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

Student Name: Marco Ermini
Email: mermini@asu.edu
Submission Date: 11th Jun, 2021
Class Name and Term: CSE548 Summer 2021

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.

All the files and configurations used for this lab have been uploaded on GitHub; references are provided throughout the text and in the Appendix A at the bottom of the file.

I. 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.

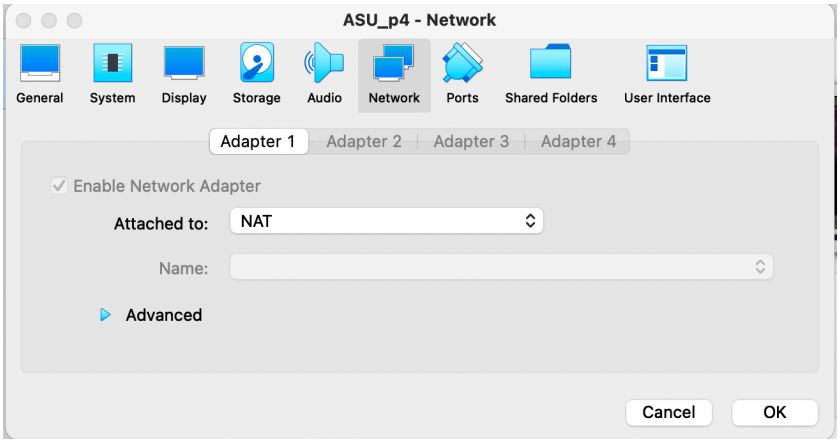


Figure 1 - Bridged network setup in VirtualBox

II. 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

III. Project Description

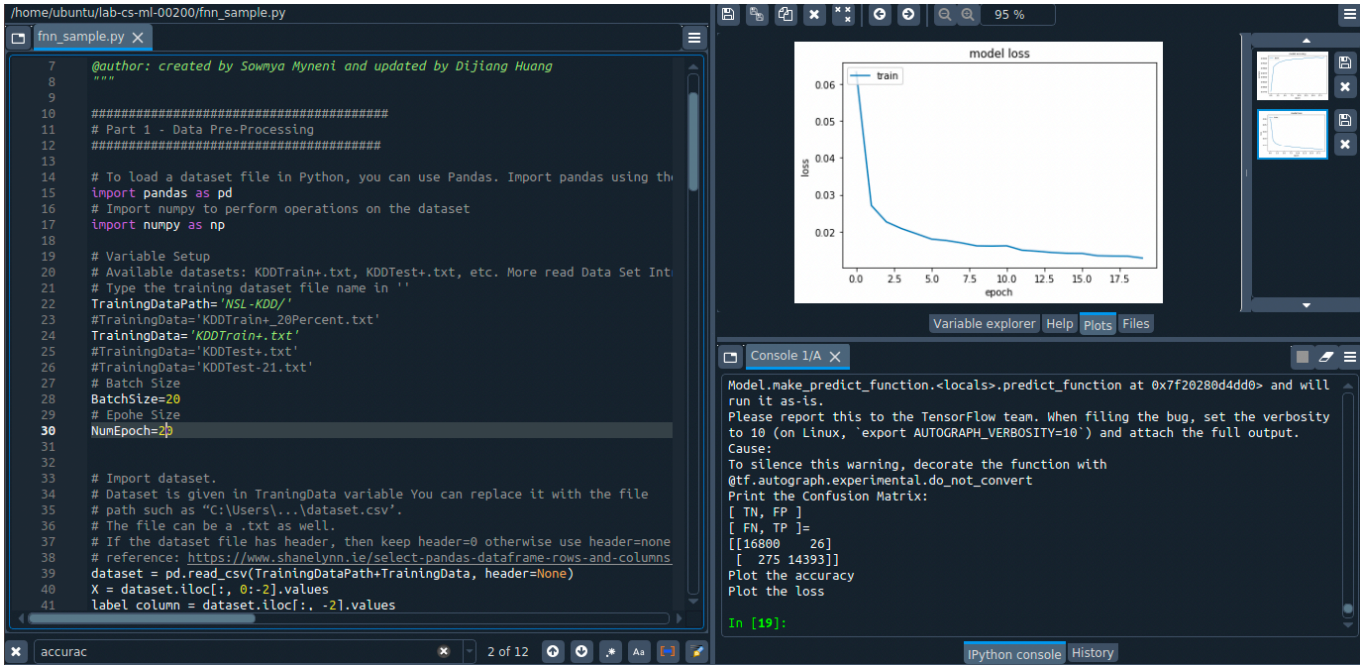
In this assignment, I have executed the various labs assignments, obtaining the proofs that they have been successfully completed.

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. FNN will use the well-labeled data to build FNN-based data pattern to differentiate normal and

abnormal network traffic. After training, we use similar data set with labeled data to validate the accuracy of the generate anomaly detection pattern.

