HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 2

BÁO CÁO CUỐI KỲ

IoT và ứng dụng

Đề tài: Phát hiện URL độc hại sử dụng phương pháp học sâu

Giảng viên hướng dẫn: Đàm Minh Lịnh

Lớp: D20CQCNPM01 - N

Sinh viên thực hiện:

N20DCCN034 – Vũ Văn Lâm

N20DCCN035 – Trần Gia Long

Thành phố Hồ Chí Minh, ngày 05 tháng 01 năm 2024

Đề tài: Phát hiện url độc hại sử dụng phương pháp học sâu

Thông tin nhóm 8:

STT	MSSV	Họ và tên	Email
1	N20DCCN034	Vũ Văn Lâm	n20dccn034@student.ptithcm.edu.vn
2	N20DCCN035	Trần Gia Long	n20dccn035@student.ptithcm.edu.vn

Mục lục

Chương 1: Tổng quan

1. Giới thiệu

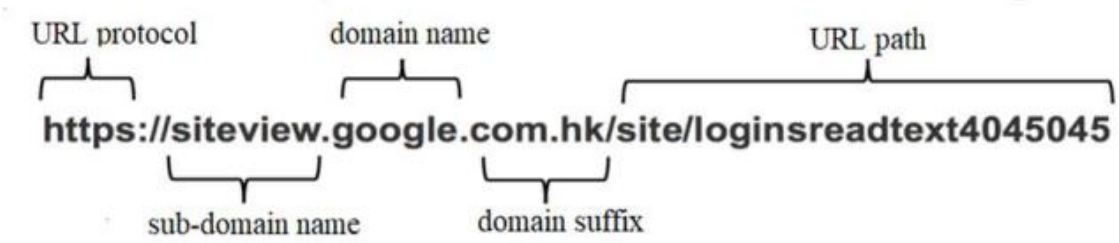
Trong thời đại công nghệ thông tin đang phát triển như hiện nay, internet là một phần không thể thiếu trong cuộc sống và công việc của nhiều người. Tuy nhiên, internet cũng tiềm tàng những nguy cơ an ninh mạng, trong đó có URL độc hại. URL hay còn gọi là Uniform Resource Locator được dùng như một đường dẫn tới một tài nguyên trên mạng. Những kẻ tấn công thường thay đổi một hoặc nhiều thành phần cấu trúc của URL để đánh lừa người dùng và phân tán URL độc hại. Qua đó khi người dùng nhấp vào link URL, họ sẽ bị gây hại theo những cách khác nhau tùy theo loại URL mà kẻ tấn công thiết kế.

Chúng em chọn đề tài này là vì chúng em quan tâm đến lĩnh vực an ninh mạng, và muốn khai thác tiềm năng của học sâu trong bài toán phát hiện url độc hại. Đây là một bài toán có ý nghĩa thực tiễn, góp phần bảo vệ người dùng và hệ thống khỏi các mối đe dọa trên internet. Đồng thời, đây cũng là một bài toán có nhiều thách thức và khó khăn, đòi hỏi sự kết hợp của nhiều kỹ thuật và phương pháp khác nhau.

Trong đề tài này, chúng em đề xuất một phương pháp sử dụng học sâu để phân loại URL. Học sâu là một nhánh của học máy, dựa trên các mạng nơ-ron nhân tạo có nhiều lớp ẩn, có khả năng học được các đặc trưng phức tạp và trừu tượng của dữ liệu. Chúng em đã phối hợp nhiều kĩ thuật trong học sâu như CNN, LSTM và Embedding để huấn luyện mô hình. Từ đó đánh giá và ứng dụng mô hình vào thực tế, sử dụng extension trên các trình duyệt để bắt sự kiện vào trang của người dùng, thông báo về nguy cơ của trang web.

2. Tổng quan về URL

URL hay thường được gọi là địa chỉ web, nó tham chiếu đến một vị trí tài nguyên trên mạng máy tính. Từ tham chiếu này ta có thể truy xuất đến tài nguyên đó. URL là một thành phần con của URI nên đôi khi người ta đồng nhất hai thuật ngữ này. Cú pháp của URL tuân theo cú pháp chung của URI và được viết bằng một chuỗi phân cấp gồm năm thành phần [1]:



Hình 1: Cấu trúc của URL

+ URL protocol: chỉ định giao thức nào được áp dụng như “http”, “https”, v.v. tùy vào chương trình phía người dùng mà có thể phần này được tự động thêm vào [1].

+ Hostname: Là vị trí cần truy cập đến, có nhiệm vụ lấy một địa chỉ IP, nó có thể tồn tại dưới dạng tên hoặc IP, một hostname bao gồm:

- Sub-domain name: Tên miền con, bao gồm bất kỳ từ hoặc cụm từ nào đứng trước dấu chấm đầu tiên của URL. Ví dụ như “www”, “mail”, “news”, v.v.
- Domain name: Tên miền của trang web truy cập, là vị trí cần truy cập đến, có thể là tên hoặc một địa chỉ IP.
- Domain suffix: Đuôi của tên miền (tên miền cấp cao nhất), những tên này được Tập đoàn Internet cấp số và tên miền (ICANN) tạo và quản lý. Có ba tên miền cấp cao nhất phổ biến là .com., .net, .gov. Hầu hết các quốc gia đều có trên miền cấp cao nhất gồm hai chữ cái, như .us (Mỹ), .vn (Việt Nam), .ca (Canada), v.v… Có một số tên miền cấp cao nhất bổ sung (như .museum) được các tổ chức cá nhân tài trợ và quản lý. Ngoài ra, cũng có một số tên miền cấp cao nhất dùng chung như. club, .life và .news [2].

+ URL path: Cho phép đưa trình duyệt của client đến đúng file hoặc thư mục trên máy chủ, gồm cả phần truy vấn và tham số liên quan để thực thi một tác vụ nào đó. Phần path bắt đầu bằng dấu “/”, phần cuối cùng là tên file được truy cập đến. Phần truy vấn được thể hiện ngay sau dấu chấm hỏi (“?”) sau đường dẫn. Có nhiều thành phần trong URI bắt buộc phải có và một số thành phần khác chỉ cần có trong một số trường hợp nhất định. Ví dụ: Dấu chấm hỏi (“?”) tại vị trí sau đường dẫn và yêu cầu truy vấn dữ liệu với dấu thăng (“#”) cũng như dấu hai chấm (“:”), dấu a móc (“@”) chỉ cần thiết khi việc truy cập cần phải cung cấp thông tin tài khoản [1].

3. URL độc hại

Các URL độc hại là những đường dẫn tới các trang web gây nguy hại cho người dùng và hệ thống mà họ đang làm việc.

Qua đó nhằm mục đích:
+ Thực hiện đánh cắp thông tin cá nhân của người dùng và của tổ chức [1].
+ Máy tính bị kiểm soát, trở thành những botnet sẵn sàng nhận lệnh từ máy chủ kẻ xấu [1].
+ Cài đặt các phần mềm độc hại như virus, worm, trojan horse từ các tập tin được tải về máy tính hoặc lừa người dùng dưới dạng một trò giải trí [1].

+ Thể hiện thông tin lừa đảo, các quảng cáo độc hại, … [1]
Các hình thức tấn công sử dụng URL phổ biến như: Drive-by Download, lừa đảo (phishing), thư rác (spamming), Social Engineering, Defacement. Drive-by Download thường tự cài các phần mềm độc hại vào máy tính bằng cách khai thác lỗ hổng của trình duyệt. Lừa đảo và Social Engineering là đánh lừa người dùng tiết lộ thông tin riêng tư bằng nhiều cách. Thư rác là việc sử dụng tin nhắn, thư điện tử để gửi thư mà người dùng không mong muốn hoặc để lừa đảo. Khi người dùng vô tình truy cập đến một URL nào đó thì cũng có thể trở thành nạn nhân của các cuộc tấn công này [1]. Defacement là hình thức tấn công mạng mà kẻ tấn công thay đổi nội dung hoặc giao diện của một trang web nào đó, thường là để phá hoại hoặc truyền tải thông điệp chính trị.

4. Phân tích đề tài

Có nhiều cách để ngăn chặn các URL độc hại, phương pháp cơ bản là so sánh với danh sách các URL độc hại với những thông tin những danh sách tên miền độc hại, danh sách IP độc hại. Các danh sách này được cung cấp bởi người dùng. Khi một trang web được yêu cầu thì yêu cầu này được tìm trong các danh sách trên, nếu kết quả là có thì trang web này được báo là trang web độc. Một cách khác là phát hiện thông qua các dấu hiệu như những chương trình phòng chống virus thương mại. Với cách thức này, các dấu hiệu là một danh sách được lưu trữ với các dấu hiệu mã độc và danh sách này được cấp định kỳ thông qua nhà cung cấp dịch vụ nào đó [1].

Tuy nhiên, những cách thức trên dường như không còn hiệu quả bởi lẽ hiện nay, tội phạm công nghệ ngày càng tinh vi và phát triển hơn nhiều những cách thức mới để đánh lừa các chương trình này. Qua đó, việc sử dụng mô hình học sâu để phát hiện những đặc trưng mang tính nguy cơ của mã URL sẽ phù hợp hơn và hiệu quả hơn những phương pháp đó.

Trong đề tài này, bọn em sẽ sử dụng các phương pháp để lấy ra danh sách những đặc trưng cơ bản của một đoạn mã URL như độ dài, số lượng các kí tự đặc biệt, số lượng chữ, số lượng số, url bất thường, v.v. để phân loại khả năng lành tính của URL, tiếp sau đó phân tách URL thành 5 phần sử dụng mô hình Embedding để trích xuất đặc trưng này và tạo một cấu trúc CNN để huấn luyện các đặc trưng từ 5 thành phần đó. Cuối cùng thông qua mô hình CNN cùng với LSTM cho ra nhãn phân loại URL độc hại.

Các nhiệm vụ cần thực hiện:
1. Thu thập dữ liệu huấn luyện và kiểm định.
2. Trích xuất đặc trưng cơ bản của URL.
3. Khởi tạo một mô hình neural network để phân loại lành tính hay độc hại.

4. Tạo một từ điển các kí tự để chuẩn hóa chuỗi URL sang một dãy số.
5. Phân tách URL thành 5 phần: protocol, sub-domain, domain, domain suffix, path.
6. Tạo mô hình CNN kết hợp kĩ thuật LSTM để phân loại nhãn URL.
7. Đánh giá và kiểm định mô hình học sâu.
8. Tạo ứng dụng khởi chạy extension.
9. Tích hợp mô hình phát hiện URL độc hại vào ứng dụng extension.
10. Chạy và kiểm định ứng dụng extension.

Chương 2: Mô hình đề xuất cho ứng dụng

1. Bộ dữ liệu thu thập

Bộ dữ liệu được sử dụng cho đề tài này là bộ URL dataset (ISCX-URL2016) từ trang UNB (University of new brunswick).

Benign URLs: Hơn 35.300 URL lành tính đã được thu thập từ các trang web hàng đầu của Alexa. Các tên miền đã được chuyển qua trình thu thập dữ liệu web Heritrix để trích xuất các URL. Khoảng nửa triệu URL duy nhất được thu thập dữ liệu ban đầu và sau đó được chuyển để loại bỏ các URL trùng lặp và chỉ tên miền. Sau đó, các URL được trích xuất đã được kiểm tra thông qua Virustotal để lọc các URL lành tính [3].

Spam URLs: Khoảng 12.000 URL spam đã được thu thập từ bộ dữ liệu WEBSPAM-UK2007 có sẵn công khai [3].

Phishing URLs: Khoảng 10.000 URL lừa đảo đã được lấy từ OpenPhish, một kho lưu trữ các trang web lừa đảo đang hoạt động [3].

Malware URLs: Hơn 11.500 URL liên quan đến các trang web chứa phần mềm độc hại đã được lấy từ DNS-BH, một dự án duy trì danh sách các trang web phần mềm độc hại [3].

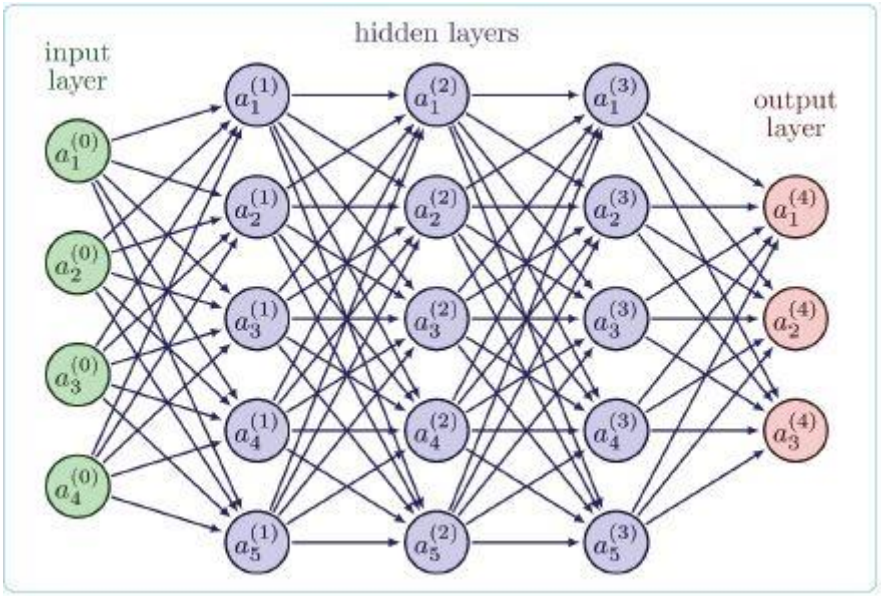
Defacement URLs: Hơn 45.450 URL thuộc danh mục URL Defacement. Chúng là các trang web đáng tin cậy được xếp hạng Alexa, mà có lưu trữ URL lừa đảo hoặc ẩn chứa cả trang web độc hại [3].

