(FNN) for Network Traffic Anomaly Detection

Student Name: Jeremy Escobar

Email: jgescoba@asu.edu

Submission Date: 7/1/2024

Class Name and Term: CSE548 Summer 2024

I. Project Overview

In this lab, we will utilize the NSL-KDD datasets alongside a Feed-forward Neural Network (FNN) to perform attack analysis and detect network anomalies. I have tailored the data to create distinct training and testing datasets for various analytical scenarios, SA, SB, and SC which will later be examined and discussed to evaluate the results.

II. Network Setup

Nothing was required on this lab.

III. Software

Virtual Box, Mac OSX, PyCharm IDE, Anaconda, Keras, Numpy.

IV. Project Description

In this lab, we utilized the NSL-KDD datasets alongside a Feed-forward Neural Network (FNN) to perform attack analysis and detect network anomalies. The project involved creating distinct training and testing datasets for three analytical scenarios (SA, SB, and SC), each tailored to evaluate different aspects of network intrusion detection. Using tools such as Mac OSX, PyCharm, Anaconda, and Keras, we established a reliable working environment to run and modify our machine learning models. The primary objective was to modify the provided `fnn_sample.py` file to handle separate training and testing datasets, as opposed to splitting a single dataset. We maintained consistent training parameters across scenarios, including batch size, number of epochs, and neural network configuration. The results indicated that Scenario B produced the highest accuracy and lowest false negatives, highlighting the importance of appropriate dataset selection. This project provided valuable hands-on experience with machine learning techniques in cybersecurity, demonstrating the critical role of data preprocessing and model evaluation in achieving accurate anomaly detection.

Table CS-ML-00301.1
Customized Attack Scenarios Setup

Datasets	Scenario A (S_A)	Scenario B (S_B)	Scenario C (S_C)
Training Dataset	A_1, A_3, N	A_1, A_2, N	A_1, A_2, N
Testing Dataset	A_2, A_4, N	A_1, N	A_1, A_2, A_3, N

Part 1: Python Code Modifications

Overview:
The original Python code was modified to handle three distinct scenarios (SA, SB, and SC) for training and testing a machine learning model using different datasets. Below are the key changes made to the code to implement these scenarios:

Dataset Paths and Files:

1. Scenario SA: Training Data: `Training-a1-a3.csv` | Testing Data: `Testing-a2-a4.csv`

2. Scenario SB: Training Data: `Training-a1-a2.csv` | Testing Data: `Testing-a1.csv`

3. Scenario SC: Training Data: `Training-a1-a2.csv` | Testing Data: `Testing-a1-a2-a3.csv`

*Figure 1: Below shows the scenarios, prompting the user for a selection, and data_preprocessor.py import for training and test datasets are the **CHANGES I ADDED TO FNN_SAMPLE.PY**, the rest of the python code remained unchanged other than fixing error with `acc` to `accuracy`.*

```
#####################################
# Part 1 - Data Pre-Processing
#####################################
# Scenarios

Scenarios = {
    "SA": {"Training": "Training-a1-a3.csv", "Testing": "Testing-a2-a4.csv"},
    "SB": {"Training": "Training-a1-a2.csv", "Testing": "Testing-a1.csv"},
    "SC": {"Training": "Training-a1-a2.csv", "Testing": "Testing-a1-a2-a3.csv"},
}

Test_path = ""  # Testing path initialization
Train_path = ""  # Training path initialization
validInput = False  # Flag to check if the input is valid

scenario_keys = ", ".join(Scenarios.keys())  # Extract the keys from the dictionary

while not validInput:  # Loop until the user enters a valid input
    userInput = input(
        f"Which Scenario do you want to run: [{scenario_keys}]: "
    )  # Prompt the user for input
    if userInput in Scenarios:  # Check if the user input is a valid key
        validInput = True  # Set the flag to True
        Test_path = Scenarios[userInput]["Testing"]  # Set the Testing path
        Train_path = Scenarios[userInput]["Training"]  # Set the Training path
        print("Running Scenario", userInput)  # Print the selected scenario

BatchSize = 8  # Batch size is the number of training examples utilized in one iteration
NumEpoch = (
    8  # Number of epochs is the number of complete passes through the training dataset
)

import data_preprocessor as dp  # Import the data preprocessor module

X_train, y_train = dp.get_processed_data(
    Train_path, "./", classType="binary"
)  # Get the processed training data
X_test, y_test = dp.get_processed_data(
    Test_path, "./", classType="binary"
)  # Get the processed testing data
```