1. Run the provided fnn sample.py python program with different NSL-KDD datasets as the input, and compare their detection accuracy

I have created the Python code to allow using the different training data by just changing a couple of comments, as shown below:



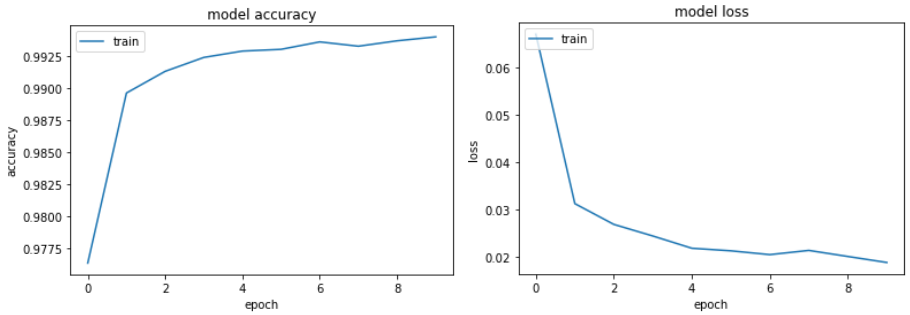
It is very easy in this way to collect the results and observe the differences from the various datasets. For instance, it is obvious to observe that accuracy increases with each epoch run:

```
Epoch 1/20
4724/4724 [==============================] - 17s 3ms/step - loss: 0.0806 - accuracy: 0.9741
Epoch 2/20
4724/4724 [==============================] - 11s 2ms/step - loss: 0.0374 - accuracy: 0.9809
Epoch 3/20
4724/4724 [==============================] - 15s 3ms/step - loss: 0.0308 - accuracy: 0.9878
Epoch 4/20
4724/4724 [==============================] - 16s 3ms/step - loss: 0.0251 - accuracy: 0.9918
Epoch 5/20
4724/4724 [==============================] - 11s 2ms/step - loss: 0.0219 - accuracy: 0.9924
Epoch 6/20
4724/4724 [==============================] - 9s 2ms/step - loss: 0.0208 - accuracy: 0.9931
Epoch 7/20
4724/4724 [==============================] - 13s 3ms/step - loss: 0.0186 - accuracy: 0.9937
Epoch 8/20
4724/4724 [==============================] - 17s 4ms/step - loss: 0.0177 - accuracy: 0.9941
Epoch 9/20
4724/4724 [==============================] - 19s 4ms/step - loss: 0.0172 - accuracy: 0.9941
Epoch 10/20
4724/4724 [==============================] - 10s 2ms/step - loss: 0.0167 - accuracy: 0.9943
Epoch 11/20
4724/4724 [==============================] - 15s 3ms/step - loss: 0.0162 - accuracy: 0.9945
Epoch 12/20
4724/4724 [==============================] - 23s 5ms/step - loss: 0.0162 - accuracy: 0.9946
Epoch 13/20
4150/4724 [=========================>....] - ETA: 1s - loss: 0.0159 - accuracy: 0.9948
```

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

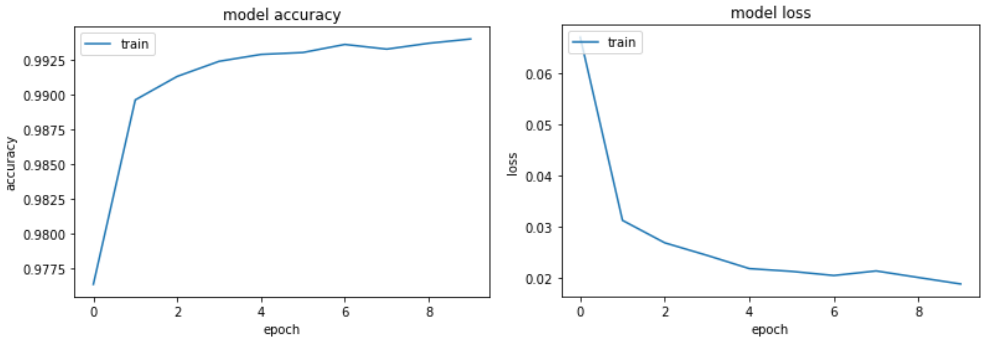
KDDTrain+: Loss 0.0221, Accuracy: 0.9912
Confusion Matrix: TN=16806, FP=20, FN=533, TP=14135

```
Loss [0,1]: 0.0221 Accuracy [0,1]: 0.9912
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x7f3d9d753a70> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function
at 0x7f3d9d753a70> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[16806    20]
 [  533 14135]]
```



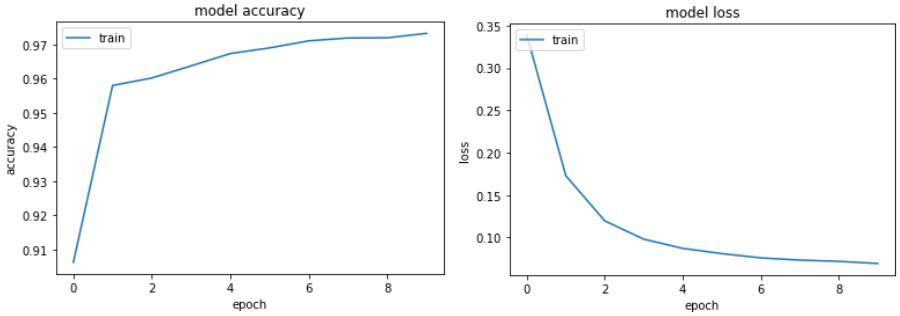
KDDTrain+20Percent: Loss 0.0319, Accuracy: 0.9835
Confusion Matrix: TN=3386, FP=1, FN=174, TP=2737

```
Loss [0,1]: 0.0319 Accuracy [0,1]: 0.9835
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x7f3d9c3f5290> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function
at 0x7f3d9c3f5290> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[3386    1]
 [ 174 2737]]
```



