3.6 Analiza traficului de date

Analiza traficului de date reprezintă un element esențial în asigurarea securității rețelelor de calculatoare și în identificarea rapidă a activităților anormale. Această analiză se bazează pe monitorizarea continuă a fluxului de informații, prin intermediul unor instrumente și tehnologii avansate, care permit atât detectarea incidentelor de securitate, cât și reacția promptă în caz de atac.

Pentru a implementa o astfel de analiză, am folosit un model de învățare automată bazat pe rețele neuronale convoluționale (CNN) și arhitectura Inception, adaptată pentru date 1D. Iată cum am implementat acest model și interfața de analiză a traficului în codul nostru:

### Importarea Bibliotecilor

```python

import os

import tkinter as tk

from tkinter import scrolledtext

from datetime import datetime

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import load\_model, Model

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dropout, Dense, Flatten, Input, concatenate

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import MinMaxScaler, LabelEncoder

```

### Definirea Căilor și Coloanelor

```python

MODEL\_DIR = os.path.join(".", "saved\_model")

MODEL\_FILE\_PATH = os.path.join(MODEL\_DIR, "model\_traffic\_FINAL.h5")

LOG\_DIR = os.path.join(".", "analysis\_logs")

columns = ['duration', 'protocol\_type', 'service', 'flag', 'src\_bytes', 'dst\_bytes', 'land', 'wrong\_fragment', 'urgent',

'hot', 'num\_failed\_logins', 'logged\_in', 'num\_compromised', 'root\_shell', 'su\_attempted', 'num\_root',

'num\_file\_creations', 'num\_shells', 'num\_access\_files', 'num\_outbound\_cmds', 'is\_host\_login', 'is\_guest\_login',

'count', 'srv\_count', 'serror\_rate', 'srv\_serror\_rate', 'rerror\_rate', 'srv\_rerror\_rate', 'same\_srv\_rate',

'diff\_srv\_rate', 'srv\_diff\_host\_rate', 'dst\_host\_count', 'dst\_host\_srv\_count', 'dst\_host\_same\_srv\_rate',

'dst\_host\_diff\_srv\_rate', 'dst\_host\_same\_src\_port\_rate', 'dst\_host\_srv\_diff\_host\_rate', 'dst\_host\_serror\_rate',

'dst\_host\_srv\_serror\_rate', 'dst\_host\_rerror\_rate', 'dst\_host\_srv\_rerror\_rate', 'attack', 'level']

```

### Funcția de Grupare a Etichetelor de Atac

```python

def group\_attack\_label(attack):

attack = attack.lower()

if attack in ['satan', 'portsweep', 'nmap', 'jpsweep']:

return 'Probe'

elif attack in ['spy', 'phf', 'multihop', 'imap', 'guess\_passwd', 'ftp\_write', 'warezmaster', 'warezclient']:

return 'R2L'

elif attack in ['rootkit', 'perl', 'loadmodule', 'buffer\_overflow']:

return 'U2R'

elif attack in ['teardrop', 'smurf', 'pod', 'neptune', 'land', 'back']:

return 'DoS'

elif attack == 'normal':

return 'NORMAL'

else:

return 'NORMAL'

