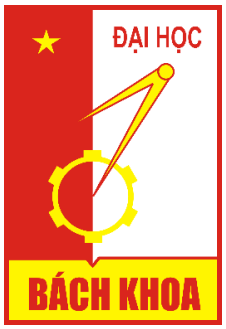
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

----- ☙ 🕮 ❧ -----



BÁO CÁO PROJECT

Đề tài: Ứng dụng graph neural network trong phát hiện ứng dụng độc hại

Giảng viên hướng dẫn: ThS. Bùi Trọng Tùng

Sinh viên thực hiện:
Họ và tên: Lê Thị Thảo Vân
Lớp: IT2 – 05 – K66
MSSV: 20215664

Hà Nội, tháng 7 năm 2024

MỤC LỤC

LỜI MỞ ĐẦU

Hiện nay, Học máy (Machine learning) là một trong những lĩnh vực của trí tuệ nhân tạo (AI) được ứng dụng rất nhiều trong rất nhiều mặt của đời sống. Một trong số đó phải kể đến lĩnh vực an toàn thông tin như: Quản lí lỗ hổng bảo mật, Phân tích tệp tĩnh, Phân tích hành vi,…

Trong bối cảnh ngày càng gia tăng của các mối đe dọa an ninh mạng, việc phát hiện và ngăn chặn các ứng dụng độc hại trở thành một nhiệm vụ cấp bách và cần thiết. Các phương pháp truyền thống thường dựa trên việc so sánh mẫu với cơ sở dữ liệu hiện có, nhưng điều này có thể không hiệu quả trước các mối đe dọa mới chưa được ghi nhận. Vì vậy, việc áp dụng các công nghệ tiên tiến như mạng neural đồ thị (Graph Neural Networks - GNNs) trong phát hiện ứng dụng độc hại đã trở thành một xu hướng mới, mang lại những kết quả hứa hẹn.

Đề tài "Ứng dụng graph neural networks trong phát hiện ứng dụng độc hại" được thực hiện nhằm nghiên cứu và ứng dụng công nghệ GNNs trong việc phát hiện và phân loại các ứng dụng độc hại. Đề tài tập trung vào việc xây dựng một mô hình học máy có khả năng nhận diện và phân biệt các ứng dụng độc hại từ các ứng dụng hợp lệ, giúp nâng cao hiệu quả của các hệ thống an ninh mạng.

Trong quá trình thực hiện đề tài, em đã tiến hành khảo sát, thu thập dữ liệu, xây dựng và huấn luyện mô hình, cũng như đánh giá và tối ưu hóa mô hình để đạt được kết quả tốt nhất. Qua đó, em hy vọng đề tài này sẽ góp phần nhỏ vào việc bảo vệ hệ thống thông tin khỏi các mối đe dọa ngày càng phức tạp và tinh vi.

Em xin chân thành cảm ơn sự hướng dẫn và hỗ trợ của thầy trong quá trình thực hiện đề tài này.

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

Việc áp dụng các công nghệ tiên tiến như mạng neural đồ thị (Graph Neural Networks - GNNs) trong phát hiện ứng dụng độc hại đã trở thành một xu hướng mới, mang lại những kết quả hứa hẹn. GNNs có khả năng phân tích và xử lý dữ liệu phức tạp dưới dạng đồ thị, giúp mô hình học máy nhận diện và phân biệt các ứng dụng độc hại từ các ứng dụng hợp lệ một cách hiệu quả hơn.

Đề tài "Ứng dụng Mạng Neural đồ thị phát hiện ứng dụng độc hại" được thực hiện nhằm nghiên cứu và ứng dụng công nghệ GNNs trong việc phát hiện và phân loại các ứng dụng độc hại. Mục tiêu của đề tài bao gồm:

- Xây dựng một mô hình học máy sử dụng GNNs để phát hiện ứng dụng độc hại.
- Đánh giá hiệu quả của mô hình trong việc nhận diện và phân biệt các ứng dụng độc hại
- Tối ưu hóa mô hình để đạt được kết quả tốt nhất.

CHƯƠNG 2. GRAPH NEURAL NETWORKS

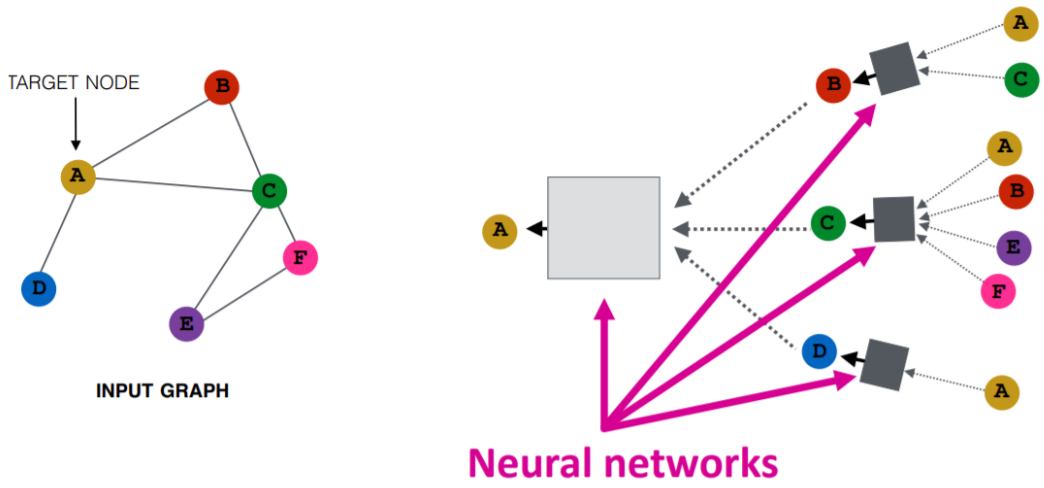
2.1. Giới thiệu về graph neural networks (GNNs)

GNNs áp dụng khả năng dự đoán của một mô hình học sâu đối với các dữ liệu mô tả các đối tượng mà chúng có những kết nối hay liên kết với nhau, hoặc ta có thể biểu diễn những dữ liệu đó ở dạng đồ thị.

