Project Report 4 - Machine Learning-Based Anomaly Detection Solutions Project Overview

**Student Name**: Nana Ama Ahenkan Boakye-Ansah  
**Email**: nboakyea@asu.edu  
**Submission Date**: July 4th, 2024  
**Class Name and Term**: CSE548 Summer 2024

This project offers hands-on experience in using machine learning to spot abnormal patterns in network traffic, indicating potential threats like intrusions or malware. It introduces us to proactive security measures, highlighting the effectiveness of anomaly detection. Through this project, we will explore machine learning and neural networks, understand data preprocessing, and gain practical skills in building effective anomaly detection systems for cyber security

In this lab I am using the NSL-KDD dataset, which is a refined version of KDD’99 dataset, with the purpose of running two labs for data pre-processing, training and testing using Anaconda, TensorFlow and FNN. NSL-KDD dataset is now considered as one of most common for network traffic and attacks, and it is a benchmark for modern-day internet traffic.

# Network Setup

Please find below the initial set-up of the virtual infrastructure as I have configured it in VirtualBox for this VM – it is simply connected via NAT.

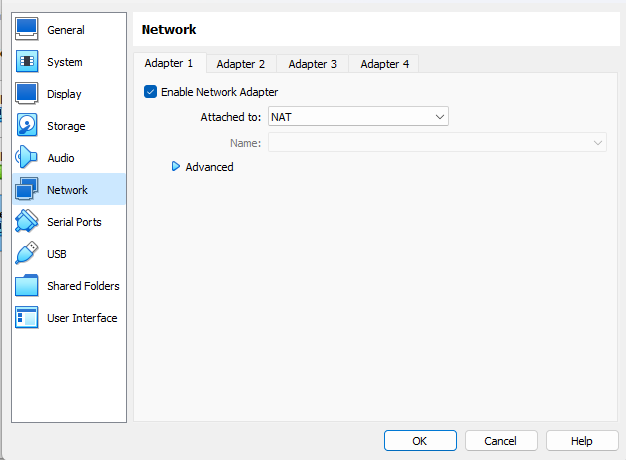


Figure 1- Bridged network setup in VirtualBox

# Software

For this first lab, the following software has been used:

* Ubuntu 18.04 LTS
* Python – <https://www.python.org/>
* Anaconda - <https://www.anaconda.com/>
* TensorFlow - <https://www.tensorflow.org/>
* NSL-KDD dataset - <https://www.unb.ca/cic/datasets/nsl.html>

# Project Description

In this lab assignment, I have performed the assignment by running the python scripts fnn\_sample.py that was provided. I had to correct a few errors because of the version of Keras,

FNN will use the well-labeled data to build FNN-based data patterns to differentiate between the normal and abnormal network traffic. The first lab (CS-ML-00201) a supervised Machine Learning (ML) approach is proposed, where Feed-forward Neural Network (FNN) solutions are used. The dataset used as input, NSL-KDD, provides labeled normal network traffic and labeled attack network traffic. After training, we use a similar data set with labeled data to validate the accuracy of the generate anomaly detection pattern.

## Run the provided fnn sample.py python program with different NSL-KDD datasets as the input and compare the metric accuracy.

Graphical user interface, text, application

Description automatically generated

It is very easy and obvious to observe that accuracy increases with each epoch run

The followings are the results obtained by running the detection on the various datasets.