Bên cạnh đó, để tăng cường cho các trường hợp Benign URLs, bọn em đã thêm 10 nghìn dòng Benign URLs từ bộ dataset Malicious And Benign URLs trên Kaggle do Siddharth Kumar chia sẻ. Các URL là những URL phổ biến thường được sử dụng bởi người dùng.

2. Mô hình sử dụng

2.1 Mô hình Neural Network

Neural Network nhân tạo được mô phỏng theo bộ não con người. Nó phân tích dữ liệu phức tạp, hoàn thành các phép toán, tìm kiếm các mẫu và sử dụng thông tin thu thập được để đưa ra dự đoán và phân loại. Cũng giống như bộ não con người, Neural Network nhân tạo có một đơn vị chức năng cơ bản được gọi là nơ-ron. Những nơ-ron này còn được gọi là các nút, truyền thông tin trong mạng [4].



Hình 2: Cấu trúc của mạng Neural Network

Một Neural Network cơ bản có các nút được kết nối với nhau trong các lớp (layer) đầu vào (input layer), layer ẩn (hidden layer) và layer đầu ra (output layer) [4].

Layer đầu vào xử lí và phân tích thông tin trước khi gửi nó đến layer tiếp theo [4].

Layer ẩn nhận dữ liệu từ layer đầu vào hoặc các layer ẩn khác. Sau đó, layer ẩn tiếp tục xử lí và phân tích dữ liệu bằng cách áp dụng một tập hợp các phép toán để chuyển đổi và trích xuất những tính năng có liên quan từ dữ liệu đầu vào [4]. Layer ẩn thường sử dụng các hàm kích hoạt để làm cho kết quả đầu ra trở nên phi tuyến. Hàm phổ biến được dùng trong lớp này là sigmoid, ReLU, TanH, …

Layer đầu ra cung cấp thông tin cuối cùng bằng cách sử dụng các tính năng được trích xuất. Layer này có thể có một hoặc nhiều nút, tùy thuộc vào kiểu thu thập dữ liệu. Đối với phân loại nhị phân, người ta thường dùng hàm kích hoạt sigmoid và số unit là 1 để phân loại, kết quả ra thường là số thực từ 0 đến 1 [4].

2.2. Character Embedding

Character Embedding có cách hiểu tương tự với Word Embedding. Chỉ khác là Character Embedding xét chuyển đổi trên từng kí tự còn Word Embedding chuyển đổi các từ trên chuỗi.

Embedding từ (Word Embedding) là tên chung cho một tập hợp các mô hình ngôn ngữ và các phương pháp học đặc trưng trong xử lý ngôn ngữ tự nhiên (NLP), nơi các từ hoặc cụm từ từ vựng được ánh xạ tới vectơ số thực. Về mặt khái niệm, nó liên quan đến việc nhúng toán học từ một không gian với một

chiều cho mỗi từ vào một không gian vectơ liên tục với kích thước thấp hơn nhiều [5].

Embedding cung cấp một biểu diễn cô đọng của các từ/ kí tự và độ tương tự giữa các từ này. Và để tạo ra vector biểu diễn của một từ/ kí tự ta cần chuyển đổi ngữ liệu sang kiểu số.

Các bước có thể khái quát như sau:

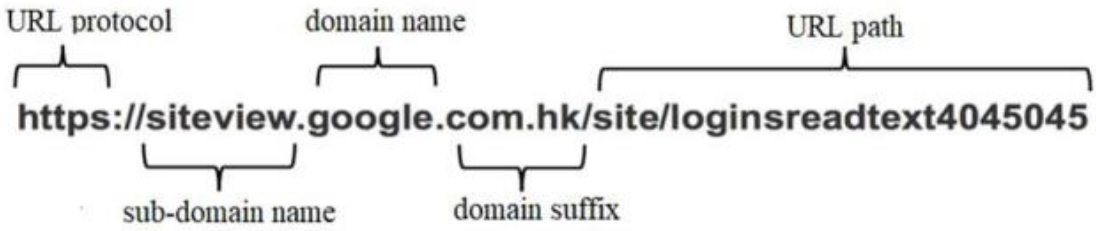
Ngữ liệu → tokenized văn bản → kí tự → chuyển tài liệu thành mảng số.

Và để làm được điều này, chúng em đã sử dụng đối tượng Tokenizer của Keras. Tokenizer sẽ học từ tập dữ liệu để tạo ra một từ điển các kí tự tương ứng với số. Các kí hiệu xuất hiện càng nhiều lần trong tập dữ liệu thì số biểu diễn của nó sẽ càng nhỏ.

Sau đó ta sẽ thiết lập lớp mạng Embedding vào trong mô hình học máy, thường là neural network để lớp này hình thành nên Embedding Matrix, và đó cũng chính là trọng số của lớp mạng, dựa theo đó, ta có thể biết được từng kí tự được biểu diễn bằng vector như thế nào. Sự tương quan cho thấy, những kí tự thường đi cùng nhau sẽ nằm rất gần khi tính khoảng cách theo vector nhiều chiều.

2.3. URL2Vec

Một nhóm nghiên cứu đã triển khai mô hình URL2Vec bằng cách phân chia URL sang 5 phần dựa theo cấu trúc sau:

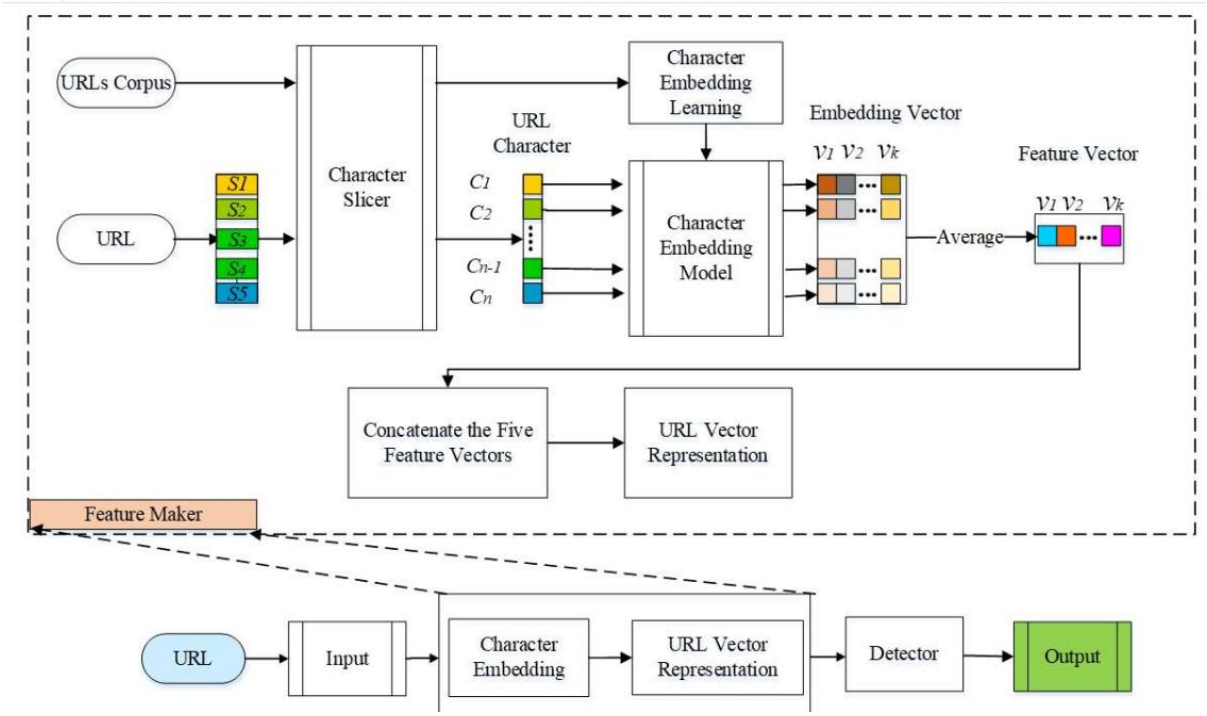


Hình 1: Cấu trúc của URL

Họ tiến hành sử dụng kĩ thuật Character Slicer để mã hóa chuỗi URL sang số và huấn luyện bằng phương pháp học Embedding để trích xuất đặc trưng của từng kí tự. Qua đó tiếp tục dùng model đã train được để trích xuất các đặc trưng ở từng thành phần. Sau đó, họ ghép năm kết quả nhúng vectơ cho năm phần tương ứng để có được URL thống nhất biểu diễn vector. Đối với một số URL có ít hơn năm phần, nhóm đã đặt các phần tử trong vectơ tương ứng với các phần còn thiếu tất cả số không. Ví dụ, giả sử rằng chúng ta thu được ký tự nhúng 10 chiều, biểu diễn vector của mỗi phần là 10 chiều bằng cách tiến hành lấy giá trị trung bình của từng vector của kí tự trong chuỗi. Cuối cùng, biểu diễn vector

của URL nằm ở dạng 50 chiều bằng cách ghép các biểu diễn của năm các phần nói trên [6].

Sau khi có đặc trưng của URL, ta tiến hành dùng model để phân loại chúng. Dưới đây là mô hình đề xuất của nhóm nghiên cứu:



Hình 3: Mô hình ứng dụng URL2Vec

Algorithm 1 Learning Character Embedding for Phishing Detection

Input: URLs

Output: Phishing or Legitimate

Step 1: Divide each URL into N parts (refer to Fig.1).

Step 2: Learn feature vector $\emptyset$ of each part using language model:

Initialize feature vector $\emptyset$ from corpus $C^{|1\times d|}$

Build a binary tree T

for $i = 0$ to num (leaves) **do**

  for each $c_i \in C$ **do**

    for each $c_k \in u[i-w:i+w]$ **do**

      $J(\emptyset) = -Pr(c_k|\emptyset(c_i))$

      $\emptyset = \emptyset - \alpha * \frac{\partial J}{\partial \emptyset}$

    end for

  end for

end for

Step 3: Concatenate feature vectors $\emptyset$ of N parts as URL representation

Step 4: Feed the vector representation of URLs to classifiers

Step 5: Output classification result

Hình 4: Mã giả cho mô hình phân loại theo kĩ thuật URL2Vec

2.4. Mô hình CNN

CNN (Convolutional Neural Network) như tên gọi, mô hình được phát triển vẫn dựa trên ý tưởng từ Neural Network, ngoài ra còn sử dụng thêm một phương pháp đó là tính tích chập (Convolutional).

Cấu trúc cơ bản của một mạng CNN thường bao gồm:

1. Lớp đầu vào (Input layer)
2. Lớp tích chập (Convolutional layer)
3. Lớp gộp (Pooling layer)
4. Lớp làm phẳng (nếu cần) - làm phẳng nhiều chiều sang 1 chiều
5. Lớp kết nối đầy đủ (Fully connected layer)
6. Lớp phân loại

Lớp đầu vào:

Lớp đầu vào thường dùng để định dạng hình dáng (shape) của dữ liệu đầu vào.

Lớp tích chập:

Xét tín hiệu một chiều a và bộ lọc (filter) w. Phép tính tích chập của tín hiệu và bộ lọc là một tín hiệu một chiều b được xác định theo công thức [7]:

$$b(n) = a(n) * w(n) = \sum_{k=-\infty}^{\infty} a(k)w(n-k)$$

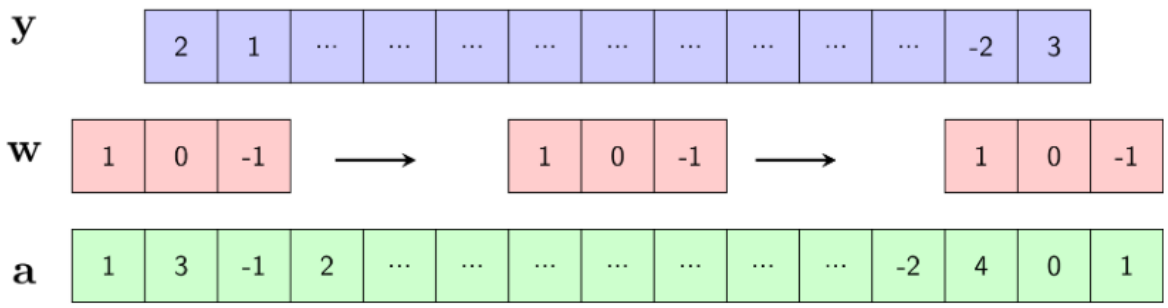
Hình 5: Công thức tính tích chập tín hiệu 1 chiều

Trong mạng neural tích chập, tích chập được định nghĩa khác đi một chút. Cho tín hiệu đầu vào và bộ lọc lần lượt là các vector $a \in R^N$ và $w \in R^f$ (f có thể là một số tự nhiên bất kỳ nhưng thường là số lẻ). Khi đó đầu ra là một vector y với từng phần tử được tính bởi công thức [7]:

$$y_n = \sum_{i=0}^{f-1} a_{n+i} w_i$$

Hình 6: Công thức tính tích chập trong mạng CNN 1 chiều

Với n thỏa mãn $0 <= n + I < N$, $\forall$ i= 0, 1, …, f - 1. Điều này tương đương với $0 <= n < N – f + 1$. Vậy $y \in R^{N-f+1}$ [7].