Trong GNN, các điểm dữ liệu được gọi là đỉnh, liên kết bằng cạnh, các phần tử được thể hiện dưới dạng toán học để thuật toán học máy có thể đưa ra dự đoán ở cấp độ nút (node level), cấp độ cạnh (edge level) hoặc toàn bộ đồ thị (graph level).

GNN sử dụng 1 quy trình gọi là truyền thông điệp, GNN sẽ sắp xếp các biểu đồ để các thuật toán học máy có thể sử dụng chúng.

Mô hình học thông qua việc cập nhật và kết hợp thông tin từ các hàng xóm của mỗi đỉnh cho phép nắm bắt thông tin cấu trúc và tương tác giữa các đối tượng trong đồ thị. Từ đó tổng hợp được những thông tin về đặc trưng của các đỉnh và cấu trúc đồ thị. Điều này giúp mang lại hiệu quả khi dự đoán tốt hơn so với dựa vào những thông tin đơn lẻ từ riêng các đỉnh hoặc các cạnh.



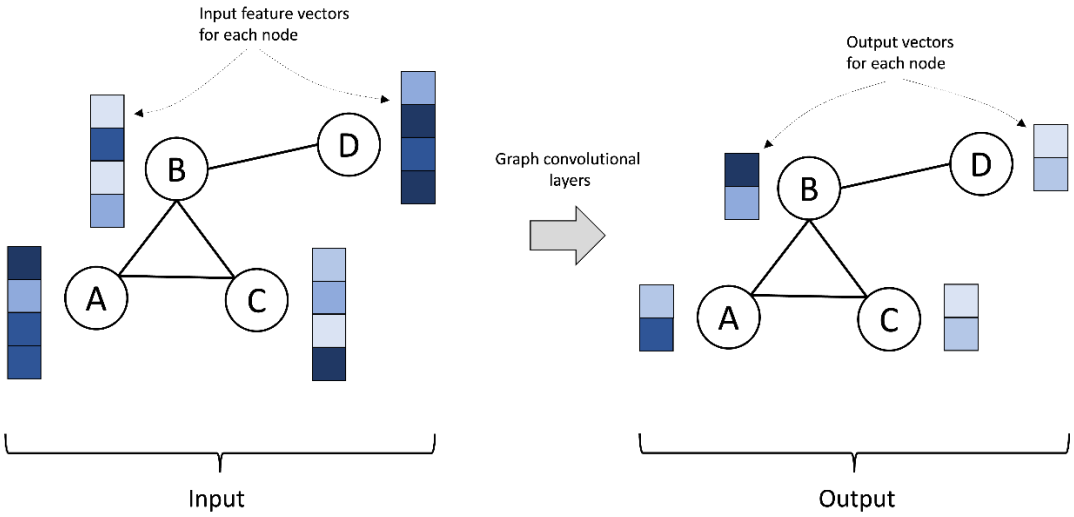
2-1: Graph neural network (nguồn:https://viblo.asia/p/gioi-thieu-ve-graph-neural-networks-gnns-yZjJYG7MVOE)

Đồ thị đầu vào được đi qua một loạt mạng neural. Cấu trúc đồ thị đầu vào được chuyển đổi thành nhúng đồ thị, cho phép chúng ta duy trì thông tin về các nút, cạnh và ngữ cảnh toàn cục. Sau đó, vectơ đặc trưng của các nút A và C được thông qua lớp mạng neural. Nó tổng hợp những đặc trưng này và truyền chúng vào lớp tiếp theo.[1]

2.2. Một số loại Graph Neural Network

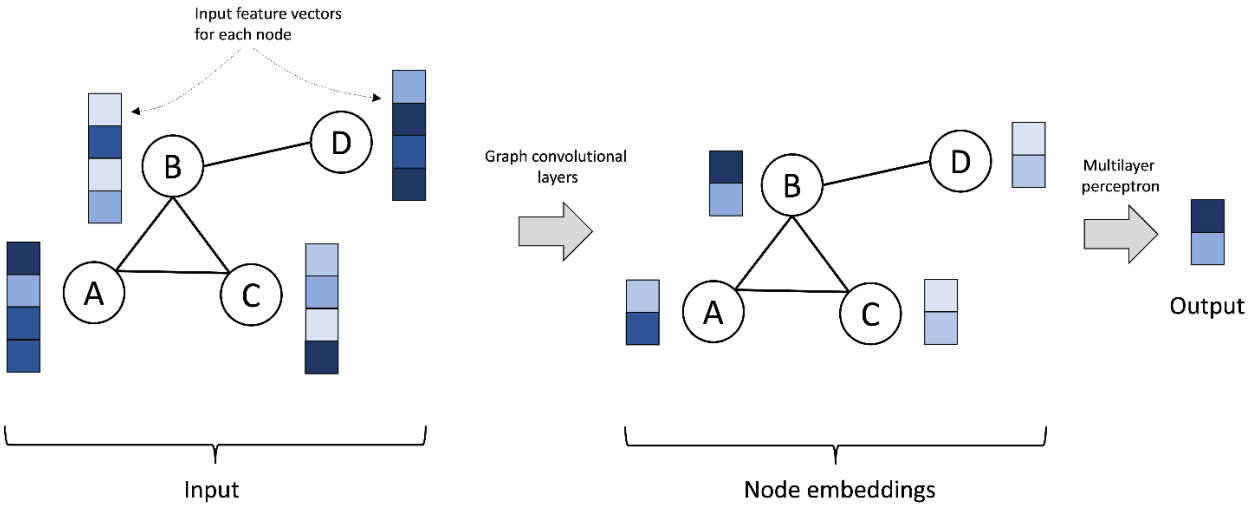
2.2.1. Graph convolutional neural networks (GCN)

Về tổng quát, 1 mô hình GCN có đầu vào là 1 đồ thị với tập các feature vectors đại diện cho các đỉnh. Sau đó GNN bao gồm 1 loạt các graph convolutional layers biến đổi các feature vectors ở mỗi đỉnh. Đầu ra khi đó là biểu đồ được liên kết với các vector đầu ra nhỏ hơn vector đầu vào