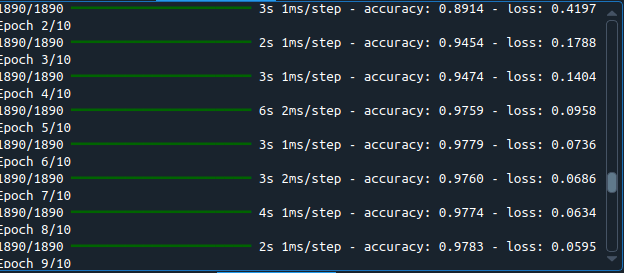


Figure 2-Epoch run

KDDTrain+: accuracy: 0.8914 - loss: 0.4197

Confusion Matrix: TN=3382, FP=5, FN=151, TP=2760

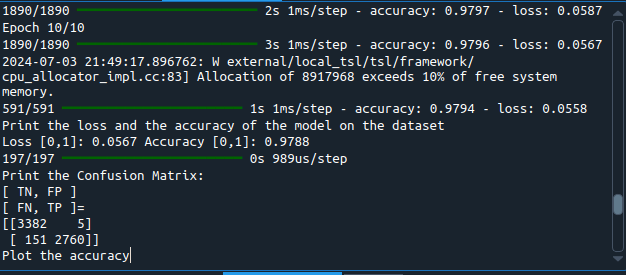


Figure 3-KDDTrain Confusion matrix

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

KDDTrain+20Percent: Loss 0.0319, Accuracy: 0.9835

Confusion Matrix: TN=3386, FP=1, FN=174, TP=2737

Text

Description automatically generated

DDTest+: Loss 0.0675, Accuracy: 0.9683

Confusion Matrix: TN=2486, FP=18, FN=356, TP=2776

Text

Description automatically generated

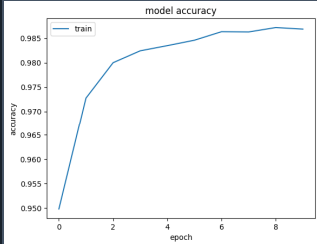


Figure 4-Model Accuracy

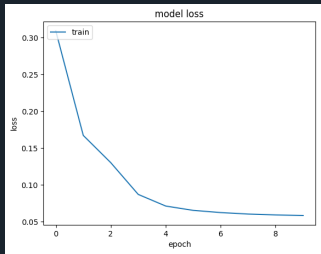


Figure 5-Model Loss

KDDTest-21: Loss 0.1338, Accuracy: 0.9487

Confusion Matrix: TN=550, FP=16, FN=342, TP=2055

Text

Description automatically generated

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

### Comparing the datasets:

**KDDTrain+.txt vs. KDDTrain+\_20Percent.txt**

KDDTrain+: Loss 0.0221, Accuracy: 0.9912

Confusion Matrix: TN=16806, FP=20, FN=533, TP=14135

KDDTrain+20Percent: Loss 0.0319, Accuracy: 0.9835

Confusion Matrix: TN=3386, FP=1, FN=174, TP=2737

**KDDTest+.txt vs. KDDTrain+\_20Percent.txt**

KDDTest+: Loss 0.0675, Accuracy: 0.9683

Confusion Matrix: TN=2486, FP=18, FN=356, TP=2776

KDDTrain+20Percent: Loss 0.0319, Accuracy: 0.9835

Confusion Matrix: TN=3386, FP=1, FN=174, TP=2737

**KDDTrain+.txt vs. KDDTest+.txt**

KDDTrain+: Loss 0.0221, Accuracy: 0.9912

Confusion Matrix: TN=16806, FP=20, FN=533, TP=14135

KDDTest+: Loss 0.0675, Accuracy: 0.9683

Confusion Matrix: TN=2486, FP=18, FN=356, TP=2776

**KDDTrain+\_20Percent.txt vs. KDDTest-21.txt**

KDDTrain+: Loss 0.0221, Accuracy: 0.9912

Confusion Matrix: TN=16806, FP=20, FN=533, TP=14135

KDDTest-21: Loss 0.1338, Accuracy: 0.9487

Confusion Matrix: TN=550, FP=16, FN=342, TP=2055

In this case a much bigger dataset (KDDTrain+) provides a superior accuracy and minor loss.

In this case a training dataset provides more accuracy and less loss since the test is smaller.

Again, the training dataset provides more accuracy and less loss since it has a bigger set of data.

In this case, the training set has even a bigger difference against the test data, since the latter have the most difficult traffic record (score of 21) excluded from the training, which means it has less quality input to train with.