Hình 7: Biểu diễn quá trình tính tích chập của CNN 1 chiều

Quá trình tính đầu ra y có thể được thực hiện như sau [7]:

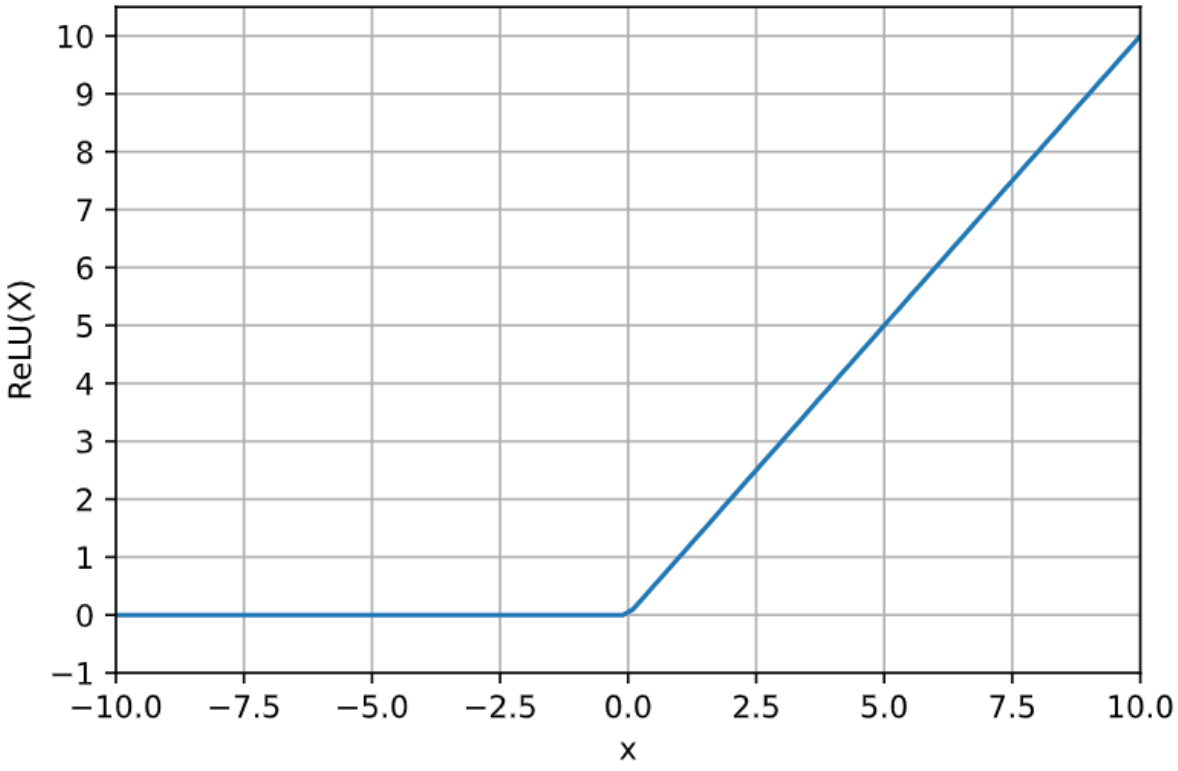
1. Đặt bộ lọc w vào vị trí tương ứng với f phần tử đầu tiên của a.

2. Nhân từng phần tử tương ứng của w và a rồi cộng các kết quả lại để được phần tử tương ứng của y (bản đồ đặc trưng – feature map).
3. Trượt bộ lọc w một bước sang bên phải. Nếu phần tử cuối cùng của bộ lọc không vượt ra ngoài phần tử cuối cùng của tín hiệu, quay lại Bước 2. Ngược lại, dừng các bước tính toán.

Bên cạnh đó đầu ra tương ứng phần tử của y có thể đi qua một hàm kích hoạt (activaion function) để làm mượt kết quả đầu ra. Hàm kích hoạt ReLu thường được sử dụng trong các lớp này. Hàm này có chức năng đưa về kết quả dương. Cụ thể công thức như sau:

$$f(x) = max(0, x)$$

Hình 8: Công thức hàm kích hoạt ReLU



Hình 9: Biểu diễn hàm kích hoạt ReLU

Ở lớp ẩn này của CNN, ta cần chú ý 3 yếu tố quan trọng:

+ Kích cỡ của bộ lọc (Filter size): Định nghĩa việc sẽ tính toán bao nhiêu phần tử trên một lần trượt. Filter size càng lớn thì tốc độ trượt sẽ càng nhanh và kết quả sẽ càng thấp.

+ Bước nhảy (Stride size): Định nghĩa ta sẽ sử dụng bộ lọc (filter/ window) trượt dài bao nhiêu trên đầu vào. Với bước trược lớn hơn thì sẽ tạo ra ít hơn các output.

+ Chế độ mở rộng (Padding): quy định phần đệm 0 được thêm vào xung quanh input trước khi trượt. Có 3 chế độ phổ biến:

- Valid padding – không thêm.
- Same padding – thêm vào sao cho kết quả bản đồ đặc trưng đầu ra bằng với kích cỡ ma trận đầu vào.
- Full padding – thêm vào sao cho tất cả các phần tử của ma trận đầu vào đều được tham gia vào phép tính bắt đầu với chỉ 1 phần tử đầu tiên. Chế độ này thường làm tăng kích cỡ của đầu ra.

Lớp gộp:

Lớp gộp có chức năng chính là làm giảm kích thước của bản đồ đặc trưng, còn được gọi là lấy mẫu con nhưng vẫn giữ lại được thông tin quan trọng trên bản đồ. Lớp này có cách thức hoạt động như lớp tích chập là có một bộ lọc trượt trên ma trận đầu vào và từng loại bộ lọc khác nhau có cách trích xuất khác nhau. Dựa trên loại bộ lọc người ta cũng phân lớp pooling theo loại bộ lọc đó. Các pooling layer có nhiều loại, loại phổ biến là:

- Max pooling: lấy số lớn nhất trong cửa sổ trượt
- Average pooling: lấy số là trung bình tất cả các số có mặt trong cửa sổ trượt
- Sum pooling: tính tổng các số trong cửa sổ trượt

Lớp kết nối đầy đủ:

Lớp này có cấu trúc như một mạng neural network.

Sau khi input được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của nó. Với FC layer được kết hợp với các tính năng lại với nhau để tạo ra một mô hình. Lớp này cũng thường sử dụng hàm ReLU để tính đầu ra.

Cuối cùng sử dụng softmax hoặc sigmoid để phân loại đầu ra.

Hàm softmax:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

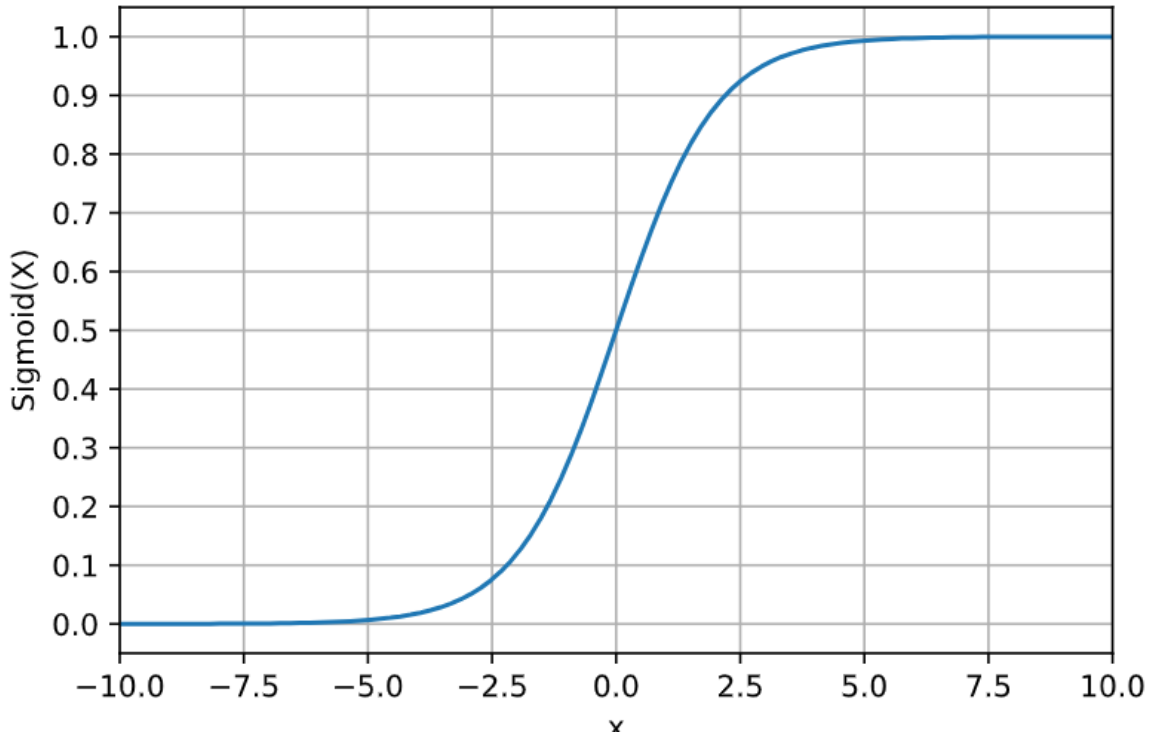
Hình 10: Công thức hàm kích hoạt Softmax

Với zi là kết quả đầu ra của nút hiện tại, zj là kết quả của mỗi nút ở lớp cuối cùng. Kết quả của hàm này là một con số từ 0 đến 1 thể hiện xác suất của dữ liệu vào với nhãn thuộc vị trí này. Hàm softmax thường dùng trong những bài toán phân loại nhiều nhãn.

Hàm sidmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Hình 11: Công thức hàm kích hoạt sigmoid



Hình 12: Biểu diễn của hàm sigmoid

Hàm sigmoid thường nằm ở lớp cuối của mô hình với bài toán phân loại nhị phân (2 nhãn). Lúc này lớp cuối thường chỉ có một unit. Kết quả luôn nằm trong khoảng 0 đến 1, thuộc số thực. Qua đó, có thể phân biệt giữa 2 nhãn bằng cách xem kết quả trả về là nhỏ hơn hoặc lớn hơn 5.

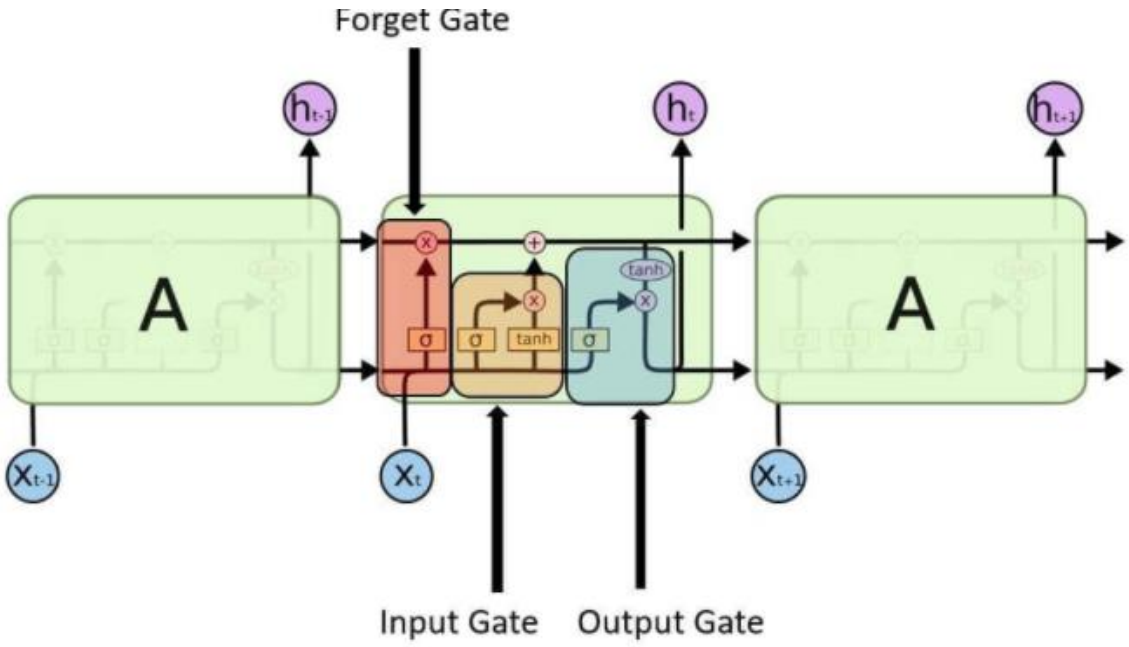
2.5. Kiến trúc LSTM

Long short-term memory (LSTM) là một kiến trúc artificial recurrent neural network (RNN) được sử dụng trong lĩnh vực Deep learning. Nó được đề xuất vào năm 1997 bởi Sepp Hochreiter và Jurgen schmidhuber. Không giống như các feed-forward neural networks, LSTM có các kết nối phản hồi. Nó có thể xử

lý không chỉ các điểm dữ liệu đơn lẻ (chẳng hạn như hình ảnh) mà còn toàn bộ chuỗi dữ liệu (chẳng hạn như speech hoặc video) [8].