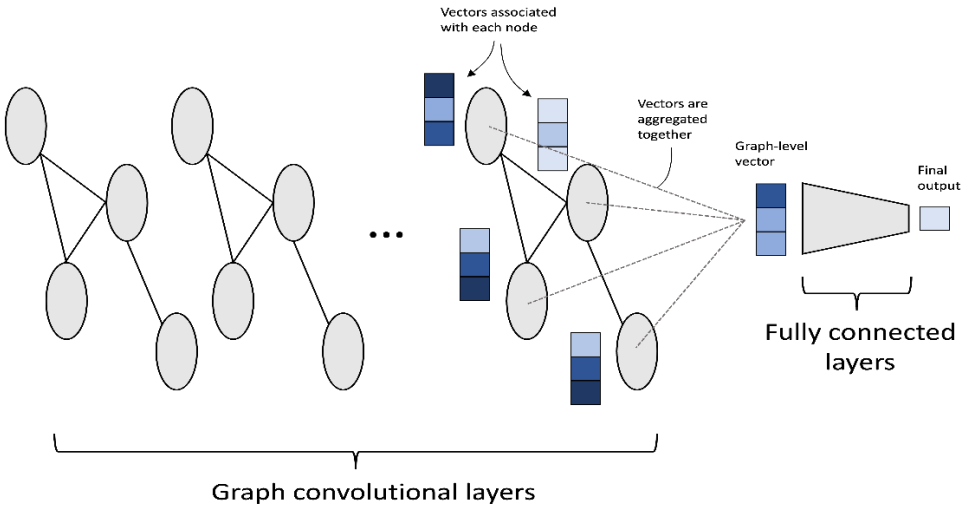
2-2: Đầu vào và đầu ra của một mô hình GCN (nguồn: https://mbernste.github.io/posts/gcn/)

Nếu bài toán chỉ là "node-level" như phân loại đỉnh thì đó có thể coi là đầu ra cuối cùng. Nhưng nếu là "graph-level", ta cần phải xây dựng một mô hình dựa trên toàn bộ đồ thị. Khi đó tất cả các vector trên mỗi đỉnh được đưa chung vào một mạng neural để đưa ra một vector đầu ra duy nhất biểu diễn cho đồ thị đó



2-3: nguồn: https://mbernste.github.io/posts/gcn/

Để thực hiện một tác vụ "graph level", chẳng hạn như phân loại đồ thị, chúng ta cần tổng hợp thông tin trên các nút. Một cách đơn giản để thực hiện việc này là thực hiện một bước tổng hợp đơn giản, trong đó chúng ta tổng hợp tất cả các vectơ liên kết với mỗi nút thành một vectơ duy nhất liên kết với toàn bộ đồ thị. Việc gộp nhóm này có thể được thực hiện bằng cách lấy giá trị trung bình của mỗi tính năng (gộp nhóm trung bình) hoặc giá trị tối đa của mỗi tính năng (gộp nhóm tối đa). Sau đó, vectơ tổng hợp này có thể được sử dụng làm đầu vào cho một perceptron nhiều lớp được kết nối đầy đủ tạo ra đầu ra cuối cùng [2]

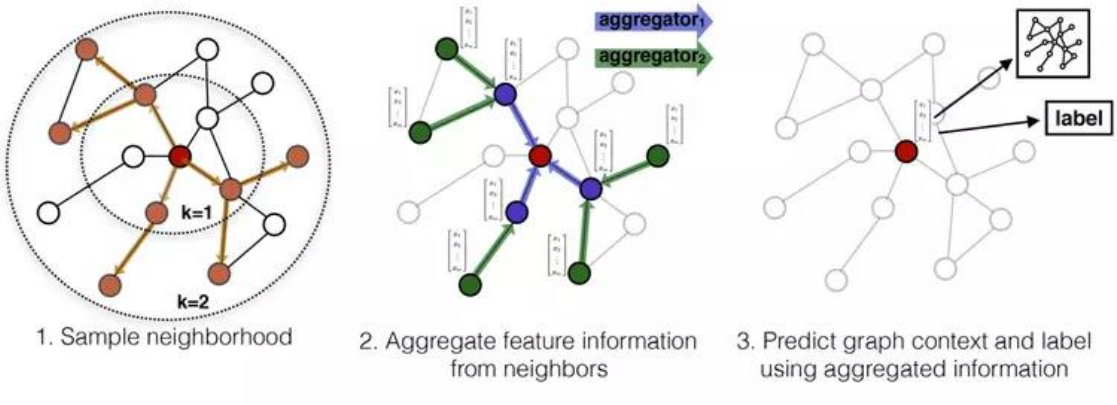


2-4: nguồn: https://mbernste.github.io/posts/gcn/

2.2.2. Graph Sample and Aggregated (GraphSAGE)

GraphSAGE sử dụng một quá trình lấy mẫu và tổng hợp thông tin từ hàng xóm của các đỉnh để cập nhật đặc trưng của mỗi đỉnh. Điều này giúp GraphSAGE học được biểu diễn đồ thị tổng thể và khả năng xử lý các đồ thị lớn.

Việc biểu diễn các đặc trưng đối tượng thành các đặc trưng của nút được nhúng trong đồ thị lớn đã được chứng minh hiệu quả trong nhiều các task dự đoán như gợi ý nội dung, ... Tuy nhiên các phương pháp transductive learning như Deep Walk đều kém hiệu quả khi trong đồ thị xuất hiện các nút mới, chưa từng xuất hiện ở tập huấn luyện do GraphSage ra đời như một trong số lời giải cho vấn đề này



2-5: Minh họa trực quan về mẫu GraphSAGE và cách tiếp cận tổng hợp (Nguồn: William L. Hamilton, Rex Ying, Jure Leskovec, "Inductive Representation Learning on Large Graphs")

2.2.3. Graph Autoencoders (GAE)

GAE là một công cụ mạnh mẽ trong học biểu diễn dữ liệu là đồ thị.

