***ĐÁNH GIÁ HỆ THỐNG PHÁT HIỆN XÂM NHẬP BẰNG CÁC GIẢI THUẬT HỌC SÂU***

1. **CHUẨN BỊ DATASET**

Dataset là CSE-CIC-IDS2018 có thể tải trục tiếp từ kaggle bằng link <https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv>

1. **GIỚI THIỆU CÁC GIẢI THUẬT HỌC SÂU SỬ DỤNG TRONG DỰ ÁN**

**1. Giới thiệu về giải thuật KNN**

K-Nearest Neighbors (KNN) là một giải thuật học máy đơn giản nhưng hiệu quả, đặc biệt trong các bài toán phân loại. Ý tưởng chính của KNN là khi cần dự đoán nhãn của một mẫu mới, thuật toán sẽ tìm ra K mẫu gần nhất trong không gian đặc trưng và sử dụng đa số phiếu từ các nhãn của chúng để đưa ra kết quả cuối cùng. Do đó, KNN không cần huấn luyện phức tạp, mà chủ yếu dựa trên khoảng cách giữa các điểm dữ liệu, thường sử dụng khoảng cách Euclidean. Tuy nhiên, nhược điểm của KNN là tốn nhiều thời gian khi dữ liệu lớn vì phải tính toán khoảng cách cho toàn bộ tập mẫu. Trong Python, để triển khai KNN, ta thường sử dụng thư viện scikit-learn với lớp KNeighborsClassifier, cho phép dễ dàng xác định số láng giềng K, huấn luyện mô hình, và đánh giá độ chính xác trên dữ liệu thử nghiệm.

Ví dụ code cho mô hình KNN như sau:

from sklearn.neighbors import KNeighborsClassifierfrom sklearn.metrics import accuracy\_score

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)print("Độ chính xác KNN:", accuracy\_score(y\_test, y\_pred))

**2. Giới thiệu về giải thuật RNN**

Recurrent Neural Network (RNN) là một dạng mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu tuần tự, chẳng hạn như chuỗi thời gian, văn bản hoặc tín hiệu mạng. Điểm mạnh của RNN là có khả năng ghi nhớ trạng thái từ bước tính toán trước, nhờ đó giữ được ngữ cảnh của dữ liệu theo thời gian. Một trong những biến thể phổ biến nhất của RNN là LSTM (Long Short-Term Memory), giúp khắc phục vấn đề gradient biến mất hoặc bùng nổ khi huấn luyện chuỗi dài. LSTM sử dụng các cổng điều khiển (input gate, forget gate, output gate) để duy trì và điều chỉnh thông tin trong bộ nhớ, khiến mô hình học được cả quan hệ ngắn hạn lẫn dài hạn trong dữ liệu.

Trong Python, RNN hoặc LSTM thường được cài đặt bằng TensorFlow hoặc Keras. Thư viện này cung cấp các lớp LSTM và SimpleRNN, cho phép định nghĩa số lượng đơn vị ẩn, độ sâu của mạng, và hàm kích hoạt. Việc huấn luyện cũng tương tự như các mô hình học sâu khác với hàm mất mát và bộ tối ưu hoá.

Ví dụ code cho mô hình LSTM như sau:

from tensorflow.keras import layers, models

rnn = models.Sequential([

layers.LSTM(64, input\_shape=(timesteps, features)),

layers.Dense(64, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

rnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

rnn.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

**3. Giới thiệu về giải thuật CNN**

Convolutional Neural Network (CNN) là một trong những giải thuật học sâu mạnh mẽ nhất trong xử lý ảnh và dữ liệu có cấu trúc không gian. CNN hoạt động dựa trên phép tích chập (convolution), trong đó các bộ lọc (filters) được trượt qua ảnh để phát hiện ra các đặc trưng cục bộ như cạnh, góc, hình dạng. Thông qua nhiều lớp tích chập và gộp (pooling), mô hình CNN có khả năng học ra các đặc trưng ngày càng phức tạp và trừu tượng. Cuối cùng, các đặc trưng này được đưa vào các lớp kết nối đầy đủ (dense layers) để phân loại.

Ưu điểm lớn nhất của CNN là giảm số lượng tham số cần huấn luyện so với mạng fully connected, đồng thời vẫn giữ được tính hiệu quả trong nhận dạng. Trong Python, CNN thường được triển khai với TensorFlow hoặc Keras, nơi có các lớp Conv2D, MaxPooling2D, Flatten, và Dense để xây dựng một mô hình hoàn chỉnh.

Ví dụ code cho mô hình CNN như sau:

from tensorflow.keras import layers, models

cnn = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 1)),

layers.MaxPooling2D(2, 2),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D(2, 2),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

cnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

cnn.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

**4. Các thư viện cần thiết để cài đặt mô hình**

Để triển khai đầy đủ ba mô hình CNN, RNN và KNN, ta cần sử dụng một số thư viện quan trọng trong Python. Thư viện NumPy dùng để xử lý mảng số liệu hiệu quả, Matplotlib để trực quan hoá dữ liệu và kết quả, PIL để xử lý ảnh, và scikit-learn để cung cấp công cụ xây dựng mô hình KNN cũng như tách dữ liệu huấn luyện và kiểm tra. Với các mô hình học sâu như CNN và RNN, ta cần sử dụng TensorFlow hoặc Keras để xây dựng kiến trúc mạng, huấn luyện và đánh giá.