Một đơn vị LSTM chung bao gồm một cell, một input gate, một output gate và một forget gate. Cell ghi nhớ các giá trị trong khoảng thời gian tùy ý và ba gate điều chỉnh luồng thông tin input và output. LSTM rất phù hợp để classify, process, và predict có khoảng thời gian không xác định [8].

Mạng Long short-term memory (LSTM) là một phiên bản sửa đổi của mạng nơ-ron tuần hoàn, giúp dễ dàng ghi nhớ dữ liệu quá khứ trong bộ nhớ [8].



Hình 13: Mô hình LSTM

Input gate– Nó phát hiện ra giá trị nào từ đầu vào sẽ được sử dụng để sửa đổi bộ nhớ. Hàm Sigmoid quyết định giá trị nào sẽ cho qua 0 hoặc 1. Và hàm tanh đưa ra trọng số cho các giá trị được truyền, quyết định mức độ quan trọng của chúng trong khoảng từ -1 đến 1 [8].

$$i_t = \sigma(W_i . [h_t - 1, x_t] + bi)$$
$$C_t = \tanh(W_C . [h_t - 1, x_t] + b_C)$$

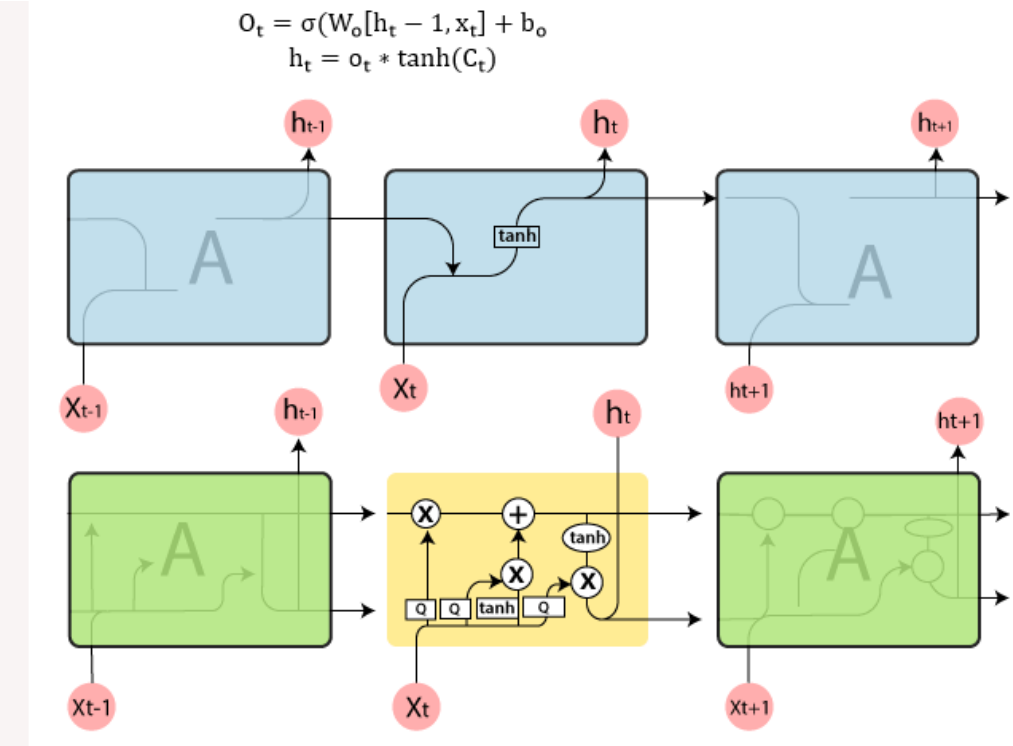
Hình 14: Công thức tính i, c

Forget gate – Nó khám phá các chi tiết cần loại bỏ khỏi khối. Một hàm sigmoid quyết định nó. Nó xem xét trạng thái trước đó (ht-1) và đầu vào nội dung (Xt) và xuất ra một số giữa 0 (bỏ qua điều này) và 1 (giữ nguyên điều này) cho mỗi số trong trạng thái ô Ct-1.

$$f_t = \sigma(W_f . [h_t - 1, x_t] + b_f)$$

Hình 15: Công thức tính f

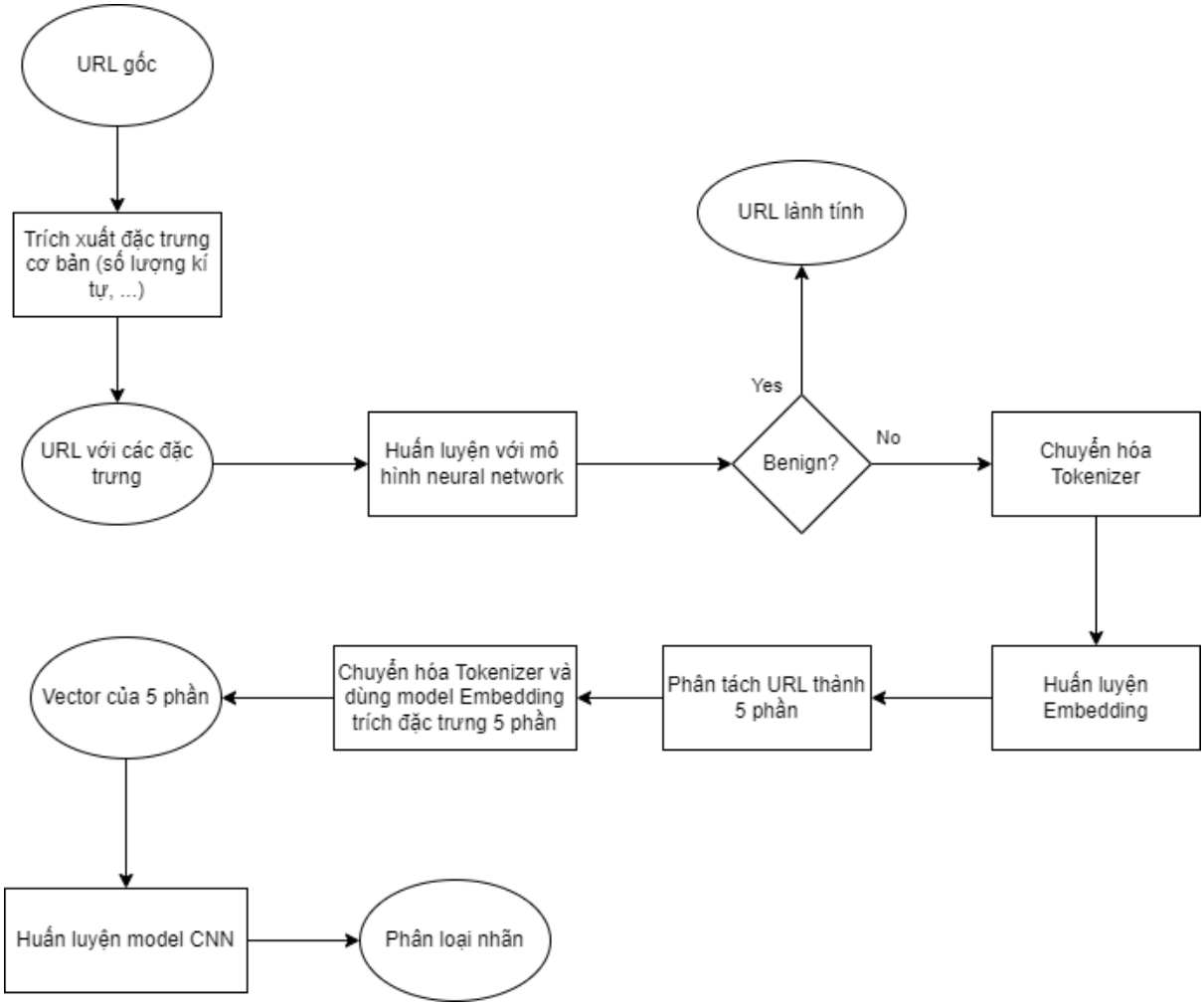
Output gate – Đầu vào và bộ nhớ của khối được sử dụng để quyết định đầu ra. Hàm Sigmoid quyết định giá trị nào cho qua 0 hoặc 1. Và hàm tanh quyết định giá trị nào cho qua 0, 1. Và hàm tanh đưa ra trọng số cho các giá trị được truyền, quyết định mức độ quan trọng của chúng trong khoảng từ -1 đến 1 và nhân lên với đầu ra là sigmoid.



Hình 16: Công thức tính O, h

2.6. Mô hình đề xuất

Chúng em đã thiết kế một mô hình gồm các bước như hình sau:



Hình 17: Mô hình đề xuất

Theo đó, bọn em sẽ tiến hành rút trích một số các đặc trưng của URL như chiều dài url, số chữ số, số chữ cái, số lượng của những kí tự đặc biệt, dùng thư viện hỗ trợ để kiểm tra sự bất thường của URL, để phân loại trước lành tính hay nguy hại từ mô hình Neural Network, trong trường hợp có nguy cơ sẽ tiếp tục dùng phương pháp URL2Vec để trích xuất đặc trưng của 5 thành phần trong URL, tiến hành huấn luyện chúng cùng với model CNN để phân loại nhãn.

Chương 3: Triển khai

1. Mô tả môi trường thực nghiệm và công cụ sử dụng

1.1. Ngôn ngữ lập trình Python (Phiên bản 3.10)

Python là một ngôn ngữ bậc cao, có mục đích chung, thông dịch và hướng đối tượng. Python cho phép chúng em viết các đoạn mã ngắn gọn, rõ ràng và dễ đọc. Python cũng có nhiều thư viện hỗ trợ các lĩnh vực như phân tích dữ liệu, học máy, phát triển web, tự động hóa và phát triển phần mềm. Python là một ngôn ngữ linh hoạt, có thể chạy trên nhiều hệ điều hành khác nhau và tương thích với nhiều nền tảng tính toán đám mây. Đó cũng là lý do đề tài được thực hiện bằng python.

1.2. Trình soạn thảo Visual Studio Code

Visual Studio Code là một trình biên tập mã nguồn đa nền tảng, để viết và chỉnh sửa các đoạn mã Python. Visual Studio Code hỗ trợ nhiều tính năng như gợi ý mã, kiểm tra lỗi, gỡ lỗi, định dạng mã và nhiều phần mở rộng khác.

1.3. Google Colab

Google Colab là một nền tảng tính toán đám mây miễn phí cho phép chạy các đoạn mã Python trên trình duyệt web mà không cần cài đặt bất kỳ phần mềm nào. Google Colab cung cấp miễn phí các tài nguyên tính toán như CPU, GPU và TPU, cũng như các thư viện khoa học và học máy phổ biến như NumPy, Pandas, Matplotlib, Scikit-learn, TensorFlow, Keras và PyTorch. Google Colab cũng cho phép chúng em lưu trữ và chia sẻ các đoạn mã Python dưới dạng các notebook có thể được kết nối với Google Drive, GitHub và nhiều dịch vụ khác.

1.4. Thư viện Tensorflow và Keras

Tensorflow là một nền tảng học máy mã nguồn mở, hỗ trợ xây dựng, huấn luyện, và triển khai các mô hình học sâu với Python, C++, hoặc các ngôn ngữ khác. TensorFlow cung cấp nhiều công cụ và tài nguyên để giải quyết các bài toán thực tế với học máy, như xử lý và tải dữ liệu, sử dụng hoặc tạo các mô hình đã được huấn luyện, triển khai mô hình trên nhiều môi trường.

