Towards Artificial Neural Network Based Intrusion Detection with Enhanced Hyperparameter Tuning

Andrei Nicolae Calugar*, Weizhi Meng* and Haijun Zhang†
*SPTAGE Lab, DTU Compute, Technical University of Denmark, Denmark
†Harbin Institute of Technology, Shenzhen, China

***Abstract*—Due to the development of complex communication paradigms and the rise in the number of inter-connected digital devices, intrusion detection system (IDS) has become one basic and important security mechanism to identify cyber intrusions and protect computer networks. Currently, various deep learning algorithms have been studied in intrusion detection to achieve a high detection rate, whereas the detection performance may be still dependent on specific datasets. To maintain the detection performance, parameter optimization is believed as an effective solution. Motivated by this observation, in this work, we propose a concise but effective hyperparameter tuning process to enhance the artificial neural network (ANN) based IDS. In the evaluation, we consider three ANN variants and four datasets. The experimental results indicate that our approach can outperform similar studies and typical learning algorithms.**

***Index Terms*—Intrusion Detection, Deep Learning, Hyperparameter Optimization, Artificial Neural Network, Tuner Search, Internet of Things.**

I. Introduction

In the past decade, there has been a significant expansion in computer networks and networked digital devices, such as Internet of Things (IoT) [7]. While how to preserve the security of information and communication has become a critical issue. For example, the number of attacks continued increasing as malicious intruders become more versatile across a broader network, even beyond embarrassing privacy leaks [1]. This could lead to being very costly and dangerous for businesses and governments as there is a high risk of leaking sensitive information, i.e., infrastructure could be sabotaged. According to a cyber-incident report from the Center for Strategic and International Studies (CSIS) [13], cyber attacks on government institutions and high tech companies have resulted in financial losses more than millions of dollars.

For protection, an intrusion detection system (IDS) is one of the most essential and important security mechanisms by continuously monitoring the network systems and detecting unwanted events based on the predefined security policies [10]. An IDS can be classified into two broad categories: signature-based detection and anomaly-based detection. For signature-based method, it works by comparing the monitored data with stored signatures or patterns [11]. This kind of approach is potent and accurate such as Suricata [15] and Snort [14], whereas it has a significant drawback, that is, it can only detect documented attacks that have been described and stored in the signature / pattern database.

978-1-6654-3540-6/22

On the other hand, anomaly-based detection method has to first create a model of the system's normal behavior and then examine the network data to spot any deviations [12]. The big merit is the capability of detecting unknown threats, while it may also produce a large number of false alarms. To help model normal behavior, numerous machine learning approaches have been investigated in the literature [6]. With the current advance in deep learning, artificial neural networks (ANNs), inspired by the simplification of neurons in a brain, have been widely studied about how to enhance the detection performance [16]. ANNs typically consist of three basic layers: the input layer, the hidden layers and the output layer. The nonlinear processing units can extract useful features and transform data, which is promising for intrusion detection.

Motivation and Contributions. However, the high accuracy and detection performance of deep learning still relies on the adequate quantity of labeled data. To improve the unstable performance, parameter optimization is a promising solution. In the literature, most studies are investigating how to perform hyperparameter optimization using ANNs. For example, Khan *et al.* [5] introduced how to apply ANNs to improve bug prediction accuracy through hyperparameter optimization. Hyperparameters are the parameters that characterize the model architecture, hence the method of finding the optimal model architecture is referred to as hyperparameter tuning. Motivated by this observation, in this work, we aim to design a concise while effective hyperparameter tuning process to enhance the artificial neural network based IDS. Our contributions can be summarized as below.

- We propose a hyperparameter tuning process with tuner search loop that can search for the optimal model architecture and enhance the performance of ANN-based IDS, including hyperparameter combination, model training and model evaluation. We further consider three ANN variants such as deep neural network (DNN), recurrent neural network (RNN), and convolutional neural network (CNN).
- In the evaluation, we consider four datasets to investigate the performance of our approach, such as KDDCUP'99, UNSW15, CICIDS2017 and CSECICIDS2018. In comparison with typical leaning algorithms and similar approaches, the experimental results demonstrate that our approach can help maintain an overall accuracy higher than 91%, by considering the complexity of different datasets and various features.

