

Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic Configuration Manual

MSc Research Project
Programme Name

Forename Surname
Student ID: 21176221

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Erick Ayala Rodríguez.....

Student ID: 21176221.....

Programme: MSc in Cloud Computing..... **Year:** 2023.....

Module: Research Project

Lecturer: Punit Gupta


Submission Due Date: 14/08/2023.....

Project Title: Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic.....

Word Count: 1602..... **Page Count:** 8 pages.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Erick Ayala Rodriguez.....


Date: 12/08/2023.....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic Configuration Manual

Erick Ayala Rodríguez
Student ID: 21176221

1 Hardware and tools

The present section will list the hardware, tools, and dependencies to set up the environment needed to execute all the tests done in the research.

The hardware utilized for the tests was implemented in two architectures, the on-premise architecture and the cloud architecture. The on-premise architecture has one AMD Ryzen 5 4600h¹ CPU with 12 cores and one Nvidia RTX 2060 mobile², for the cloud architecture was created an EC2 instance g3.8xlarge³ from AWS. Ubuntu 22.04.2 LTS is the operating system for the cloud architecture in the EC2 Instance and Windows 11 home for the on-premise architecture.

The main tools are Jupyter Notebook as the web-based interactive computing platform to create the machine learning algorithms and TensorFlow as the framework to manage the workloads across the hardware available. To configure the tools to make use of all the hardware available especially the GPUs is necessary to install all the dependencies with the correct versions to make the tools installed to work properly. The dependencies are described in the next section.

2 Dependencies

To do the correct use of the CPUs and GPUs available it is important to use versions that are compatible with the version of TensorFlow installed. Here is the list of versions of dependencies and drivers required for the test environments.

1. Python 3.9.17

¹ <https://www.amd.com/en/product/9086>

² <https://www.nvidia.com/en-gb/geforce/gaming-laptops/compare-20-series/>

³ <https://www.amazonaws.cn/en/ec2/instance-types/>

2. The Nvidia drivers version 535.86.05 for the cloud architecture which has two Nvidia Tesla M60 and for the on-premise architecture the drivers version 536.67 which has one Nvidia RTX 2060 mobile
3. CUDA libraries version 11.3.58
4. Miniconda for the creation of the TensorFlow virtual environments to allocate all the dependencies, in the present research the name of the virtual environment was “tf”
5. PIP installer to install all the necessary dependencies

3 Framework and libraries

As mentioned in Section 1, TensorFlow is implemented to manage the workloads and how are they distributed across the different CPUs and GPUs of the systems. The TensorFlow version installed is 2.12.1⁴ which is compatible with the CUDA version installed. It is important to have installed versions that are compatible because if not, the framework will work but it is not going to detect the GPU and in consequence, will not use it.

By default, TensorFlow will utilize the GPU to process data but to control how many GPUs should be used for was necessary to configure a list with the device IDs and the “tf.distribute.MirroredStrategy⁵” strategy to distribute the dataset in the devices configured. Figure 1 shows how was configured the list and Figure 2 the strategy configuration.

```
GPUS = ["GPU:0", "GPU:1"]
strategy = tf.distribute.MirroredStrategy( GPUS )
print('Number of devices: %d' % strategy.num_replicas_in_sync)
```

Figure 1: TensorFlow mirror strategy definition for multiple GPUs.

```
with strategy.scope():
    # Binary classification model
    model_binary = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_binary.shape[1], 1)),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=128, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=256, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
```

Figure 2: TensorFlow mirror strategy implemented in the CNN model.

It is worth to mention that these lines of code in the configuration for the on-premise architecture is not necessary because they have only one GPU and the Framework by default will make use of it.