```

### Funcțiile de Încărcare și Preprocesare a Datelor

```python

def load\_and\_preprocess\_train\_data(file\_path):

df = pd.read\_csv(file\_path, delimiter=',', header=None)

df.columns = columns

X = df.iloc[:, :-1]

y\_raw = df['attack'].apply(group\_attack\_label)

label\_encoders = {}

for column in X.select\_dtypes(include=['object']).columns:

le = LabelEncoder()

X[column] = le.fit\_transform(X[column])

label\_encoders[column] = le

label\_encoder\_y = LabelEncoder()

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

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

X\_scaled = X\_scaled.reshape(X\_scaled.shape[0], X\_scaled.shape[1], 1)

return X\_scaled, y, label\_encoders, label\_encoder\_y, scaler

def load\_and\_preprocess\_test\_data(file\_path, label\_encoders, label\_encoder\_y, scaler):

df = pd.read\_csv(file\_path, delimiter=',', header=None)

df.columns = columns

raw\_df = df.copy()

y\_raw = df['attack'].apply(group\_attack\_label)

mask\_attack = y\_raw.isin(label\_encoder\_y.classes\_)

if not mask\_attack.all():

unknown\_labels = y\_raw[~mask\_attack].unique()

print("Următoarele etichete de atac nu sunt prezente în setul de antrenare și vor fi eliminate:", unknown\_labels)

df = df[mask\_attack]

raw\_df = raw\_df[mask\_attack]

y\_raw = y\_raw[mask\_attack]

X = df.iloc[:, :-1]

for column in X.select\_dtypes(include=['object']).columns:

le = label\_encoders[column]

mask\_col = X[column].isin(le.classes\_)

if not mask\_col.all():

unknown\_values = X.loc[~mask\_col, column].unique()

print(f"În coloana '{column}', următoarele valori necunoscute vor fi eliminate:", unknown\_values)

X = X[mask\_col]

y\_raw = y\_raw[mask\_col]

raw\_df = raw\_df[mask\_col]

X[column] = le.transform(X[column])

y = label\_encoder\_y.transform(y\_raw)

X\_scaled = scaler.transform(X)

X\_scaled = X\_scaled.reshape(X\_scaled.shape[0], X\_scaled.shape[1], 1)

return X\_scaled, y, raw\_df

```

### Definirea Blocului Inception pentru Date 1D

```python

def inception\_block(x, filters):

branch1 = Conv1D(filters, kernel\_size=1, padding='same', activation='relu')(x)

branch3 = Conv1D(filters, kernel\_size=1, padding='same', activation='relu')(x)

branch3 = Conv1D(filters, kernel\_size=3, padding='same', activation='relu')(branch3)

branch5 = Conv1D(filters, kernel\_size=1, padding='same', activation='relu')(x)

branch5 = Conv1D(filters, kernel\_size=5, padding='same', activation='relu')(branch5)

branch\_pool = MaxPooling1D(pool\_size=3, strides=1, padding='same')(x)

branch\_pool = Conv1D(filters, kernel\_size=1, padding='same', activation='relu')(branch\_pool)

x = concatenate([branch1, branch3, branch5, branch\_pool], axis=-1)

return x

```

### Crearea Modelului VGG16IncepNet pentru Date 1D

```python

def create\_vgg16incepnet\_model(input\_shape, num\_classes):

inputs = Input(shape=input\_shape)

x = Conv1D(64, kernel\_size=3, activation='relu', padding='same')(inputs)

x = Conv1D(64, kernel\_size=3, activation='relu', padding='same')(x)

x = MaxPooling1D(pool\_size=2)(x)

x = Conv1D(128, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(128, kernel\_size=3, activation='relu', padding='same')(x)

x = MaxPooling1D(pool\_size=2)(x)

x = inception\_block(x, filters=64)

x = Conv1D(256, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(256, kernel\_size=3, activation='relu', padding='same')(x)

x = MaxPooling1D(pool\_size=2)(x)

x = Flatten()(x)

x = Dense(512, activation='relu')(x)

x = Dropout(0.5)(x)

outputs = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.summary()

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

return model

```

### Încărcarea și Pregătirea Datelor

```python

train\_file = "../Data/nsl-kdd/KDDTrain+.txt"

test\_file = "../Data/nsl-kdd/KDDTest+.txt"

X\_train, y\_train, label\_encoders, label\_encoder\_y, scaler = load\_and\_preprocess\_train\_data(train\_file)

X\_test, y\_test, raw\_test\_df = load\_and\_preprocess\_test\_data(test\_file, label\_encoders, label\_encoder\_y, scaler)

num\_classes = len(np.unique(y\_train))

if os.path.exists(MODEL\_FILE\_PATH):

model = load\_model(MODEL\_FILE\_PATH, compile=False)

model.summary()

print("Modelul a fost încărcat din fișierul salvat.")

else:

os.makedirs(MODEL\_DIR, exist\_ok=True)

model = create\_vgg16incepnet\_model(X\_train.shape[1:], num\_classes)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

model.fit(X\_train, y\_train, epochs=100, batch\_size=64,

validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])

model.save(MODEL\_FILE\_PATH)

model.summary()

print("Modelul a fost antrenat și salvat în format .h5.")