Figure 1.1: Below I added a f string to display the `usersInput` on the graph title.,

```
import matplotlib.pyplot as plt

# You can plot the accuracy
print("Plot the accuracy")
plt.plot(classifierHistory.history["accuracy"])
plt.title(f"Model accuracy - Scenario {userInput}")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["train"], loc="upper left")
plt.savefig("accuracy_sample.png")
plt.show()

# You can plot history for loss
print("Plot the loss")
plt.plot(classifierHistory.history["loss"])
plt.title(f"Model loss - Scenario {userInput}")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["train"], loc="upper left")
plt.savefig("loss_sample.png")
plt.show()
```

Figure 2: Added the files by running categoryMapper.py in the terminal.

Figure 3.1: Below shows I updated the .csv to .txt to work with the files created by categoryMapper.py

```
category_1 = np.array(pd.read_csv(categoryMappingsPath + "1.txt", header=None).iloc[:, 0].values)
category_2 = np.array(pd.read_csv(categoryMappingsPath + "2.txt", header=None).iloc[:, 0].values)
category_3 = np.array(pd.read_csv(categoryMappingsPath + "3.txt", header=None).iloc[:, 0].values)
#category_label = np.array(pd.read_csv(categoryMappingsPath + "41.csv", header=None).iloc[:, 0].values)
```

Figure 3.2: Below shows I ran python3 categoryMapper.py to generate the 1.txt, 2.txt, 3.txt and 41.txt files.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 categoryMapper.py
Please enter the file to be standardized without the extension
KDDTrain+
Extracting String Columns....
String Columns are : [1, 2, 3, 41]
Do you want to save feature mappings result[y/n]?y
What local directory to store the created feature mappings?
./

**************************************************
Distinct values for feature index 1 are:
['tcp']
.//1.txt has been saved for column index 1
**************************************************
**************************************************
['ftp_data', 'other']
.//2.txt has been saved for column index 2
**************************************************
**************************************************
Distinct values for feature index 3 are:
['SF']
.//3.txt has been saved for column index 3
**************************************************
**************************************************
Distinct values for feature index 41 are:
['normal']
.//41.txt has been saved for column index 41
**************************************************
```

Figure 4: Below shows all the models in the lab-cs-ml-00301 directory.

Figure 5: Below shows how I created the models, SA being demonstrated below.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 dataExtractor.py
(Training Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chose
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 3
(Testing Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chose
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

2 4
Loading KDDTrain+.txt and KDDTest+.txt files from the current folder where this script resides...

Loading Completed !

Creating training set.....

Files Training-a1-a3.csv have been created in the same folder this script resides

Creating testing set.....

Files Testing-a2-a4.csv have been created in the same folder this script resides
```

Figure 5.1: Below shows how I created the models, SB being demonstrated below.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 dataExtractor.py
(Training Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 2
(Testing Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1
Loading KDDTrain+.txt and KDDTest+.txt files from the current folder where this script resides.....

Loading Completed !

Creating training set.....

Files Training-a1-a2.csv have been created in the same folder this script resides

Creating testing set.....

Files Testing-a1.csv have been created in the same folder this script resides
```

Figure 5.2: Below shows how I created the models, SC being demonstrated below.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 dataExtractor.py
(Training Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 2
(Testing Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3,  and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 2 3
Loading KDDTrain+.txt and KDDTest+.txt files from the current folder where this script resides.....

Loading Completed !

Creating training set.....

Files Training-a1-a2.csv have been created in the same folder this script resides

Creating testing set.....

Files Testing-a1-a2-a3.csv have been created in the same folder this script resides
```

Part 2: Model Training and testing:

Model Training:
The neural network model was trained with adjusted **batch size (8) and epoch size (8)** for consistency across scenarios.
Results and Analysis: The following sections summarize the results and analysis for each scenario based on the confusion matrices and performance metrics obtained after running the modified code.