Các câu lệnh import điển hình như sau:

import osimport timeimport numpy as npimport matplotlib.pyplot as pltfrom PIL import Imagefrom sklearn.model\_selection import train\_test\_splitfrom sklearn.preprocessing import LabelEncoderfrom sklearn.neighbors import KNeighborsClassifierfrom sklearn.metrics import accuracy\_scoreimport tensorflow as tffrom tensorflow.keras import layers, modelsfrom tensorflow.keras.utils import to\_categorical

Như vậy, ba giải thuật CNN, RNN và KNN mỗi giải thuật đều có ưu điểm và hạn chế riêng, đồng thời phù hợp với từng loại dữ liệu khác nhau. Việc triển khai nhiều giải thuật và so sánh kết quả sẽ giúp ta đánh giá được mô hình nào hoạt động hiệu quả nhất trên bộ dữ liệu IDS.

1. **CHUẨN BỊ MÔI TRƯỜNG ĐỂ THỰC HIỆN**

Đề tài được thực hiện trên môi trường googlecolab với thông số TPU v2-8 156GB, thực hiện trên ngôn ngữ lập trình là python với phiên bản là python 3. Kết nối với googledrive với thư mục chứa tập dataset bằng lệnh:

from google.colab import drive

drive.mount('/content/drive')

1. **CÁC BƯỚC TIỀN XỬ LÍ DỮ LIỆU**
2. **Các thư viện cần thiết**

Trước tiên, ta cần cài thêm thư viện imbalanced-learn để phục vụ việc **cân bằng dữ liệu** bằng kỹ thuật RandomOverSampler.

!pip install -q imbalanced-learn

1. **Thêm các thư viện**

* Pandas, numpy: xử lí dữ liệu dạng bảng.
* Matplotlib, seaborn: vẽ biểu đồ phân tích.
* Scikit-learn: hỗ trợ mã hóa, chuẩn hóa, chia tập dữ liệu.
* Os, gc, datetime: thao tác file/thư mục và quản lí bộ nhớ.

import pandas as pd

import numpy as np

import gc, os, pickle

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

from collections import Counter

from datetime import datetime

from imblearn.over\_sampling import RandomOverSampler

1. **Xác định thư mục dư liệu**

Thư mục chứa tập dư liệu dataset trên drive có đường dẫn /content/drive/MyDrive/ColabNotebooks/nienluancoso1/DATA/CSE-IDS2018

Sau đây là lênhk đọc dữ liệu của file trong thư mục chứ dataset

csv\_folder = '/content/drive/MyDrive/ColabNotebooks/nienluancoso1/DATA/CSE-IDS2018'

print(f"[1] Đọc dữ liệu từ thư mục: {csv\_folder}")

1. **Đọc và gộp các file csv**

Ở bước này ta sẽ duyệt toàn bộ file trong tập dataset với mỗi file ta sẽ loại bỏ cột không cần như flowid, timesttamp.Sau đó ta tiếp tục xử lí các giá trị NaN/Inf rồi gộp nó thành 1 dataFrame chung.

df\_list = []

for filename in os.listdir(csv\_folder):

if filename.endswith('.csv'):

filepath = os.path.join(csv\_folder, filename)

print(f" → Đang đọc: {filename}")

try:

df = pd.read\_csv(filepath, low\_memory=False)

df.drop(columns=['Flow ID', 'Timestamp'], inplace=True, errors='ignore')

df.replace([np.inf, -np.inf], np.nan, inplace=True)

df.dropna(inplace=True)

if 'Label' in df.columns:

df\_list.append(df)

except Exception as e:

print(f" ❌ Lỗi đọc {filename}: {e}")

df = pd.concat(df\_list, ignore\_index=True)

print(f"[2] Tổng số dòng sau khi gộp: {df.shape[0]}, số cột: {df.shape[1]}")

gc.collect()

1. **Tách đặt trưng và nhãn**

Sau khi gộp dữ liệu từ nhiều file CSV thành một bảng lớn df, ta cần phân chia dữ liệu thành hai phần: là đặt trưng(Features-X) và nhãn(lable-Y)

Đặt trưng là các cột chứa thông tin dùng để mô hình học ví dụ như trong tập dữ liệu IDS có các đặt trung như protocol, flow Duration,Fwd Pakets,v.v.

Nhãn là các cột mục tiêu ta muốn mô hình dự đoán ví dụ như trong tập dữ liệu IDS,cột này có tên lable, với các giá trị dạng chuỗi như BENIGN, Dos Hulk, PortScan, Bot,v.v.Nói cách khác đây chính là mô hình phải học cách phân loại.

X = df.drop(columns=['Label']) #Lấy toàn bộ dữ liệu trừ cột 'Label'

y = df['Label'] # Lấy riêng cột 'Label' làm nhãn

1. **Mã hóa dữ liệu dạng chuỗi**

Trong tập dữ liệu, có một số cột đặc trưng là **chuỗi ký tự (string)**, ví dụ: Protocol.  
Mô hình học máy/học sâu chỉ làm việc với **số** nên ta cần chuyển các chuỗi này thành số.

Ta dùng **LabelEncoder** để biến mỗi chuỗi thành một số nguyên duy nhất.

print("[3] Mã hoá các cột dạng chuỗi...")

for col in X.select\_dtypes(include=['object']).columns: # Tìm tất cả cột kiểu object (chuỗi)

X[col] = LabelEncoder().fit\_transform(X[col].astype(str))

print(f" → {col} đã mã hoá.")

Ví dụ:

Nếu cột Protocol có giá trị ['TCP', 'UDP', 'ICMP']  
→ Sau mã hoá thành [0, 1, 2].

1. **Mã hóa nhãn**

Tương tự, cột Label cũng là chuỗi (BENIGN, DoS Hulk, …).  
Ta cũng dùng **LabelEncoder** để chuyển thành số.’

print("[4] Mã hoá nhãn Label...")

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

Ví dụ: BENIGN -> 0, Dos Hulk -> 1,...