The paper structure is summarized as follows. Section II presents the background of artificial neural network with three variants, our proposed hyperparameter optimization process, and the implementation details. Section III presents the evaluation details, including four datasets, evaluation metrics and the results. Section IV discusses the limitations and the related work on deep learning in intrusion detection. Section V finally concludes our work.

II. Our Approach

A. Artificial Neural Network

Artificial Neural Networks (ANNs) are one of the most widely used machine learning techniques for intrusion detection and have been shown to yield successful results in identifying different malicious events.

An ANN can be defined as "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" [2]. The inspiration for neural networks comes from the structure of the mammalian cerebral cortex but with lesser complexity.

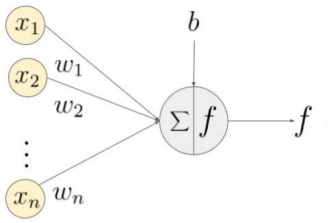

Fig. 1. Schematic of a neuron.

The processing elements of a neural network are called *neurons* and the way they work can be observed in Fig. 1. The input vector x of elements $x_1, x_2, ..., x_n$ has their corresponding weights $w_1, w_2, ..., w_n$. The weighted sum is performed and then a bias b and the activation function f can be applied to obtain the output.

The neurons are typically organized in layers that, when put together, can form the architecture of the neural network. The data is fed into the network via the input layer, which would not do any computation but feed one or more hidden layers where the actual calculation is done using a system of weighted connections [9]. Lastly, the results are given in the output layer. The way that the weights are modified uses a learning rule that comes as an algorithm, providing input to the network, and constructing a favored output.

1) Hyperparameter: The properties that govern the whole training process are called *hyperparameters*, which have a high impact on the training performance, as they control how the training algorithm behaves. In this work, we aim to investigate how to configure and optimize the hyperparameters in order to enhance the detection performance in the best case. Below is a list of variables that can be configured:

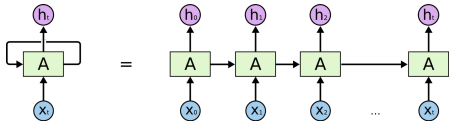

Fig. 2. A recurrent neural network unit.

- Learning rate – how quickly a network can update the parameters;
- Number of Epochs – the number of epochs through the whole training data;
- Batch size – the number of samples given to the network;
- Dropout – regularization technique that approximates the training and helps avoid over-fitting;
- Hidden layers – the layers between input and output layer;
- Activation function – introducing nonlinearity to models.

2) ANN variants: There are several variants for ANN in the literature. In this work, we focus on deep neural network (DNN), recurrent neural network (RNN), and convolutional neural network (CNN).

a) DNN: Deep neural networks often contain multiple hidden layers between the input and the output, making it being a higher level of complexity and more computationally demanding than a simple neural network. This is not given only by higher number of neurons but also the fact that this neuron may influence the computation by triggering different actions. Mathematically, DNN can be described as below:

$$O : \mathbf{R}^m \times \mathbf{R}^n$$

with m being the size of the input vector $x = x_1, x_2, ..., x_m$, and n being the size of the output vector $y = y_1, y_2, ..., y_n$.

b) RNN: A recurrent neural network is a type of neural network that checks the current input and considers the past state information to determine its output [9]. Fig. 2 presents the representation of such a recurrent neural network. Each green block represents a simple feed-forward neural network. Considering x of elements $x_1, x_2, ..., x_t$ as an input that maps them to hidden and output elements h of elements $h_1, h_2, ..., h_t$. For each time step, an input x_0 enters the network and produces an output h_0. In the next step, the input x_1 is taken into block A and additional input from the previous block. In this way, the neural network considers the context of the previous input to determine the next results.