GAE là một lớp mô hình mạng neural được thiết kế để học các biểu diễn có ý nghĩa của dữ liệu đồ thị, có thể được sử dụng cho nhiều tác vụ khác nhau như phân loại đỉnh, dự đoán liên kết và phân cụm đồ thị. GAE bao gồm một bộ mã hóa (encode) cấu trúc topo và nội dung đỉnh của đồ thị, và một bộ giải mã (decoder) tái tạo lại đồ thị từ biểu diễn đó.[3]

Ngoài những tiến bộ này, các nhà nghiên cứu đã đề xuất nhiều kỹ thuật khác nhau để cải thiện hiệu suất của GAE. Ví dụ, Symmetric Graph Convolutional Autoencoder giới thiệu một bộ giải mã đối xứng dựa trên việc làm sắc nét Laplacian, trong khi Adversarially Regularized Graph Autoencoder (ARGA) và biến thể của nó, Adversarially Regularized Variational Graph Autoencoder (ARVGA), thực thi biểu diễn tiềm ẩn để khớp với phân phối trước thông qua đào tạo đối nghịch.

Autoencoders là mô hình học không giám sát được sử dụng cho các tác vụ như giảm chiều, học tính năng và học biểu diễn. Chúng bao gồm một bộ mã hóa nén dữ liệu đầu

vào thành biểu diễn tiềm ẩn chiều thấp hơn và một bộ giải mã tái tạo dữ liệu gốc từ biểu diễn tiềm ẩn. Autoencoders có thể được áp dụng cho nhiều loại dữ liệu khác nhau, bao gồm hình ảnh, văn bản và đồ thị. **[3]**

2.3. Đánh giá mô hình

		Model dự đoán	
		Positive	**Negative**
Thực tế	**Positive**	True Positive (Dự đoán Đúng là Positive)	False Negative (Dự đoán Sai là Negative)
	Negative	False Positive (Dự đoán Sai là Positive)	True Negative (Dự đoán Đúng là Negative)

- **True positives**: Các điểm Positive thực được nhận Đúng là Positive
- **False positives**: Các điểm Negative thực được nhận Sai là Positive
- **True negatives**: Các điểm Negative thực được nhận Đúng là Negative
- **False negatives**: Các điểm Positive thực được nhận Sai là Negative

2.3.1. Precision-Recall Curve

- **Precision**: Precision là thước đo tần suất mô hình học máy dự đoán chính xác positive class. Bạn có thể tính toán độ chính xác bằng cách chia số lượng dự đoán TruePositive cho tổng số trường hợp mà mô hình dự đoán là positive

$$Precision = \frac{TruePositive}{TruePositive + FalsePositives}$$

Và khi đó 0< Precision <=1, Precision càng lớn có nghĩa là độ chính xác của các điểm tìm được càng cao.

- **Recall** đo lường tỷ lệ của các trường hợp dự đoán true positive so với tổng số trường hợp thực sự thuộc vào positive class.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Recall rất hữu ích khi xử lý các tập dữ liệu mất cân bằng trong đó một class (positive hoặc negative) chiếm ưu thế hơn class kia vì nó tập trung vào khả năng của mô hình trong việc tìm kiếm các đối tượng của class mục tiêu.

Đường cong PR chỉ đơn giản là một đồ thị với các giá trị Precision trên trục y và các giá trị Recall trên trục x.

2.3.2. Đường cong AUC-ROC

Sử dụng đường cong AUC-ROC:

Đường cong AUC-ROC, hay Đường cong đặc tính hoạt động của máy tính, là biểu đồ mô tả hiệu suất của phân loại nhị phân mô hình ở nhiều loại phân loại ngưỡng khác nhau. Đường cong này thường được sử dụng trong máy học để đánh giá khả năng của hình học trong công việc phân biệt giữa hai lớp, thường là positive class và negative class.[4]

ROC là biểu diễn đồ họa về hiệu quả của mô hình phân loại nhị phân. Nó vẽ biểu đồ tỷ lệ dương tính thật (TPR) so với tỷ lệ dương tính giả (FPR) ở các ngưỡng phân loại khác nhau.

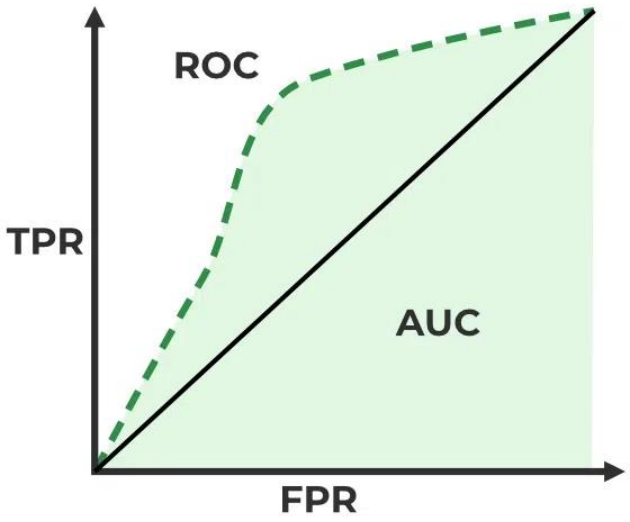
AUC biểu thị vùng dưới đường cong ROC. Nó đo lường hiệu suất tổng thể của mô hình phân loại nhị phân. Vì TPR và FPR đều nằm trong khoảng từ 0 đến 1, nên diện tích sẽ luôn nằm trong khoảng 0 đến 1 và giá trị AUC càng lớn biểu thị hiệu suất mô hình tốt hơn. Mục tiêu chính là tối đa hóa TPR và tối thiểu hóa FPR ở một ngưỡng nhất định.

