الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

Université Badji Mokhtar - Annaba

Badji Mokhtar – Annaba University

جامعة باجي مختار – عنابــة

Faculté : Technologie

Département : Informatique

Domaine : Mathématique-Informatique

Filière : Informatique

Spécialité : Systèmes informatiques

Mémoire

Présenté en vue de l'obtention du Diplôme de Licence

Thème

Detection d'intrusion par deep learning

Présenté par : Bouaicha Salah

Bouacha Najmedine Mohammed

Encadrant : Yahiouche Salima

Grade : MAA

Année Universitaire : 2022/2023

Remerciements

We would like to take a moment to extend our sincere thanks and gratitude to our supervisor, Mrs. Yahiouche Salima, for her guidance and support throughout every phase of our project.

Furthermore, we would also like to express our heartfelt to our friends and family for their love, encouragement, and support throughout this journey

Dédicaces

This project is dedicated to all people who stand with us through our journey

.

Table des matières

Table des figures

Introduction

Projet’s context:

Technology has seen a lot of improvements these days as the network has been playing essential part in our modern Life which being the most common way to gain knowledge and information The network has a risky side that puts our private information in danger Intrusion true purpose is to gain this private information or had access to our software To prevent this, we need to use an Intrusion Detection System (IDS) which is an important part of a cyber Security techniques designed to detect unauthorized access or malicious activities within a computer network or system. The primary function of an Intrusion Detection System IDS is to Monitor network traffic and system activity for any types of attacks by giving us alerts.

Problem

The problem that the IDS had is processing a high volume of data to monitor all Network traffic and system activity which can generate a large amount of data. This can be difficult for IDSs that are not designed to handle a high volume of data which can lead to failed detection.

Motivation

Computer networks and systems face a growing range of security risks and malicious acts in the modern era of increasing interconnection. It is crucial to safeguard sensitive data and maintain the reliability of these systems. Therefore, our objective is to use deep learning's capabilities to create an AI model that can accurately identify a variety of anomalous behavior and probable security breaches.

Objective

We are building an Ai model by using Deep Neural Network (DNN deep learning) to enhance the security of the computer network by creating an intrusion detection system that can handle multiple packet and alert on potential security breaches or malicious activities.

Chapters organization

- **Chapter 1:** definitions of IDS, deep learning and the methods used to build our model.

- **Chapter 2*:*** understanding how our IDS works and how to process the data and network architecture including model training and evaluation.

- **Chapter 3*:*** in this last chapter we discuss different tools that helped developing our project along with the project implementation.

Finally, we will end with a general conclusion and discuss some perspectives.

Chapter 1: On Intrusion Detection Systems and Deep Learning

In this chapter, we provide a general overview of IDSs (Intrusion Detection Systems) and some basic information. We will also cover some fundamental concepts and definitions of deep learning and machine learning that are crucial for understanding our project.

1. Intrusion Detection Systems (IDS):

1.1 Definition :

An IDS stands for Intrusion Detection System It is a security technology that monitors network traffic and system activity for signs of unauthorized access, malicious activities, or policy violations. In simple words an IDS is a tool that helps detect if someone is trying to break into a computer system or network and alerts the system administrator or security team to take action.

1.2 How IDS works:

An intrusion detection system is a monitor-only application designed to identify and report on anomalies before hackers can damage the network infrastructure. IDS is either installed on the network or on a client system (host-based IDS).

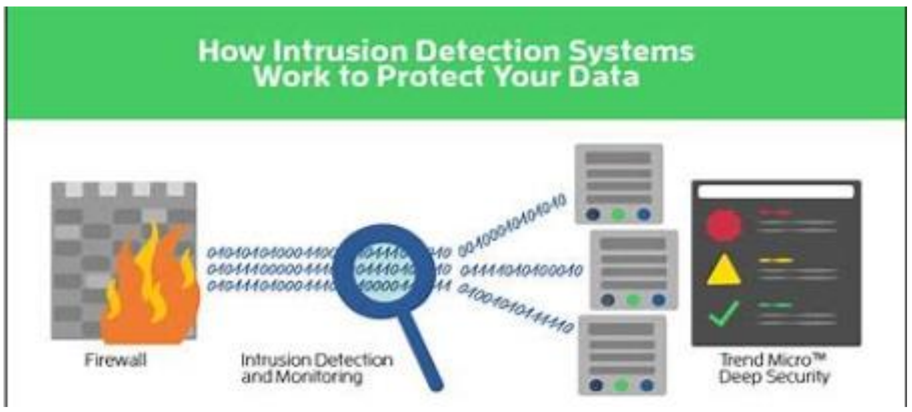

How Intrusion Detection Systems Work to Protect Your Data
Firewall
Intrusion Detection and Monitoring
Trend Micro™ Deep Security

Figure 1 : IDS to protect systems [1]

1.3 Types of IDS:

There are two types of IDS:

• Network Intrusion Detection System (NIDS): NIDS is set up at a planned point within the network to examine traffic from all devices on the network. It performs an observation of passing traffic on the entire subnet and matches the traffic that is passed on the subnets to the collection of known attacks. Once an attack is identified or an abnormal behavior is observed, the alert can be sent to the administrator. An example of NIDS is installing it on the subnet where firewalls are located to see if someone is trying to crack the firewall

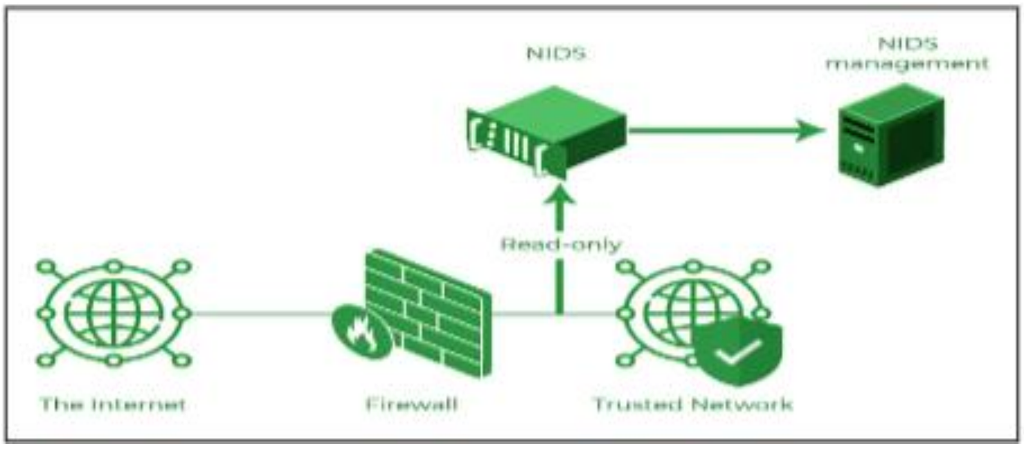


Figure 2 : NIDS system architecture [2]

• Host Intrusion Detection System (HIDS): HIDS runs on independent hosts or devices on the network. An HIDS monitors the incoming and outgoing packets from the device only and will alert the administrator if suspicious or malicious activity is detected. It takes a snapshot of existing system files and compares it with the previous snapshot. If the analytical system files were edited or deleted, an alert is sent to the administrator to investigate. An example of HIDS usage can be seen on mission-critical machines, which are not expected to change their layout.

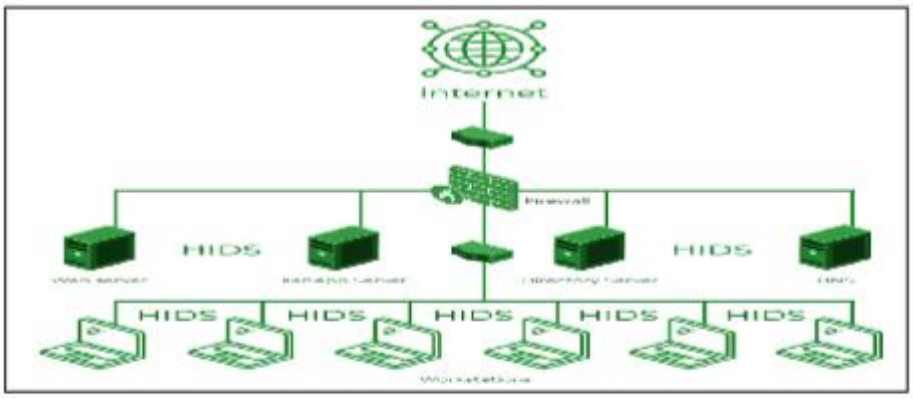


Figure 3 : HID system architecture . [2]

.

2. Artificial intelligence subsets:

Machine learning and deep learning are both part of artificial intelligence. These approaches both aim to give computers the ability to make intelligent decisions. However, deep learning is a subcategory of machine learning as it relies on unsupervised learning. In both cases, the intelligence is limited to specific uses.

This is known as weak artificial intelligence, as opposed to strong artificial intelligence, which would be capable of making intelligent decisions similar to humans in many domains and circumstances. Both technologies require large amounts of data as their learning foundation. The similarities end there.

