	0.000013	232	8471	434	0	NaN	0.0	2	148	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12124	6534	0.000377	759	6958	351	54	0.0	0.0	2	231	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12125	6535	0.000475	726	6107	351	54	0.0	0.0	2	559	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12126	6537	0.000241	207	7785	139	279	0.0	0.0	2	208	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

12127 rows × 85 columns

2.3.2 Chuyển đổi và phân chia dữ liệu hình ảnh

Ở đây ta sẽ sử dụng MinMaxScaler để đưa dữ liệu sang miền giá trị của các pixels (0, 255) trước khi chuyển đổi chúng sang dữ liệu dạng hình ảnh. Mỗi ảnh sẽ mang kích thước 192x192 và được lưu trữ trong thư mục

Images_Data/Botnet đối với hình ảnh có nhãn Botnet và Images_Data/Normal cho nhãn Normal, tương tự cho nhãn C&C. Ta sử dụng thư viện PIL để chuyển sang hình ảnh.

Sau khi đã hoàn tất các công đoạn trên ta tiến hành bước cuối cùng, chia dữ liệu thành tập train, test với tỉ lệ 80:20; Sau đó, từ tập train vừa chia xong ở trên ta chia tiếp train, validation theo tỉ lệ 75:25 để kiểm tra và đánh giá độ chính xác của model trong quá trình training, như vậy ta có 3 tập dữ liệu là train, test, validation được chia theo tỉ lệ 60:20:20.

2.3.3 Xây dựng mô hình phân loại

Bước tiếp theo, ta thực hiện khởi tạo Resnet-18 CNN và thực hiện kiểm tra số lượng feature của vector đầu vào với số lớp đầu ra tương ứng, ở đây là 512 feature cho một vector đặc trưng và 1000 lớp. Kết quả trên không phù hợp với bài toán hiện đang thực hiện nên ta cần phải khởi tạo lại số lớp đầu ra cho mô hình

```
resnet = models.resnet18(pretrained=True)
print(resnet)
print(resnet.fc.in_features)
print(resnet.fc.out_features)
# check if CUDA is available
use_cuda = torch.cuda.is_available()
```

Kế tiếp, ta thực hiện transforms và load dữ liệu từ các tập train, test và validation**.** Ta phải transform dữ liệu sang kiểu tensor để phù hợp với mô hình của thư viện tensorflow

Ở bài toán này chúng ta cần phân loại ra 3 lớp là Normal, Botnet và C&C nên ta sẽ khởi tạo lại số output đầu ra cho mô hình là 3.

Trích xuất vector đặc trưng và nhãn từ ảnh của của các tập train, test và validation để đưa vào mô hình thực hiện train

Tiếp theo ta thực hiện xây dựng hàm train để thực hiện train mô hình, sau đó trả về mô hình đã được luyện.

```
##################
# train the model #
##################
model.train()
for batch_idx, (data, target) in enumerate(loaders['train']):

    # move to GPU
    if use_cuda:
        model = model.to('cuda')
        data, target = data.to('cuda'), target.to('cuda')
    ## find the loss and update the model parameters accordingly
    ## record the average training loss, using something like
    ## train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))
    optimizer.zero_grad()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # backward pass: compute gradient of the loss with respect to model parameters
    loss.backward()
    # perform a single optimization step (parameter update)
    optimizer.step()
    # update training loss
    train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))
    trainLoss += train_loss
    train_batch += 1
```

```
######################
# validate the model #
######################
model.eval()
for batch_idx, (data, target) in enumerate(loaders['valid']):

    # move to GPU
    if use_cuda:
        model = model.to('cuda')
        data, target = data.to('cuda'), target.to('cuda')
    ## update the average validation loss
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))
    validLoss += valid_loss
    valid_batch += 1

trainLoss = trainLoss.cpu().numpy()
validLoss = validLoss.cpu().numpy()
```

```
    # print training/validation statistics
    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch,
        train_loss,
        valid_loss
        ))
    valid_loss_ls.append(validLoss/valid_batch)
    train_loss_ls.append(trainLoss/train_batch)
    ## TODO: save the model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model.state_dict(), 'model.pt')
        valid_loss_min = valid_loss
    if len(valid_loss_ls) > 1 and abs(valid_loss_min - valid_loss) <= stopping_threshold:
      break
# return trained model
plt.plot(train_loss_ls,'-o')
plt.plot(valid_loss_ls,'-o')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train','Valid'])
plt.title('Train vs Valid Loss')
plt.show()
return model
```