Sau khi thực hiện thì các nhãn này đã sẵn sàng đưa vào mô hình.

1. **Phân tích dữ liệu trước khi cân bằng**

Vì dữ liệu IDS có sự **mất cân bằng lớp** (lớp BENIGN thường chiếm số lượng rất lớn so với các lớp tấn công), nên ta cần trực quan hóa số lượng mẫu mỗi lớp trước khi cân bằng.

print("[📊] Phân phối nhãn trước khi cân bằng:")

class\_counts = pd.Series(y).value\_counts() # Đếm số mẫu theo lớp

class\_labels = label\_encoder.inverse\_transform(class\_counts.index) # Đổi về tên nhãn gốc

plt.figure(figsize=(10, 5))

sns.barplot(x=class\_labels, y=class\_counts.values, palette='viridis')

plt.title("Số lượng mẫu theo lớp trước khi cân bằng", fontsize=14)

plt.xlabel("Lớp (Label)")

plt.ylabel("Số lượng mẫu")

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

Biểu đồ sẽ cho ta thấy lớp nào nhiều, lớp nào ít (thường BENIGN >> DoS >> các loại tấn công khác).

1. **Cân bằng dữ liệu với RamdomOverSampler**

Vì mô hình học sâu cần dữ liệu phân phối cân đối, ta dùng **RandomOverSampler** để **sao chép thêm dữ liệu của lớp thiểu số** cho đến khi tất cả lớp có số lượng bằng nhau.

print("[⚖️] Cân bằng dữ liệu bằng RandomOverSampler...")

ros = RandomOverSampler(random\_state=42)

X\_resampled, y\_resampled = ros.fit\_resample(X, y)

# In ra kết quả phân phối sau cân bằng

balanced\_counts = Counter(y\_resampled)

for idx, count in balanced\_counts.items():

label = label\_encoder.inverse\_transform([idx])[0]

print(f" {label}: {count} mẫu")

1. **Chuẩn hóa dữ liệu**

Đưa tất cả đặc trưng về cùng thang đo bằng **StandardScaler**.  
Điều này giúp mô hình hội tụ nhanh hơn và tránh bị lệ thuộc vào cột có giá trị lớn và điều này cũng giúp cho việc tiết kiệm bộ nhớ, CPU, GPU,.. cho môi trường là máy tính cá nhân.

print("[5] Chuẩn hoá đặc trưng bằng StandardScaler...")

X\_scaled = StandardScaler().fit\_transform(X\_resampled)

1. **Chia dữ liệu train/test**

Để đánh giá mô hình, ta cần chia dữ liệu thành hai phần: tập train 80% để huấn luyện và tập test 20% để kiểm tra đọo chính xác.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_scaled, y\_resampled, test\_size=0.2, random\_state=42, stratify=y\_resampled

)

print(f"[6] Train: {X\_train.shape}, Test: {X\_test.shape}")

1. Lưu kết quả

Lưu train/test ra CSV để dễ xem và .npy để huấn luyện nhanh hơn sau này.

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

output\_dir = '/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output'

os.makedirs(output\_dir, exist\_ok=True)

# Train

train\_df = pd.DataFrame(X\_train, columns=X.columns)

train\_df['Label'] = y\_train

train\_csv = os.path.join(output\_dir, f"train\_data\_{timestamp}.csv")

train\_df.to\_csv(train\_csv, index=False)

# Test

test\_df = pd.DataFrame(X\_test, columns=X.columns)

test\_df['Label'] = y\_test

test\_csv = os.path.join(output\_dir, f"test\_data\_{timestamp}.csv")

test\_df.to\_csv(test\_csv, index=False)

# Numpy

np.save(os.path.join(output\_dir, "X\_train.npy"), X\_train)

np.save(os.path.join(output\_dir, "X\_test.npy"), X\_test)

np.save(os.path.join(output\_dir, "y\_train.npy"), y\_train)

np.save(os.path.join(output\_dir, "y\_test.npy"), y\_test)

Sau khi tiền xử lý, dữ liệu đã được chuẩn hoá và lưu dưới dạng **mảng numpy** (X\_train.npy). Tiếp theo, ta sẽ chuyển các vector này thành ảnh để sử dụng cho huấn luyện mô hình học sâu dạng CNN.

1. **TẠO TẬP ẢNH DATASET ĐỂ HUẤN LUYỆN**

1. **Cài đặt các thư viện cần thiết:**

Trước tiên, dữ liệu trong X\_train.npy được tải vào bộ nhớ bằng thư viện Numpy. Việc nạp dữ liệu này giúp ta biết được kích thước của tập dữ liệu, ví dụ như số lượng mẫu và số chiều đặc trưng của mỗi mẫu. Trong trường hợp dữ liệu có hàng trăm nghìn mẫu, ta không thể tạo ảnh cho tất cả trong một lần chạy, vì vậy để thử nghiệm ban đầu, chỉ 10.000 mẫu đầu tiên được lựa chọn để xử lý. Điều này giúp tiết kiệm thời gian và tài nguyên, đồng thời vẫn đảm bảo đủ số lượng dữ liệu để thử nghiệm mô hình.

import numpy as np

import os

from PIL import Image

1. **Tải dữ liệu vector đã xử lí**

Sau bước tiền xử lý, bạn đã lưu X\_train.npy trong thư mục output. Đây chính là dữ liệu đầu vào để chuyển thành ảnh.

# Đường dẫn đến file X\_train.npy

X\_train\_path = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/X\_train.npy"

# Tải dữ liệu X\_train

X\_train = np.load(X\_train\_path)

# In thông tin cơ bản

print("📂 Kích thước X\_train:", X\_train.shape)

Ví dụ kết quả thu được là 1 file có kích thước x\_train: (200000, 78)

Tức là có 200000 mẫu, mỗi mẫu có 78 đặt trưng hoặc có thể hiểu là vector 78 chiều.