Đánh giá hiệu quả của mô hình:

AUC cao (gần bằng 1) cho thấy khả năng phân biệt tuyệt vời. Điều này có nghĩa là mô hình có hiệu quả trong việc phân biệt giữa hai lớp và dự đoán của nó là đáng tin cậy.[4]

AUC thấp (gần bằng 0) cho thấy hiệu suất kém. Trong trường hợp này, mô hình gặp khó khăn trong việc phân biệt giữa lớp tích cực và tiêu cực và dự đoán của nó có thể không đáng tin cậy.[4]

AUC khoảng 0,5 ngụ ý rằng về cơ bản mô hình đang đưa ra các dự đoán ngẫu nhiên. Nó cho thấy không có khả năng phân tách các lớp, cho thấy rằng mô hình không học bất kỳ mẫu có ý nghĩa nào từ dữ liệu.[4]



2-6: Đường cong AU-ROC (nguồn:https://www.geeksforgeeks.org/auc-roc-curve/)

Trong đường cong ROC, trục x thường biểu thị Tỷ lệ dương tính giả (FPR) và trục y biểu thị Tỷ lệ dương tính thực (TPR), còn được gọi là Độ nhạy hoặc Thu hồi. Vì vậy, giá trị trục x cao hơn (về phía bên phải) trên đường cong ROC sẽ biểu thị Tỷ lệ dương tính giả cao hơn và giá trị trục y cao hơn (về phía trên) biểu thị Tỷ lệ dương tính thực cao hơn. Đường cong ROC là một đường cong thể hiện sự cân bằng giữa tỷ lệ dương tính thực và tỷ lệ dương tính giả ở các ngưỡng khác nhau. Nó cho thấy hiệu suất của mô hình phân loại ở các ngưỡng phân loại khác nhau. AUC (Diện tích dưới đường cong) là thước đo tóm tắt về hiệu suất của đường cong ROC. Việc lựa chọn ngưỡng phụ thuộc vào các yêu cầu cụ thể của vấn đề mà bạn đang cố gắng giải quyết và sự cân bằng giữa kết quả dương tính giả và âm tính giả. chấp nhận được trong bối cảnh của bạn.[4]

Việc chọn ngưỡng (thresh) có ảnh hưởng đến kết quả dự đoán:

Đối với bài toán đang xét, phân loại các ứng dụng android độc hại và bình thường. Sau khi sử dụng mô hình, em tính được các anomaly score của từng đồ thị (chi tiết ở chương 3)

Ví dụ:

Nhãn thực: [1, 0, 1, 0, 1, 1, 0, 0, 1, 0]

Anomaly scores: [0,8, 0,3, 0,6, 0,2, 0,7, 0,9, 0,4, 0,1, 0,75, 0,55]

Nhãn thật	Anomaly scores	Nhãn được dự đoán nếu ngưỡng = 0.6	Nhãn được dự đoán nếu ngưỡng = 0.7	Nhãn được dự đoán nếu ngưỡng = 0.8	Nhãn được dự đoán nếu ngưỡng = 0.9
1	0.8	1	1	1	0
0	0.3	0	0	0	0
1	0.6	0	0	0	0
0	0.2	0	0	0	0
1	0.7	1	0	0	0
1	0.9	1	1	1	0
0	0.4	0	0	0	0
0	0.1	0	0	0	0
1	0.75	1	1	0	0
0	0.55	0	0	0	0

Theo đó, nếu lấy ngưỡng là 0.7:

Ma trận nhầm lẫn:

	Dự đoán = 0	Dự đoán = 1
Thực tế = 0	TP = 5	FN = 0
Thực tế = 1	FP = 2	TN = 3

Tỷ lệ dương tính thực sự (TPR): Tỷ lệ dương tính thực tế được bộ phận phân loại chính xác

$$TPR = \frac{TP}{TP + FN} = \frac{5}{5 + 0} = 1$$

Tỷ lệ dương tính giả (FPR): Tỷ lệ âm tính thực tế được phân loại không chính xác thành dương tính.

$$FPR = \frac{FP}{FP + TN} = \frac{1}{1 + 4} = 0.2$$

Theo đó, nếu lấy ngưỡng là 0.9:

Ma trận nhầm lẫn:

	Dự đoán = 0	Dự đoán = 1
Thực tế = 0	TP = 5	FN = 0
Thực tế = 1	FP = 5	TN = 0

Tỷ lệ dương tính thực sự (TPR): Tỷ lệ dương tính thực tế được bộ phận phân loại chính xác

$$TPR = \frac{TP}{TP + FN} = \frac{5}{5 + 0} = 1$$

Tỷ lệ dương tính giả (FPR): Tỷ lệ âm tính thực tế được phân loại không chính xác thành dương tính.

$$FPR = \frac{FP}{FP + TN} = \frac{5}{5 + 0} = 1$$

CHƯƠNG 3. THỰC HIỆN ĐỀ TÀI

3.1. Xây dựng NETWORK FLOW GRAPH từ dữ liệu ban đầu

3.1.1. Cơ sở dữ liệu nguồn

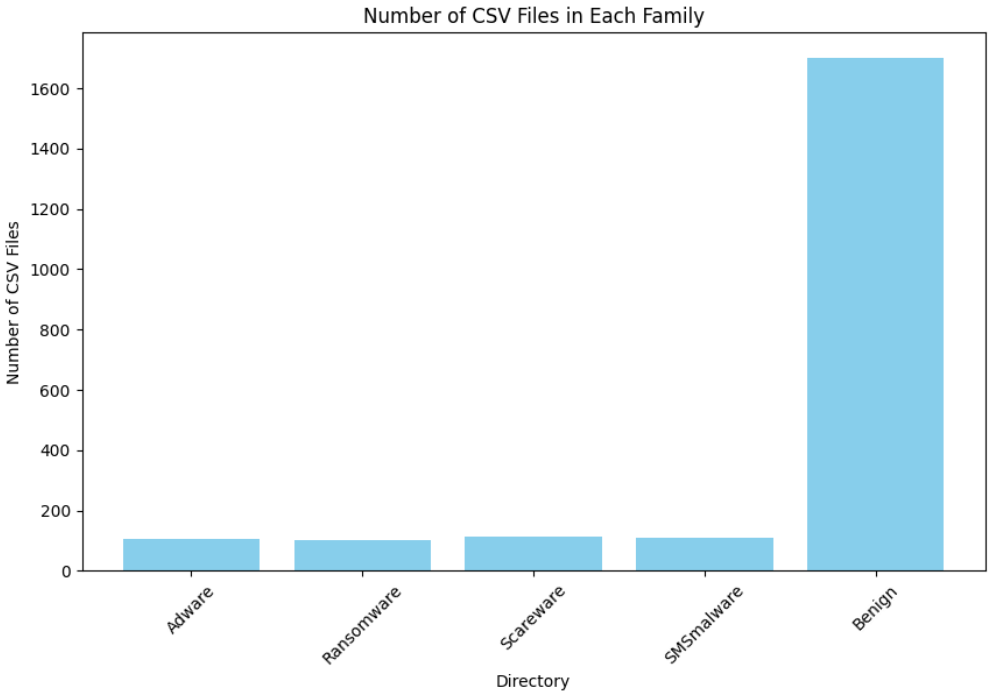
Nguồn dữ liệu: dựa vào bộ dữ liệu Android malware dataset (CIC-AndMal2017) *[5]*