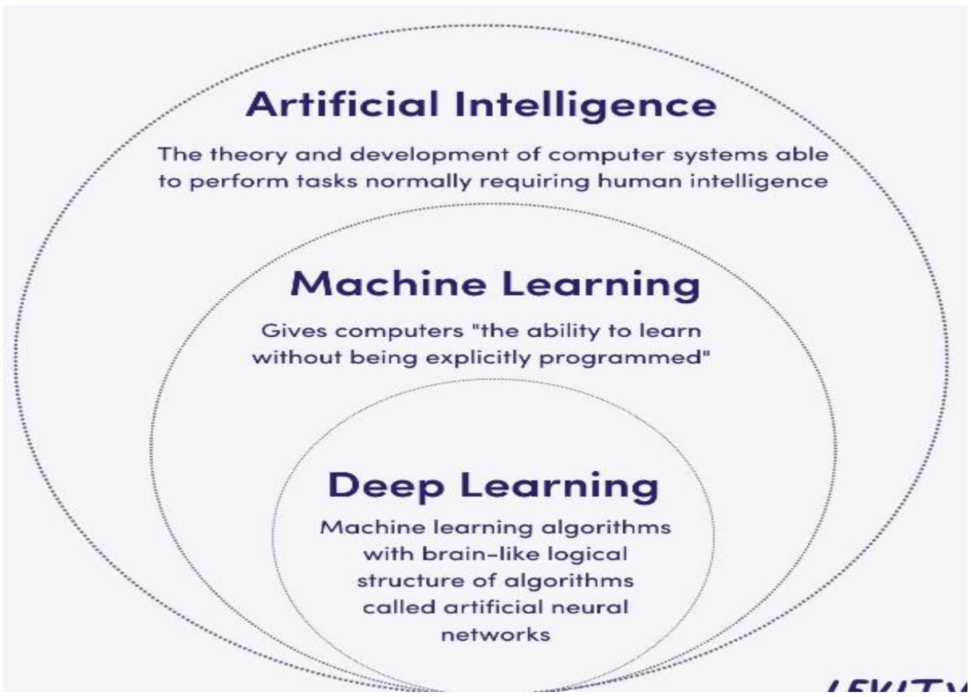

Artificial Intelligence
The theory and development of computer systems able to perform tasks normally requiring human intelligence
Machine Learning
Gives computers "the ability to learn without being explicitly programmed"
Deep Learning
Machine learning algorithms with brain-like logical structure of algorithms called artificial neural networks

Figure 4 : Machine learning and deep learning. [3]

2.1 Deep learning Definition:

Deep Learning is a machine learning technique that constructs artificial neural networks to mimic the structure and function of the human brain. In practice, deep learning, also known as deep structured learning or hierarchical learning, uses a large number of hidden layers -typically more than 6but often much higher - of nonlinear processing to extract features from data and transform the data into different levels of abstraction

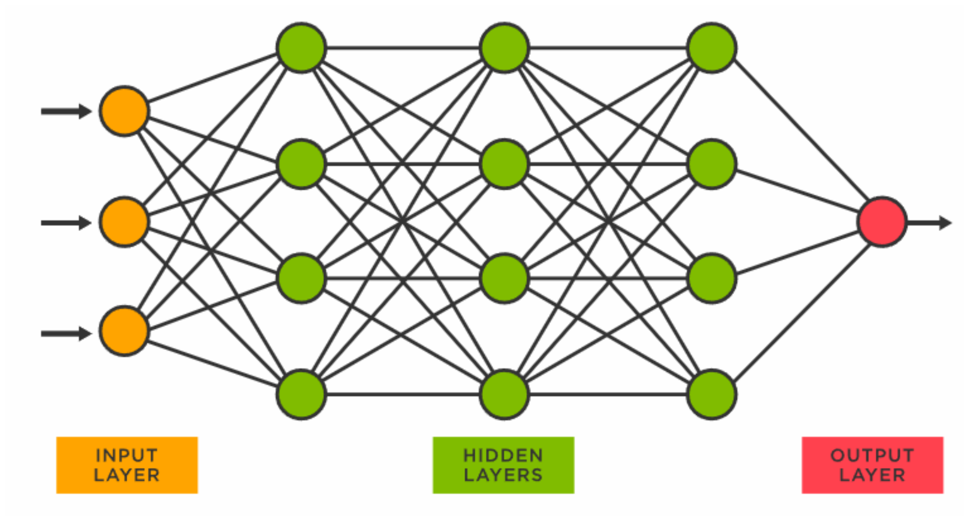


Figure 5: deep learning architecture. [4]

2.2 Neural Networks:

A neural network is structured like the human brain and consists of artificial neurons, also known as nodes. The nodes are stacked next to each other in three layers:

1 The input layer

2 The hidden layer(s)

3 The output layer

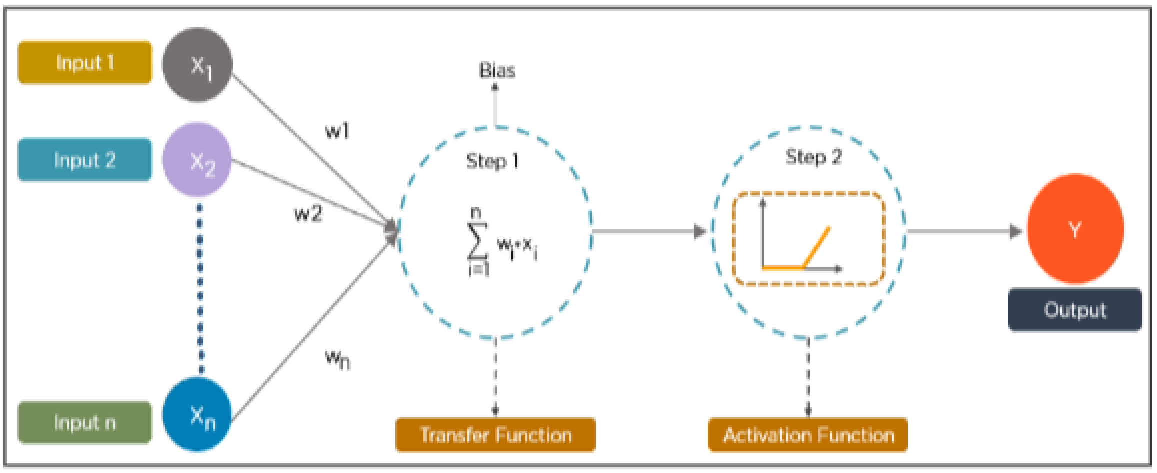


Figure 6 : Neural Network architecture. [5]

Data provides each node with information in the form of inputs. The node multiplies the inputs with random weights, calculates them, and adds a bias.

Finally, nonlinear functions, also known as activation functions, are applied to determine which neuron to fire.

2.2 The hidden layer :

Hidden layers are the intermediate layers between the input and output layers. They play a vital role in capturing complex patterns and representations from the input data

- Dense Layers (Fully Connected Layers):

Dense layer are the most common type of layer in a DNN. Every neuron is connected to every neuron in the previous layer. Each neuron applies an activation function as the hyperbolic tangent function that adds the result in tanh(z) value .

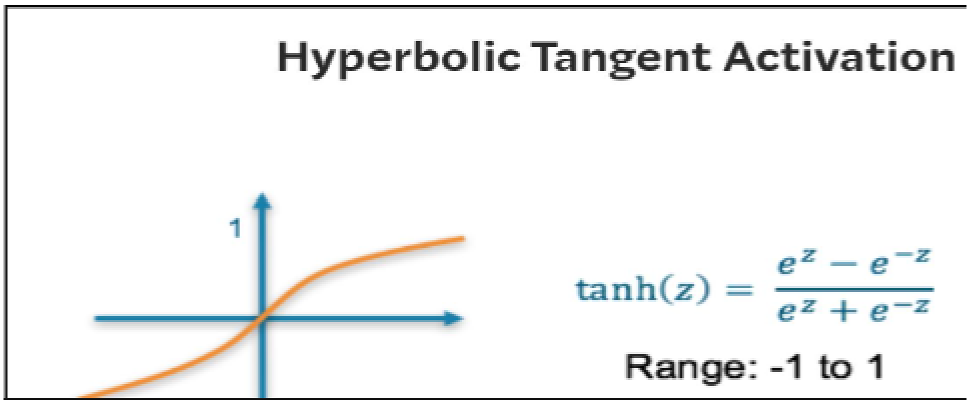


Figure 10 : Dense Layers [7]

2.3 Types of Algorithms of Deep Learning:

- **Convolutional Neural Networks (CNNs)**

CNNs, also known as Conv Nets, consist of multiple layers and are mainly used for image processing and object detection. Yann Le Cun developed the first CNN in 1988 when it was called Le Net. It was used for recognizing characters like ZIP codes and digits

CNNs are widely used to identify satellite images, process medical images, forecast time series, and detect anomalies.

- **Generative Adversarial Networks (GANs)**

GANs are generative deep learning algorithms that create new data instances that resemble the training data. GAN has two components: a generator, which learns to generate fake data, and a discriminator, which learns from that false information.

- **Long Short Term Memory Networks (LSTMs)**

LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies Recalling past information for long periods is the default behavior.

2.Conclusion:

In this chapter, we learned about intrusion detection systems (IDS) and how they may assist individuals and companies identify and prevent illegal network access. IDS is essential for guaranteeing network security and safeguarding sensitive data.

As a foundation for our application in the next chapters, we also examined several key ideas and the idea of deep learning. Deep Learning is a cutting-edge artificial intelligence technique that enables computers to learn and make intelligent judgments based on massive volumes of data.

By employing Deep Learning capabilities, we may improve the efficacity of intrusion detection systems, allowing for more accurate and efficient identification of possible threats and assaults.

Chapter 2: System Conception

In this chapter, we will focus on the overall architecture and design of deep neural networks (DNNs) and the data collection process. We will also examine the process of building our implemented Python desktop application and discuss its design using the appropriate UML diagrams.

1 DNN Architecture :

In this chapter, we will discuss the components of a DNN (Deep Neural Network) architecture, with a specific focus on hidden layers. Additionally, we will explore the significance of data collection and data cleaning in the context of deep learning.