The following Equations 1 and 2 are governing the computation at any time t, with the activation function $g()$, X input, H output, b the bias and W the weight matrix.

$$A_t = g \cdot (W_{ax} \cdot X_t + W_{aa} \cdot A_{t-1} + b_a) \quad (1)$$

$$H_t = g \cdot (W_{ya} \cdot A_t + b_y) \quad (2)$$

c) CNN: Convolutional neural networks are a typical deep learning method by extending traditional feed-forward neural networks. They are typically used as a model to process data with a grid pattern such as an image. This type of neural network has proven a substantial success in image recognition

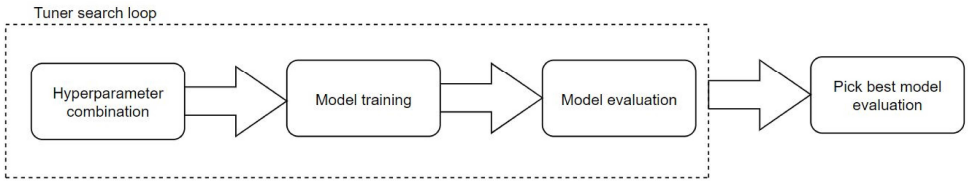

Fig. 3. Hyperparameter tuning process

```
cnn_dense_layers =[ 1, 2, 3,]
cnn_conv_layers = [ 1 , 2 ,3 ]
cnn_layer_sizes  = [64, 128 ]
# Build model for each possible value
for cnn_dense_layer in cnn_dense_layers:
    for cnn_layer_size in cnn_layer_sizes :
        for cnn_conv_layer in cnn_conv_layers :
            model = Sequential()
            model.add(Conv1D(cnn_layer_size ,3,input_shape=(78, 1)))
            cnn3.add(MaxPooling1D((2, 2)))
            model.add(LeakyReLU(alpha=0.1))
            model.add(Dropout(0.4))
        for l in range(cnn_conv_layer - 1):
            model.add(Conv1D(cnn_layer_size ,3))
            cnn3.add(MaxPooling1D((2, 2)))
            model.add(LeakyReLU(alpha=0.1))
            model.add(Dropout(0.4))
        model.add(Flatten())
        for l in range(cnn_dense_layer):
            model.add(Dense(cnn_layer_size, input_dim=78)
            model.add(LeakyReLU(alpha=0.1))

        model.add(Dense(15, activation='softmax'))
        model.compile(loss ='categorical_crossentropy', optimizer= 'adam',
            metrics = ['accuracy'])
        model.fit(features_train,labels_train,epochs=25,batch_size = 1000)
        # Print model
        print(NAME ="Number of conv layers-{}-Number of nodes-{}-number of
            dense layers-{}.".format(cnn_conv_layer, cnn_layer_size,
            cnn_dense_layer))
        # Evaluating model accuracy.
        model.evaluate(features_test , labels_test)
```

Fig. 4. Grid search tuning method

```
from tensorflow.keras.callbacks import TensorBoard
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
    histogram_freq=1)

model.fit(features_train,
          labels_train,
          epochs=30,
          validation_data=(features_validation, labels_validation),
          callbacks=[tensorboard_callback])
```

Fig. 5. Implementation with TensorBoard

and classification, speech processing application, and self-driving cars and robotics applications [18].

In a traditional CNN, a three-dimensional input (x rows, y columns, z depths) is required [18]. For example, an image that has x width, y height and z channels. In this work, we study the one-dimensional input data as a three-dimensional picture that has one channel and the height is also one. For this purpose, we convert the dataset into the input format of one-dimensional convolution architecture.

The architecture of a CNN is comprised of several building blocks such as *convolution layer*, *pooling layers* and *fully connected layers*. After the input is converted into a three-dimensional form, it will be fed into the convolution layer. This layer is a fundamental component of CNN that is in charge of feature extraction. A combination of linear and nonlinear operations is carried out, namely convolution operation and activation function.