Keras là một API cao cấp của TensorFlow, giúp tạo và huấn luyện các mô hình học sâu một cách đơn giản, linh hoạt, và mạnh mẽ. Keras tập trung vào tốc độ gỡ lỗi, sự gọn gàn và ngắn gọn của mã, khả năng bảo trì, và khả năng triển khai. Khi chọn Keras, mã lệnh sẽ nhỏ hơn, dễ đọc hơn, dễ tái sử dụng hơn. Mô hình sẽ chạy nhanh hơn nhờ biên dịch XLA với JAX và TensorFlow, và dễ triển khai hơn trên mọi nền tảng nhờ các thành phần phục vụ từ các hệ sinh thái

TensorFlow và PyTorch, như TF Serving, TorchServe, TF Lite, TF.js, và nhiều hơn nữa.

1.5. Thư viện urllib

Thư viện urllib là một gói cung cấp nhiều mô-đun để làm việc với các URL, bao gồm urllib.request để mở và đọc các URL, urllib.error chứa các ngoại lệ được ném ra bởi urllib.request, urllib.parse để phân tích các URL, và urllib.robotparser để phân tích các tệp robots.txt. urllib hỗ trợ xác thực cơ bản và tiêu hóa, chuyển hướng, cookie, HTTPS, FTP, file và dữ liệu URL, và nhiều hơn nữa.

2. Triển khai mô hình

2.1. Trích xuất một số đặc trưng URL

Khởi tạo các hàm kiểm tra

```
import re
def get_domain(url):
    try:
        res=get_tld(url,as_object=True,fail_silently=False,fix_protocol=True)
        pri_domain=res.parsed_url.netloc
    except:
        pri_domain=None
    return pri_domain
def abnormal_url(url):
    hostname=urlparse(url).hostname
    hostname=str(hostname)
    match=re.search(hostname,url)
    if match:
        return 1
    else:
        return 0
def http_secure(url):
    http=urlparse(url).scheme
    match=str(http)
    if match=='https':
        return 1
    else:
        return 0
def digit_count(url):
    digits=0
    for c in url:
        if c.isnumeric():
            digits+=1
    return digits
def letter_count(url):
```

```
    letters=0
    for c in url:
        if c.isalpha():
            letters+=1
    return letters
def find_shortening_service(url):
    match=re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|t
inyurl|tr\.im|is\.gd|cli\.gs|'
                    'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|
su\.pr|twurl\.nl|snipurl\.com|'
                    'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\
.com|snipr\.com|fic\.kr|loopt\.us|'
                    'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|
to\.ly|bit\.do|t\.co|lnkd\.in|'
                    'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyur
l\.com|ow\.ly|bit\.ly|ity\.im|'
                    'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|bu
zurl\.com|cutt\.us|u\.bb|yourls\.org|'
                    'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vztur
l\.com|qr\.net|1url\.com|tweez\.me|v\.gd|'
                    'tr\.im|link\.zip\.net', url)
    if match:
        return 1
    else:
        return 0
def contain_ip_address(url):
    match=re.search('(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-
4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-
5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4 with port
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-
F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}|'
        '([0-9]+(?:\.[0-9]+){3}:[0-9]+)|'
        '((?:(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])\.){3}(?:25[0-5]|2[0-
4]\d|[01]?\d\d|\d)(?:\/\d{1,2})?)', url)
    if match:
        return 1
    else:
        return 0
```

Thực hiện thêm đặc trưng trên tập dữ liệu

```
feature0['url_len']=feature0['url'].apply(lambda x: len(str(x)))
feature0['domain']=feature0['url'].apply(lambda x: get_domain(x))
chars=['@','?','-','=','.','#','%','+','$','!','*',',','//']
```

```
for char in chars:
    feature0[char]=feature0['url'].apply(lambda x: x.count(char))
feature0['abnormal_url']=feature0['url'].apply(lambda x: abnormal_url(x))
feature0['https']=feature0['url'].apply(lambda x: http_secure(x))
feature0['digits']=feature0['url'].apply(lambda x: digit_count(x))
feature0['letters']=feature0['url'].apply(lambda x: letter_count(x))
feature0['shortening_service']=feature0['url'].apply(lambda x:
find_shortening_service(x))
feature0['contain_ip_address']=feature0['url'].apply(lambda x:
contain_ip_address(x))
feature0['type']=feature0['label'].apply(lambda x: 1 if x>0 else 0)
feature0
```

Bảng 1: Tập dữ liệu sau khi thêm đặc trưng

url_len	@	?	-	=	.	#	%	+	$	...	*	,	//	abnormal_url	https	digits	letters	shortening_service	contain_ip_address	type
46	0	0	0	0	5	0	0	0	0	...	0	0	1	1	0	4	31	0	0	1
42	0	0	0	0	3	0	0	0	0	...	0	0	1	1	0	0	35	0	0	1
57	0	1	0	3	3	0	0	0	0	...	0	0	1	1	0	7	37	0	0	1
33	0	0	0	0	3	0	0	0	0	...	0	0	1	1	0	0	26	0	0	1
92	0	0	0	0	1	0	21	0	0	...	0	0	1	1	0	28	36	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
66	0	0	6	0	3	0	0	0	0	...	0	0	1	1	0	6	46	0	0	1
71	0	0	3	0	2	0	0	0	0	...	0	0	1	1	0	4	57	0	0	1
92	0	0	8	0	1	0	0	0	0	...	0	0	1	1	0	8	67	0	0	0
52	0	0	1	0	3	0	0	0	0	...	0	0	1	1	0	0	44	0	0	1
130	0	1	10	4	2	0	0	0	0	...	0	0	1	1	0	5	99	0	0	1

2.2. Dự đoán nhị phân với Neural Network

Chia tập dữ liệu huấn luyện

```
X0=feature0.drop(columns=['type'],axis=1)
y0=feature0['type']
X_train, X_test, y_train, y_test = train_test_split(X0, y0, test_size=0.2,
random_state=42)
```

Xây dựng mô hình

```
def create_model_2label(input_shape):
  model = keras.Sequential([
      Dense(64, activation='relu', input_shape=input_shape),
      Dropout(0.3),

      Dense(32, activation='relu'),
      BatchNormalization(),
      Dropout(0.3),

      Dense(16, activation='relu'),
      BatchNormalization(),
      Dropout(0.3),
```

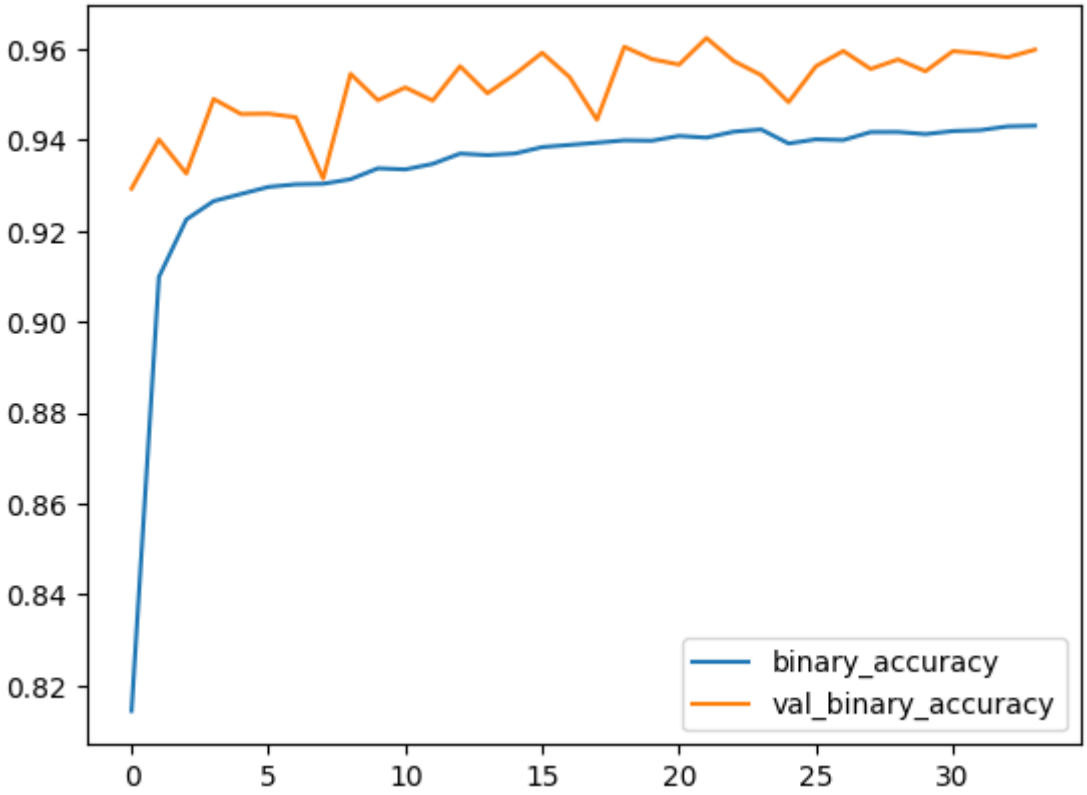
```
    Dense(1, activation='sigmoid'),
    ])
return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 64)                1344

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 batch_normalization (Batch  (None, 32)                128
 Normalization)

 dropout_2 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 16)                528

 batch_normalization_1 (Bat  (None, 16)                64
 chNormalization)

 dropout_3 (Dropout)         (None, 16)                0

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 4161 (16.25 KB)
Trainable params: 4065 (15.88 KB)
Non-trainable params: 96 (384.00 Byte)
_________________________________________________________________
```

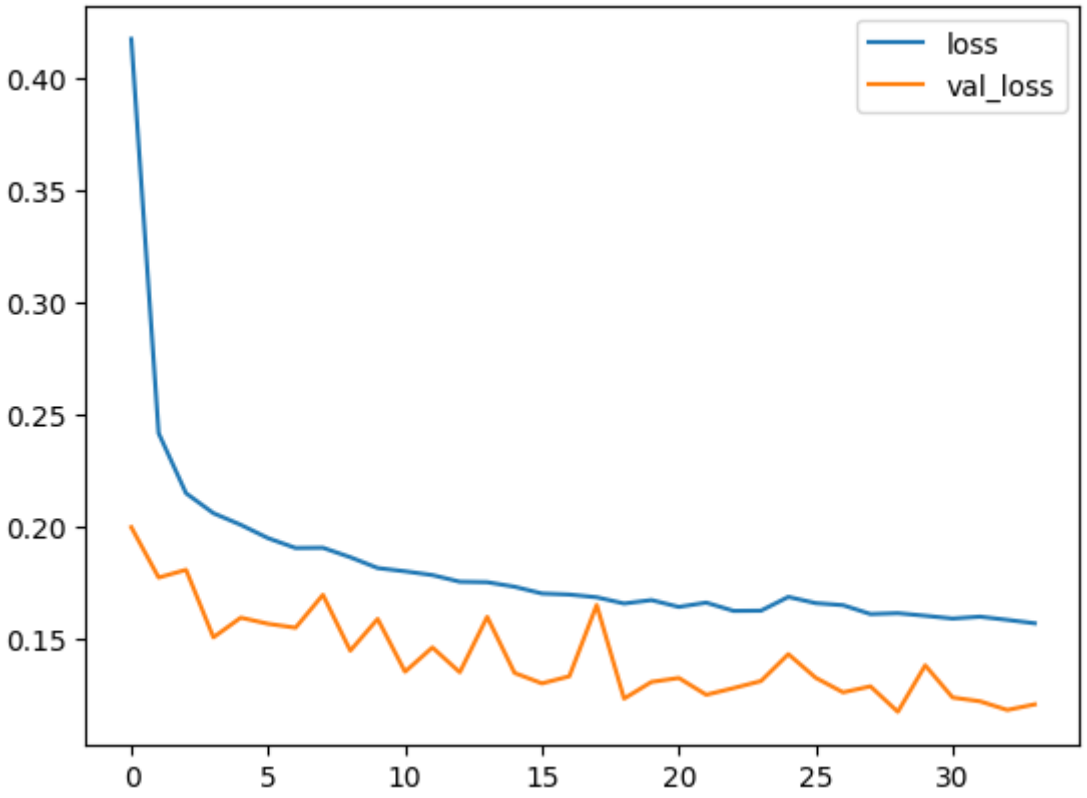
Hình 18: Cấu trúc Neural Network

Kết quả quá trình huấn luyện 33 epochs và đánh giá:

Biểu đồ 1: Binary_accuracy của tập train và validation



Biểu đồ 2: Loss của tập train và validation



Nhận xét: Mô hình hạn chế hiện tượng quá khớp, sau khi huấn luyện, mô hình đã có khả năng phân loại URL lành tính hay độc hại, hiệu quả rất khả quan sau khi kiểm thử trên tập test, chỉ số chính xác accuracy lên tới 95,5%, nhưng độ mất mát vẫn còn ở mức 0,13. Trên thực tiễn, mô hình hoạt động khá ổn định khi nhận diện được những URL thực tế.

	url
0	http://www.beriva.lt/index.html?di=akcijos&V
1	http://www.amicidelgiocodelponte.it/ck.htm
2	https://www.youtube.com/watch?v=jfKfPfyJRdk
3	https://study4.com/tests/?term=ETS
4	https://drive.google.com/drive/u/0/my-drive
5	https://genshin.hoyoverse.com/vi/gift
6	https://www.google.com/?hl=vi
7	http://1337x.to/torrent/1048648/American-Snipe...
8	https://www.youtube.com/?app=desktop
9	https://www.youtube.com/watch?v=EzNyPTVnblQ
10	https://www.kaggle.com/datasets/sid321axn/mali...

```
array([[0.9977215 ],
       [0.9975042 ],
       [0.01452936],
       [0.01298384],
       [0.02031535],
       [0.01599664],
       [0.01261552],
       [0.07565102],
       [0.01353913],
       [0.01452936],
```

Hình 19: Kết quả kiểm định mô hình NN trên URL thực tế

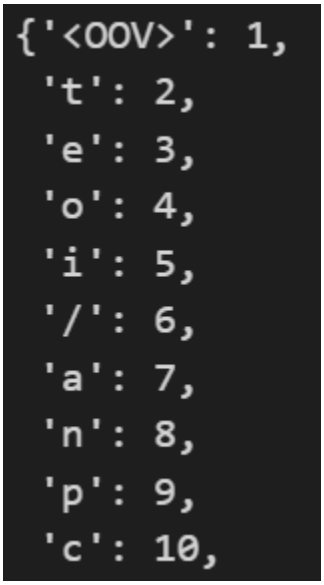
Theo đó, trong 10 URL trên có 2 URL đầu tiên là có nguy cơ độc hại. Tiếp theo đó, bọn em sẽ tạo model cnn để nhận dạng loại nhãn của URL khi nó được mô hình NN nhận diện có nguy cơ.