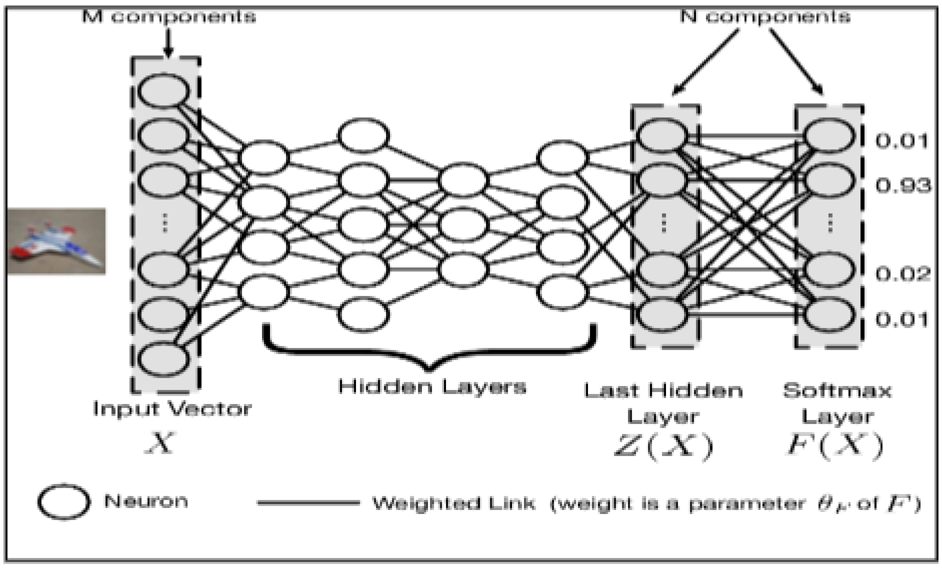


Figure 7 : DNN Architecture. [6]

1. Data collection

The most important step in deep learning is data collection and data cleaning, as the quality and relevance of the collected data directly impacts the performance and accuracy of deep learning models.

Data collection is the process of collecting information or data from different sources. It is an important step in any data analysis or machine learning project.

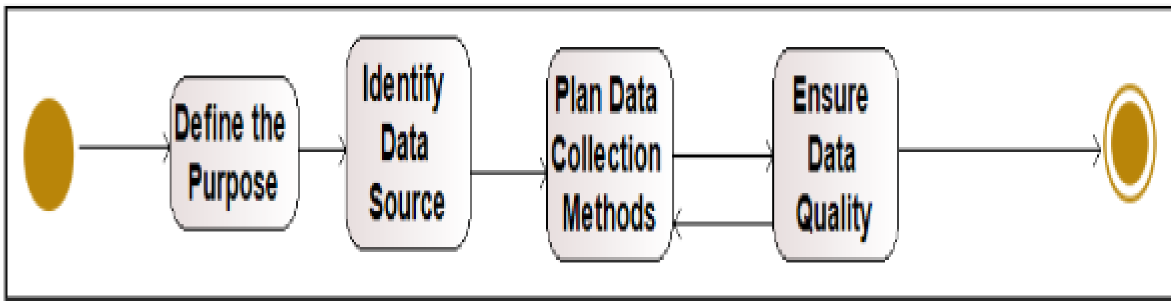


Figure 8 : Data collection

Data cleaning, also known as data cleansing or data preprocessing, refers to the process of identifying and correcting or removing errors, inconsistencies, or inaccuracies in the collected data. It is an essential step before performing any data analysis or training machine learning models.

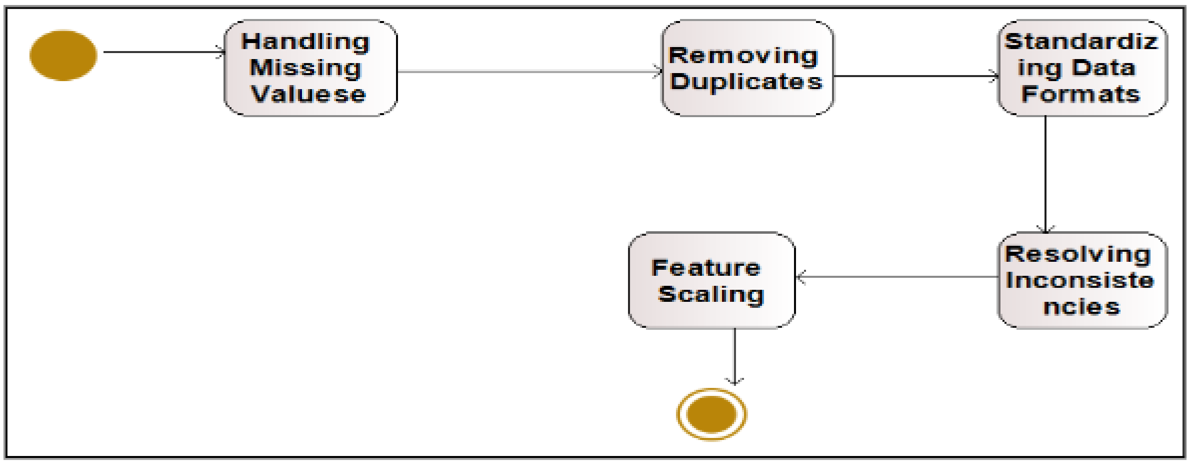


Figure 9 : data preprocessing

2. Presentation of the output layers :

The sigmoid function is a commonly used activation function that maps the input to a range between 0 and 1.

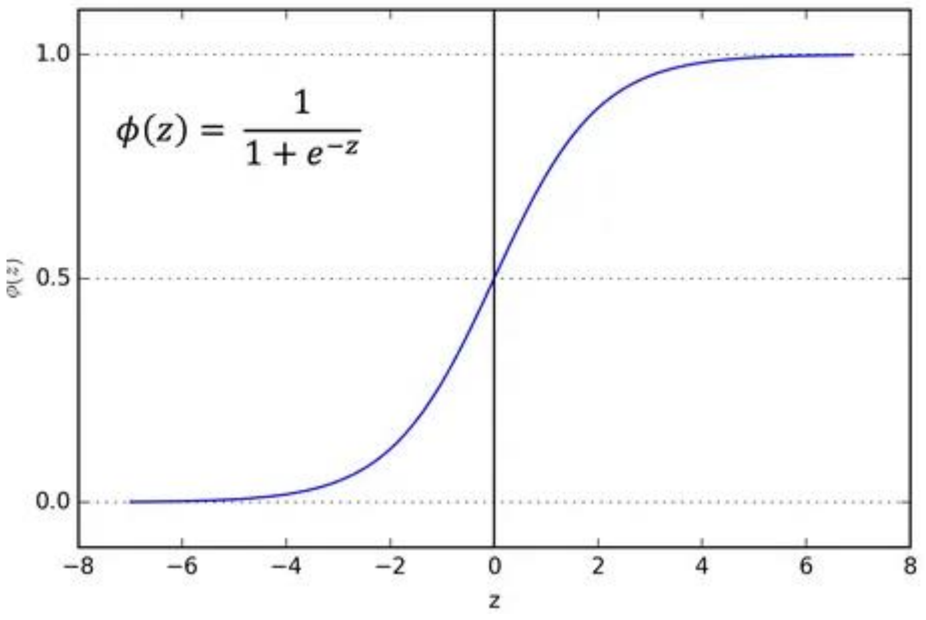


Figure 11 : Sigmoid function [8]

The sigmoid function squeezes the input values into the range of 0 to 1, which can be interpreted as probabilities.

2.5 Loss function :

A loss function is a mathematical function that measures the discrepancy between a machine learning model's anticipated outputs and the actual outputs or labels connected to the input data.

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\\text{size}}} \sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Part II : Application Conception :

In this section, we will discuss the conception of our application using three UML diagrams: use case diagram, activity diagram and the sequence diagram.

Use case diagram: is a diagram that illustrates the interactions between actors and a system in terms of the system's functionalities or use cases. It provides a high-level view of how users interact with a system and the various actions that can be performed.

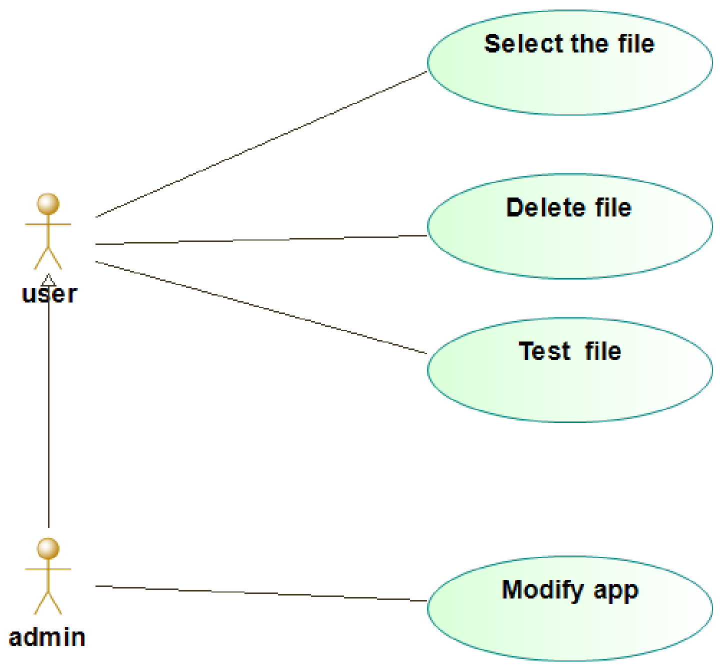


figure 12 : Uml diagram use case

- **In this diagram**, there are two actors: User and Admin. The User can perform actions such as Log-in, Remove User, Load, show and Test Data. The Admin, on the other hand has additional actions available,
- **Activity diagram** :is a graphical representation used to model the behavior and flow

of activities within a system, process, or workflow. It visually illustrates the sequence of actions, decisions, and transitions that occur during the execution of a particular activity.