CHƯƠNG 3 : THÍ NGHIỆM VÀ ĐÁNH GIÁ

3.1. Các trường hợp thí nghiệm

Với mục tiêu khám phá độ phù hợp cũng như chất lượng của mô hình sẽ tăng giảm như thế nào khi điều chỉnh kích thước của hình ảnh đầu vào, luận văn đề xuất việc áp dụng ba trường hợp với kích thước ảnh lần lượt là: 192x192, 200x200, 224x244. Thí nghiệm sẽ thực hiện từng trường hợp và quan sát độ chính xác, độ mất mát cũng như thời gian thực thi của mô hình, từ đó hiểu được sự ảnh hưởng của kích thước ảnh với mô hình phân loại

3.2. Luyện và kiểm thử mô hình

Để thực nghiệm mô hình đã xây dựng, luận văn đã sử dụng tính năng GPU của Google Colab để cải thiện tốc độ tính toán.

Sau khi chạy training mô hình với các kích thước như trên, ta thu được kết quả được kết quả

Kích thước ảnh đầu vào	Epoch dừng	Độ mất mát	Thời gian thực thi
192x192	93	17.75%	28 phút 33 giây
200x200	46	12.33%	17 phút 02 giây
224x224	79	22.59%	30 phút 31 giây

3.3. Kết quả và nhận xét.

Sau khi hoàn thành quá trình kiểm thử, kết quả của cả ba trường hợp được tổng hợp trong bảng dưới đây:

Kích thước ảnh đầu vào	Độ chính xác	Số điểm phân loại chính xác
192x192	96.29%	2336/2426
200x200	97.16%	2357/2426
224x224	93.16%	2260/2426

Với ba trường hợp thí nghiệm, ta đã nhận được kết quả cao nhất về độ chính xác là 97.16% từ kích thước ảnh 200x200, mô hình đã phân loại được 2357 điểm chính xác trên tổng số 2426 điểm dữ liệu. Xếp thứ hai chính là trường hợp 192x192 với độ chính xác 96.29% và cuối cùng là 224x224 với độ chính xác 93.16%.

Qua các trường hợp thí nghiệm, luận văn đã tìm hiểu và xây dựng được mô hình phân loại sử dụng Representation Learning cụ thể là gray scale image cũng như khám phá ra sự ảnh hưởng của kích thước ảnh đến chất lượng của mô hình.

Tuy nhiên, do giới hạn về phần cứng khi chạy thử mô hình và số lượng dữ liệu kiểm nghiệm chưa nhiều mặc dù mô hình phân loại đạt kết quả khá cao nhưng cũng cần phải cải thiện thêm, vì thực tế nếu ta bỏ sót một vài trường hợp cũng đủ để các hacker tấn công và làm hại đến hệ thống.

KẾT LUẬN

1. Kết quả đạt được

1.1. Về mặt lý thuyết

Nắm được nguyên lý các kỹ thuật tấn công cơ bản, cách thức của tấn công Botnet.

Tìm hiểu về Trí tuệ nhân tạo (AI), các kỹ thuật Representation Learning và ứng dụng vào để phân tích dữ liệu.

Các loại kiến trúc mạng CNN của công nghệ DeepLearning.

1.2. Về mặt thực tiễn

Luận văn đã đưa ra được giải pháp cảnh báo tấn công dựa vào phân tích logs, có thể cho người quản trị biết được mối nguy hiểm trước khi có xảy ra tấn công.

Đưa ra giải pháp phân tích logs dựa vào ứng dụng Trí tuệ nhân tạo (AI), các kỹ thuật Representation Learning.

Xây dựng thành công phần mềm có thể dựa vào phân tích các pha ban đầu của tấn công để cảnh báo đến người quản trị một cách sớm nhất trước khi xảy ra cuộc tấn công thực sự nói chung và Botnet nói riêng.

2. Hạn chế

Giao diện chương trình chưa được chuyên nghiệp. Tập dữ liệu ứng dụng nghiên cứu đã cũ cộng với phần cứng giới hạn nên độ chính xác có thể không như mong muốn. Kết quả đạt được chưa bao quát được hết các trường hợp, dữ liệu mẫu cần được training và mở rộng môi trường áp dụng.

3. Hướng phát triển

Tập trung nghiên cứu rút trích các đặc trưng phù hợp hơn cho quá trình phân tích, tăng độ chính xác trong việc nhận dạng các hành động trinh sát .

Nghiên cứu các mô hình tấn công mạng, các phương pháp trinh sát mới nhằm phát hiện và cảnh báo tốt hơn. Thực nghiệm mô hình trên bộ dữ liệu mới hơn.