B. Our Proposed Hyperparameter Optimization

Intuitively, when deploying a machine learning algorithm into intrusion detection, the model settings are usually not optimized from the beginning. Ideally, the optimization process should be automatically run by an algorithm in order to search for the optimal model architecture. Hyperparameters are the parameters that characterize the model architecture.

The types of parameters present in a machine learning model can be divided into trainable parameters that the algorithm learns through testing. The weights of a neural network, for example, are trainable parameters and hyperparameters that must be set before initiating the learning process. Some critical metrics include the learning rate, the number of hidden layers, and the number of nodes in a dense layer.

Fig. 3 describes the steps of our proposed tuning process (with a tuner search loop): hyperparameter combination, model training, model evaluation, and the best model determination. It is observed that the tuner search loop runs each time with a different hyperparameter selection. The training process is completed with a set of parameters on each iteration, after which some evaluation metrics are computed. Finally, possible combinations are discovered, and the models with the best results can be output. Fig. 4 shows an example by detailing the process as a grid search. In the example, three hyperparameters are chosen for a convolutional neural network (CNN) model: the number of dense layers, the number of convolutional layers, and size of the layers. The process is straightforward, the accuracy of the trained model is determined after each run, and the optimal combination of parameters would be maintained till the end.

C. Implementation with TensorBoard

TensorBoard [19] is a powerful optimization tool that can serve as a user interface for visualizing graphs and accessing additional resources to understand and debug models. It can calculate and visualize the workflows of a machine learning algorithm. It enables tracking metrics such as loss and precision, the analysis of concept graphs, and the embedding of projects in lower-dimensional spaces among other things. A key feature of TensorBoard is the capability of updating metrics constantly. In this case, the time required for training and adjusting a model can be shortened.

In this work, we therefore implemented our proposed hyperparameters optimization method based on TensorBoard, as illustrated in Fig. 5. Firstly, the library needs to be imported from $tensorflow.keras.callbacks$, and then create and compile a model. The next step is to start initialization of a log_dir variable with the location where the logs are saved and a time-stamp followed by a $tensorboard\_callback$ variable that takes the log_dir, and to set $histogram\_freq$ to one (allowing histogram computation per one epoch). Finally, we pass this to Keras's $Model.fit()$ together with the rest of the parameters.

III. Evaluation

To explore the performance of our approach, we conduct an evaluation on four datasets, and consider some similar studies and typical learning algorithms in the comparison.

A. Datasets

In the evaluation, we consider four publicly known datasets: KDDCUP'99, UNSW15, CICIDS2017 and CSECICIDS2018.

- **KDDCUP'99.** This is one of the most known and widely used datasets for intrusion detection applications. It is a collection of data transferred from a virtual environment on a typical US Air Force Local Area Network (LAN). It contains around five million single connection records where each has a number of 41 features [20].
- **UNSW15.** This dataset is composed of regular and malicious network traffic data in a packet-based format using the IXIA Perfect Storm tool. It captured a large amount of raw network traffic data, and after processing that data, 49 features were extracted, and a number of 257,673 records were stored [21].
- **CICIDS2017.** This dataset contains up-to-date common network attacks combined with normal network behavior. The data capturing period spammed a total of five days: only benign traffic data was recorded on Monday, while the following days having up to 14 types of attacks, including brute-force attack, denial of service (DoS) attack, Heartbleed, distributed denial of service (DDoS), network infiltration attacks, port scanning, botnet, and web attacks. Overall, it contains 3,119,345 records with a total of 83 features [22].
- **CSECICIDS2018.** This dataset is an extended version of CICIDS2017. The main difference is that it provides ten days of traffic from 18 February 2018 to 2 March 2018. The 14 attack types are the same as CICIDS2017. Overall, it contains approximately 15 million instances of network traffic with nine files that have 79 features and one file with 83 features [23].