1. **Lấy mẫu dữ liệu để tạo ảnh**

Vì dữ liệu có thể rất lớn, ta chỉ lấy một phần (ví dụ **10.000 mẫu đầu tiên**) để thử nghiệm.

X\_sample = X\_train[:10000]

print("📊 Số vector được chọn:", X\_sample.shape[0])

1. **Chuẩn bị thư mục lưu ảnh**

Ảnh sẽ được lưu dưới dạng .png trong một thư mục riêng.

output\_dir = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/dataset\_10k"

os.makedirs(output\_dir, exist\_ok=True)

print("📁 Thư mục lưu ảnh:", output\_dir)

1. **Xác định kich thước ảnh**

Vì vector không phải lúc nào cũng có số chiều phù hợp để tạo ảnh vuông, ta cần chọn kích thước cố định.  
Ví dụ: 32x32 = 1024 pixel trong khi vector chỉ có 78 giá trị. Giải pháp đặt ra là ngắn hơn 1024 thì pad thêm số 0 cho đủ, nếu dài hơn thì cắt bỏ cho vừa.

image\_size = (32, 32) # Kích thước ảnh mong muốn

size = image\_size[0] \* image\_size[1]

1. **Chuyển vector thành ảnh**

Duyệt qua từng vector và xử lý

for i, vector in enumerate(X\_sample):

# Nếu vector ngắn hơn số pixel → pad thêm số 0

if len(vector) < size:

vector = np.pad(vector, (0, size - len(vector)))

# Nếu vector dài hơn số pixel → cắt bớt

elif len(vector) > size:

vector = vector[:size]

# Chuyển vector thành ma trận 2D (32x32)

image\_array = vector.reshape(image\_size)

# Chuẩn hoá về dạng ảnh (0–255, kiểu uint8)

image = Image.fromarray((image\_array \* 255).astype(np.uint8))

# Lưu ảnh với tên dạng 0.png, 1.png, …

image.save(os.path.join(output\_dir, f"{i}.png"))

print("✅ Đã tạo xong 10.000 ảnh đầu tiên!")

1. **Kiểm tra và lưu ảnh**

Sẽ in ra các ảnh đầu tiên để xem thử có bị lỗi không và ảnh sẽ được lưu.

from IPython.display import Image as IPyImage, display

# Hiển thị 5 ảnh đầu tiên

for i in range(5):

display(IPyImage(filename=os.path.join(output\_dir, f"{i}.png")))

vector = (vector - vector.min()) / (vector.max() - vector.min() + 1e-8)

1. **MÔ HÌNH CNN** (CNN có **64 + số lớp nhãn nơ-ron fully connected** (chưa tính filter Conv).

## Đọc ảnh và nhãn

Trong bước đầu tiên, chương trình thực hiện việc tải dữ liệu ảnh và nhãn tương ứng. Các nhãn được lưu trữ trong một file .npy, trong khi tập ảnh được lưu dưới dạng các file .png. Hàm load\_images\_labels() đảm nhiệm vai trò đọc toàn bộ dữ liệu này.

Khi thực thi, chương trình sẽ lần lượt duyệt qua từng tệp ảnh, chuyển ảnh về dạng grayscale (ảnh xám), chuẩn hóa kích thước về đúng chuẩn (32x32), sau đó chia tỷ lệ các giá trị pixel về khoảng [0,1].

import osimport cv2import numpy as np

def load\_images\_labels(image\_dir, label\_path):

# Đọc nhãn từ file .npy

labels = np.load(label\_path, allow\_pickle=True)

images = []

for file\_name in sorted(os.listdir(image\_dir)):

if file\_name.endswith(".png"):

img = cv2.imread(os.path.join(image\_dir, file\_name), cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img, (32, 32)) # Chuẩn hóa kích thước ảnh

img = img / 255.0 # Đưa về [0,1]

images.append(img)

return np.array(images).reshape(-1, 32, 32, 1), labels

## Chuẩn bị dữ liệu

Dữ liệu ảnh (X) và nhãn (y) được đưa vào xử lý. Các nhãn ban đầu ở dạng chuỗi văn bản như "BENIGN", "DoS", "PortScan",… được mã hóa thành số nguyên bằng LabelEncoder. Sau đó, chúng được chuyển thành **one-hot vector** để phục vụ cho bài toán phân loại nhiều lớp.

Tiếp đó, dữ liệu được chia làm hai phần: 80% huấn luyện và 20% kiểm thử.

from sklearn.model\_selection import train\_test\_splitfrom sklearn.preprocessing import LabelEncoderfrom tensorflow.keras.utils import to\_categorical

# Tải dữ liệu

image\_dir = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/DATA/CSE-IDS2018/images"

label\_path = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/y\_train.npy"

X, y\_raw = load\_images\_labels(image\_dir, label\_path)

# Mã hóa nhãn

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_raw)

y = to\_categorical(y\_encoded)

# Chia train/test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

## Xây dựng mô hình CNN

Mạng CNN được xây dựng gồm nhiều lớp: các lớp tích chập (Conv2D) để trích xuất đặc trưng, các lớp pooling để giảm chiều dữ liệu, một lớp Dense để học đặc trưng phức tạp, và lớp đầu ra Softmax để phân loại.

import tensorflow as tffrom tensorflow.keras import layers, models

model = models.Sequential([

layers.Conv2D(32, (3,3), activation='relu', input\_shape=(32,32,1)),

layers.MaxPooling2D((2,2)),

layers.Conv2D(64, (3,3), activation='relu'),

layers.MaxPooling2D((2,2)),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(y.shape[1], activation='softmax')

])

# Biên dịch mô hình

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

## Huấn luyện mô hình

Mô hình được huấn luyện trên dữ liệu với số epoch là 10 và batch size là 64. Một phần nhỏ (10%) dữ liệu huấn luyện được dùng để kiểm tra (validation).

import time

start\_time = time.time()

history = model.fit(

X\_train, y\_train,

epochs=10,

batch\_size=64,

validation\_split=0.1,

verbose=1

)

end\_time = time.time()print("⏱️ Thời gian huấn luyện:", round(end\_time - start\_time, 2), "giây")

## Đánh giá mô hình

Sau huấn luyện, mô hình được đánh giá trên tập test để đo độ chính xác.