Mô tả dữ liệu: Dữ liệu gồm 2126 file csv, mỗi file csv là tổng hợp các luồng được thu thập từ các ứng dụng android ở 3 thời điểm:

- Cài đặt: Trạng thái đầu tiên của việc thu thập dữ liệu xảy ra ngay sau khi cài đặt phần mềm độc hại (1-3 phút).
- Trước khi khởi động lại: Trạng thái thu thập dữ liệu thứ hai xảy ra 15 phút trước khi khởi động lại điện thoại.
- Sau khi khởi động lại: Trạng thái thu thập dữ liệu cuối cùng xảy ra 15 phút sau khi khởi động lại điện thoại.

Mỗi file csv có 85 trường để mô tả chi tiết thông tin từng luồng như: Flow ID, Source IP, Source Port, Destination IP, …, Label.

Các file csv được phân vào các nhóm Adware, Ransomware, Scareware, SMS Malware và Benign.



3-1: Số lượng các loại malware trong dữ liệu

3.1.2. Xử lí đầu vào

Thực hiện xây dựng flow graph gồm: các đỉnh là các endpoint, cạnh là các luồng. Mỗi file csv tương ứng với 1 flow graph.

1. Bỏ những trường không cần thiết khi xây dựng cạnh: Flow ID, Source Port, Destination Port, Timestamp, Protocol, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, FIN Flag Count, SYN Flag Count, RST Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate.

2. Từ một tập hợp các luồng F, trích xuất một đồ thị có hướng G = (V, E) trong đó:

- Các nút tương ứng với các điểm cuối được liên quan đến bất kỳ luồng F nào và một cạnh có hướng được thêm vào cho tất cả các cặp (si ,ti) mà tồn tại một luồng Fi trong F có nguồn và đích IP là si và ti tương ứng.

- Vectơ đặc trưng được gán cho cạnh này tổng hợp các vectơ đặc trưng $f_i \in R^d$ của tất cả các luồng Fi dọc theo cạnh này bằng cách sử dụng một tập hợp năm hàm tổng hợp. Đối với mỗi đặc trưng, hình dạng của phân phối các giá trị dọc theo cạnh được mô tả bằng cả bốn độ lệch chuẩn đầu tiên, cụ thể là trung bình, độ lệch chuẩn, độ lệch và độ nhọn, và giá trị trung vị. Các giá trị tổng hợp được tính toán cho mỗi đặc trưng và sau đó được nối lại, kết quả là một vectơ đặc trưng $x_i \in R^{5d}$ cho mỗi cạnh $e_i \in$ E.

- Sau khi xây dựng, mỗi cạnh sẽ được biểu diễn bằng 1 vector 295 chiều (59 x 5 = 295)

```
edge_attr_list = []
    for edge in edges_list:
        src_ip = edge[0]
        dst_ip = edge[1]

        # Lọc dữ liệu theo cạnh hiện tại
        edge_data = data[
            (data["Source IP"] == src_ip) & (data["Destination IP"] == dst_ip)
        ]
        if not edge_data.empty:
            edge_attr = []
            for feature in edge_data.columns:
                if feature not in ["Source IP", "Destination IP", "Label"]:
                    values = edge_data[feature]

                    # Tính trung bình
                    if not np.isnan(np.nanmean(values)):
                        mean_value = np.nanmean(values)
                    else:
                        mean_value = 0
```

```
            # Tính độ lệch chuẩn
            if not np.isnan(np.nanstd(values)):
                std_value = np.nanstd(values)
            else:
                std_value = 0

            # Tính độ lệch
            if not np.isnan(values.skew()):
                skew_value = values.skew()
            else:
                skew_value = 0

            # Tính độ nhọn
            if not np.isnan(values.kurtosis()):
                kurtosis_value = values.kurtosis()
            else:
                kurtosis_value = 0

            # Tính giá trị trung vị
            if not np.isnan(np.nanmedian(values)):
                median_value = np.nanmedian(values)
            else:
                median_value = 0

            edge_attr.extend(
                [
                    mean_value,
                    std_value,
                    skew_value,
                    kurtosis_value,
                    median_value,
                ]
            )

        edge_attr_list.append(edge_attr)
```

3. Để phục vụ cho mô hình phát hiện ứng dụng anroid độc hại, các đồ thị sẽ được gán nhãn 0 và 1, 0 là bình thường và 1 là malware. Do đó trong bộ dữ liệu này, ta sẽ có 2126 đồ thị với 1700 đồ thị là bình thường và 426 đồ thị còn lại là malware.

3.2. Xây dựng mô hình Network Flow Graph Autoencoder

3.2.1. Autoencoder model

Với bài toán phát hiện bất thường sử dụng học không giám sát (unsupervised anomaly detection), mô hình Autoencoder thường có hiệu quả tốt[6]. Nhìn chung, một mô hình Autoencoder thường gồm 2 modul mạng neural. Phần mã hóa (encode) có nhiệm vụ học các đặc trưng của dữ liệu đầu vào nhằm biểu diễn chúng có kích thước nhỏ hơn. Đầu ra của phần mã hóa là đầu vào của phần giải mã (decode), phần giải mã sẽ sử dụng những dữ liệu đã được thu gọn đó xây dựng lại đồ thị ban đầu. Khi thực hiện xây dựng lại, sẽ có những khác biệt, tổn thất so với đồ thị đầu vào, nó sẽ được tính toán và gọi là anomaly score (điểm bất thường). Với những mẫu có anomaly score lớn, có thể xem xét những mẫu đó là bất thường.