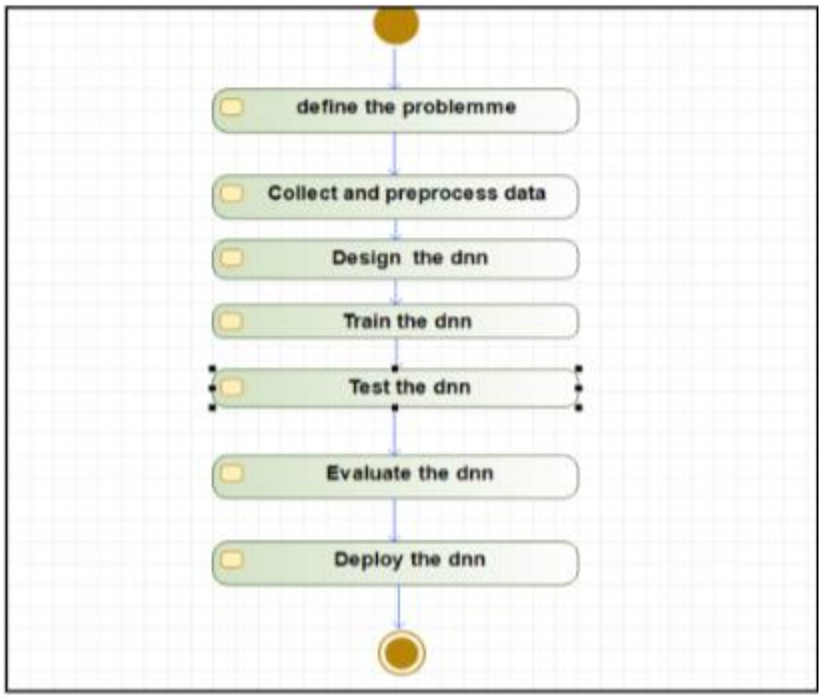


Figure 13: Uml activity diagram

Sequence Diagram: Shows the interactions and messages exchanged between objects in a particular scenario or process flow.

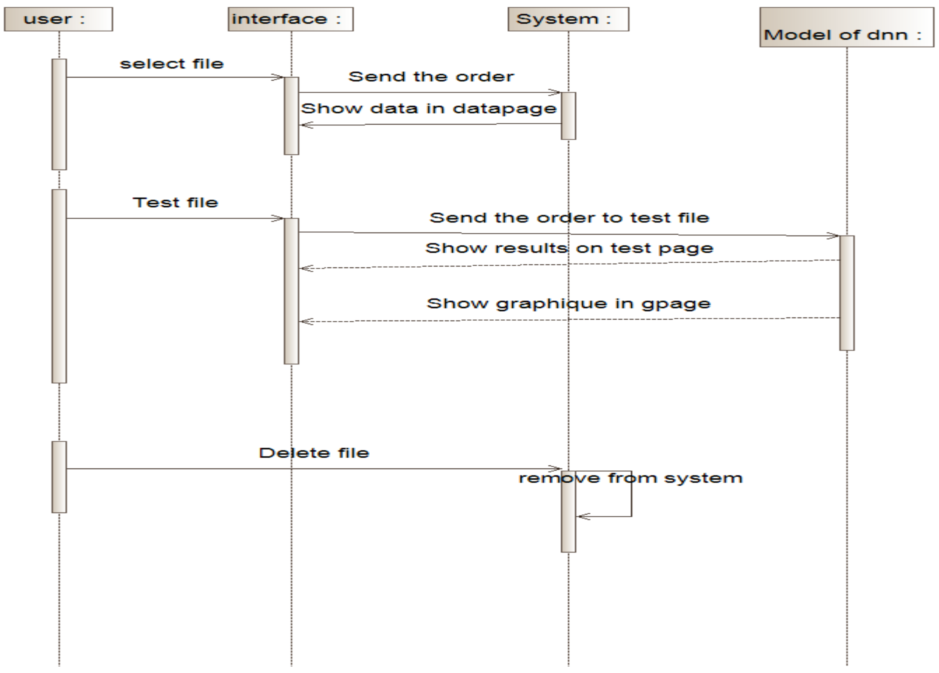


Figure 14: Uml Sequence Diagram

2. Conclusion:

In this chapter we explained the details of the DNN architecture we are using in the machine learning model, we also explained our application through some UML diagrams. In the next chapter, we’ll discuss the implementation details of our model and application.

Chapter 3: System Implementation

1. Introduction :

In this chapter, we discuss the data collection and pre-processing steps that are necessary to build a deep neural network (DNN). We used a dataset created by the University of New Brunswick, implemented various data cleaning and processing techniques using Python libraries such as pandas, numpy, and scikit-learn. We also discussed the network architecture selection using TensorFlow Keras API, and finally, we trained and saved our DNN model for later use.

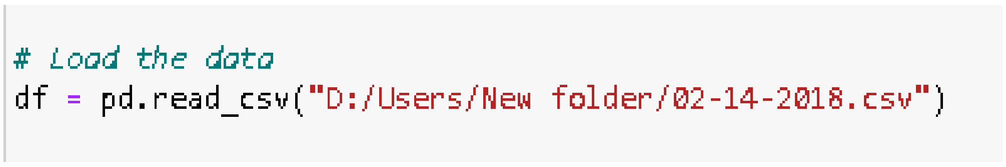
3 Data collection:

We used a dataset created by the University of New Brunswick to analyze various types of attacks, including DoS, FTP, and Hulk [10]. The dataset contains 80 columns, with each column representing an intrusion detection system (IDS) Activity System of the University of New Brunswick. The Label column is a critical component of the dataset, as it determines whether the sent packets are malicious or not.

4 Pre-processing of the data:

The first step is to load the dataset using the *pandas* library, which is an essential tool for working with data in Python. We can use pandas to perform various tasks on the dataset, such as cleaning, preprocessing, and analyzing the data.

Loading the data:



```
# Load the data
df = pd.read_csv("D:/Users/New folder/02-14-2018.csv")
```

The second step is to clean the dataset by replacing all infinite values with NaN values and then removing them from the dataset. NaN values are a standard way to represent missing or invalid data in pandas.To replace infinite values with NaN values, we can use the numpy library.

Removing NaN values:

```python
# Replace every value in columns to NaN
df.replace([np.inf, -np.inf], np.nan, inplace=True)

# Remove every row that has NaN value
df.dropna(inplace=True)
```

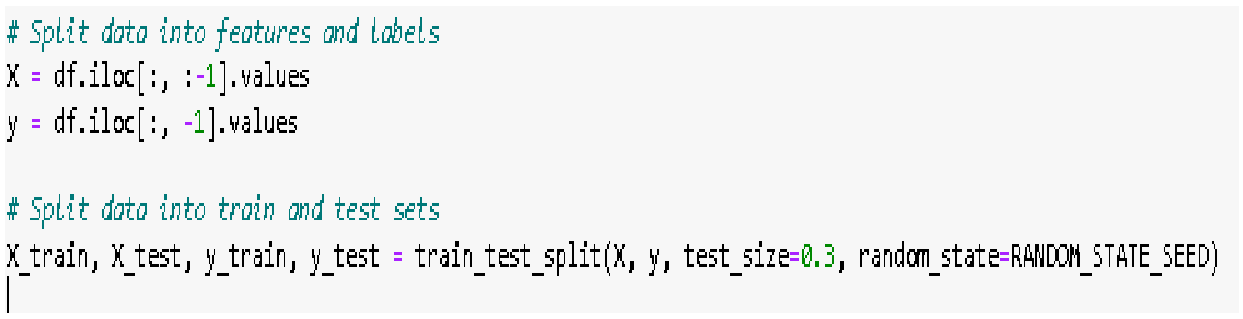
After cleaning the dataset, the *next step* is to remove any columns that are not necessary for our analysis. In this case, we can remove the "Timestamp" column as it's not needed for our analysis.

```python
if 'Timestamp' in df.columns:
    df.drop(['Timestamp'], axis=1, inplace=True)
```

After removing the unnecessary column, *the next step* is to split the dataset into input features (X) and target variable (Y), and then split them into training and test sets.

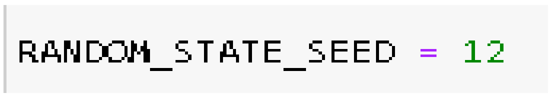
To split the dataset into X and Y, we can use pandas to select the columns we want to use as input features and target variable. We can use the iloc [] method to select columns by their index.

To split X and Y into training and test sets, we can use the train_test_split() function from the sklearn.model_selection library. We can specify the X, Y, and test size as input parameters to the function. The function returns four variables: X_train, X_test, Y_train, and Y_test.



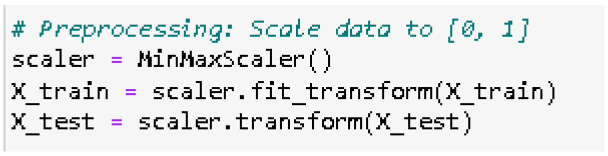
```
# Split data into features and labels
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=RANDOM_STATE_SEED)
```

The random_state in the train_test_split() function sets a seed value for the random number generator, which ensures that the data is split in the same way every time we run the code.