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)print("🎯 Độ chính xác trên tập test:", round(test\_acc\*100, 2), "%")

## Dự đoán mẫu

Mẫu ngẫu nhiên từ tập test được đưa vào mô hình để minh họa quá trình dự đoán.

import random

idx = random.randint(0, len(X\_test)-1)

sample = X\_test[idx].reshape(1, 32, 32, 1)

true\_label = np.argmax(y\_test[idx])

predicted\_label = np.argmax(model.predict(sample))

print("Nhãn thật:", label\_encoder.inverse\_transform([true\_label])[0])print("Nhãn dự đoán:", label\_encoder.inverse\_transform([predicted\_label])[0])

1. **Ma trận nhầm lẫn**

Để phân tích chi tiết hơn, ta dùng ma trận nhầm lẫn nhằm kiểm tra mô hình dự đoán đúng/sai ở từng loại nhãn.

from sklearn.metrics import confusion\_matriximport matplotlib.pyplot as pltimport seaborn as sns

y\_pred = model.predict(X\_test)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true = np.argmax(y\_test, axis=1)

cm = confusion\_matrix(y\_true, y\_pred\_classes)

plt.figure(figsize=(10,8))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",

xticklabels=label\_encoder.classes\_,

yticklabels=label\_encoder.classes\_)

plt.xlabel("Dự đoán")

plt.ylabel("Thực tế")

plt.title("Ma trận nhầm lẫn")

plt.show()

1. **HUẤN LUYỆN RNN (LSTM)**(RNN có **128 + số lớp nhãn nơ-ron** (64 LSTM + 64 Dense)

# **1. Đọc ảnh và nhãn dữ liệu**

Trước tiên, ta cần đọc dữ liệu ảnh đã được sinh ra từ vector đặc trưng ở bước trước. Các ảnh này nằm trong thư mục dataset\_10k với kích thước chuẩn hóa 32x32 pixel. Song song đó, ta cũng cần tải tệp y\_train.npy chứa nhãn tương ứng của từng ảnh.

Ở đây, ta viết một hàm load\_images\_labels để thực hiện hai công việc chính: mở từng ảnh, chuyển sang ảnh mức xám (grayscale), resize về 32x32 và chuẩn hóa giá trị điểm ảnh về khoảng [0,1]; sau đó, gom dữ liệu thành mảng numpy. Hàm cũng trả về danh sách nhãn đã cắt sao cho trùng khớp với số lượng ảnh.

def load\_images\_labels(image\_dir, label\_path, image\_size=(32, 32)):

labels = np.load(label\_path)

images = []

filenames = sorted(os.listdir(image\_dir), key=lambda x: int(x.split(".")[0]))

for fname in filenames:

img\_path = os.path.join(image\_dir, fname)

img = Image.open(img\_path).convert("L").resize(image\_size)

images.append(np.array(img) / 255.0)

images = np.array(images)

return images, labels[:len(images)]

# **2. Chuẩn bị dữ liệu**

Sau khi đã có ảnh và nhãn, ta cần chuyển nhãn từ dạng chữ (ví dụ: "BENIGN", "DoS", "PortScan", …) sang dạng số để mô hình học được. Công việc này dùng LabelEncoder của scikit-learn, sau đó mã hóa one-hot bằng to\_categorical.

Ảnh đầu vào là ma trận 32x32. Với RNN (cụ thể là LSTM), ta coi mỗi hàng ảnh là một bước thời gian (time step), và mỗi cột ảnh là một đặc trưng (feature). Như vậy, dữ liệu đầu vào có dạng (samples, 32, 32) thay vì (samples, 32, 32, 1) như trong CNN.

image\_dir = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/dataset\_10k"

label\_path = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/y\_train.npy"

X, y\_raw = load\_images\_labels(image\_dir, label\_path)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_raw)

y\_categorical = to\_categorical(y\_encoded)

X = X.reshape(-1, 32, 32)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y\_categorical, test\_size=0.2, random\_state=42

)

# **3. Xây dựng mô hình RNN (LSTM)**

Mô hình RNN sử dụng một lớp **LSTM** với 64 đơn vị nhớ để học quan hệ tuần tự trong từng hàng của ảnh. Sau đó, ta thêm một lớp Dense với 64 neuron kèm hàm kích hoạt ReLU nhằm tăng khả năng biểu diễn phi tuyến tính. Cuối cùng là lớp Dense đầu ra với số neuron bằng số lớp nhãn, kết hợp softmax để phân loại.

model = models.Sequential([

layers.LSTM(64, input\_shape=(32, 32)),

layers.Dense(64, activation='relu'),

layers.Dense(y\_categorical.shape[1], activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# **4. Huấn luyện và đo thời gian**

Tiếp theo, ta huấn luyện mô hình trong 10 epoch với batch size bằng 64. Để đánh giá quá trình học, dữ liệu được tách thêm 10% từ tập huấn luyện làm validation. Đồng thời, ta ghi lại thời gian huấn luyện để so sánh hiệu năng.