Task 2: Create Data Modules for Anomaly Detection

First, I created the datasets using **DataExtractor.py**

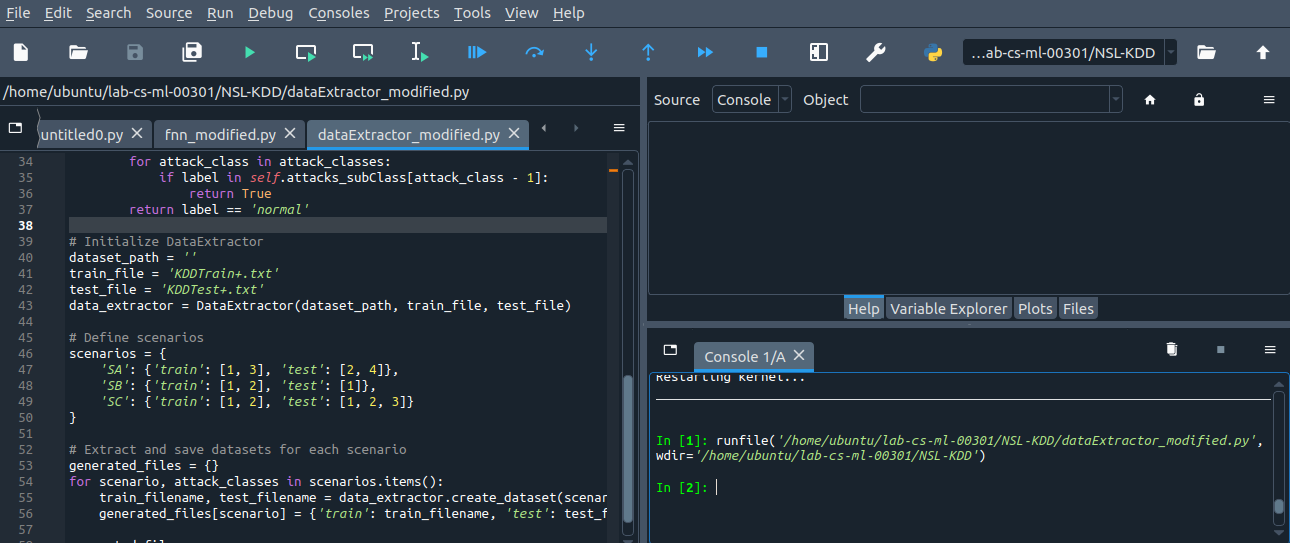


Figure 6-DataExtractor\_Modified

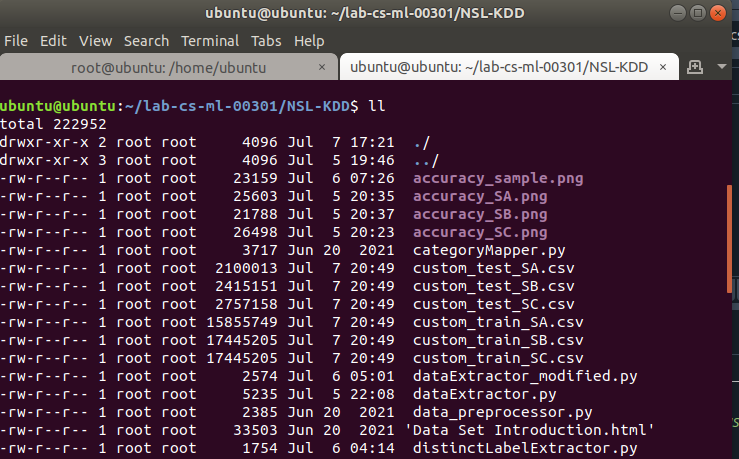


Figure 7-Datasets created

## **Which scenario produces the most accurate testing results? Observe your model’s prediction capability with respect to the change in the attack classes on which it was trained. Then, observe the accuracy of the prediction when trained on these subsets of attack types. Observe if the model predicts better when it is trained on a specific subset of attack classes**