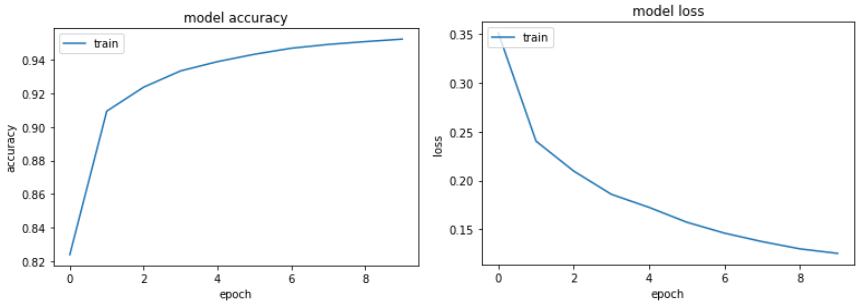
KDDTest+: Loss 0.0675, Accuracy: 0.9683
Confusion Matrix: TN=2486, FP=18, FN=356, TP=2776

```
Loss [0,1]: 0.0675 Accuracy [0,1]: 0.9683
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x7f3e04e45b00> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function
at 0x7f3e04e45b00> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[2486   18]
 [ 356 2776]]
```



KDDTest-21: Loss 0.1338, Accuracy: 0.9487
Confusion Matrix: TN=550, FP=16, FN=342, TP=2055

```
Loss [0,1]: 0.1338 Accuracy [0,1]: 0.9487
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x7f3e0efde9e0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function
at 0x7f3e0efde9e0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux,
`export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[ 550   16]
 [ 342 2055]]
```

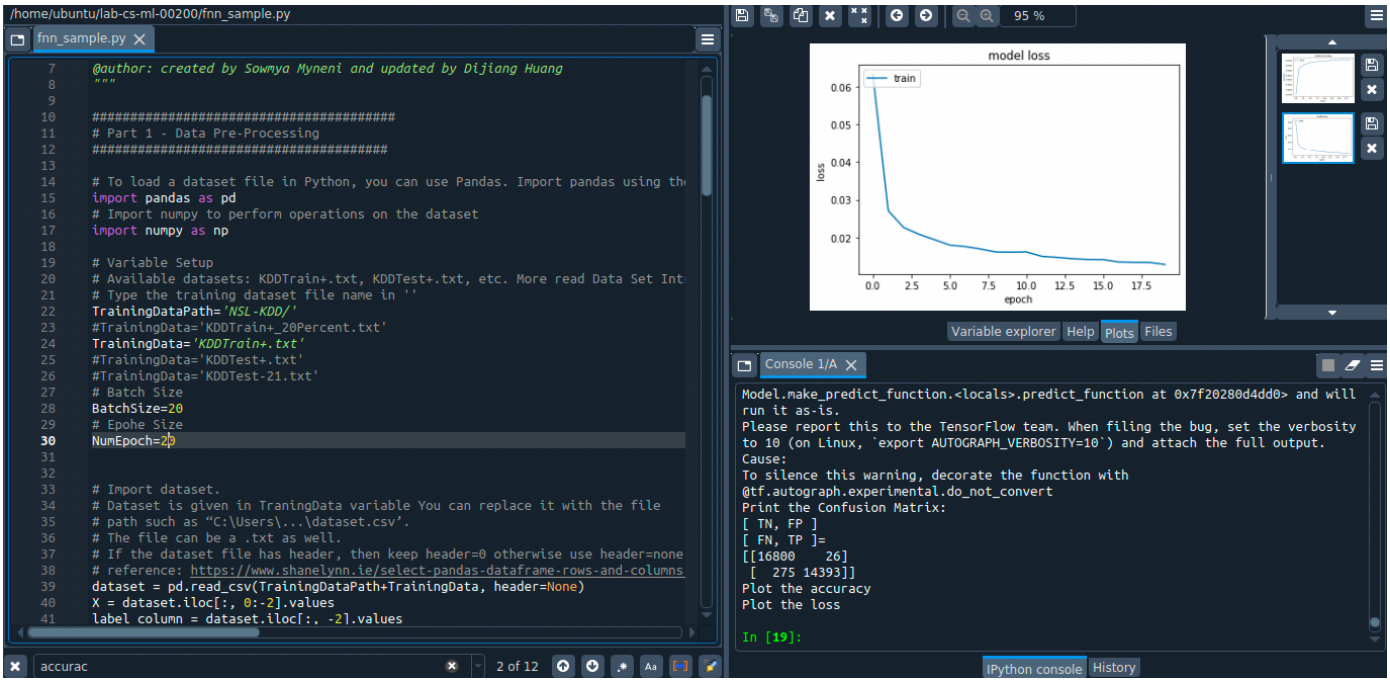


a. Comparing the datasets:

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	In this case a much bigger dataset (KDDTrain+) provides a superior accuracy and minor loss.
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	In this case a training dataset provides more accuracy and less loss since the test is smaller.
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	Again, the training dataset provides more accuracy and less loss since it has a bigger set of data.
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, 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.