Đầu vào của model:

- Ma trận biểu diễn thuộc tính cạnh (edge_attr) - kích thước: số cạnh x 295
- Ma trận cạnh đi vào(B_{in}) – kích thước: số đỉnh x số đỉnh
- Ma trận cạnh đi ra (B_{out}) – kích thước: số đỉnh x số đỉnh

Encoder gồm 4 linear layers:

$E^{(0)}$ = f1(X) (X là ma trận thuộc tính cạnh) (1)[7]

$H^{(0)}$ = f2([$B_{in} * E^{(0)}, B_{out} * E^{(0)}$]) (2)[7]

$E^{(1)}$ = f3([${B_{in}}^{T} * H^{(0)}, {B_{out}}^{T} * H^{(0)}, E^{(0)}$]) (3)[7]

$H^{(1)}$ = f4([$B_{in} * E^{(1)}, B_{out} * E^{(1)}, H^{(0)}$]) (4)[7]

Lớp mạng đầu tiên tìm hiểu cách các end point tương tác trực tiếp với nhau bằng cách trước tiên áp dụng phép chuyển đổi tính năng có thể học được cho các vectơ đặc trưng cạnh ban đầu (Phương trình 1) [7]

Layer thứ 2 tính toán biểu diễn nút bằng cách tổng hợp các vectơ đặc trưng từ các cạnh lân cận (Phương trình 2).

Lớp thứ 3 cho phép mô hình tìm hiểu cách các end point giao tiếp gián tiếp với các hàng xóm thứ 2. Trong bước đầu tiên, các tính năng biên được cập nhật lại bằng cách chuyển đổi các vectơ đặc trưng được nối của nút nguồn và nút đích cũng như các tính năng biên từ lớp trước (Phương trình 3). [7]

Lớp thứ 4 có đầu ra là biểu diễn cạnh $H^{(1)}$

```
def __init__(self, input_dim, hidden_dim, output_dim):
        super(NF_GNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim * 2, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim * 3, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim * 3, output_dim)

    def forward(self, edge_attr, Bin, Bout):
        E_0 = F.relu(self.fc1(edge_attr))
        H_0 = F.relu(
            self.fc2(
                torch.cat((torch.matmul(Bin, E_0), torch.matmul(Bout, E_0)), dim=1)
            )
        )
        E_1 = F.relu(
            self.fc3(
                torch.cat(
                    (torch.matmul(Bin.t(), H_0), torch.matmul(Bout.t(), H_0), E_0),
                    dim=1,
                )
            )
        )

        H_1 = F.relu(
            self.fc4(
                torch.cat((torch.matmul(Bin, E_1), torch.matmul(Bout, E_1), H_0),
dim=1)
            )
        )
        return H_1
```

Decoder gồm 4 linear layers:

$E^{(2)} =$ f5($[B_{in}{}^{T} * H^{(1)}, B_{out}{}^{T} * H^{(1)}]$)) (5)[7]

$H^{(2)} =$ f6($[B_{in} * E^{(2)}, B_{out} * E^{(2)}, H^{(1)}]$) (6)[7]

$E^{(3)} =$ f7($[B_{in}{}^{T} * H^{(2)}, B_{out}{}^{T} * H^{(2)}, E^{(2)}]$) (7)[7]

$E^{(4)} =$ f8($E^{(3)}$) (8)[7]

```
def __init__(self, input_dim, hidden_dim, output_dim):
        super(NF_GNN_decode, self).__init__()
        self.fc1 = nn.Linear(input_dim * 2, input_dim)
        self.fc2 = nn.Linear(input_dim * 3, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim * 2 + input_dim, hidden_dim)
```

```
        self.fc4 = nn.Linear(hidden_dim, output_dim)

    def forward(self, H_1, Bin, Bout):
        E_2 = F.relu(
            self.fc1(
                torch.cat(
                    (torch.matmul(Bin.t(), H_1), torch.matmul(Bout.t(), H_1)), dim=1
                )
            )
        )
        H_2 = F.relu(
            self.fc2(
                torch.cat((torch.matmul(Bin, E_2), torch.matmul(Bout, E_2), H_1),
dim=1)
            )
        )
        E_3 = F.relu(
            self.fc3(
                torch.cat(
                    (torch.matmul(Bin.t(), H_2), torch.matmul(Bout.t(), H_2), E_2),
                    dim=1,
                )
            )
        )
        E_4 = F.relu(self.fc4(E_3))
        return E_4
```

3.2.2. Hàm anomaly score

$$L_{AE} = \frac{1}{N} \sum \frac{1}{m_i} \sqrt{\| X_i - E_i^{(4)} \|_F^2}$$

```
def anomalyScores_NF_GNN_AE(original_tensors, reduced_tensors):
    loss = []
    for i in range(len(original_tensors)):
        original_tensor = original_tensors[i]
        reduced_tensor = reduced_tensors[i]
        diff = original_tensor - reduced_tensor
        abs_diff = torch.abs(diff)
        sum = torch.sum(abs_diff, dim=1)
        sum = torch.sum(sum)

        frobenius_norm_per_sample = torch.sqrt(torch.sqrt(torch.sqrt(sum))) / (
            original_tensor.shape[0]
        )
```

```
    loss.append(frobenius_norm_per_sample)
min_score = min(loss)
max_score = max(loss)
normalized_scores = torch.tensor(
    [((max_score - score)) / (max_score - min_score) for score in loss]
)

return normalized_scores
```