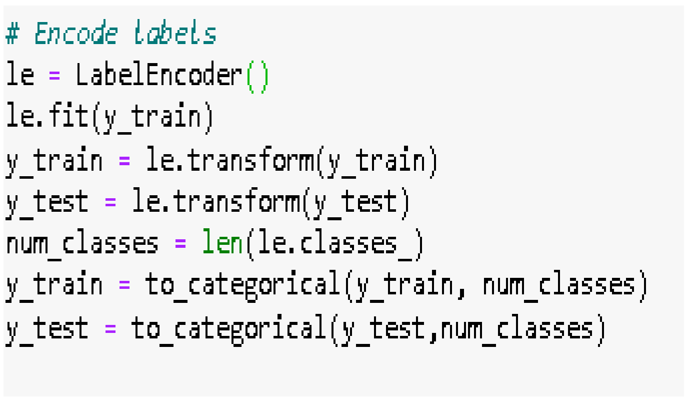
```
RANDOM_STATE_SEED = 12
```

After this step,in order to train and test the dataset it needs to change all values to integers to numerical Transforming the values to a range between 0 and 1 can be achieved using the MinMaxScaler. from the scikit-learn library. The MinMaxScaler scales the data so that the minimum value becomes 0 and the maximum value becomes 1.



```
# Preprocessing: Scale data to [0, 1]
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Next step: encoding labels to convert categorical variables into a numerical format that can be used in machine learning algorithms.



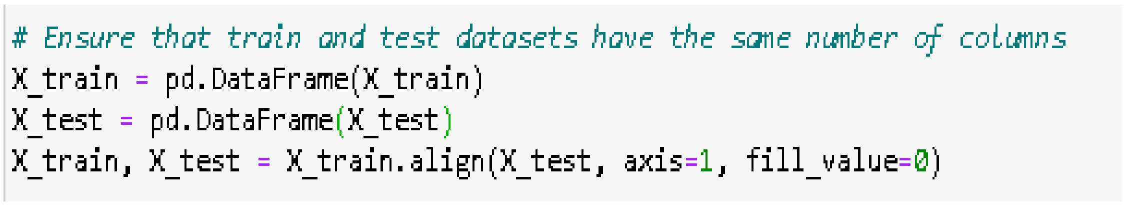
```
# Encode labels
le = LabelEncoder()
le.fit(y_train)
y_train = le.transform(y_train)
y_test = le.transform(y_test)
num_classes = len(le.classes_)
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test,num_classes)
```

LabelEncoder(): encodes categorical labels as integers.

Num_classes : numbers of the unique labels determined by len .

to_categorical(): is then used to convert these integer labels into one-hot encoded vectors.

Then ensure that X_train and X_test has the same columns set



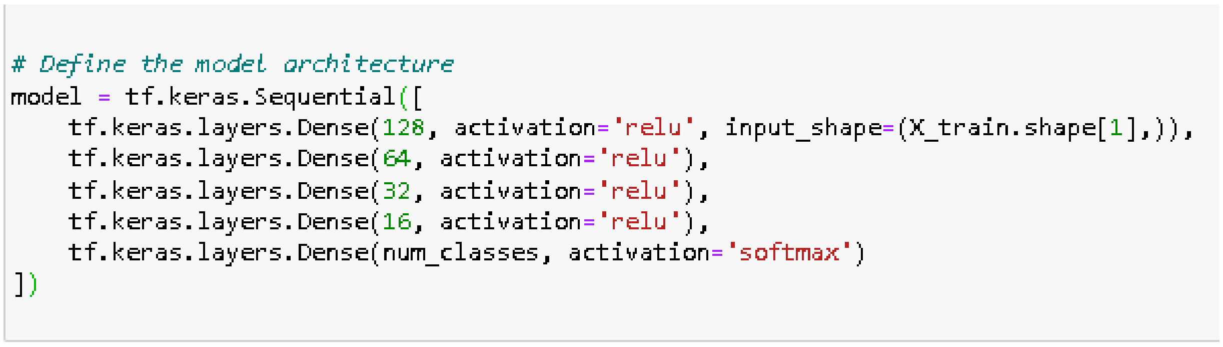
```
# Ensure that train and test datasets have the same number of columns
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
X_train, X_test = X_train.align(X_test, axis=1, fill_value=0)
```

Fill_value : fills any missing value with 0

5 Network architecture selection:

In order to create deep neural network (DNN) architecture we used TensorFlow

Keras API TensorFlow Keras is a high-level neural network API that is part of the TensorFlow library.



```
# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

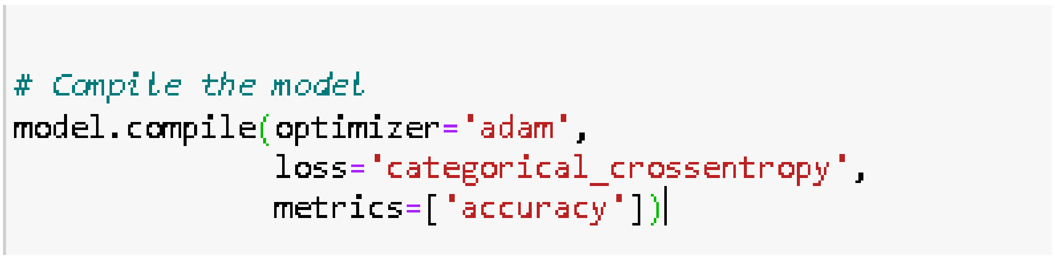
Code explanation:

Sequential: object, which means that the layers are added to the model in sequential order.

For the model define with first layer "**Dense**" with 128 neurons with relu activation which means rule return whether 0 or 1, with input_shape (the number of columns in "**X_train"**).

Same thing for all dense layers, the final layer is a **Dense** layer with **num_classes** neurons, where **num_classes** is the number of classes in the classification problem. This output layer uses a softmax activation function, which outputs probabilities of the input belonging to each class.

Dense: refers to a type of layer where each neuron is connected to every neuron in the previous layer.



```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Compile : is keras methode configures the model for training by adjusting the optimizer, loss function, and evaluation metrics.

Optimizer = adam : Adam is an optimization algorithm that adapts the learning rate based on the loss function.

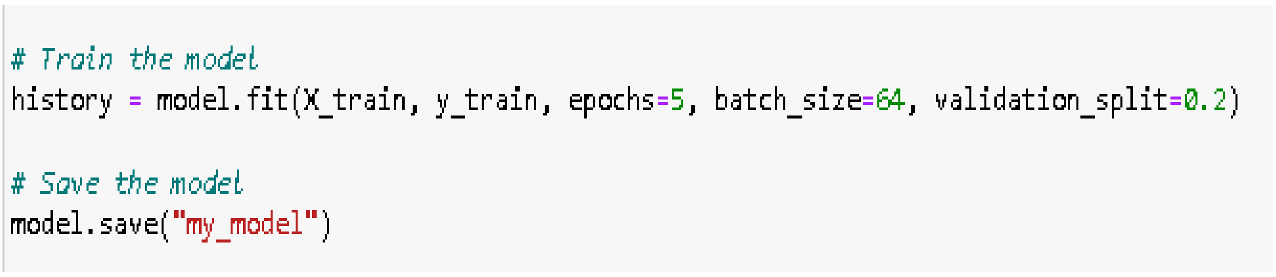
Loss = categorical_crossentropy : is a commonly used loss function for multi-class classification problems, where the goal is to predict one of several possible classes.

metrics=['accuracy']: specifies the evaluation metric to be used during training and evaluation. In this case, the metric is accuracy, which measures the percentage of correctly classified samples in the dataset.

6 Model training:

We train the compiled neural network model on the training data and saves the model for later use.

Train the model & Save it :



```
# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Save the model
model.save("my_model")
```

Model.fit: method is used to train the model. The **X_train** and **y_train** variables represent the training data and labels, respectively

Epochs : parameter specifies the number of times the model will iterate over the entire training set

batch_size: parameter specifies the number of samples that will be processed by the model in each batch

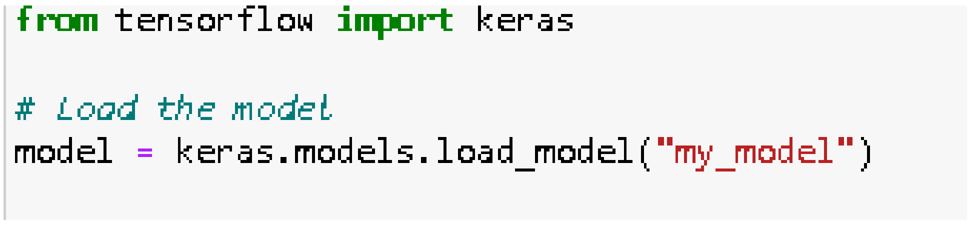
validation_split: parameter specifies the fraction of the training data to use for validation during training.

Save: method is used to save the model to a file named "my_model". This will allow you to load the trained model and use it for inference on new data in the future.

7 Model evaluation :

1- Loading the model

Using keras from tensorflew :



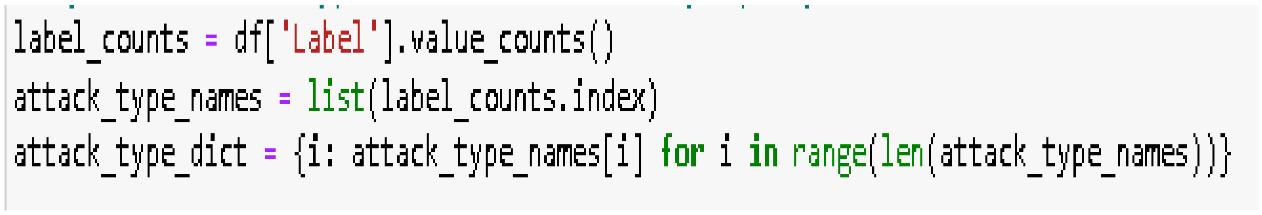
```python
from tensorflow import keras