2. Run the provided fnn sample.py python program with different one NSL-KDD dataset: KD- DTrain+.txt as the input and compare their detection accuracy with a different neural network setup. Explain your observations.

The fnn_sample.py has been modified as required, as shown below.




```
Loss [0,1]: 0.0132 Accuracy [0,1]: 0.9959
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[16795    31]
 [  257 14411]]
```

KDDTrain+.txt (1st run) vs. KDDTrain+.txt (2nd run)

KDDTrain+ 1st run: Loss 0.0221, Accuracy: 0.9912
Confusion Matrix: TN=16806, FP=20, FN=533, TP=14135

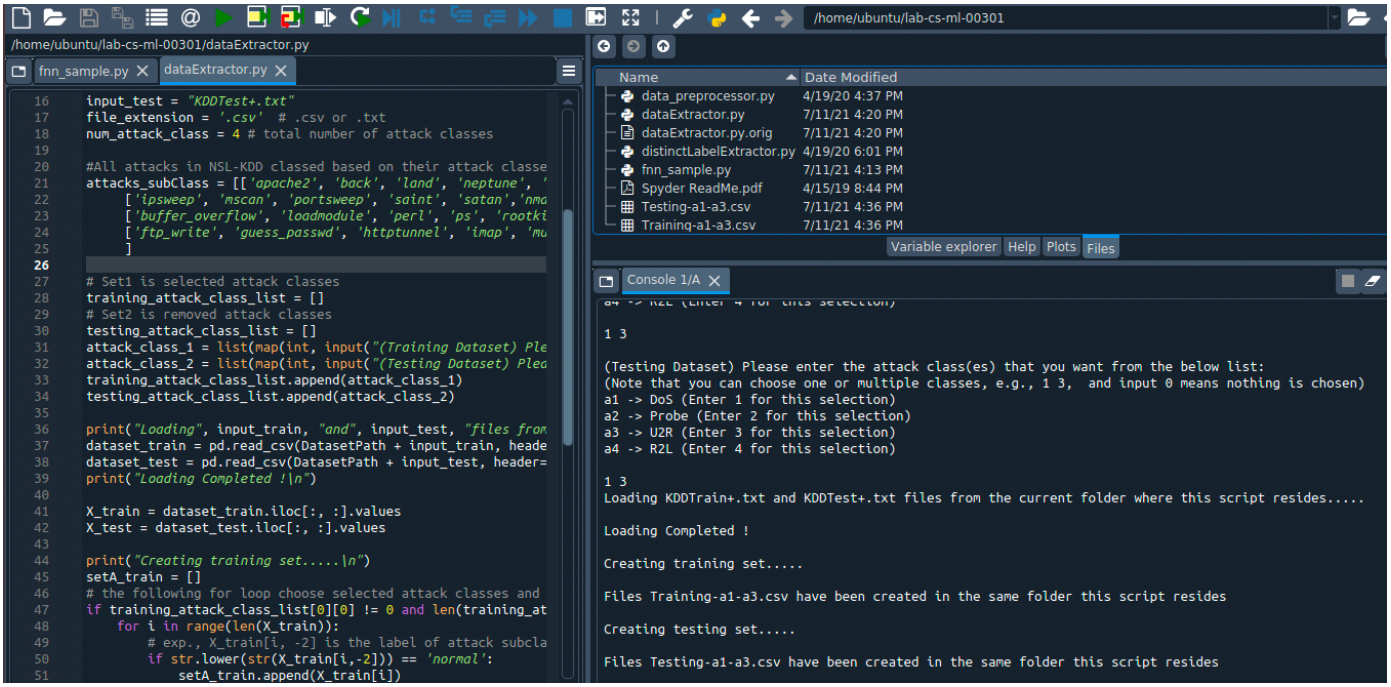
KDDTrain+ 2nd run: Loss 0.0132, Accuracy: 0.9959
Confusion Matrix: TN=16795, FP=31, FN=257, TP=14411

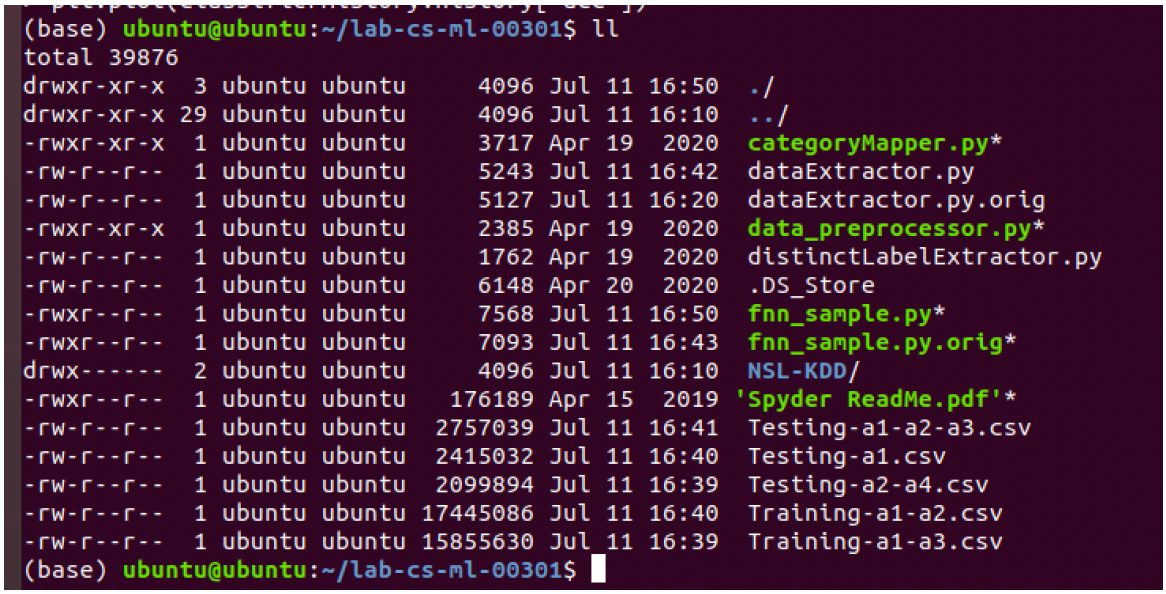
A run with an increased depth of the FNN (more layers) and increased epoch seems to increase the accuracy and diminish the loss – which is expected. The run with the increased value has ran much slower – it could be perceived although I have not measured precisely.
Interestingly enough, while the TN and TP have increased, so they are the FP – albeit the FN have diminished. So, surprisingly, it seems that not all the results have not improved. It would be interesting to investigate the reason for it (although it is out of scope for this lab).