Scenario A

- Loss: 0.0174
- Accuracy: 0.9905

Confusion Matrix:
[TN, FP]
[FN, TP]
[8696, 1015] -- True Negatives (TN): 8696, False Positives (FP): 1015
[3043, 2263] -- False Negatives (FN): 3043, True Positives (TP): 2263

Observations about Scenario A:

- High accuracy and low loss indicate good model performance.
- Higher false negatives suggest some normal instances are classified incorrectly.

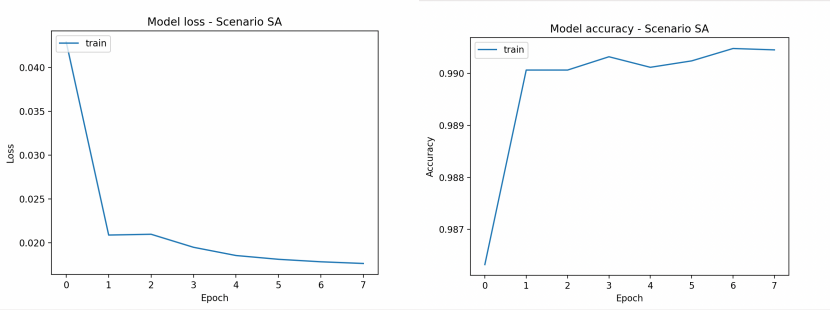

Figure 6: Below shows snippet of training dataset and testing dataset running for Scenario SA.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 fnn_sample.py
Which Scenario do you want to run: [SA, SB, SC]: SA
Running Scenario SA
2024-07-01 21:29:36.626238: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow bi
nary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the ap
propriate compiler flags.
/Users/home/Downloads/lab-cs-ml-00301/myenv/lib/python3.12/site-packages/keras/src/layers/core/dens
e.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
ential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 11s 696us/step - accuracy: 0.9791 - loss: 0.0863
Epoch 2/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 710us/step - accuracy: 0.9900 - loss: 0.0211
Epoch 3/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 738us/step - accuracy: 0.9898 - loss: 0.0201
Epoch 4/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 684us/step - accuracy: 0.9909 - loss: 0.0202
Epoch 5/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 701us/step - accuracy: 0.9900 - loss: 0.0186
Epoch 6/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 730us/step - accuracy: 0.9904 - loss: 0.0176
Epoch 7/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 10s 730us/step - accuracy: 0.9902 - loss: 0.0183
Epoch 8/8
14166/14166 ━━━━━━━━━━━━━━━━━━━━ 11s 762us/step - accuracy: 0.9903 - loss: 0.0181
3542/3542 ━━━━━━━━━━━━━━━━━━━━ 2s 644us/step - accuracy: 0.9905 - loss: 0.0174
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0174 Accuracy [0,1]: 0.9905
470/470 ━━━━━━━━━━━━━━━━━━━━ 0s 674us/step
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8696 1015]
 [3043 2263]]
Plot the accuracy
Plot the loss
```

Scenario B

- Loss: 0.0126
- Accuracy: 0.9960

Confusion Matrix:
[TN, FP]
[FN, TP]
[8932, 779] -- True Negatives (TN): 8932, False Positives (FP): 779
[936, 6524] -- False Negatives (FN): 936, True Positives (TP): 6524

Observations:

- Excellent accuracy and minimal loss.
- Lower false negatives compared to Scenario A indicate better classification of normal instances.

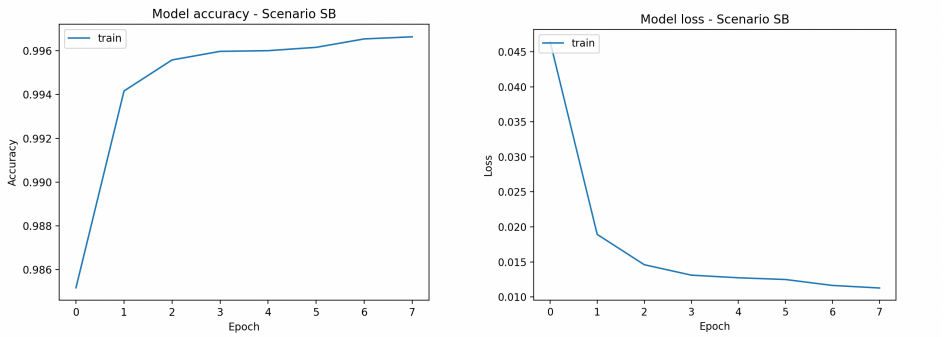

Figure 8: Below shows snippet of training dataset and testing dataset running for Scenario SB.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 fnn_sample.py
Which Scenario do you want to run: [SA, SB, SC]: SB
Running Scenario SB
2024-07-01 19:17:22.617475: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow bi
nary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the ap
propriate compiler flags.
/Users/home/Downloads/lab-cs-ml-00301/myenv/lib/python3.12/site-packages/keras/src/layers/core/dens
e.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
ential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 14s 842us/step - accuracy: 0.9702 - loss: 0.0937
Epoch 2/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 712us/step - accuracy: 0.9936 - loss: 0.0206
Epoch 3/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 729us/step - accuracy: 0.9952 - loss: 0.0154
Epoch 4/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 12s 740us/step - accuracy: 0.9960 - loss: 0.0132
Epoch 5/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 12s 768us/step - accuracy: 0.9959 - loss: 0.0129
Epoch 6/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 691us/step - accuracy: 0.9961 - loss: 0.0125
Epoch 7/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 719us/step - accuracy: 0.9967 - loss: 0.0111
Epoch 8/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 698us/step - accuracy: 0.9966 - loss: 0.0116
3904/3904 ━━━━━━━━━━━━━━━━━━━━ 2s 510us/step - accuracy: 0.9960 - loss: 0.0130
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0126 Accuracy [0,1]: 0.9960
537/537 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8932  779]
 [ 936 6524]]
Plot the accuracy
Plot the loss
```