# Load the model
model = keras.models.load_model("my_model")
```

Loading the saved model allows user to use the trained model for making predictions

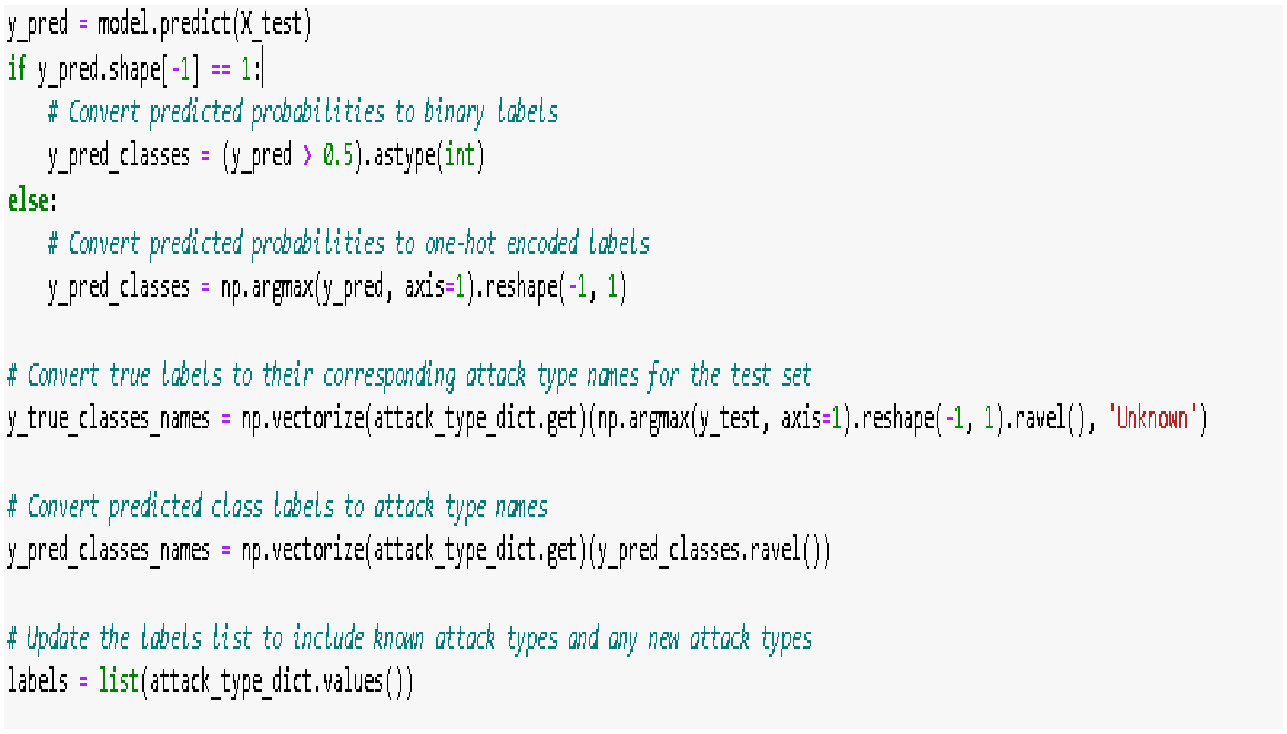
2- Defining the attack type names :

To convert the numerical attack values back to their original names, we used the pandas value_counts method to obtain a list of unique values in a specific column. We then took the resulting index to create a list of names (attack_type_names).

A dictionary (attack_type_dict) is created to map the names to their original values. we used this dictionary to generate the original names for the predicted values generated by my the machine learning model. To ensure there were no problems with the shape of the predicted values (y_pred), we added a condition to check its shape. If there were any issues, they would be fixed.

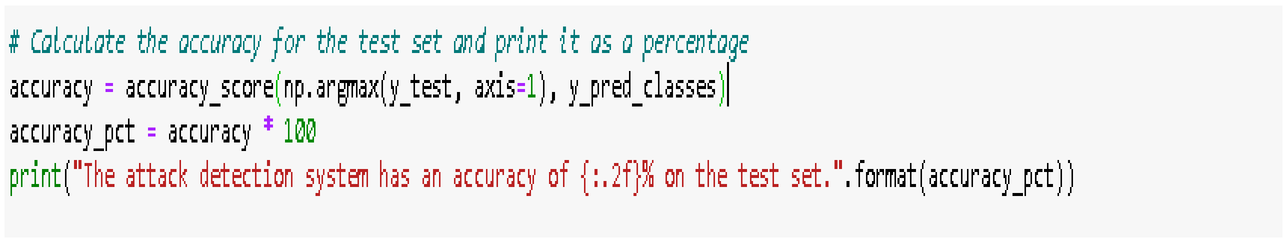


```
label_counts = df['Label'].value_counts()
attack_type_names = list(label_counts.index)
attack_type_dict = {i: attack_type_names[i] for i in range(len(attack_type_names))}
```



```
y_pred = model.predict(X_test)
if y_pred.shape[-1] == 1:
    # Convert predicted probabilities to binary labels
    y_pred_classes = (y_pred > 0.5).astype(int)
else:
    # Convert predicted probabilities to one-hot encoded labels
    y_pred_classes = np.argmax(y_pred, axis=1).reshape(-1, 1)

# Convert true labels to their corresponding attack type names for the test set
y_true_classes_names = np.vectorize(attack_type_dict.get)(np.argmax(y_test, axis=1).reshape(-1, 1).ravel(), 'Unknown')

# Convert predicted class labels to attack type names
y_pred_classes_names = np.vectorize(attack_type_dict.get)(y_pred_classes.ravel())

# Update the labels list to include known attack types and any new attack types
labels = list(attack_type_dict.values())
```

3- Calculating accuracy:



```
# Calculate the accuracy for the test set and print it as a percentage
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred_classes)
accuracy_pct = accuracy * 100
print("The attack detection system has an accuracy of {:.2f}% on the test set.".format(accuracy_pct))
```

Using the accuracy_score: calculate the accuracy from y_test and y_pred_classes.

4- Predicting attacks:

After we have the y_pred_names, we use it to give the score and recall in same attacks that is found in the dataset, and if both of them are 0% that does not mean that it is not in the dataset but it means that prediction failed to detect them which means that the model have not been trained for this attack. In this case, it will automatically detect as new attack that has not been seen in the training set.

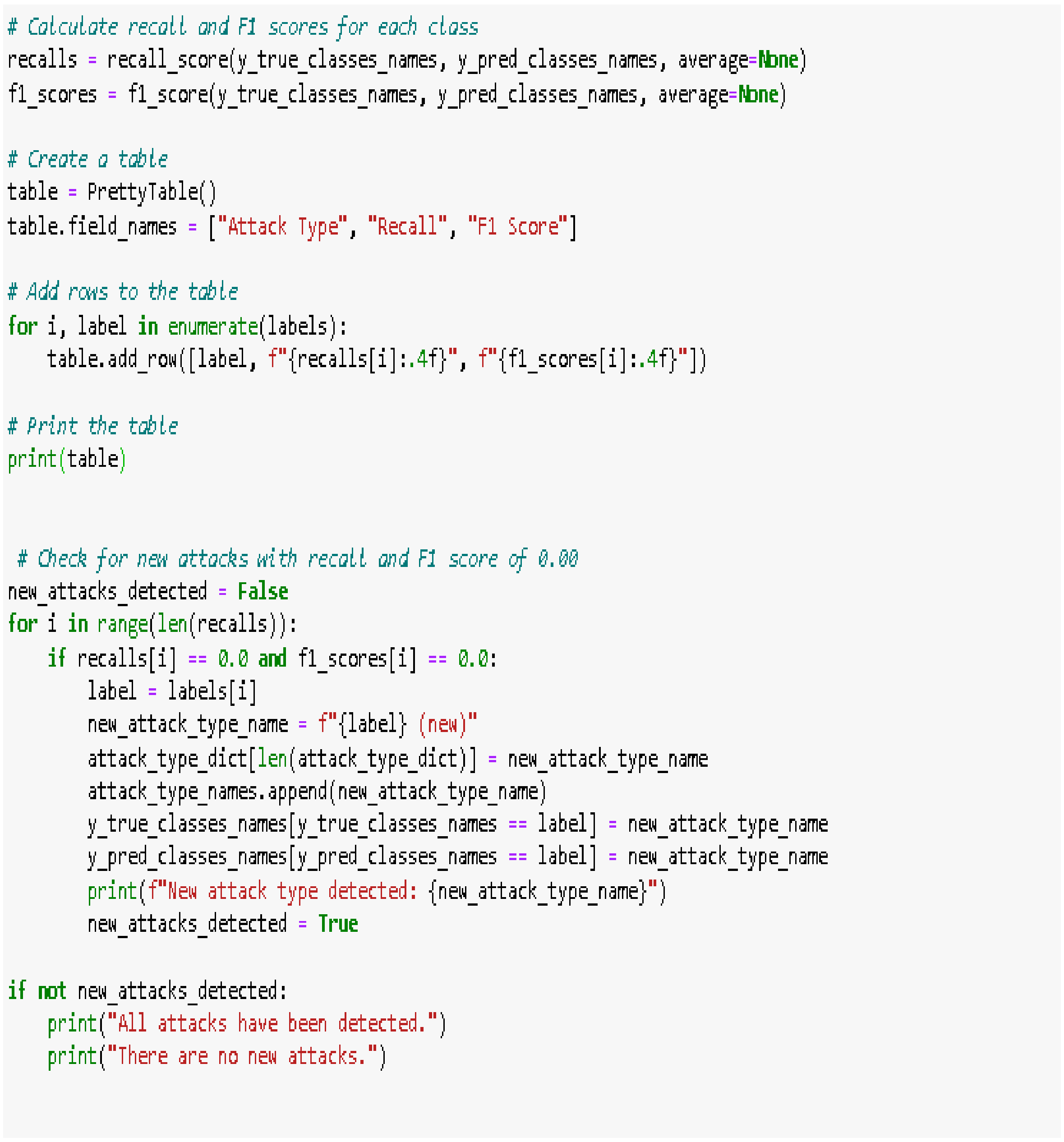
Table representing f1-score and recall:

Recall is a performance metric that shows how many of the actual positive instances were correctly predicted as positive by the model. Its formula is as follows: Recall = TP / (TP + FN).

F1-score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance by considering both precision and recall. Its formula is as follows:

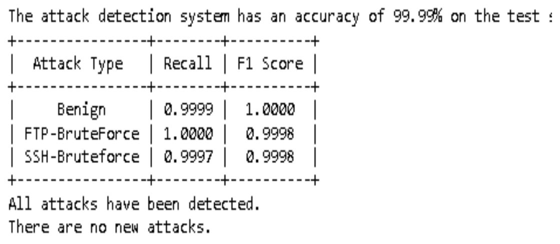
F1 score = 2 * (precision * recall) / (precision + recall)

Precision is the ratio of true positives to the sum of true positives and false positives. F1 score gives equal weight to precision and recall. It is particularly useful when we want to balance the trade-off between precision and recall and get an overall performance measure that considers both metrics.

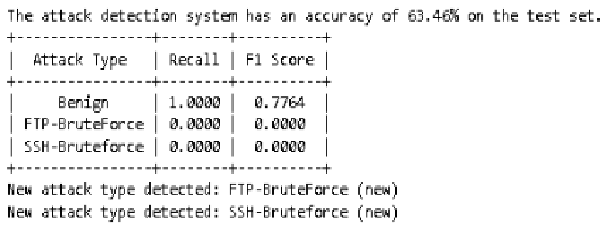