In the second lab (CS-ML-00301), I use basic FNN model to create an anomaly detection model for network intrusion detection.

3. Task 2: Create Data Modules for Anomaly Detection

First of all, I create the datasets using DataExtractor.py. The datasets are all available on the GitHub repository.





4. Lab Assessment

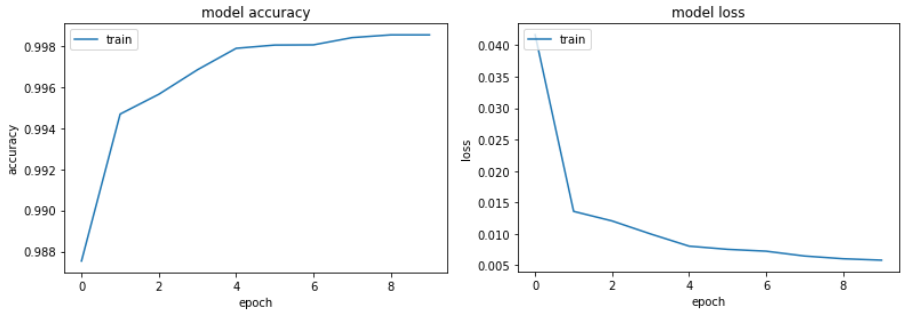
a. Using the lab screenshot feature to document ML running results only. Provide sufficient but concise explanations on the generated results for each given training/testing scenarios.

The followings are the results from the three scenarios.

a) ***Scenario A****: Loss=0.0052, Accuracy=0.9988, Confusion Matrix: TN=8701, FP=1010, FN=2502, TP=2804*

```
In [1]: runfile('/home/ubuntu/lab-cs-ml-00301/fnn_sample.py', wdir='/home/ubuntu/
lab-cs-ml-00301')

Please enter the scenario you wish to run - either a, b or c:a
2021-07-11 19:25:09.071681: I tensorflow/core/platform/cpu_feature_guard.cc:143]
Your CPU supports instructions that this TensorFlow binary was not compiled to use:
SSE4.1 SSE4.2 AVX AVX2
2021-07-11 19:25:09.091889: I tensorflow/core/platform/profile_utils/cpu_utils.cc:
102] CPU Frequency: 2592000000 Hz
2021-07-11 19:25:09.092085: I tensorflow/compiler/xla/service/service.cc:168] XLA
service 0x55aa45fab370 initialized for platform Host (this does not guarantee that
XLA will be used). Devices:
2021-07-11 19:25:09.092101: I tensorflow/compiler/xla/service/service.cc:176]
StreamExecutor device (0): Host, Default Version
2021-07-11 19:25:09.093911: I tensorflow/core/common_runtime/process_util.cc:147]
Creating new thread pool with default inter op setting: 2. Tune using
```

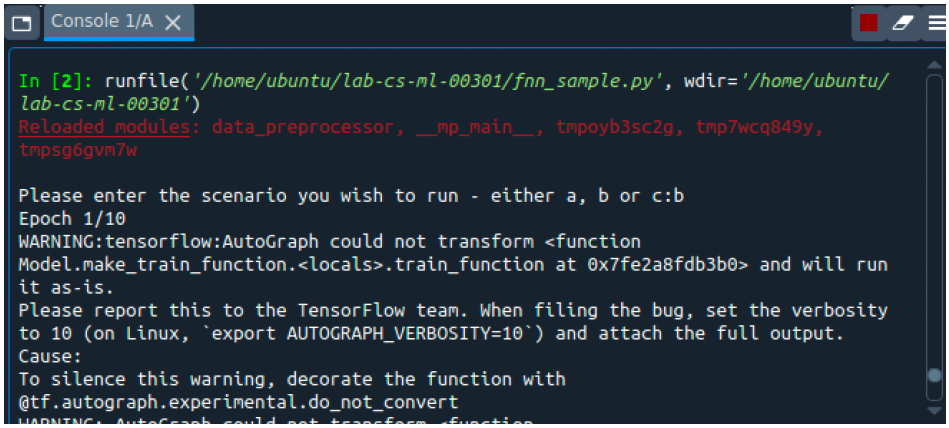
```
Epoch 10/10
11333/11333 [==============================] - 22s 2ms/step - loss: 0.0058 -
accuracy: 0.9985
3542/3542 [==============================] - 8s 2ms/step - loss: 0.0052 - accuracy:
0.9988
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0052 Accuracy [0,1]: 0.9988
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8701 1010]
 [2502 2804]]
Plot the accuracy
Plot the loss

In [2]:
```