⁴ https://www.tensorflow.org/install/pip#linux_1

⁵ https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy

To validate the correct use of the two GPUs can be used the command “nvidia-smi -l 2⁶” where the numeric value can be set as frequently as needed to refresh the information. Figure 3 shows the output using two GPUs in the cloud architecture.

```

Fri Aug 11 22:46:36 2023
+-----+
| NVIDIA-SMI 535.86.05                  Driver Version: 535.86.05          CUDA Version: 12.2                |
+-----+-----+-----+-----+-----+-----+
| GPU  Name           Persistence-M | Bus-Id              Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |      Memory-Usage   | GPU-Util  Compute M. |
|                                           |              Memory Usage   |      GPU-Util  Compute M. |
|-----+-----+-----+-----+-----+-----+-----+
|   0   Tesla M60             Off          | 00000000:00:1D.0  Off          |         0      Default |
| N/A   40C    P0              57W / 150W   | 7176MiB / 7680MiB |        38%      Default |
|                                           |              |
+-----+-----+-----+-----+-----+-----+
|   1   Tesla M60             Off          | 00000000:00:1E.0  Off          |         0      Default |
| N/A   38C    P0              52W / 150W   | 7174MiB / 7680MiB |        37%      Default |
|                                           |              |
+-----+-----+-----+-----+-----+-----+
| Processes:                               |
| GPU   GI    CI          PID   Type   Process name                               GPU Memory |
|      ID    ID                                   |             Usage   |
+-----+-----+-----+-----+-----+-----+
|   0   N/A  N/A           2250    C   ...buntu/miniconda3/envs/tf/bin/python   7173MiB |
|   1   N/A  N/A           2250    C   ...buntu/miniconda3/envs/tf/bin/python   7171MiB |
+-----+-----+-----+-----+-----+-----+

```

Figure 3: EC2 Instance x2 Nvidia Tesla M60 GPUs in usage.

To configure the use of the CPU is necessary to setup the “os” environmental variable and set the value “-1” to disable the use of GPU at the beginning of the code as shown in Figure 4, to then configure the number of threads to be split on the workload and control the number of cores to be used in the tests as shown on Figure 5.

```

import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

```

Figure 4: Disabling the use of GPUs.

```

# Number of threads
os.environ['TF_NUM_INTEROP_THREADS'] = '32'
os.environ['TF_NUM_INTRAOP_THREADS'] = '32'

```

Figure 5: Configuration of threads.

These configurations are needed in bout the on-premise and cloud architecture.

It is important to mention that these configurations need to be removed or commented if it is required to use the code with GPUs, if not TensorFlow will skip the GPUs for data

⁶ https://nvidia.custhelp.com/app/answers/detail/a_id/3751/~/useful-nvidia-smi-queries
⁷ <https://docs.python.org/3/library/os.html>

processing, and vice versa if the CPU configuration is needed, the code shown in Figure 1 must be eliminated or commented and the line number one in Figure 2 should be also commented or removed in combination with the tabular space to avoid errors.

For the configuration of the models where used different python libraries listed below:

- a) Numpy⁸: To work with tabular data from the UNSW-NB15 dataset (Moustafa, N., & Slay, J. 2015).
- b) Pandas⁹: To work with numerical data inside of the dataset.
- c) Keras API¹⁰: Required to build sequential and convolutional neural networks in the CNN and SNN models implemented in the research.
- d) tensorflow-estimator (2.12.0): To simplify the mechanics of machine learning tasks like training, evaluation, prediction, and export of the ML models.
- e) tensorboard (2.12.3): TensorBoard provides visualization tools to understand, debug, and optimize TensorFlow programs. It offers a suite of visualization tools to make it easier to understand, debug, and optimize TensorFlow programs.
- f) tensorflow-io-gcs-filesystem (0.32.0): This is an extension of TensorFlow that provides support for the Google Cloud Storage (GCS) filesystem.
- g) Keras (2.12.0): Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It allows for easy and fast prototyping and supports both convolutional networks and recurrent networks.
- h) scikit-learn (1.3.0): ML library for calculations and evaluations of how the models are performing.
- i) opt-einsum (3.3.0): This library can help to handle many operations required in the layers on neurons of the neural network models tested. It optimizes TensorFlow tensor contractions.