2.3. Khởi tạo từ điển

```
tokenizer=Tokenizer(oov_token='<OOV>',char_level=True)
tokenizer.fit_on_texts(urls_corpus['url'])
word_index=tokenizer.word_index
word_index
```

Đối tượng Tokenizer cho phép các kí tự trong các chuỗi URL được quy đổi sang số dựa theo tần số xuất hiện của chúng trong tập dữ liệu. Tần suất xuất hiện càng nhiều chỉ số của chúng càng nhỏ.

Danh sách 10 kí tự đầu xuất hiện nhiều nhất, trong đó <OOV> là kí tự chưa xuất hiện trong từ điển sẽ được gán vào URL khi mã hóa.



Hình 20: Danh sách top 10 trong từ điển Tokenizer

Khởi tạo các tham số cho lớp Embedding:

```
maxlen=300
embedding_dims=32
len_dictionary= 138
```

Trong đó, maxlen là số kí tự giữ lại của URL, embedding_dims là số vector mà lớp Embedding sẽ trích xuất, len_dictionary là số lượng kí tự trong từ điển.

Lệnh chuyển đổi chuỗi URL sang dãy số là:

```
token=tokenizer.texts_to_sequences(urls_corpus['url'])
```

Vì kích thước của những URL sẽ khác nhau nên sẽ dùng kĩ thuật padding để điền vào chỗ trống những số 0 hoặc dùng kĩ thuật truncating để cắt bớt các kí tự thừa vượt quá maxlen.

```
token = pad_sequences(token, maxlen=maxlen, padding='post',truncating='post')
```

2.4. Tạo model Embedding

Model có chứa lớp Embedding nhằm để trích xuất các đặc trưng của URL dựa trên mối tương quan của chúng. Mẫu code thiết kế model:

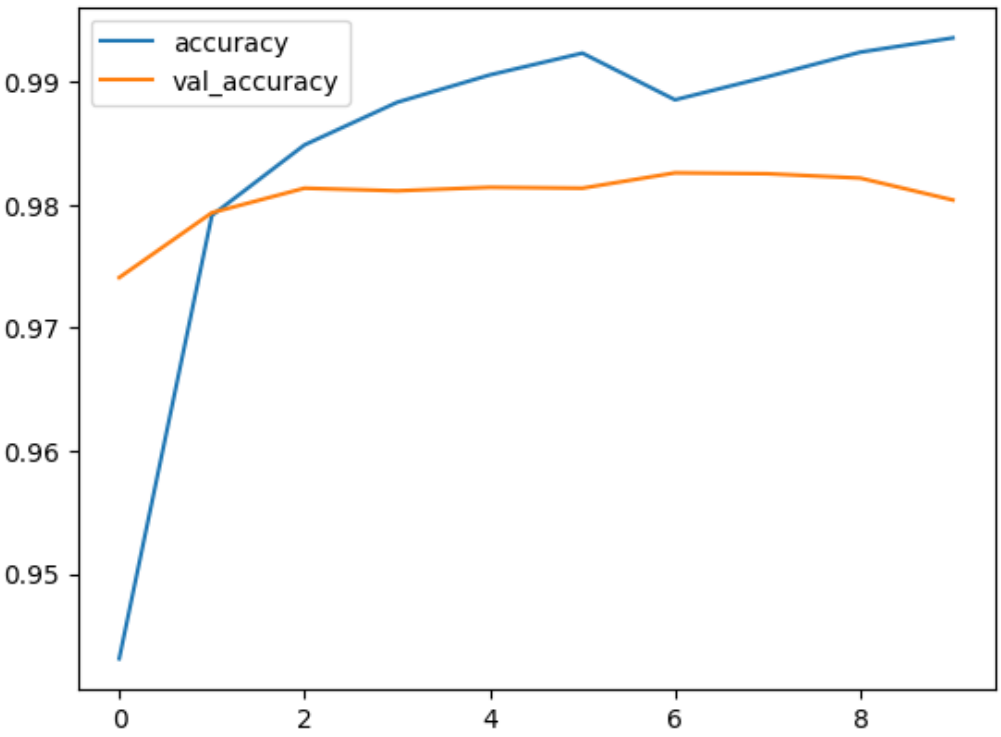
```
def create_embedding(size_label=5):
    model = keras.Sequential([
    Embedding(len_dictionary,embedding_dims,input_length=maxlen),
    Flatten(),
    Dense(10,activation='relu'),
    Dense(size_label, activation='softmax'),
    ])
    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 32)           4416

 flatten (Flatten)           (None, 9600)              0

 dense (Dense)               (None, 10)                96010

 dense_1 (Dense)             (None, 5)                 55

=================================================================
Total params: 100481 (392.50 KB)
Trainable params: 100481 (392.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

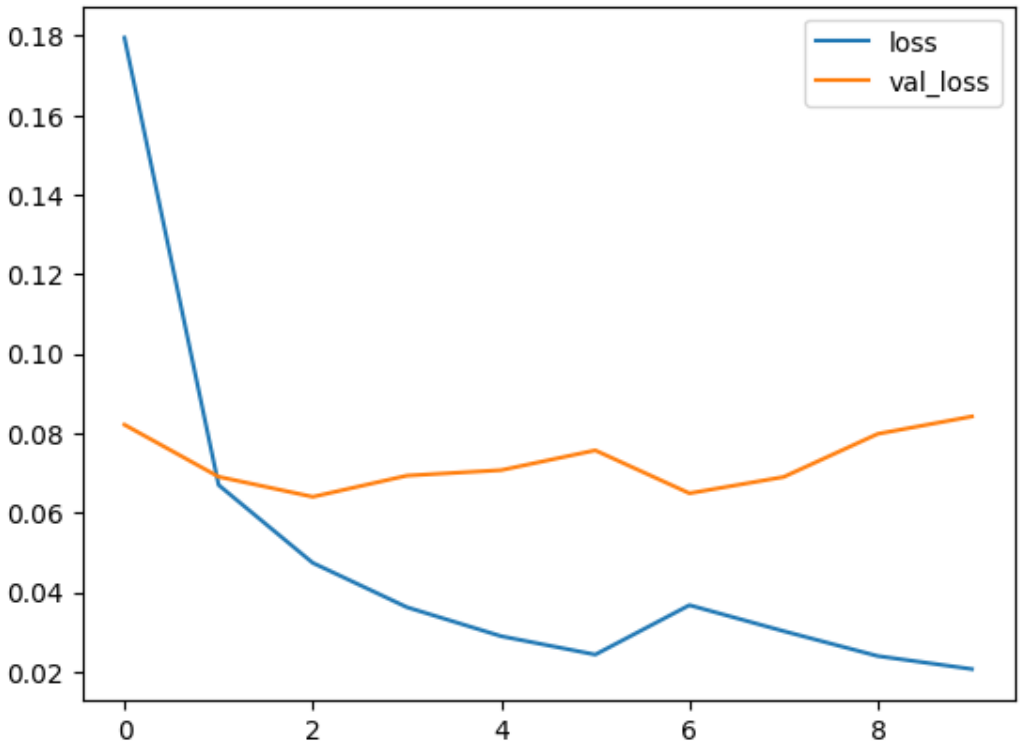
Hình 21: Cấu trúc Embedding model và Param trên mỗi lớp

Kết quả quá trình huấn luyện và đánh giá:

Biểu đồ 3: Accuracy của train và validation



Biểu đồ 4: Loss của train và validation



Model được đánh giá trên tập test cho thấy có độ chính xác lên tới 98%, độ mất mát chỉ ở 0,071. Như vậy mô hình đạt hiệu suất cao trên việc phân loại. Bên cạnh đó khi kiểm nghiệm trên thực tế, mô hình cho thấy còn dự đoán chính xác với những trang web lành tính.

```
benign https://www.youtube.com/
benign https://www.kaggle.com/code/shujian/blend-of-lstm-and-cnn-with-4-embeddings-1200d
benign https://study4.com/tests/?term=ETS
benign https://phamdinhkhanh.github.io/2019/04/22/Ly_thuyet_ve_mang_LSTM.html
benign https://drive.google.com/drive/u/0/my-drive
benign https://genshin.hoyoverse.com/vi/gift
benign https://www.google.com/?hl=vi
benign http://1337x.to/torrent/1048648/American-Sniper-2014-MD-iTALiAN-DVDSCR-X264-BST-MT/
benign https://www.youtube.com/?app=desktop
benign https://www.youtube.com/watch?v=EzNyPTVnblQ
benign https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset
```

Hình 22: Kết quả thử nghiệm trên những URL thực tế của Embedding model

2.5. Model CNN & LSTM phân loại nhãn URL

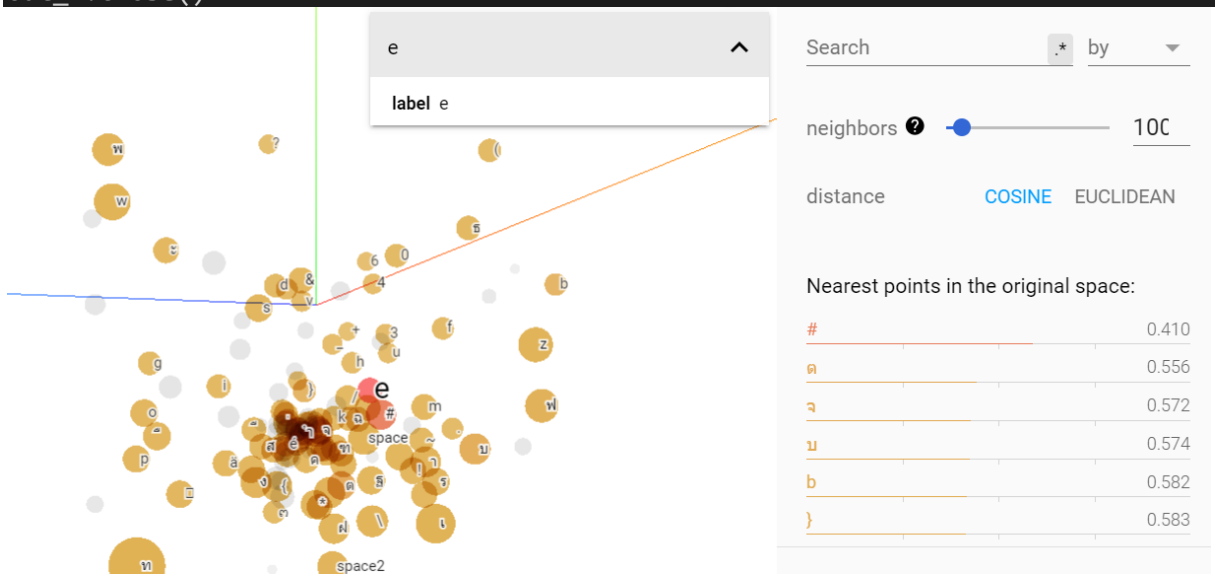
Trước khi huấn luyện mô hình, tập URL cần được phân tách thành 5 phần và cho qua lớp Embedding để trích xuất đặc trưng của chúng và tính trung bình theo từng kí tự trong chuỗi.

Đầu tiên ta cần lấy ra ma trận Embedding trong model vừa train ở mục 2.4:

```
embedding_layer=model.get_layer('embedding')
embedding_matrix=embedding_layer.weights[0].numpy()
```

Ma trận này có thể được lưu trữ và phóng lên không gian nhiều chiều để quan sát sự tương quan của các kí tự nằm gần nhau.