start\_train = time.time()

history = model.fit(

X\_train, y\_train,

epochs=10,

batch\_size=64,

validation\_split=0.1,

verbose=1

)

end\_train = time.time()

train\_time = end\_train - start\_trainprint(f"⏱️ Thời gian huấn luyện: {train\_time:.2f} giây")

# **5. Đánh giá mô hình**

Sau huấn luyện, mô hình được đánh giá trên tập test. Kết quả gồm loss và accuracy, trong đó accuracy là chỉ số quan trọng để biết khả năng phân loại của RNN.

loss, accuracy = model.evaluate(X\_test, y\_test)print(f"✅ Độ chính xác trên tập test: {accuracy \* 100:.2f}%")

# **6. Dự đoán một mẫu và đo thời gian suy luận**

Ở bước này, ta thử đưa một mẫu ảnh trong tập test vào mô hình để kiểm tra dự đoán. Đồng thời, đo thời gian suy luận (inference) để đánh giá tốc độ phản ứng của mô hình khi chạy thực tế.

sample = X\_test[0:1]

start\_pred = time.time()

prediction = model.predict(sample)

end\_pred = time.time()

predicted\_class = label\_encoder.inverse\_transform([np.argmax(prediction)])print(f"📌 Dự đoán lớp: {predicted\_class[0]}")print(f"⏱️ Thời gian đưa ra phán đoán: {(end\_pred - start\_pred) \* 1000:.2f} ms")

# **7. Vẽ ma trận nhầm lẫn**

Cuối cùng, ta vẽ ma trận nhầm lẫn để biết chi tiết mô hình phân loại nhầm giữa các lớp nào. Đây là công cụ rất hữu ích để phân tích lỗi và cải thiện mô hình.

y\_test\_labels = np.argmax(y\_test, axis=1)

y\_pred\_probs = model.predict(X\_test)

y\_pred\_labels = np.argmax(y\_pred\_probs, axis=1)

cm = confusion\_matrix(y\_test\_labels, y\_pred\_labels)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=label\_encoder.classes\_)

disp.plot(xticks\_rotation=45, cmap=plt.cm.Blues)

plt.title("🔁 Confusion Matrix - RNN (LSTM)")

plt.show()

1. **HUẤN LUYỆN MÔ HÌNH KNN**

## Đọc dữ liệu ảnh và nhãn

Ở bước đầu tiên, ta cần xây dựng một hàm có nhiệm vụ đọc toàn bộ ảnh trong thư mục dataset, đồng thời liên kết chúng với nhãn tương ứng được lưu trong file y\_train.npy. Ảnh được resize về kích thước 32x32, chuyển sang grayscale (mức xám), sau đó chuẩn hóa giá trị pixel về khoảng [0,1]. Cuối cùng, ảnh được làm phẳng (flatten) thành vector để có thể đưa vào mô hình KNN.

def load\_images\_labels(image\_dir, label\_path, image\_size=(32, 32)):

labels = np.load(label\_path)

images = []

filenames = sorted(os.listdir(image\_dir), key=lambda x: int(x.split(".")[0]))

for fname in filenames:

img\_path = os.path.join(image\_dir, fname)

img = Image.open(img\_path).convert("L").resize(image\_size)

images.append(np.array(img) / 255.0) # chuẩn hóa về [0,1]

images = np.array(images).reshape(len(images), -1) # flatten

return images, labels[:len(images)]

## Chuẩn bị dữ liệu huấn luyện và kiểm thử

Sau khi ảnh và nhãn được load thành công, ta tiến hành mã hóa nhãn về dạng số nguyên (Label Encoding). Tiếp đó, dữ liệu được chia thành hai phần: tập huấn luyện (80%) và tập kiểm thử (20%) để đánh giá hiệu suất của mô hình.

image\_dir = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/dataset\_10k"

label\_path = "/content/drive/MyDrive/ColabNotebooks/nienluancoso1/output/y\_train.npy"

X, y\_raw = load\_images\_labels(image\_dir, label\_path)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_raw)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y\_encoded, test\_size=0.2, random\_state=42

)

## Huấn luyện mô hình KNN

Ở bước này, ta xây dựng mô hình KNN với số láng giềng gần nhất là 3 (n\_neighbors=3). Sau đó, mô hình được huấn luyện trên tập huấn luyện. Đồng thời, ta đo thời gian huấn luyện để đánh giá chi phí tính toán.

knn = KNeighborsClassifier(n\_neighbors=3)

start\_train = time.time()

knn.fit(X\_train, y\_train)

end\_train = time.time()

train\_time = end\_train - start\_trainprint(f"⏱️ Thời gian huấn luyện KNN: {train\_time:.2f} giây")

## Đánh giá mô hình trên tập kiểm thử

Sau khi huấn luyện, mô hình được kiểm thử trên dữ liệu test. Ta tính độ chính xác (Accuracy) để xem khả năng phân loại của KNN.

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)print(f"✅ Độ chính xác trên tập test: {accuracy \* 100:.2f}%")

## Dự đoán mẫu và đo thời gian phản hồi

Để kiểm chứng khả năng dự đoán, ta lấy ngẫu nhiên một mẫu từ tập test và đưa vào mô hình. Kết quả dự đoán sẽ được chuyển lại về dạng nhãn ban đầu. Đồng thời, ta đo thời gian phản hồi của mô hình.

sample = X\_test[0:1]

start\_pred = time.time()

predicted\_class = knn.predict(sample)

end\_pred = time.time()

predicted\_label = label\_encoder.inverse\_transform(predicted\_class)print(f"📌 Dự đoán lớp: {predicted\_label[0]}")print(f"⏱️ Thời gian đưa ra phán đoán: {(end\_pred - start\_pred) \* 1000:.2f} ms")

## Vẽ Confusion Matrix

Cuối cùng, để trực quan hóa hiệu suất mô hình, ta vẽ **ma trận nhầm lẫn (Confusion Matrix)**. Ma trận này cho thấy số lượng mẫu dự đoán đúng và sai trên từng lớp, giúp đánh giá chi tiết hơn thay vì chỉ dựa vào Accuracy.