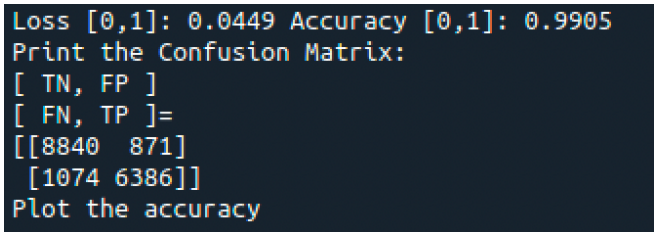


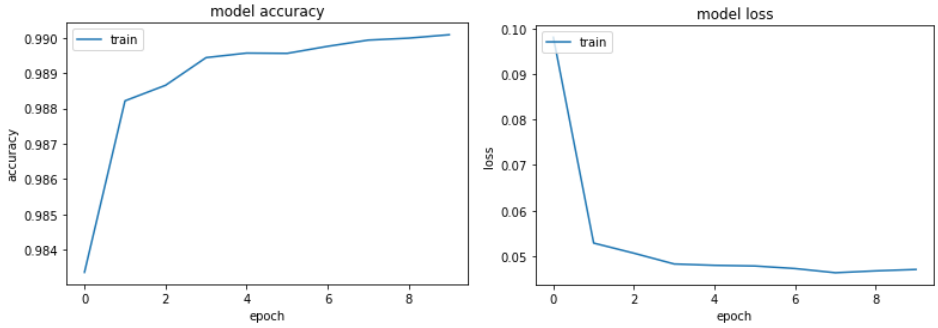
Explanation of Scenario A: this scenario uses "DoS" and "U2R" attacks as training and "Probing" and "R2L" attacks in the test cases. Basically, there is no overlap between the training and test cases, and therefore we would expect this scenario to be the one that is worst performing.

Although it is true that this Scenario produces the worst result in terms of Confusion Matrix – surprisingly enough, it is not the worst performer on the Loss and Accuracy values, meaning that the dataset used for training it is probably not very far from the test cases. This can likely be attributed to a certain level of overlapping between the behavior of the different attacks – for instance, a buffer overflow pattern observed amongst the U2R can also be seen during certain DoS attacks.

b) ***Scenario B****: Loss=0.0449, Accuracy=0.9905, Confusion Matrix: TN=8840, FP=871, FN=1074, TP=6386*