```
import io
out_v=io.open('/content/drive/MyDrive/Model/vecs.tsv','w',encoding='utf-8')
out_m=io.open('/content/drive/MyDrive/Model/meta.tsv','w',encoding='utf-8')

for idx in range(1,len_dictionary+1):
  word=tokenizer.index_word[idx]
  vector=embedding_matrix[idx-1]
  if idx==91:
    word='space'
  elif idx==94:
    word='space2'
  out_m.write(word+"\n")
  out_v.write('\t'.join([str(x) for x in vector])+"\n")
out_m.close()
out_v.close()
```



Hình 23: Ảnh chụp trực quan từ ma trận Embedding

(thực hiện trên trang https://projector.tensorflow.org/)

Tiếp theo tạo một model như sau để trích xuất đặc trưng:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 300, 32)           4416

 global_average_pooling1d (  (None, 32)                0
 GlobalAveragePooling1D)

=================================================================
Total params: 4416 (17.25 KB)
Trainable params: 4416 (17.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Hình 24: Cấu trúc model trích đặc trưng Embedding

Tạo hàm phân tách các thành phần trong URL:

```
def protocol(url):
    result=urlparse(url)
    return result.scheme
def sub_domain(url):
    domain=tldextract.extract(url)
    return domain.subdomain
def domain(url):
    domain=tldextract.extract(url)
    return domain.domain
def suffix(url):
    domain=tldextract.extract(url)
    return domain.suffix
def path(url):
    result=urlparse(url)
    return result.path
```

Thêm cột mang thành phần vào tập URL:

```
slice_data['protocol_url']=slice_data['url'].apply(lambda x: protocol(x))
slice_data['sub_domain_url']=slice_data['url'].apply(lambda x: sub_domain(x))
slice_data['domain_url']=slice_data['url'].apply(lambda x: domain(x))
slice_data['suffix_domain_url']=slice_data['url'].apply(lambda x: suffix(x))
slice_data['path_url']=slice_data['url'].apply(lambda x: path(x))
slice_data.head(1)
```

	url	protocol_url	sub_domain_url	domain_url	suffix_domain_url	path_url
0	http://amber.ch.ic.ac.uk/archive/all/5075.html	http	amber.ch	ic	ac.uk	/archive/all/5075.html

Hình 25: Mẫu URL đã phân tách

Tạo hàm lấy đặc trưng:

```
def get_feature(data):
  temp_data = tokenizer.texts_to_sequences(data)
  temp_data = pad_sequences(temp_data, maxlen=maxlen,
padding='post',truncating='post')
  feature=model.predict(temp_data)

  del(temp_data)
  return feature
```

Tiến hành lấy đặc trưng từng phần:

```
feature=get_feature(slice_data['protocol_url'])
feature2=get_feature(slice_data['sub_domain_url'])
feature3=get_feature(slice_data['domain_url'])
feature4=get_feature(slice_data['suffix_domain_url'])
feature5=get_feature(slice_data['path_url'])
feature=np.concatenate((feature,feature2,feature3,feature4,feature5),axis=1)
feature.shape
feature=pd.DataFrame(feature,columns=['f'+str(x)for x in
range(1,feature.shape[1]+1)])
feature.head(1)
```

Mỗi thành phần trích xuất ra 32 đặc trưng x 5 sẽ có tổng cộng 160 cột trong data feature

Bảng 2: Tập dữ liệu mẫu của feature

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	...	f152
f1	1.000000	-0.941773	0.553775	-0.750186	0.503830	0.331230	-0.539199	0.681233	-0.684243	0.580848	...	0.030426
f2	-0.941773	1.000000	-0.789923	0.806295	-0.713146	-0.190920	0.675229	-0.472847	0.452389	-0.670465	...	-0.034060
f3	0.553775	-0.789923	1.000000	-0.792268	0.959425	0.101277	-0.855531	0.100358	0.161631	0.786022	...	0.040273
f4	-0.750186	0.806295	-0.792268	1.000000	-0.875051	-0.654505	0.951820	-0.672598	0.126649	-0.973287	...	-0.054121
f5	0.503830	-0.713146	0.959425	-0.875051	1.000000	0.334000	-0.957488	0.249031	0.259460	0.908370	...	0.048839
...	...	...	...	...	...	...	...	...	...	...	...	...
f157	0.057846	-0.064825	0.068686	-0.082688	0.075712	0.052406	-0.081027	0.050757	-0.003331	0.081913	...	-0.136296
f158	0.010091	-0.013931	0.021746	-0.025641	0.026252	0.019605	-0.028394	0.013002	0.009791	0.028188	...	0.404011
f159	0.012120	-0.004733	-0.021313	0.027320	-0.033518	-0.033970	0.038664	-0.009310	-0.039725	-0.037960	...	-0.318405
f160	0.025085	-0.026745	0.026103	-0.033530	0.028946	0.022509	-0.031774	0.022978	-0.004676	0.032613	...	-0.213159
label	-0.216202	0.234639	-0.227197	0.274082	-0.244055	-0.163170	0.259653	-0.176210	0.042581	-0.263375	...	-0.106457

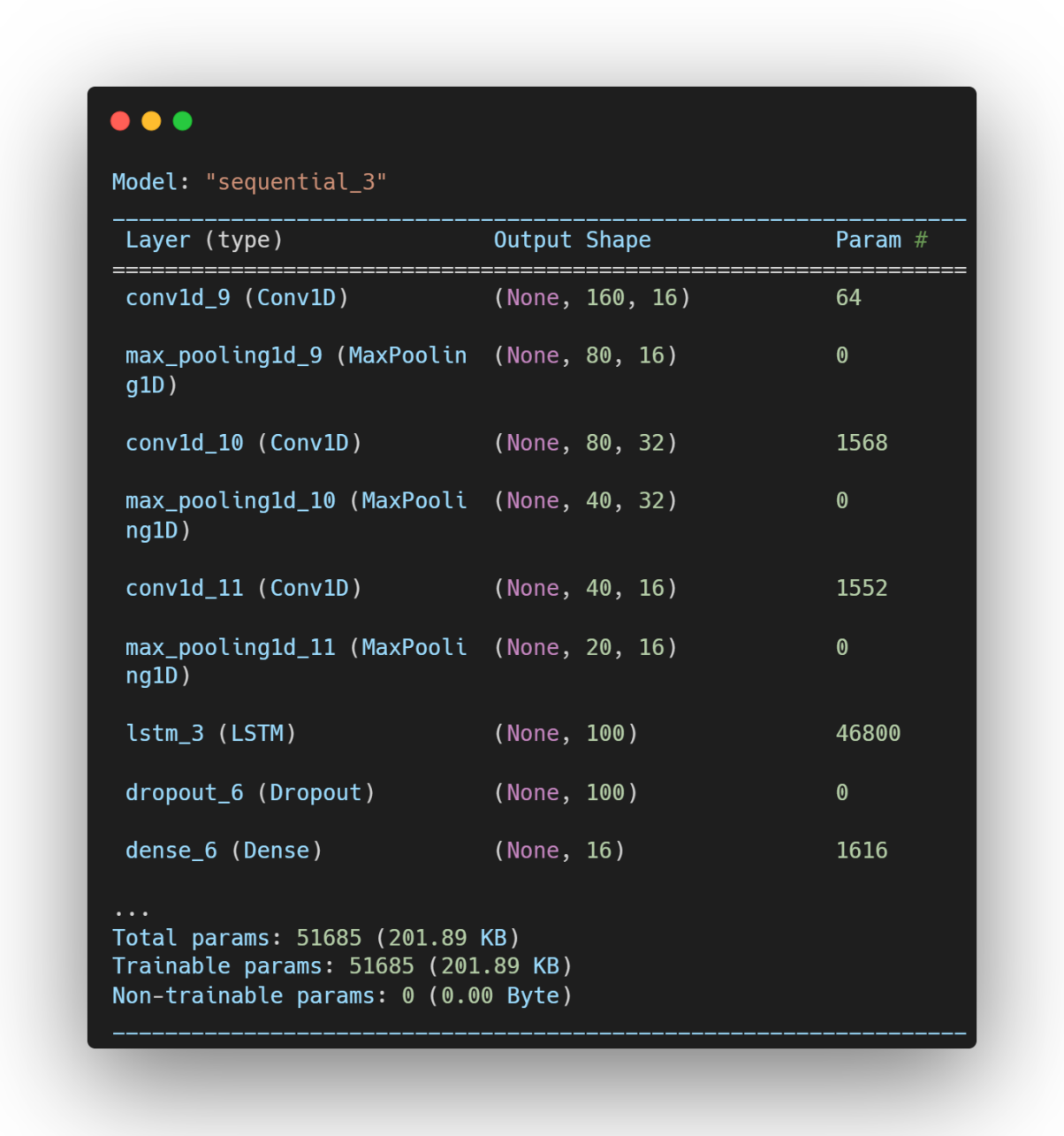
Ta tiến hành phân chia tập dữ liệu sang train và test:

```
X_train, X_test, y_train, y_test = train_test_split(feature,
urls_corpus['label'], test_size=0.2, random_state=42)
```

((140292, 160), (140292,), (35074, 160), (35074,)) là kích cỡ của train và test.

Tạo mô hình CNN và LSTM:

```
def create_cnn_lstm_model(input_shape,size_label=5):
    model=Sequential([
        Conv1D(16,3,padding='same',activation='relu',input_shape=input_shape),
        MaxPooling1D(pool_size=2),

        Conv1D(32,3,padding='same',activation='relu'),
        MaxPooling1D(pool_size=2),

        Conv1D(16,3,padding='same',activation='relu'),
        MaxPooling1D(pool_size=2),

        LSTM(100),
        Dropout(0.3),
        Dense(16,activation='relu'),
        Dropout(0.3),
        Dense(size_label,activation='softmax')
    ])
    return model
```



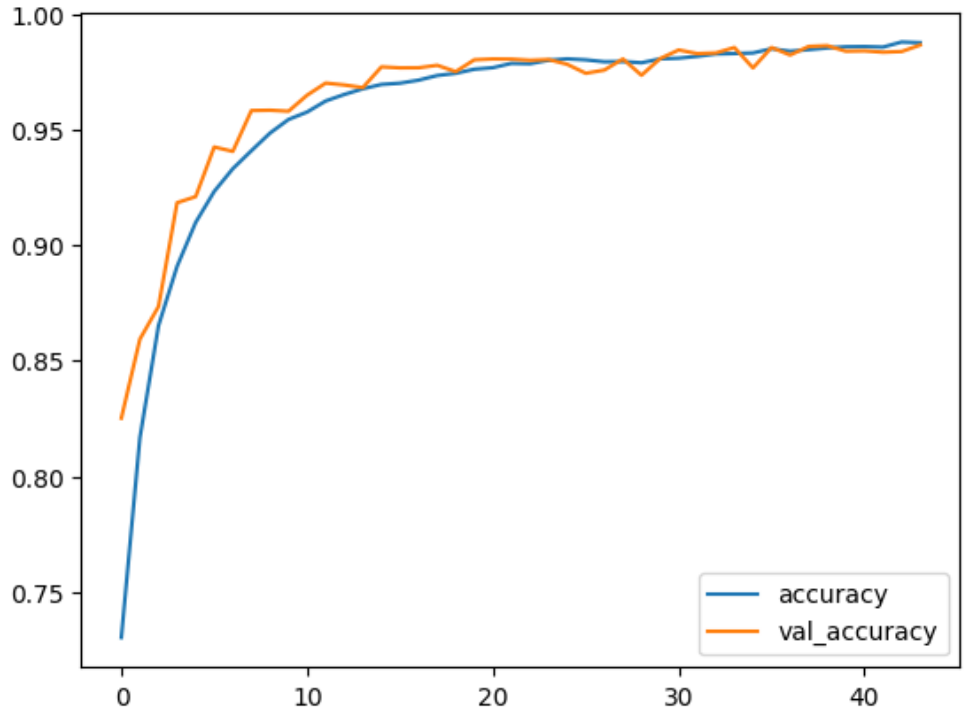
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_9 (Conv1D)           (None, 160, 16)           64

 max_pooling1d_9 (MaxPoolin  (None, 80, 16)            0
 g1D)

 conv1d_10 (Conv1D)          (None, 80, 32)            1568

 max_pooling1d_10 (MaxPooli  (None, 40, 32)            0
 ng1D)

 conv1d_11 (Conv1D)          (None, 40, 16)            1552

 max_pooling1d_11 (MaxPooli  (None, 20, 16)            0
 ng1D)

 lstm_3 (LSTM)               (None, 100)               46800

 dropout_6 (Dropout)         (None, 100)               0

 dense_6 (Dense)             (None, 16)                1616

...
Total params: 51685 (201.89 KB)
Trainable params: 51685 (201.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Hình 26: Cấu trúc của CNN và LSTM model

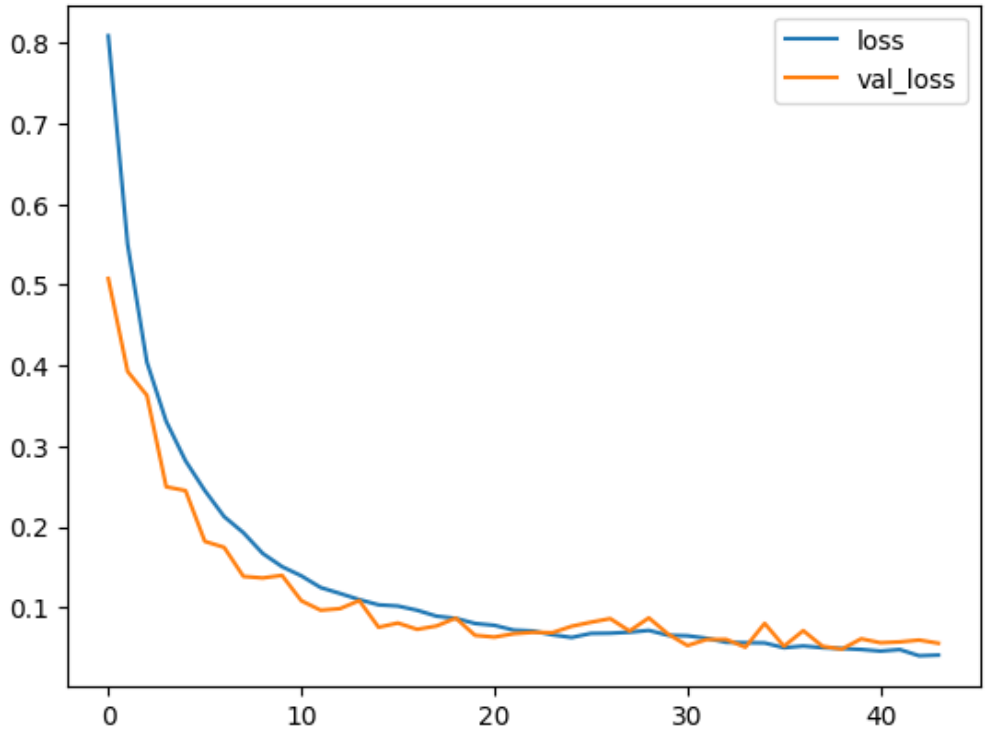
Nhận xét và đánh giá kết quả:

Sau 43 epochs, mô hình đã đạt được những kết quả nhất định trong việc phân loại URL.

Biểu đồ 5: Accuracy của train và validation



Biểu đồ 6: Loss của train và validation



Theo đó, có thể thấy trong quá trình training, mô hình hầu như không bị hiện tượng quá khớp, kết quả kiểm định trên tập test cho thấy mô hình có thể nhận

định đúng nhãn lên tới 98,5% và độ mất mát chỉ ở 0,05. Tuy nhiên mô hình vẫn nhầm lẫn một số mã URL lành tính sang độc hại.