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=label\_encoder.classes\_)

disp.plot(xticks\_rotation=45, cmap=plt.cm.Blues)

plt.title("🔍 Confusion Matrix - KNN")

plt.show()

1. **TỔNG HỢP CÁC GIẢI THUẬT**

**1. Giai đoạn nạp dữ liệu và xử lý ban đầu**

Trong giai đoạn đầu tiên, chương trình thực hiện việc nạp dữ liệu đầu vào từ thư mục chứa ảnh và file nhãn. Để làm được điều này, ta định nghĩa một hàm có tên là load\_images\_labels, hàm này nhận tham số là đường dẫn đến thư mục chứa ảnh, đường dẫn đến file nhãn cùng với kích thước ảnh mong muốn. Bên trong hàm, chương trình lần lượt duyệt qua từng tệp ảnh, mở ảnh bằng thư viện PIL, chuyển về ảnh xám (mức xám đơn kênh), sau đó thay đổi kích thước về chuẩn 32x32 pixel để đảm bảo sự thống nhất. Mỗi ảnh sau đó được chuẩn hóa bằng cách chia cho 255 để đưa giá trị điểm ảnh về khoảng từ 0 đến 1, giúp quá trình huấn luyện mô hình dễ dàng hơn. Danh sách ảnh được chuyển thành mảng numpy để thuận tiện cho các thao tác tiếp theo. Cùng lúc đó, mảng nhãn được đọc từ file .npy bằng numpy.load. Kết quả trả về của hàm là cặp dữ liệu gồm mảng ảnh và mảng nhãn tương ứng.

Sau khi dữ liệu được nạp, nhãn thô sẽ được xử lý bằng cách mã hóa với LabelEncoder để chuyển đổi từ dạng chuỗi sang dạng số. Tiếp theo, ta sử dụng to\_categorical để chuyển nhãn thành vector one-hot, điều này rất quan trọng khi làm việc với các mô hình học sâu như CNN hoặc RNN. Kết quả của giai đoạn này là một tập dữ liệu ảnh đã chuẩn hóa cùng với nhãn ở dạng số và one-hot vector, sẵn sàng để đưa vào huấn luyện.

X, y\_raw = load\_images\_labels(image\_dir, label\_path)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_raw)

y\_categorical = to\_categorical(y\_encoded)

**2. Huấn luyện mô hình CNN**

Bước tiếp theo là xây dựng và huấn luyện mạng nơ-ron tích chập (Convolutional Neural Network – CNN). Dữ liệu ảnh được reshape thành dạng bốn chiều có kích thước (số lượng ảnh, 32, 32, 1) để phù hợp với kiến trúc CNN. Sau đó, dữ liệu được chia thành hai tập con, trong đó 80% dùng để huấn luyện và 20% còn lại dành cho kiểm thử.

Kiến trúc CNN được thiết kế bao gồm hai lớp tích chập (convolution) kết hợp với hai lớp pooling nhằm rút trích đặc trưng từ ảnh. Sau đó, dữ liệu đi qua lớp Flatten để chuyển từ ma trận thành vector một chiều, tiếp đến là một lớp Dense 64 nút với hàm kích hoạt ReLU để học các đặc trưng trừu tượng hơn. Lớp cuối cùng là Dense với số nút bằng số lớp nhãn, sử dụng hàm softmax để dự đoán xác suất từng lớp. Mạng CNN được biên dịch bằng bộ tối ưu Adam với hàm mất mát categorical crossentropy. Cuối cùng, mô hình được huấn luyện với 10 epochs và batch size là 64, đồng thời dành ra 10% dữ liệu huấn luyện để làm tập validation. Sau khi huấn luyện xong, mô hình được đánh giá trên tập kiểm thử để tính toán độ chính xác.

cnn = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 1)),

layers.MaxPooling2D(2, 2),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D(2, 2),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(y\_categorical.shape[1], activation='softmax')

])

cnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

cnn.fit(X\_train\_cnn, y\_train\_cnn, epochs=10, batch\_size=64, validation\_split=0.1, verbose=0)

cnn\_loss, cnn\_acc = cnn.evaluate(X\_test\_cnn, y\_test\_cnn, verbose=0)

**3. Huấn luyện mô hình RNN với LSTM**

Sau CNN, chương trình tiếp tục triển khai mô hình RNN sử dụng lớp LSTM. Dữ liệu ảnh ở đây được reshape lại thành dạng ba chiều (số lượng ảnh, 32, 32), coi như mỗi ảnh là một chuỗi gồm 32 bước, mỗi bước có 32 đặc trưng. Mô hình LSTM được xây dựng với một lớp LSTM 64 đơn vị, sau đó kết nối đến lớp Dense 64 nút và một lớp Dense đầu ra sử dụng softmax. Quá trình biên dịch và huấn luyện tương tự CNN, với Adam optimizer và categorical crossentropy loss. Sau khi huấn luyện, mô hình được đánh giá để xác định độ chính xác và thời gian huấn luyện.

rnn = models.Sequential([

layers.LSTM(64, input\_shape=(32, 32)),

layers.Dense(64, activation='relu'),

layers.Dense(y\_categorical.shape[1], activation='softmax')

])

rnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

rnn.fit(X\_train\_rnn, y\_train\_rnn, epochs=10, batch\_size=64, validation\_split=0.1, verbose=0)

rnn\_loss, rnn\_acc = rnn.evaluate(X\_test\_rnn, y\_test\_rnn, verbose=0)