3.2.3: Chia tập train và test, tham số đầu vào

Chia tập train và test với tỉ lệ 1 : 9

input_dim = 295

hidden_dim = 64

output_dim = 32

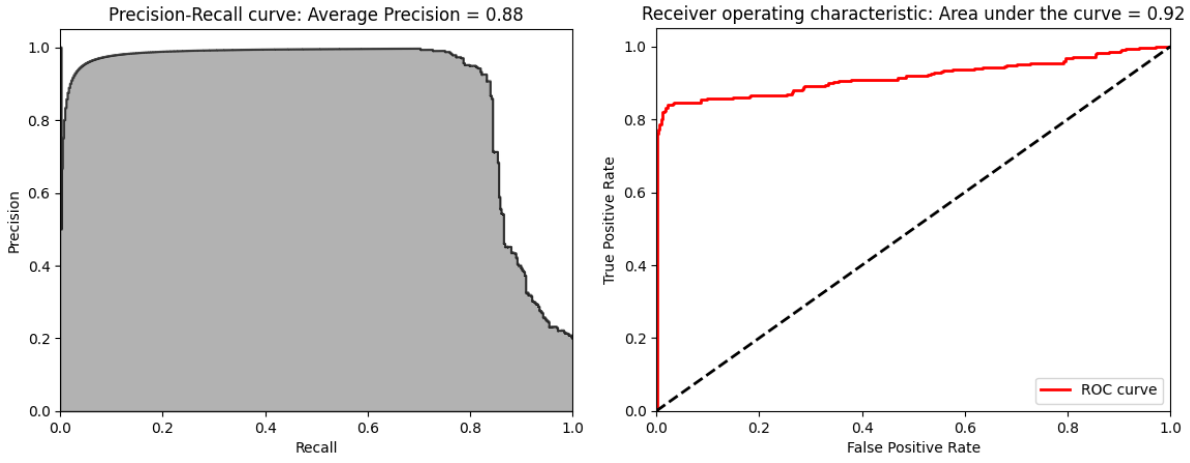
learning_rate = 0.001

num_epochs = 1

3.3. Kết quả thực nghiệm

Mô hình là học không giám sát nên sẽ dựa vào việc huấn luyện ở tập train để tính toán anomaly scores của các đồ thị trong tập test. Dựa vào đó để đưa ra nhãn dự đoán cho các đồ thị.

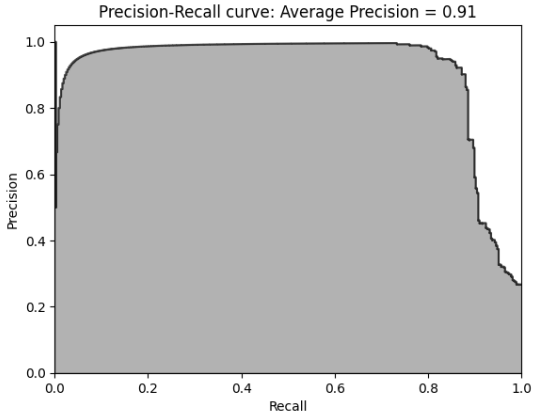
Sử dụng Precision-Recall Curve và đường cong AUC ROC để đánh giá hiệu năng của model.

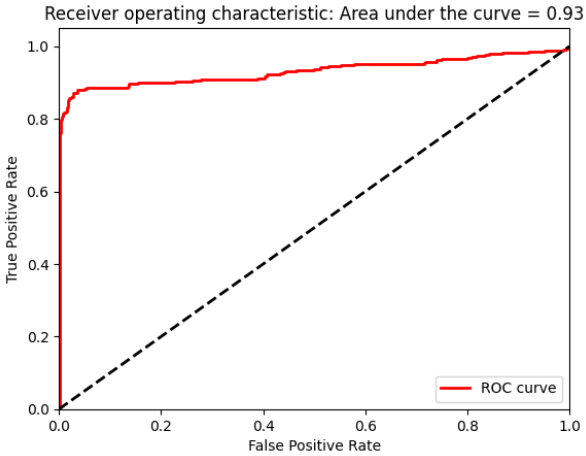


Best threshold: 0.922763

Từ kết quả của đồ thị trên, tỉ lệ Average Precision (AP) = 0.88 là giá trị trung bình độ chín xác đạt tại mỗi ngưỡng và AUC = 0.92

Nếu chỉ xét những đồ thị lớn (số cạnh lớn hơn 500) thì có 1527 cạnh, hiệu năng của đồ thị được tăng lên cho thấy ảnh hưởng của kích thước đồ thị lên các dự đoán của model.





KẾT LUẬN

Trong đề tài này, em đã nghiên cứu và ứng dụng công nghệ mạng neural đồ thị (Graph Neural Networks - GNNs) trong việc phát hiện các ứng dụng độc hại trên nền tảng Android. Qua quá trình thu thập, xử lý dữ liệu và xây dựng mô hình học không giám sát, em đã đạt được những kết quả đáng khích lệ với độ chính xác và hiệu quả cao.

Việc sử dụng GNNs để phân tích các luồng mạng đã chứng minh tính khả thi và ưu điểm vượt trội so với các phương pháp truyền thống trong việc phát hiện các mối đe dọa an ninh mạng. Kết quả thực nghiệm cho thấy mô hình Autoencoder dựa trên GNNs có thể phân biệt hiệu quả các ứng dụng độc hại từ các ứng dụng hợp lệ, góp phần nâng cao khả năng bảo mật cho các hệ thống thông tin.

Em hy vọng rằng kết quả của đề tài sẽ góp phần nhỏ vào việc bảo vệ hệ thống thông tin khỏi các mối đe dọa ngày càng phức tạp và tinh vi, đồng thời mở ra hướng nghiên cứu mới trong lĩnh vực an toàn thông tin.

Em xin chân thành cảm ơn sự hướng dẫn và hỗ trợ của thầy trong quá trình thực hiện đề tài này.

TÀI LIỆU THAM KHẢO

[1] Trinh Quang, "Giới thiệu về Graph Neural Networks (GNNs)" 30 Oct 2023, https://viblo.asia/p/gioi-thieu-ve-graph-neural-networks-gnns-yZjJYG7MVOE

[2] Matthew N. Bernstein, "Graph convolutional neural networks", 24 September 2023, https://mbernste.github.io/posts/gcn/

[3] "Graph Autoencoders", https://www.activeloop.ai/resources/glossary/graph-autoencoders/

[4] "AUC ROC Curve in Machine Learning", https://www.geeksforgeeks.org/auc-roc-curve/

[5] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification", In the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada, 2018.

[6] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407 (2019).

[7] Julian Busch, Anton Kocheturov, Volker Tresp, Thomas Seidl "NF-GNN: Network Flow Graph Neural Networks for Malware Detection and Classification" pp.4-5, July 6–7, 2021, doi: 2103.03939