```python
# Calculate recall and F1 scores for each class
recalls = recall_score(y_true_classes_names, y_pred_classes_names, average=None)
f1_scores = f1_score(y_true_classes_names, y_pred_classes_names, average=None)

# Create a table
table = PrettyTable()
table.field_names = ["Attack Type", "Recall", "F1 Score"]

# Add rows to the table
for i, label in enumerate(labels):
    table.add_row([label, f"{recalls[i]:.4f}", f"{f1_scores[i]:.4f}"])

# Print the table
print(table)


 # Check for new attacks with recall and F1 score of 0.00
new_attacks_detected = False
for i in range(len(recalls)):
    if recalls[i] == 0.0 and f1_scores[i] == 0.0:
        label = labels[i]
        new_attack_type_name = f"{label} (new)"
        attack_type_dict[len(attack_type_dict)] = new_attack_type_name
        attack_type_names.append(new_attack_type_name)
        y_true_classes_names[y_true_classes_names == label] = new_attack_type_name
        y_pred_classes_names[y_pred_classes_names == label] = new_attack_type_name
        print(f"New attack type detected: {new_attack_type_name}")
        new_attacks_detected = True

if not new_attacks_detected:
    print("All attacks have been detected.")
    print("There are no new attacks.")
```

So for this code we calculate both F1 score and recall for each class then create a table which contains attack types along with the metrics recall,f1-score; then we put condition: if both recall and f1-score for one attack type are 0.00 that means it is a new attack the model has not seen before, else it will detect all attack types.



```
The attack detection system has an accuracy of 99.99% on the test
+----------------+--------+----------+
|  Attack Type   | Recall | F1 Score |
+----------------+--------+----------+
|     Benign     | 0.9999 |  1.0000  |
| FTP-BruteForce | 1.0000 |  0.9998  |
| SSH-Bruteforce | 0.9997 |  0.9998  |
+----------------+--------+----------+
All attacks have been detected.
There are no new attacks.
```