```
malware https://colab.research.google.com/drive/1mdR7wKjBbXOvvPCZchNGZy1s18A9Qx7B?authuser=2#scrollTo=041vBLFFKgYo
phishing https://safebrowsing.google.com
benign https://www.youtube.com/watch?v=jfKfPfyJRdk
benign https://study4.com/tests/?term=ETS
phishing https://drive.google.com/drive/u/0/my-drive
phishing https://genshin.hoyoverse.com/vi/gift
benign https://www.google.com/?hl=vi
benign http://1337x.to/torrent/1048648/American-Sniper-2014-MD-iTALiAN-DVDSCR-X264-BST-MT/
benign https://www.youtube.com/?app=desktop
benign https://www.youtube.com/watch?v=EzNyPTVnblQ
benign https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset
```

Hình 27: Kết quả kiểm nghiệm trên URL thực tế

Nguyên nhân của vấn đề này có thể tới từ sự thiếu đa dạng trong dataset hoặc sai lầm trong thiết lập lớp mạng CNN. Để khắc phục cần có thêm thời gian nghiên cứu về vấn đề nêu trên. Hiện trạng trên được khắc phục tạm thời bằng cách sử dụng một mô hình Neural Network để kiểm tra trước nguy cơ vì mô hình này hoạt động trong việc nhận định URL lành tính rất tốt.

2.6. Tích hợp mô hình vào ứng dụng extension

Để ứng dụng khả năng của mô hình vào trong thực tiễn, bọn em đã tích hợp chúng vào ứng dụng extension sử dụng trong trình duyệt.

Trong ứng dụng, bọn em đã tiến hành bắt sự kiện URL có sự thay đổi trên thanh địa chỉ.

```
chrome.webNavigation.onBeforeNavigate.addListener(function (details) {
  if (details.frameId === 0 && !mainFrameNavigated) {
    const urlBeforeNavigate = details.url;
    console.log('URL trước khi truy cập:', urlBeforeNavigate);
    mainFrameNavigated = true;
    // Sử dụng phương thức startsWith để kiểm tra URL của popup
    if (urlBeforeNavigate.endsWith("popup/popup.html") ||
urlBeforeNavigate.indexOf("//new-tab-page/") != -1) {
      isPopup = true; // đặt trạng thái là popup
      return; // bỏ qua xử lý
    } else {
      isPopup = false; // đặt trạng thái là bình thường
    };


    // Kiểm tra xem người dùng có chọn tiếp tục hay không
    if (isContinue) {
      return; // bỏ qua xử lý
    };
    // Gửi url về server để xử lý
```

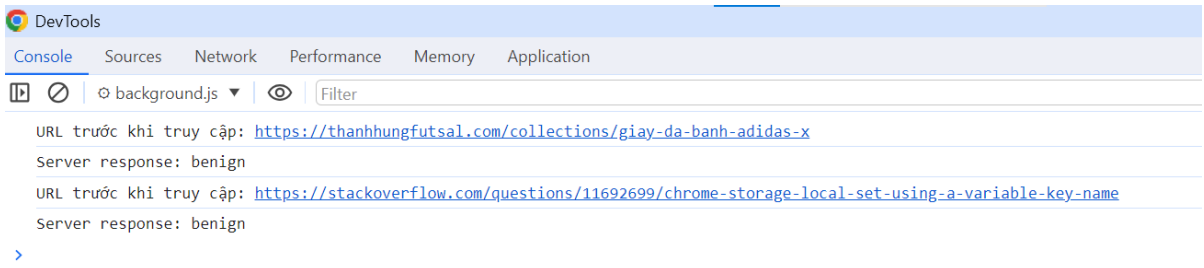
```
    const dataToSend = { "url": urlBeforeNavigate };

    fetch('http://localhost:5000/process_data', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(dataToSend),
    })
      .then(response => response.json())
      .then(data => {
        // Xử lý kết quả từ server (ví dụ: in ra console)
        console.log('Server response:', data.result);
        if (data.result != "benign" && !isPopup) {

          chrome.storage.local.set({ url: urlBeforeNavigate }, function () {
            // Tạo một tab mới với URL của popup
            chrome.tabs.update(details.tabId, { url: "popup/popup.html" });
          });
        }
      })
      .catch(error => {
        console.error('Error:', error);
      })

  }
  mainFrameNavigated = false;
});
```

Nếu URL được model dự đoán khác nhãn ‘benign’ URL hiện tại sẽ chuyển hướng sang trang popup.html hiển thị giao diện cảnh báo người dùng về nguy cơ độc hại từ trang web truy cập. popup.html nhận lựa chọn của người dùng có muốn tiếp tục truy cập hay trở lại trang trước.

Dưới đây là đoạn áp dụng dự đoán trên URL:

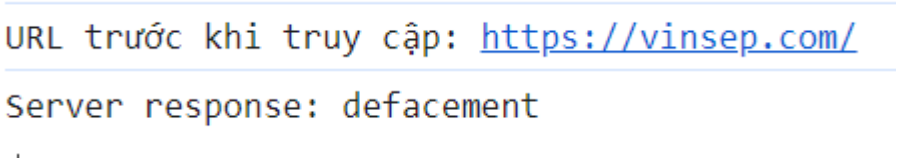
```
def predict(data):
    try:
        score=nn_predict(data)
        print('Score: ',score)
        if score < 0.5:
            return 0
        return cnn_predict(data)
    except Exception as e:
        print(e)
        return e

def process_data():
```

```
    data = request.get_json()
    # Xử lý dữ liệu ở đây (ví dụ: in ra console)
    print("Received data:", data)
    y_pred=predict(data)
    if y_pred is not None:
        print("Predict label: ",label[y_pred])
        # Trả về kết quả cho client (ví dụ: chuỗi "Processed!")
        return jsonify({"result": label[y_pred]})
    print("Predict label: Error!!!")
    return jsonify({"result": "Error predict!"})
```



Hình 28: Lịch sử ghi nhận truy cập và dự đoán nhãn

Hình 29: Ví dụ trường hợp URL có nguy cơ độc hại



Hình 30: Ghi nhận trong lịch sử truy cập

Lưu ý: Trang web trên hoàn toàn lành tính, nhóm chỉ nhúng một URL độc hại khác trong ứng dụng set mặc định để test trường hợp, không cố ý công kích cá nhân hay tổ chức nào

Chương 4: Kết luận

Qua thực nghiệm, nhóm em đã triển khai việc trích xuất đặc trưng ở URL, xây dựng thành công các mô hình học sâu để dự đoán nhãn của URL, đã đạt được nhiều kết quả tốt.

Với mô hình Neural Network, phân loại URL có lành tính hay không, mô hình đạt 95,5%, nhưng độ mất mát vẫn còn ở mức 0.13 trên tập kiểm tra. Mô hình có thể phân loại một cách ổn định trên tập URL lành tính hay có nguy cơ độc hại.

Với mô hình Character Embedding, phân loại 5 nhãn của URL, mô hình cho kết quả khả quan nhất với 98%, độ mất mát chỉ ở 0,071 trên tập kiểm tra. Hơn thế, mô hình phân loại rất hiệu quả 5 nhãn với những URL thực tế. Tuy nhiên mô hình khá đơn giản và ít tham số param, như thế có nguy cơ kém hiệu quả trong tương lai.

Với mô hình CNN và LSTM kết hợp với việc trích xuất đặc trưng, mô hình cho kết quả rất cao 98,5% và độ mất mát chỉ ở 0,05 trên tập kiểm tra. Tuy nhiên, mô hình được nhóm em nhìn nhận là có khả năng kém với việc phân loại URL lành tính trên thực tế.

Từ đó, đề tài đã xuất hiện những điểm hạn chế như mô hình CNN và LSTM chưa đạt được kết quả như mong muốn, mô hình đề xuất để dự đoán cần phải sử dụng kết hợp nhiều model với nhau gây ra sự lãng phí về tài nguyên, nảy sinh thêm thời gian cho công việc dự đoán URL.

Hướng phát triển trong tương lai, nhóm cần thu thập thêm dữ liệu làm phong phú tập training, tìm hiểu, nghiên cứu, thiết lập những mô hình có hiệu quả cao hơn, nhằm đạt được thành tựu cao trong công tác phân loại URL độc hại, góp phần tránh tình trạng lừa đảo, đánh cắp thông tin, gây hại đến người dùng mạng Internet. Bên cạnh đó, ứng dụng extension cần cải thiện thêm về mặc giao diện, khiến cho trải nghiệm người dùng trở nên sinh động.

Nguồn tài liệu kham khảo

[1] T. H. Định, "Ứng dụng máy học trong phát hiện các liên kết độc hại trên Internet," in *Trường đại học Nguyễn Tất Thành*, Thành phố Hồ Chí Minh, 2021.

[2] T. Phong, "URL là gì? Cấu trúc của URL," Quantrimang, 29 06 2023. [Online]. Available: https://quantrimang.com/cong-nghe/url-la-gi-158090. [Accessed 04 01 2024].

[3] C. I. f. Cybersecurity, "URL dataset (ISCX-URL2016)," UNB, [Online]. Available: https://www.unb.ca/cic/datasets/url-2016.html. [Accessed 04 01 2024].

[4] N. C. Minh, "Neural Network, Deep Learning và các ứng dụng trong cuộc sống," Tạp chí Ngân Hàng, 20 09 2023. [Online]. Available: https://tapchinganhang.gov.vn/neural-network-deep-learning-va-cac-ung-dung-trong-cuoc-song.htm. [Accessed 04 01 2024].

[5] Q. Vũ, "Tìm hiểu về lớp Embedding trong Keras," Medium, 15 09 2020. [Online]. Available: https://medium.com/@vudinhquyad/t%C3%ACm-hi%E1%BB%83u-v%E1%BB%81-l%E1%BB%9Bp-embedding-trong-keras-c11c3f08c2a3. [Accessed 04 01 2024].

[6] Huaping Yuan, Zhenguo Yang, Xu Chen, Yukun Li, Wenyin Liu, "URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection," *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications*, pp. 265-272, 2018.

[7] V. H. Tiệp, "Bài 37: Tích chập hai chiều," Machine Learning cơ bản, 03 10 2018. [Online]. Available: https://machinelearningcoban.com/2018/10/03/conv2d#-tich-chap-mot-chieu. [Accessed 04 01 2024].

[8] D. Đ. Trình, "Long short-term memory (LSTM) là gì?," websitehcm.com, 01 04 2022. [Online]. Available: https://websitehcm.com/long-short-term-memory-lstm-la-gi/. [Accessed 04 01 2024].