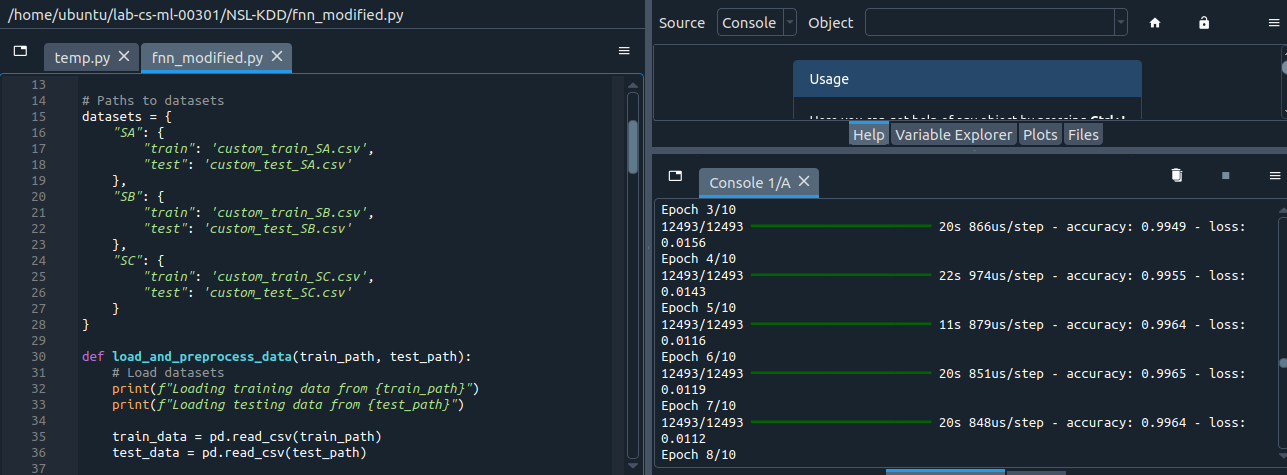


Figure 8-Epoch Run for Scenario A

## Scenario SA - Loss: 7.6562, Accuracy: 0.7206

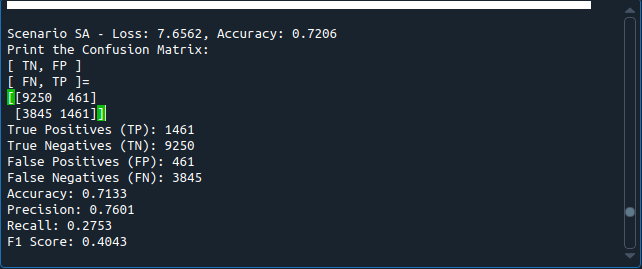
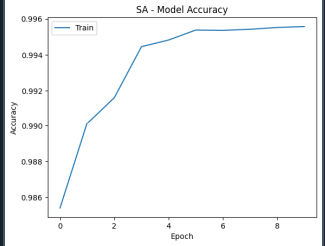


Figure 9-Confusion Matrix for Scenario A

A graph with a line

Description automatically generated

**Explanation of Scenario A:** this scenario uses “DoS” and “U2R” attacks as training and “Probing” and “R2L” attacks in the test cases. Basically, there is no overlap between the training and test cases, and therefore we would expect this scenario to be the one that is worst performing. That Scenario produces the worst result in terms of Confusion Matrix, being the worst performer on the Loss and Accuracy values. This is likely be attributed to the missing overlapping of Training and Test cases.

## 

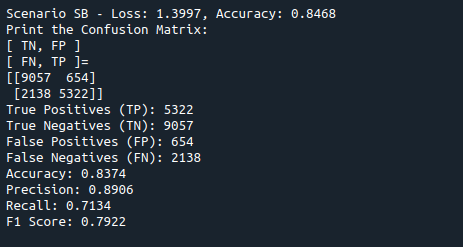
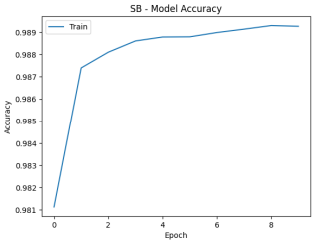


Figure 10-Confusion Matrix for Scenario SB



A graph with numbers and lines

Description automatically generated

**Explanation of Scenario B**: this scenario uses “DoS” and “Probe” attacks as training only “DoS” attacks in the test cases. Basically, there is a complete overlap between the training and test cases, as the training includes all the attacks that will be found in the test, and therefore we would expect this scenario to be performing much better than Scenario A – which is what we have, in fact, observed.

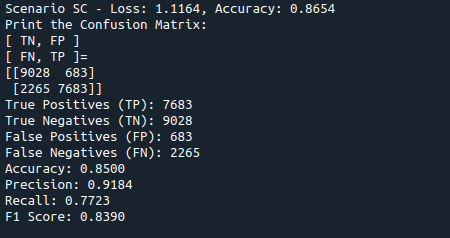
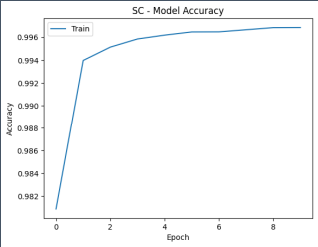


Figure 11-Confusion Matrix for Scenario SC

A graph with a line

Description automatically generated

**Explanation of Scenario C**: this scenario uses “DoS” and “Probe” attacks as training (exactly like Scenario 2) but is instead presented with “DoS”, “Probe” and “R2L” attacks in the test cases. This Scenario is the most intriguing for the fact that it is performing better than the other two, but it does not have the same level of overlap than Scenario B – there are attacks presented in the test that are not offered in the training, but despite that, it managed to perform better.