B. Evaluation Measures

There are four possible primary outcomes of detection. The number of positive samples in the test dataset is denoted by TP. FP denotes the number of samples that are potentially negative but are counted as positive. The number of negative samples measured in the test dataset is denoted by TN. FN denotes the number of test samples that are currently positive but are counted as negative samples by the model. To measure the performance, we adopted the following metrics:

- **Accuracy** is defined as the proportion between correctly predicted observations and the total number of observations.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- The **precision** of a prediction is defined as the ratio of correctly predicted positive observations to all predicted positive observations.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

TABLE I
HYPERPARAMETER CONFIGURATIONS/RANGES USED IN THE TUNING PROCESS.

Model	**DNN**	**CNN**	**RNN**
Number of layers	1,2,3	1,2,3	1,2,3
Number of nodes	256,128,64	64,64,64	32,32,32
Layer activations	ReLU	LeakyReLU	ReLU
Output layer activation	softmax/sigmoid	softmax/sigmoid	softmax/sigmoid
Loss function	cross-entropy	cross-entropy	cross-entropy
Layer dropout rates	0.1	0.3	0.1 - 0.4
Batch size	64 - 128	256 or 512	512 - 1024
Number of epochs	25 - 30	25 - 30	25 - 30
Optimizer	Adam	Adam	Adam

- The **recall** ratio is the number of correctly estimated positive observations divided by the total number of observations in the actual class.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

- The **F1-Score** is calculated by averaging Precision and Recall.

$$Recall = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

C. Experimental Result

In this work, we used the hyperparameter values recorded in the literature as a reference for defining the range of random search. These values can serve as a baseline for the first model. Thus, the final set of hyperparameters for each model on each dataset can be determined after applying our approach. Table I describes the final hyperparameter configurations based on the highest accuracy on the validation set (40% of each dataset by random selection).

To validate the obtained results, we adopted the 5-fold cross validation. The detection performance results are shown in each Table II, Table III, Table IV and Table V. We also compared the detection performance with and without our hyperparameter optimization. Below are the main observations.

- By comparing the accuracy with and without our hyperparameter optimization, it is observed that our approach could greatly enhance the detection performance across all datasets. For example, our approach could generally improve the accuracy by 5% according to Table II.

TABLE II
PERFORMANCE RESULTS FOR KDDCUP'99.

Model	Acc. (Org)	Acc. (After)	Precision (After)	Recall (After)	F1-score (After)
DNN1	94.2323	99.9588	99.9570	99.9589	99.9577
DNN2	95.7238	99.9547	99.9527	99.9548	99.9527
DNN3	96.6377	99.9614	99.9595	99.9615	99.9598
CNN1	96.7328	99.9406	99.9355	99.9406	99.9322
CNN2	95.3342	99.9609	99.9583	99.9609	99.9586
CNN3	94.2763	99.9059	99.9063	99.9059	99.9036
RNN1	95.2883	99.9812	99.9799	99.9812	99.9805
RNN2	96.2388	99.9755	99.9746	99.9756	99.9746
RNN3	96.4434	99.9843	99.9838	99.9843	99.9840

TABLE III
PERFORMANCE RESULTS FOR UNSW15.

Model	Acc. (Org)	Acc. (After)	Precision (After)	Recall (After)	F1-score (After)
DNN1	95.7783	99.6052	99.6080	99.6053	99.6053
DNN2	94.8892	98.8279	98.8546	98.8280	98.8279
DNN3	94.7236	98.9433	98.9650	98.9434	98.9433
CNN1	95.6621	99.6660	99.6677	99.6660	99.6660
CNN2	95.8827	99.6644	99.6662	99.6645	99.6645
CNN3	96.2366	99.9423	99.9424	99.9423	99.9423
RNN1	92.1873	96.9120	97.0920	96.9121	96.9093
RNN2	94.6621	99.9954	99.9954	99.9954	99.9954
RNN3	94.2133	99.0860	99.1009	99.0861	99.0860

TABLE IV
PERFORMANCE RESULTS FOR CICIDS2017.