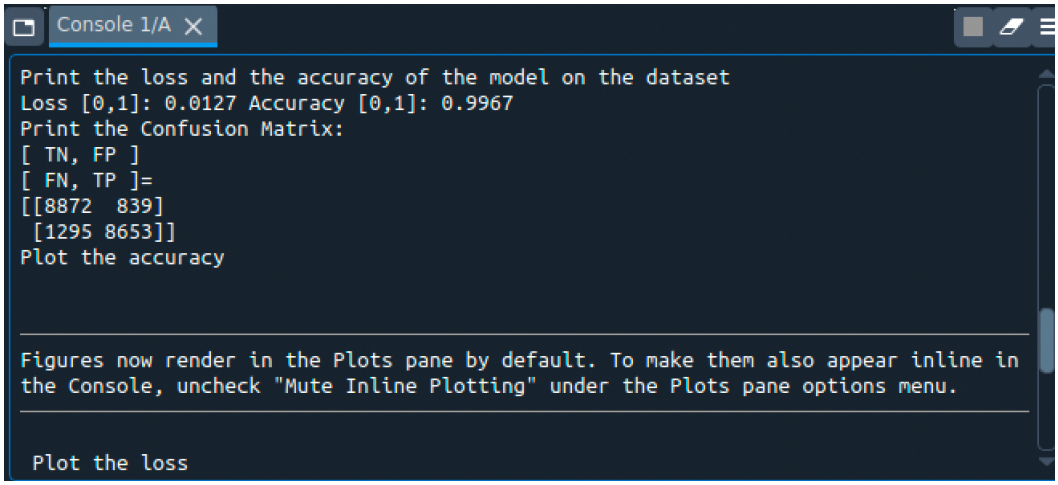


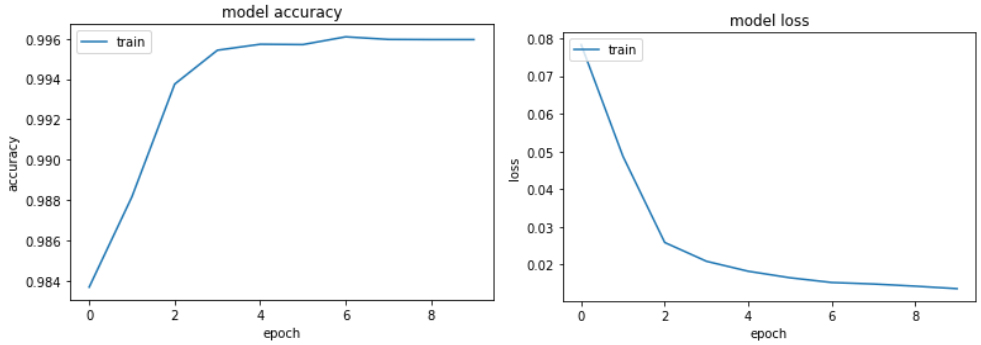




Explanation of Scenario B: this scenario uses "DoS" and "Probe" attacks as training only "DoS" attacks in the test cases. Basically, there is a complete overlap between the training and test cases, as the training includes all the attacks that will be found in the test, and therefore we would expect this scenario to be the one that is performing better.
What we observe instead, is that the Confusion Matrix is much better than Scenario A – where we detect as many as three times the True Positives – but with worst Loss and Accuracy. Furthermore, general results are inferior to Scenario C.
Most likely, since this scenario uses less data for the Test compared to Scenario C, it has less possibilities to perform better and detect more attacks, causing an overall worst performance.

c) ***Scenario C****: Loss=0.0127, Accuracy=0.9967, Confusion Matrix: TN=8872, FP=839, FN=1295, TP=8653*



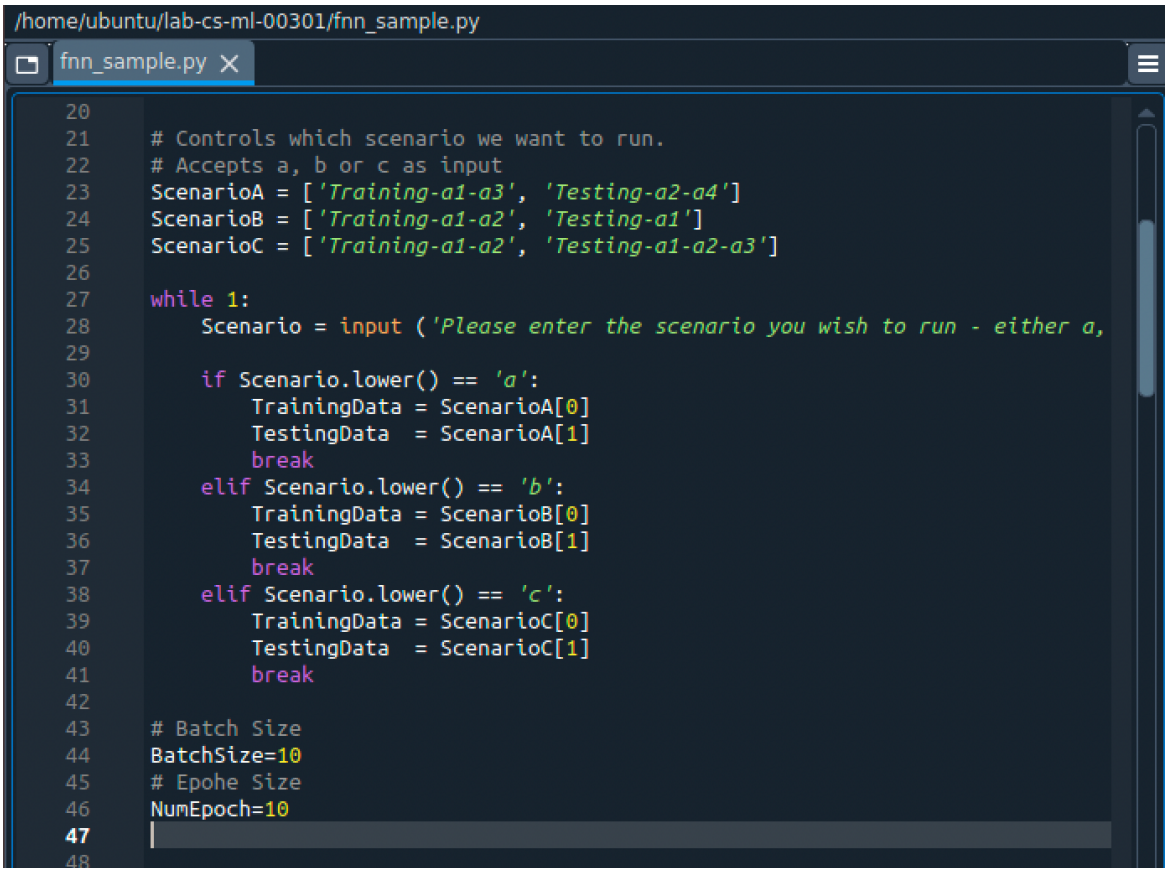


Explanation of Scenario C: this scenario uses "DoS" and "Probe" attacks as training (exactly like Scenario 2) and is presented with "DoS", "Probe" and "R2L" attacks in the test cases. This Scenario is the one that is fed with the biggest number of tests, and it can be therefore the reason why performances seem to be superior to the other Scenarios for what concerns the Confusion Matrix. However, it is inferior to Scenario A for what concerns Loss and Accuracy, which is to be attributed to the fact that there is a class of attacks ("R2L") for which it is not trained.