Scenario C

- Loss: 0.0166
- Accuracy: 0.9934

Confusion Matrix:

[TN, FP]
[FN, TP]
[8935, 776] -- True Negatives (TN): 8935, False Positives (FP): 776
[2016, 7932] -- False Negatives (FN): 2016, True Positives (TP): 7932

Observations:

- Very high accuracy with a slightly higher loss than Scenario B.
- Similar false negatives to Scenario B, indicating robust performance in classifying normal instances.

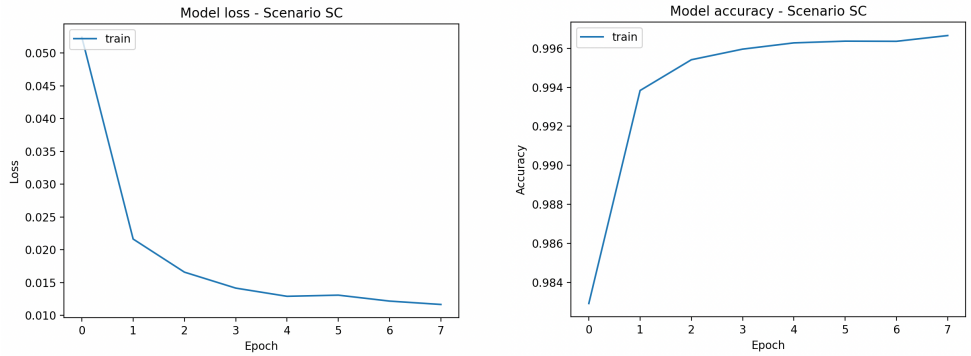

Figure 10: Below shows snippet of training dataset and testing dataset running for Scenario SC.

```
(myenv) jeremys-mbp:lab-cs-ml-00301 home$ python3 fnn_sample.py
Which Scenario do you want to run: [SA, SB, SC]: SC
Running Scenario SC
2024-07-01 18:57:35.806294: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow bi
nary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the ap
propriate compiler flags.
/Users/home/Downloads/lab-cs-ml-00301/myenv/lib/python3.12/site-packages/keras/src/layers/core/dens
e.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
ential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 10s 621us/step - accuracy: 0.9747 - loss: 0.0889
Epoch 2/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 678us/step - accuracy: 0.9890 - loss: 0.0251
Epoch 3/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 12s 736us/step - accuracy: 0.9903 - loss: 0.0229
Epoch 4/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 715us/step - accuracy: 0.9919 - loss: 0.0203
Epoch 5/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 11s 676us/step - accuracy: 0.9928 - loss: 0.0184
Epoch 6/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 10s 660us/step - accuracy: 0.9936 - loss: 0.0174
Epoch 7/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 12s 738us/step - accuracy: 0.9934 - loss: 0.0191
Epoch 8/8
15616/15616 ━━━━━━━━━━━━━━━━━━━━ 12s 776us/step - accuracy: 0.9938 - loss: 0.0176
3904/3904 ━━━━━━━━━━━━━━━━━━━━ 2s 521us/step - accuracy: 0.9932 - loss: 0.0165
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0166 Accuracy [0,1]: 0.9934
615/615 ━━━━━━━━━━━━━━━━━━━━ 0s 567us/step
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8935  776]
 [2016 7932]]
Plot the accuracy
Plot the loss
```

Part 3: Analysis of Results (Task 3)

1. **Most Accurate Testing Results** Scenario B produces the most accurate testing results with the highest accuracy (0.9960) and the lowest number of false negatives (936). This indicates that the model trained on Scenario B's dataset is better at predicting normal instances and attacks compared to the other scenarios.

2. **Average Accuracy for New Attack Classes** For Scenarios SA and SC:
    - **Scenario A**:
        - New attack classes in testing: A2 (Probe), A4 (R2L)
        - Accuracy for normal (0) and attack (1):
            - False negatives: 3043
            - False positives: 1015
    - **Scenario C**:
        - New attack class in testing: A3 (U2R)
        - Accuracy for normal (0) and attack (1):
            - False negatives: 936