**4. Huấn luyện mô hình KNN**

Song song với các mô hình học sâu, chương trình cũng thử nghiệm thuật toán học máy truyền thống K-Nearest Neighbors (KNN). Đối với KNN, dữ liệu ảnh được chuyển đổi thành vector một chiều bằng cách reshape thành mảng hai chiều (số lượng ảnh, số pixel). Sau đó, dữ liệu được chia thành tập huấn luyện và tập kiểm thử tương tự như trước. Mô hình KNN được khởi tạo với k = 3, nghĩa là mỗi điểm dữ liệu mới sẽ được phân loại dựa trên 3 láng giềng gần nhất. Sau khi huấn luyện, mô hình được dùng để dự đoán trên tập kiểm thử và tính toán độ chính xác.

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train\_knn, y\_train\_knn)

y\_pred\_knn = knn.predict(X\_test\_knn)

knn\_acc = accuracy\_score(y\_test\_knn, y\_pred\_knn)

**5. So sánh kết quả và trực quan hóa**

Cuối cùng, chương trình thực hiện so sánh kết quả của ba mô hình CNN, RNN và KNN. Các chỉ số được ghi nhận bao gồm độ chính xác trên tập kiểm thử và thời gian huấn luyện. Sau đó, một biểu đồ cột được vẽ bằng thư viện matplotlib để trực quan hóa kết quả, giúp người đọc dễ dàng so sánh ưu nhược điểm của từng mô hình. Qua đó, có thể đánh giá được sự khác biệt giữa các giải thuật học sâu và học máy truyền thống khi áp dụng vào cùng một tập dữ liệu ảnh.

plt.bar(['CNN', 'RNN', 'KNN'], [cnn\_acc\*100, rnn\_acc\*100, knn\_acc\*100], color=['skyblue', 'orange', 'green'])

plt.ylabel("Accuracy (%)")

plt.title("So sánh mô hình")

plt.ylim(0, 100)

plt.grid(axis='y')

plt.show()

1. **HƯỚNG PHÁT TRIỂN**

Hệ thống hiện tại mới dừng lại ở việc thử nghiệm trên tập dữ liệu CIC-IDS2018 với ba giải thuật học máy và học sâu phổ biến là KNN, RNN và CNN. Kết quả so sánh cho thấy CNN có độ chính xác cao nhất khi xử lý dữ liệu dưới dạng ảnh, tuy nhiên hệ thống vẫn còn nhiều điểm cần phát triển thêm để trở nên hoàn thiện và có khả năng ứng dụng thực tế.

Trong tương lai, một trong những hướng phát triển quan trọng là mở rộng quy mô dữ liệu huấn luyện. Thay vì chỉ sử dụng một phần nhỏ của bộ dữ liệu, hệ thống có thể được áp dụng trên toàn bộ tập CIC-IDS2018 hoặc thậm chí kết hợp thêm các bộ dữ liệu khác như NSL-KDD, UNSW-NB15 để tăng tính đa dạng của mẫu tấn công. Việc mở rộng dữ liệu sẽ giúp mô hình học được nhiều đặc trưng hơn, từ đó nâng cao độ chính xác và khả năng tổng quát hóa khi gặp dữ liệu mới.

Bên cạnh đó, hệ thống có thể được cải thiện về mặt mô hình. Các mạng CNN hiện tại chỉ sử dụng hai lớp tích chập đơn giản, trong khi đó nhiều nghiên cứu đã chứng minh rằng các kiến trúc sâu hơn hoặc các biến thể nâng cao như ResNet, Inception hay LSTM-CNN kết hợp sẽ cho hiệu quả vượt trội trong phát hiện xâm nhập. Việc thử nghiệm các mô hình lai ghép giữa CNN và RNN cũng là một hướng đi tiềm năng nhằm khai thác cả đặc trưng không gian lẫn đặc trưng chuỗi thời gian của dữ liệu mạng.

Ngoài ra, hệ thống có thể được triển khai trên môi trường thực tế thay vì chỉ chạy thử nghiệm trên Google Colab. Điều này bao gồm việc xây dựng pipeline tiền xử lý tự động, triển khai mô hình trên server hoặc container, đồng thời tích hợp với hệ thống mạng để thu thập dữ liệu trực tiếp từ gói tin. Khi đó, hệ thống IDS sẽ không chỉ dừng ở mức đánh giá mô hình mà còn có thể hoạt động trong môi trường mạng doanh nghiệp với khả năng phát hiện và cảnh báo tấn công theo thời gian thực.

Cuối cùng, một hướng phát triển khác là tối ưu hiệu năng hệ thống. Việc huấn luyện trên dữ liệu lớn và mô hình phức tạp thường tiêu tốn nhiều tài nguyên phần cứng. Do đó, các kỹ thuật tối ưu như giảm chiều dữ liệu, chọn đặc trưng quan trọng, áp dụng mô hình nhẹ hơn như MobileNet hoặc sử dụng tăng tốc phần cứng GPU/TPU sẽ giúp rút ngắn thời gian xử lý mà vẫn đảm bảo hiệu quả.

Như vậy, từ phiên bản thử nghiệm hiện tại, hệ thống IDS hoàn toàn có thể được phát triển theo nhiều hướng: mở rộng dữ liệu, cải thiện kiến trúc mô hình, triển khai thực tế và tối ưu hiệu năng. Các hướng đi này sẽ giúp hệ thống ngày càng tiến gần hơn đến một giải pháp an ninh mạng toàn diện, có khả năng phát hiện và ngăn chặn tấn công hiệu quả trong môi trường phức tạp của thực tế.