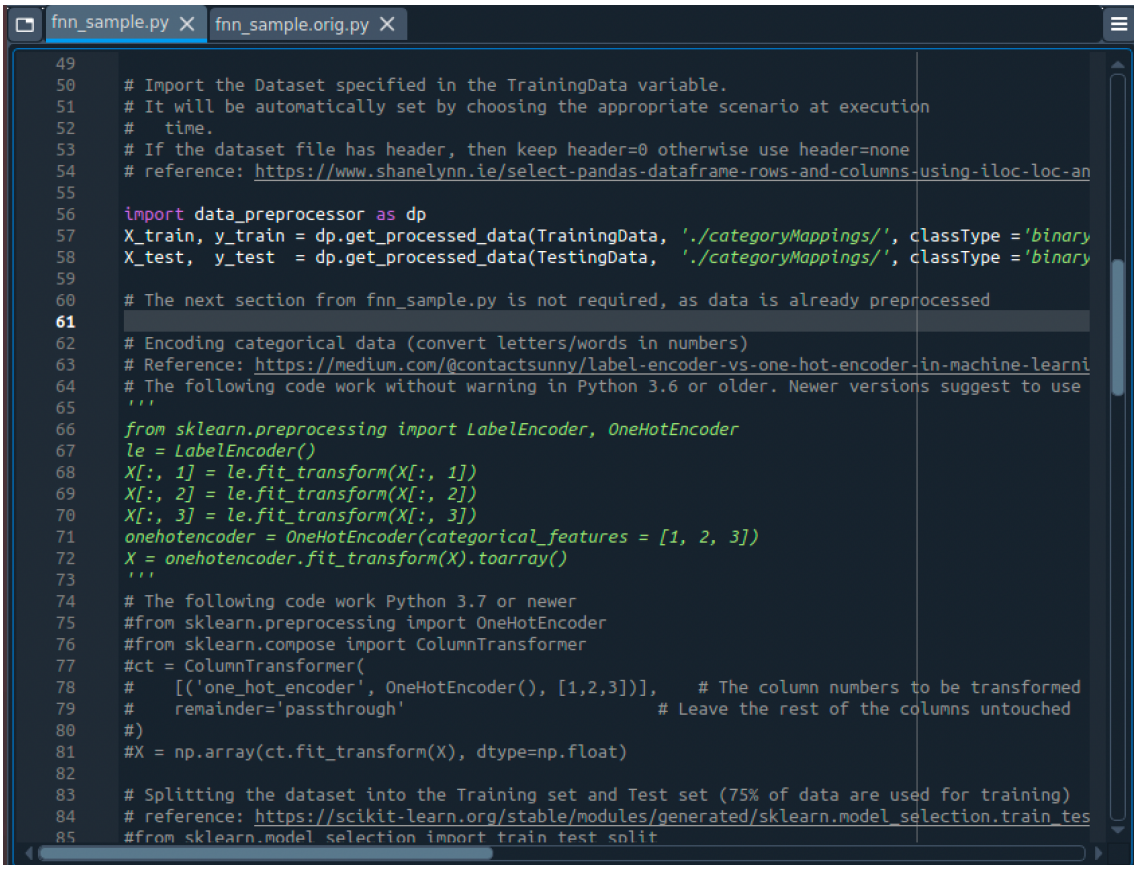
b. Submit the updated fnn simple.py and provide sufficient comments to illustrate your updated codes.

All the codes shown in the pictures below are available on GitHub, as well as in the provided ZIP file.

My first change is that I have modified fnn_sample.py so that it takes one of the three scenarios (A, B, C) as input. Selecting the scenario will automatically select the respective datasets.



From line 56 up to the implementation of the FNN, I have implemented the code that loads the two pre-processed set (training and test) and then commented out all the code that creates the “hot” split of test and training data, as this is not necessary.



From line 98 (after the comment “Part 2: Building FNN”), the code is basically unmodified.

5. Task 3: Anomaly detection analysis

1. The scenario producing the best results is Scenario C. As already described above, there seems to be a better outcome when the sets used in the Test are also present in the Training, which is unsurprising.
2. The average accuracy between Scenario A and Scenario C is 0.99775.
3. As noticed, when a Scenario is presented with attacks for which it has not been trained, it will perform worst in terms of Accuracy and Loss, as it to be expected. However, this does not automatically translate into more attacks blocked.
4. The R2L and U2L use similar patters – for instance they are mostly using TCP protocol as they must transmit an exploit, deposit a rootkit or being transported via HTTP.
   Another issue is determined by the fact that “U2R” and “R2L” attacks are numerically much inferior to “DoS” and “Probe” attacks, which reflects negatively on the training.

IV. 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

Project Report 4 - Machine Learning-Based Anomaly Detection Solutions.docx	https://github.com/markoer73/CSE-548/blob/main/Project%204%20-%20Machine%20Learning-Based%20Anomaly%20Detection%20Solutions/Project-Report-4%20-%20Machine%20Learning-Based%20Anomaly%20Detection%20Solutions.docx
fnn_sample.py	https://github.com/markoer73/CSE-548/blob/main/Project%204%20-%20Machine%20Learning-Based%20Anomaly%20Detection%20Solutions/CS-ML-00301/fnn_sample.py

V. References

Data preprocessing:

- https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
- 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://scikit-learn.org/stable/modules/preprocessing.html

Build ANN:

- 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/
- https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

Contents