```
The attack detection system has an accuracy of 63.46% on the test set.
+----------------+--------+----------+
|  Attack Type   | Recall | F1 Score |
+----------------+--------+----------+
|     Benign     | 1.0000 |  0.7764  |
| FTP-BruteForce | 0.0000 |  0.0000  |
| SSH-Bruteforce | 0.0000 |  0.0000  |
+----------------+--------+----------+
New attack type detected: FTP-BruteForce (new)
New attack type detected: SSH-Bruteforce (new)
```

8 Application implementation

In this section we discuss the language used and show some graphical user interface implemented to test our application

Used Language :

In general, Python is widely used as the programming language for deep learning projects due to its simplicity, flexibility, and rich set of libraries and frameworks specifically designed for building and training deep neural network

Python

Python is a high-level programming language designed to be easy to read and simple to implement. It is open source, which means it is free to use, even for commercial applications. It has been stated since 2012 as one of the most languages used in data science (figure 6).

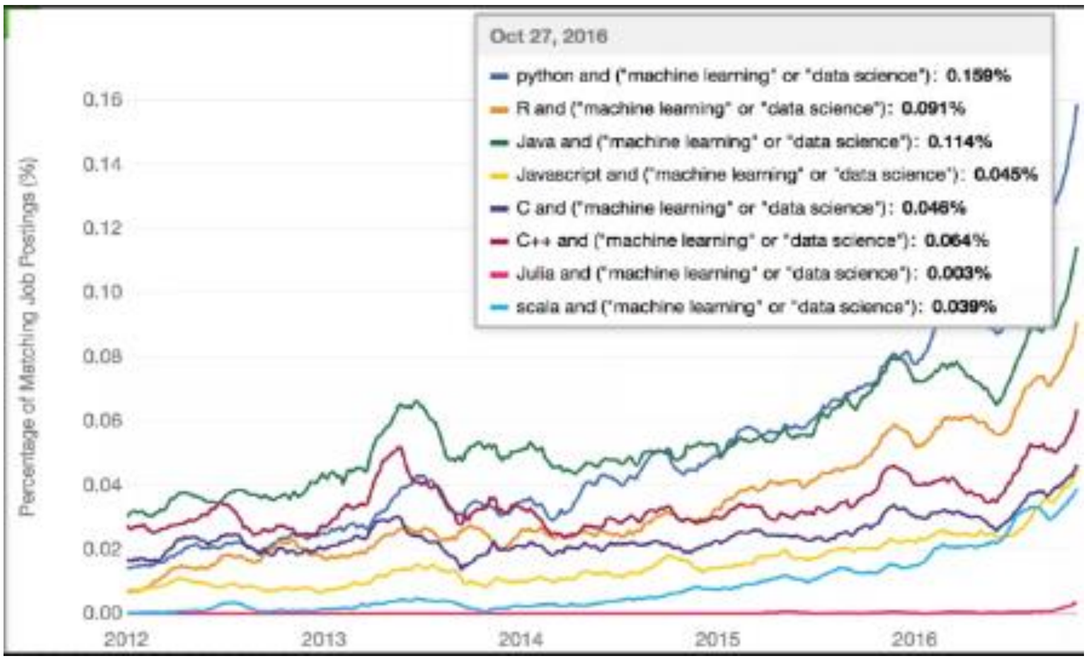


Figure 6 : Python's job positing in data science [9]

CoLibraries used in the project:

Python's popularity for AI is primarily due to the availability of excellent libraries. Here are some commonly used libraries in AI projects:

- Scikit-learn: Used for handling basic ML algorithms such as clustering, linear and logistic regressions, classification, and others.
- Pandas: Used for high-level data structures and analysis, enabling data merging, filtering, and gathering from external sources like Excel.
- Keras: Used for deep learning, providing fast calculations and prototyping capabilities by utilizing both the CPU and GPU of the computer.
- TensorFlow: Developed by Google, TensorFlow is a powerful open-source library for numerical computation and large-scale machine learning, offering a comprehensive ecosystem of tools, libraries, and resources for deep learning.
- PyQt6: A popular cross-platform framework for creating desktop applications with a graphical user interface (GUI), PyQt6 is a Python binding for the Qt framework.

- Matplotlib: Used for creating 2D plots, histograms, charts, and other forms of visualization.

2. Conception of the user interface :

A user interface (UI) refers to the visual and interactive components of a software application or system that allow users to interact with and control the system. It encompasses all the elements that enable users to input commands or data, view information, and receive feedback from the system.

Qt designer :

Qt Design Tools refer to a set of software tools provided by the Qt framework for designing and prototyping user interfaces (UIs) in Qt applications, developers and designers can quickly and effectively construct aesthetically pleasing and engaging UIs for their Qt-based apps.

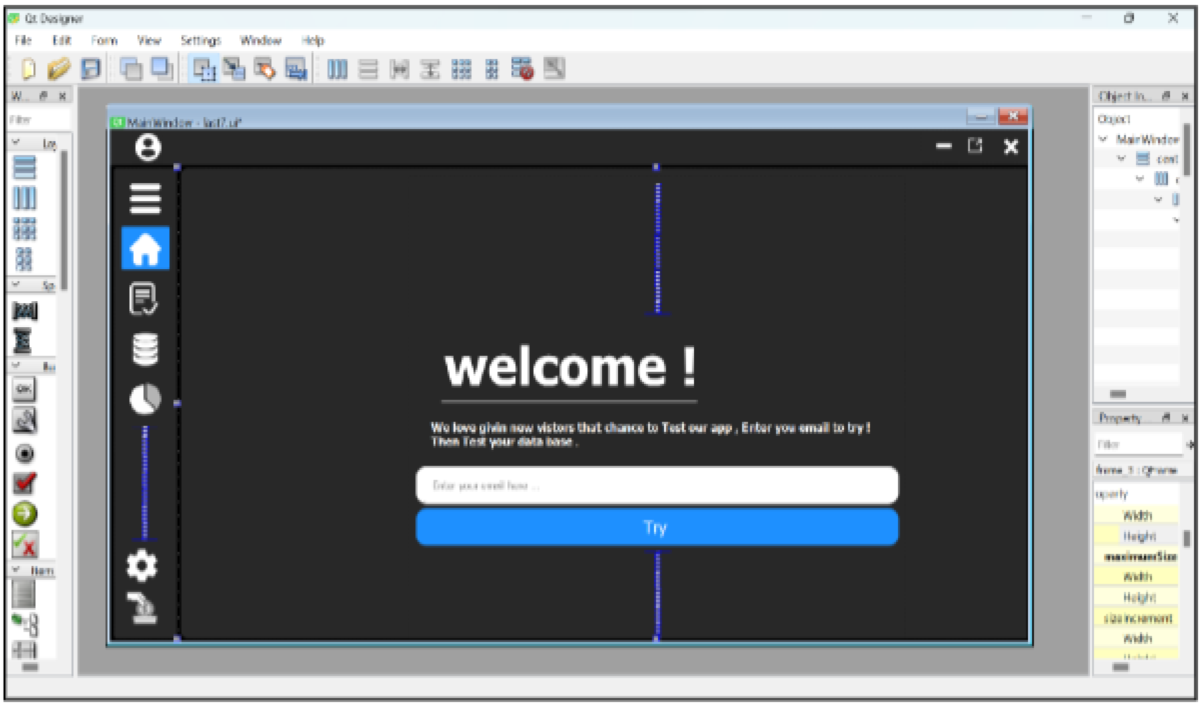
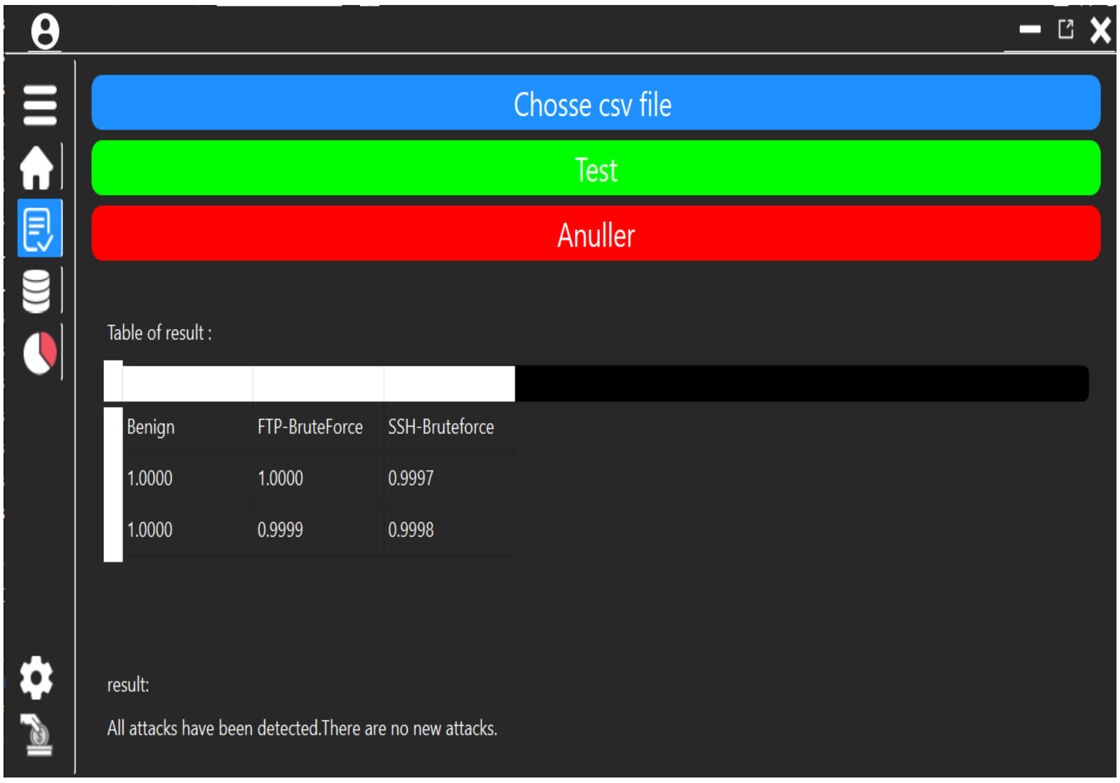

Qt Designer
File
Edit
Form
View
Settings
Window
Help
MainWindow - last7.ui*
welcome !
We love givin new visitors that chance to Test our app , Enter you email to try !
Then Test your data base .
Try

Figure 11 : Application interface .

Testing the application :

loading dataset contains attacks (FTP , SSH). And then press test



Benign	FTP-BruteForce	SSH-Bruteforce
1.0000	1.0000	0.9997
1.0000	0.9999	0.9998

result:

All attacks have been detected.There are no new attacks.

Figure 11 : Application menu.

After the test finished you can check recall score

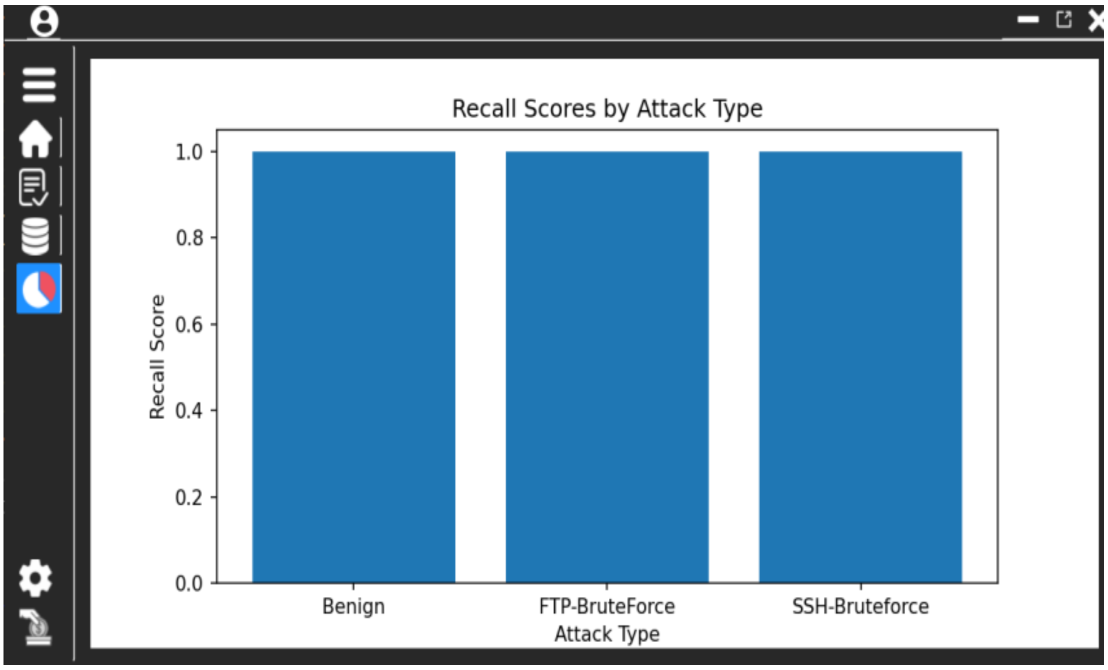


Figure 12 : Recall Score by attack type

8- Conclusion:

Building an effective DNN model for attack detection requires careful data collection, pre-processing, and network architecture selection. By following the steps outlined in this chapter, we have demonstrated how to clean and preprocess raw data, encode categorical variables, and train a DNN model using TensorFlow Keras API. This DNN model can be used as a powerful tool for real-time attack detection.

General sConclusion and perspectives

In this report, we took a general view of intrusion detection systems (IDS), which are an important part of cyber security to detect unauthorized access or malicious activities within a computer network system. We found that IDS major difficulty was in handling high volumes of data, which can lead to failed detection or false positive results.

To fix this issue, Artificial Intelligence (AI) techniques, such as deep Learning models, have been implemented to enhance IDS's performance. These AI models, like Deep Neural Networks (DNNs), can improve the detection accuracy and speed of IDSs, as well as enable them to detect new types of attacks.

Training an Artificial Intelligence (AI) model for intrusion detection systems (ids) requires huge system resources and is time-consuming, that's why we used Google -Colab to train and test the CSV files of CSE-CIC-IDS2018 dataset.

We achieved an accuracy approximating 90% and obtained encouraging F1-score and Recall scores.

As perspectives, we want to test our model on various datasets and compare our performance results with other existing models on the same datasets.

References :

1 : https://attaa.sa/library/view/1442 5/27/2023

2 : https://stacklima.com/difference-entre-les-hid-et-les-nid/ 5/27/2023

3: https://levity.ai/blog/difference-machine-learning-deep-learning 5/27/2023

4:https://www.tibco.com/fr/reference-center/what-is-a-neural-network 5/27/2023

5:https://www.analyticsvidhya.com/blog/2021/05/introduction-to-supervised- deep-learning-algorithms/ 5/27/2023

6:https://www.researchgate.net/figure/Overview-of-a-DNN-architecture-This-architecture-suitable-for-classification-tasks_fig1_306304648 5/27/2023

7 : https://ibelieveai.github.io/ActivationFunctions/#sigmoid 5/28/2023

8: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 5/28/2023

9: https://www.pinterest.com/pin/python-machine-learning-pdf--536421005625529099/ 5/28/2023

10: NSL-KDD dataset:

https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=02-14-2018.csv
https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=02-15-2018.csv
https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=02-21-2018.csv
https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=02-22-2018.csv
https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=02-23-2018.csv
https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv?select=03-02-2018.csv

Abstract

This project is about enhancing the computer security networks and systems by building an IDS.

We want to overcome the challenges faced by IDS in processing large amounts of data while detecting all types of attacks without generating false positives. Our project covers topics such as the definitions of IDS and deep learning, the method used to build the model, how our proposed IDS works, the network architecture, the model training and evaluation. We trained the AI model, using deep learning DNN architecture, on the CSE-CIC-IDS2018 dataset, created by the University of New Brunswick.

Our proposed model achieved acceptable performance, with interesting recall and f1-scores.

Keys words:

Intrusion Detection System, Computer network security, AI model, deep learning, DNN architecture, malicious activities.