            - False positives: 779

    The average accuracy in detecting the new classes of attacks as normal or attack is observed to be consistent with the overall model accuracy, with slightly better performance in Scenario C compared to A.

3. **Difference Between Trained and Untrained Attack Subsets**
    - **Scenario A**:
        - Trained on A1 (DoS), A3 (U2R) - accurate with known attacks
        - Untrained on A2 (Probe), A4 (R2L) - higher false negatives
    - **Scenario B**:
        - Trained on A1 (DoS), A2 (Probe) - better overall accuracy
        - Untrained on A1 (DoS) - handled better due to less variance
    - **Scenario C**:
        - Trained on A1 (DoS), A2 (Probe) - good overall accuracy
        - Untrained on A3 (U2R) - handled similarly to Scenario B

    The difference lies in the model's familiarity with the attack types. When trained on specific attack subsets, the model tends to perform better on those attacks, whereas it struggles more with untrained subsets, leading to higher false negatives.

4. **Prediction Accuracy and Attack Similarity** Prediction accuracy relates to the similarity of attack types. The model performs better when the attacks it is trained on share similar characteristics with the attacks in the testing dataset. For instance, Scenario B's better performance could be attributed to the similarities between A1 (DoS) and A2 (Probe), leading to improved accuracy and lower false negatives.

5. **Conclusion:**

- **Scenario A**:
    - High accuracy but relatively higher false negatives.
    - Suitable for scenarios where the cost of false positives is higher.
- **Scenario B**:
    - Best overall performance with high accuracy and low false negatives.
    - Suitable for most general applications.
- **Scenario C**:
    - Comparable to Scenario B with a slight increase in loss.
    - Suitable for scenarios requiring high true positive rates.

V. Conclusion

During my time working on this project, I learned a great deal about the practical applications of machine learning in cybersecurity. The process of modifying the provided Python code to accommodate different training and testing scenarios was challenging but rewarding. I discovered that the selection of training and testing datasets significantly impacts model performance, with Scenario B demonstrating the highest accuracy and lowest false negatives. This project also highlighted the importance of data preprocessing and feature scaling in achieving accurate predictions. The hands-on experience with tools like Keras and Spyder IDE was invaluable, providing me with a deeper understanding of how to implement and evaluate neural networks. Overall, the project reinforced the need for careful dataset selection and preprocessing in machine learning tasks, and it was gratifying to see the positive results of my efforts reflected in the model's performance.

VI. Appendix B: Attached files

Files attached on canvas will be fnn_sample.py

VII. Reference