4 Machine learning models configuration

The configuration of the CNN model for the detection of network threads (binary classification) is shown in Figure 6 where it has three Conv1D filters, two Dense layers of neurons with dropout to prevent overfitting.

```
# Binary classification model
model_binary = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_binary.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=128, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=256, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

⁸ <https://numpy.org/install/>

⁹ https://pandas.pydata.org/docs/getting_started/install.html

¹⁰ https://keras.io/getting_started/

Figure 6: CNN model for binary classification.

Continuing with the configuration of the CNN model Figure 7 shows the code for the classification of network threats (multi-class classification) where it has one Conv1D filter, and one Dense layer of neurons with dropout to avoid overfitting.

```
# Multi-class classification model
model_multiclass = Sequential([
    Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train_multiclass.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(y_train_multiclass.shape[1], activation='softmax')
])
```

Figure 7: CNN model for multi-class classification.

It is worth mentioning that the model has fewer layers than the CNN binary classification because if it is added more neuron layers or filters are adding time to train the model but it is not adding much improvement in accuracy, recall, precision, or F1 score.

To train the CNN model was configured with 10 epochs and a batch size of 128 because even if the GPUs can handle all the data in parallel the model is not able to detect proper patterns and the accuracy goes down. To measure the time to train was implemented the “time” function. Figure 8 shows how these configurations were implemented in the code.

```
# Train the binary classification model
start_time_binary = time.time()
model_binary.fit(
    X_train_binary, y_train_binary, epochs=10, batch_size=128, validation_split=0.2, callbacks=[early_stopping_callback]
)
end_time_binary = time.time()

# Train the multi-class classification model
start_time_multiclass = time.time()
model_multiclass.fit(
    X_train_multiclass, y_train_multiclass, epochs=10, batch_size=128, validation_split=0.2, callbacks=[early_stopping_callback]
)
end_time_multiclass = time.time()

# Calculate training time for binary classification
training_time_binary = end_time_binary - start_time_binary
print("Binary Classification - Training Time:", training_time_binary, "seconds")

# Calculate training time for multi-class classification
training_time_multiclass = end_time_multiclass - start_time_multiclass
print("Multi-Class Classification - Training Time:", training_time_multiclass, "seconds")
```

Figure 8: CNN model training configuration and time measurement.

For the configuration of the SNN implemented three Neuron layers in decreasing order with relu and sigmode as activation methods. The training was configured with 10 epochs and the default batch size of 32 because the training times were low and also because the model showed to be more sensitive to changes and it is reflected in accuracy and precision. The code is shown in Figure 9.

```

with strategy.scope():
    model = Sequential()
    model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid')) # Binary classification output layer

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model using fit()
    start_time_binary = time.time()
    model.fit(X_train, y_train, epochs=10, verbose=1)
    end_time_binary = time.time()

# Calculate training time for binary classification
training_time_binary = end_time_binary - start_time_binary
print("Binary Classification - Training Time:", training_time_binary, "seconds")

```

Figure 9: SNN model build and training configuration binary classification.

For the configuration of the SNN model for the classification of network threats (multi-class classification) was removed one layer of neurons because is just adding time to train the model and does not get any improvement in the evaluation metrics. Figure 10 shows the code for the SNN in multi-class classification.

```

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))

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

# Train the model using fit()
start_time_multiclass = time.time()
model.fit(X_train, y_train, epochs=10, verbose=1)
end_time_multiclass = time.time()

# Calculate training time for multi-class classification
training_time_multiclass = end_time_multiclass - start_time_multiclass
print("Multi-Class Classification - Training Time:", training_time_multiclass, "seconds")

```

Figure 10: SNN model build and training configuration multi-class classification.

The configuration of the DTC model was implemented with a random state of 42 to provide randomness in finding the best features to predict the results. Figure 11 shows the configuration of the model.