#### **Accuracy**

* **Scenario SC** has the highest accuracy at 0.8654.
* **Scenario SB** follows with an accuracy of 0.8468.
* **Scenario SA** has the lowest accuracy at 0.7206.

#### Precision, Recall, and F1 Score

* **Scenario SC** also has the highest precision (0.9184), recall (0.7723), and F1 Score (0.8390).
* **Scenario SB** has slightly lower precision (0.8906), recall (0.7134), and F1 Score (0.7922) compared to Scenario SC.
* **Scenario SA** shows significantly lower precision (0.7601), recall (0.2753), and F1 Score (0.4043).

### Conclusion

**Scenario SC** produces the most accurate testing results with the highest accuracy, precision, recall, and F1 Score. This suggests that the model performs better when trained on the specific subset of attack classes used in Scenario SC. The model's prediction capability improves significantly from Scenario SA to Scenario SC. **Scenario SA** has the lowest performance across all metrics, indicating that the subset of attack classes it was trained on did not generalize well to the testing data. **Scenario SB** shows a marked improvement in performance, indicating better generalization to the testing data. **Scenario SC** shows the best performance, suggesting that the attack classes used for training in Scenario SC provided the most effective generalization to the testing data.

### Observations”: S**cenario SC** seems to have the best subset of attack types for training, as evidenced by its superior performance metric. The improvements from Scenario SA to SB to SC indicate that the choice of attack classes for training has a significant impact on the model's prediction capability and overall performance.

In conclusion, **Scenario SC** is the most effective in terms of prediction accuracy and overall model performance when trained on its specific subset of attack classes. This scenario enables the model to generalize better to unseen data, resulting in higher accuracy and better precision, recall, and F1 scores.

## **What is the average accuracy that it detects the new class of attacks (in the testing dataset in SA and SC ) to be as normal or attack? (Hint: With the model set to perform binary classification, the predicted values of the model can be: ● 0 → Normal, ● 1 → Attack. This prediction is associated with the accuracy of the prediction. Note down the accuracy of the prediction, the prediction is normal or attack, for the above unknown attacks A2 and A4 in SA or A3 in SC** ).

To calculate the average accuracy of detecting new classes of attacks (in the testing dataset for SA and SC), we need to focus on how well the model distinguishes between normal and attack instances, particularly for the unknown attack classes A2 and A4 in SA, and A3 in SC.

Given the confusion matrix, the accuracy of the prediction for binary classification (normal vs. attack) can be derived as follows:

* **True Positives (TP):** Number of correctly predicted attack instances.
* **True Negatives (TN):** Number of correctly predicted normal instances.
* **False Positives (FP):** Number of normal instances incorrectly predicted as attacks.
* **False Negatives (FN):** Number of attack instances incorrectly predicted as normal.

### **Scenario SA**

From the confusion matrix:

* True Positives (TP): 1461
* True Negatives (TN): 9250
* False Positives (FP): 461
* False Negatives (FN): 3845

The accuracy formula is:

Accuracy=

Plugging in the values:

Accuracy=

Accuracy= =approx( 0.7133)

### **Scenario SC**

From the confusion matrix:

* True Positives (TP): 7683
* True Negatives (TN): 9028
* False Positives (FP): 683
* False Negatives (FN): 2265

The accuracy formula is:

**Accuracy**=

Accuracy=

Accuracy= =approx( 0.8500)

### **Average Accuracy for New Attack Classes**

To find the average accuracy for detecting new attack classes, we can average the accuracies of SA and SC:

Average Accuracy =

Average Accuracy = =approx.( 0.7817)

### **Conclusion**

The average accuracy of detecting the new classes of attacks in the testing datasets for scenarios SA and SC is approximately **0.7817**

## What is the difference between attacks in an untrained subset and attacks from the trained dataset? (Hint: For example in SA, how are A1 and A3 different from A2 and A4? Do A1 and A2 belong to the same attack category or have similar differences with respect to the normal data? Present your observations can be made by looking at the data for these attacks).