```

### Funcția de Preprocesare și Predicție

```python

def preprocess\_and\_predict(data, threshold=50):

prediction\_probs = model.predict(data)[0]

predicted\_index = np.argmax(prediction\_probs)

confidence = prediction\_probs[predicted\_index] \* 100

if confidence < threshold:

return "Trafic Legitim", confidence

predicted\_category = label\_encoder\_y.inverse\_transform([predicted\_index])[0]

return f"Atac: {predicted\_category}", confidence

```

### Funcția de Salvare a Atacurilor Detectate

```python

def save\_attack\_to\_file(log\_file, attack\_type, confidence, packet\_data):

current\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

with open(log\_file, 'a', encoding="utf-8") as file:

file.write(f"{current\_time}\t{attack\_type}\t\t{confidence:.2f}%\n")

file.write(f"Packet Data: {packet\_data}\n")

file.write("-"\*60 + "\n")

```

### Interfața Tkinter pentru Analiza Traficului

```python

class LiveTrafficAnalyzer:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Live Internet Traffic Analyzer")

self.label = tk.Label(root, text="Internet Traffic Analysis", font=("Arial", 16))

self.label.pack(pady=10)

self.start\_button = tk.Button(root, text="Start Analysis", command=self.start\_analysis)

self.start\_button.pack(pady=5)

self.stop\_button = tk.Button(root, text="Stop Analysis", command=self.stop\_analysis, state="disabled")

self.stop\_button.pack(pady=5)

self.history\_text = scrolledtext.ScrolledText(root, width=80, height=10, font=("Arial", 10))

self.history\_text.pack(pady=10)

self.history\_text.insert(tk.END, "Timestamp\t\tPredicted Attack Type\t\tConfidence (%)\n")

self.history\_text.insert(tk.END, "-"\*60 + "\n")

self.packet\_text = scrolledtext.ScrolledText(root, width=80, height=10, font=("Arial", 10))

self.packet\_text.pack(pady=10)

self.packet\_text.insert(tk.END, "Detalii pachet:\n")

self.packet\_text.insert(tk.END, "-"\*60 + "\n")

self.running = False

self.log\_file = None

def start\_analysis(self):

self.running = True

self.start\_button.config(state="disabled")

self.stop\_button.config(state="normal")

os.makedirs(LOG\_DIR, exist\_ok=True)

self.log\_file = os.path.join(LOG\_DIR, f"analysis\_log\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}.txt")

self.run\_analysis()

def stop\_analysis(self):

self.running = False

self.start\_button.config(state="normal")

self.stop\_button.config(state="disabled")

def run\_analysis(self):

if self.running:

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

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

attack\_type, confidence = preprocess\_and\_predict(sample\_data)

current\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

self.history\_text.insert(tk.END, f"{current\_time}\t{attack\_type}\t\t{confidence:.2f}%\n")

self.history\_text.see(tk.END)

packet\_details = raw\_test\_df.iloc[idx]

details\_str = "Pachetul de date: " + " | ".join([f"{col}: {packet\_details[col]}" for col in raw\_test\_df.columns])

self.packet\_text.delete("1.0", tk.END)

self.packet\_text.insert(tk.END, details\_str + "\n" + "-"\*60 + "\n")

save\_attack\_to\_file(self.log\_file, attack\_type, confidence, details\_str)

self.root.after(3000, self.run\_analysis)

root = tk.Tk()

app = LiveTrafficAnalyzer(root)

root.mainloop()

```

### Concluzie

Implementarea acestei soluții de analiză a traficului de date folosește un model de învățare automată pentru a detecta și clasifica atacurile de rețea. Interfața Tkinter permite vizualizarea în timp real a rezultatelor analizei, iar datele sunt preprocesate și scalate pentru a fi compatibile cu modelul. Modelul este antrenat pe un set de date etichetat și poate fi salvat și reîncărcat pentru utilizare ulterioară.