```

with strategy.scope():
    # Train the decision tree model
    start_time_binary = time.time()
    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train_encoded, y_train)
    end_time_binary = time.time()

# Calculate training time for binary classification
training_time_binary = end_time_binary - start_time_binary
print("Binary Classification - Training Time:", training_time_binary, "seconds")

```


Figure 11: DTC model build and training configuration binary classification.

And finally, for the classification of network threats with the DTC model, the configurations follow the same structure as the binary classification. This is because the only difference is the target variables which change from the “label” which determines if the network packets represent a threat or not to the column “attack_cat” which contained the classification of the network threats. This is also applicable to all the ML models tested during the research.

5 Running the environment

To start Jupyter Notebook is needed to be activated in the environment created with miniconda with the command “conda activate tf” where “tf” is the name of the virtual environment. Then needs to be executed the command “jupyter notebook”. This will start Jupyter Notebook being accessible from the web browser.

To make the local host available when is running in the EC2 instance it is needed to create an SSH connection to the instance activate the conda environment and run the “jupyter notebook” command. Open a different terminal and create a tunnel connection with the command “ssh -i "x21176221_P4.pem" -L 8888:localhost:8888 ubuntu@ec2-3-253-95-225.eu-west-1.compute.amazonaws.com” (the command needs to be changed every time the EC2 instance restarts because the DNS name changes and the command will not be able to create the tunnel) setting the tunnel on the port 8888, and now the link is accessible in the web browser as shown in Figure 12.

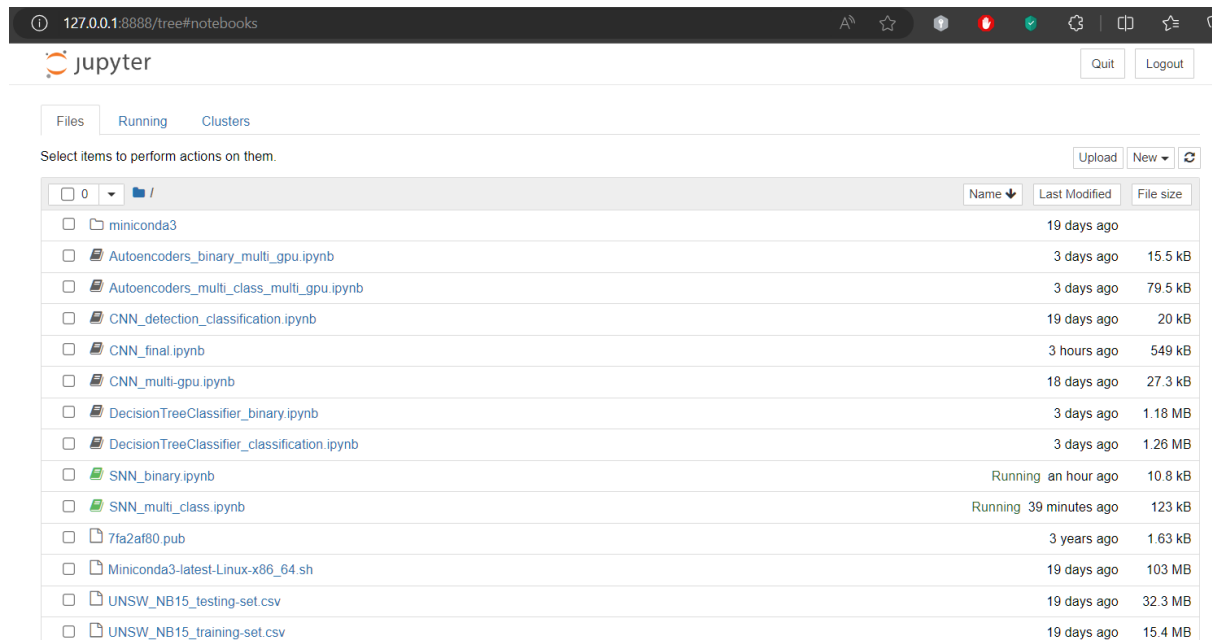


Figure 12: Jupyter Notebook.

References

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS), 1-6.