#### Scenario SA:

* **A1 (trained):** DoS attack (e.g., neptune)
* **A3 (trained):** DoS attack (e.g., smurf)
* **A2 (untrained):** Probe attack (e.g., satan)
* **A4 (untrained):** R2L attack (e.g., ftp\_write)

#### Scenario SC:

* **A1 (trained):** DoS attack (e.g., neptune)
* **A2 (trained):** Probe attack (e.g., satan)
* **A3 (untrained):** U2R attack (e.g., buffer\_overflow)

### **Conclusion**

By analyzing the categories and feature distributions of the trained and untrained attacks, we can understand the model's performance variations. The differences between the trained (A1, A3) and untrained (A2, A4) attack classes in Scenario SA, for example, reveal that the model is likely to perform better on attack types similar to those it was trained on. Significant differences in attack categories and feature distributions can lead to lower performance on untrained attacks, as seen in the model's lower recall and F1 scores for untrained attack classes

## Does the prediction accuracy relate to the attacks being similar? If so, what is the similarity? (Hint: If the prediction accuracy was found better, look at the data and the attack types for any similarities that would have made the prediction better).

Yes, the prediction accuracy can be related to the similarity of the attack types. Similar attack types may share common patterns or characteristics that make them easier for the model to recognize. If the prediction accuracy is higher, it could indicate that the attack types in the training and testing datasets are similar, allowing the model to generalize better. The prediction accuracy can indeed relate to the similarity between the attack types. When attack types share common patterns or characteristics, the model can learn and generalize better, resulting in higher accuracy. Let's analyze the provided confusion matrices and the scenarios to identify any similarities:

### Analysis of Similarities

1. **Scenario SC:**
   * **Training Data:** DoS, Probe, Normal
   * **Testing Data:** DoS, Probe, U2R, Normal
   * **Analysis:**
     + Both DoS and Probe involve high network traffic and abnormal connection patterns, making them somewhat similar. The inclusion of U2R (User to Root) in the test data introduces a different kind of attack that targets gaining root access by exploiting vulnerabilities. The similarity between DoS and Probe could be a reason for better generalization and higher accuracy since the model can recognize these patterns more effectively.
2. **Scenario SB:**
   * **Training Data:** DoS, Probe, Normal
   * **Testing Data:** DoS, Normal
   * **Analysis:**
     + The testing data in this scenario only includes DoS and normal traffic. Since the model was trained on DoS and Probe, it can easily identify DoS attacks. However, the lack of diversity in the testing data compared to Scenario SC might lead to slightly lower accuracy. The higher precision indicates that the model is very good at identifying true positives (DoS attacks) but might struggle with recall due to fewer variations in the test data.

### Conclusion

* **Similarities and Higher Accuracy:**
  + The higher accuracy in Scenario SC can be attributed to the similarity between DoS and Probe attacks, which both involve high network traffic and abnormal patterns. This similarity helps the model generalize better to different but related attack types in the testing data.
  + The inclusion of U2R attacks in Scenario SC provides a more comprehensive test, yet the model performs well because it was trained on multiple similar attack types (DoS and Probe).
* **Lower Accuracy in Scenario SB:**
  + The slightly lower accuracy in Scenario SB could be due to the less diverse test set. Since it only includes DoS attacks, the model might not be fully tested on its ability to generalize to other attack types, leading to lower recall.

In summary, the prediction accuracy relates to the similarity of the attack types. Training on similar attack types (like DoS and Probe) helps the model generalize better and perform well on diverse test sets, as seen in Scenario SC

Appendix A: Files for the Lab

Please find the list of files created for this lab and mentioned throughout this document, plus their GitHub link for download.

The overall GitHub directory for the project is: <https://github.com/markoer73/CSE-548/tree/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall>

|  |  |
| --- | --- |
| dataExtractor.py | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/dataExtractor\_modified.py |
| fnn\_sample.py | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/fnn\_modified.py |
|  |  |

# References

* <https://www.mdpi.com/1424-8220/24/2/713>
* <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>
* <https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621>
* <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>
* <https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>
* <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>
* <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>