Model	Acc. (Org)	Acc. (After)	Precision (After)	Recall (After)	F1-score (After)
DNN1	91.2833	95.8991	95.9557	95.8991	95.4269
DNN2	88.7723	93.3076	93.7944	93.3077	92.7291
DNN3	90.8722	93.7170	94.1924	93.7170	93.1555
CNN1	91.1299	96.7281	96.6678	96.7281	96.5661
CNN2	90.8827	96.9331	96.8102	96.9332	96.4291
CNN3	92.3772	98.4399	98.3506	98.4400	98.3589
RNN1	93.7263	99.6700	99.6603	99.6701	99.6329
RNN2	94.5524	99.6386	99.5815	99.6387	99.6037
RNN3	95.2311	99.6437	99.6594	99.6438	99.6032

- As compared to the DNN and CNN models, RNN models could reach a higher accuracy. For example, in Table IV, RNN could reach an accuracy over 99% while DNN and CNN could provide an accuracy rate of 93%-96%.
- As shown in Table IV and Table V, the detection performance on CSECICIDS2018 was slightly worse than the same models applied on CICIDS2017. One possible reason for this could be that the training and validation data may be in the order of millions, and in some cases, it is harder for the model to classify correctly.

Comparison with similar studies. It is worth noting that a direct comparison with similar studies is often difficult, as most research studies do not detail the pre-processing steps performed on datasets, and the experimental platforms with implementation are different in most cases. While as a reference, in this work, we still considered some similar approaches (i.e., applying deep learning for IDS) and typical learning algorithms in a comparison. As shown in Table VI, it is easily noticed that our approach could provide and maintain a better detection rate compared with three recent similar studies and typical learning algorithms (e.g., LogisticRegression, AdaBoost, GaussianNB)

TABLE V
PERFORMANCE RESULTS FOR CSECICIDS2018.

Model	Acc. (Org)	Acc. (After)	Precision (After)	Recall (After)	F1-score (After)
DNN1	81.7733	90.4057	90.9670	90.4058	88.8832
DNN2	80.6237	89.6870	90.4469	89.6871	87.9723
DNN3	84.8822	92.8383	92.8970	92.8383	92.0008
CNN1	73.5682	81.6705	75.6440	81.6706	77.3512
CNN2	85.8374	92.1221	90.4862	92.1222	90.3389
CNN3	90.7263	95.2052	94.5229	95.2053	94.0680
RNN1	82.6399	88.5359	89.6032	88.5360	86.5132
RNN2	91.7233	96.2413	95.8368	96.2413	95.4874
RNN3	89.7822	96.2692	95.5717	96.2693	95.2236

TABLE VI
DETECTION ACCURACY COMPARISON AMONG TYPICAL LEARNING ALGORITHMS AND SIMILAR STUDIES.

Detection Approach	KDD'99	UNSW15	CICIDS2017
Vinayakumar et al. [17] in 2019	0.930	0.763	0.944
Liu et al. [8] in 2021	0.807	-	-
Gupta et al. [3] in 2022	0.780	-	0.880
Our work	0.990	0.947	0.937
LogisticRegression	0.978	0.811	0.883
AdaBoost	0.975	0.882	0.796
GaussianNB	0.663	0.677	0.701

on these datasets.

IV. LIMITATIONS AND RELATED WORK

1) Limitation and Discussion: Our work is still developing at an early stage, and some challenges are considered for future work. a) This work considers four datasets, while some more datasets can be considered to analyze and validate the results, such as CAIDA dataset. b) There are many relevant studies on designing an IDS based on deep learning and ANNs, our future work has to consider more relevant studies in the comparison.

2) Related Work: In the literature, deep learning has been widely studied in intrusion detection. For instance, Vinayakumar *et al.* [17] studied how to develop a flexible and effective IDS to detect cyber-intrusions based on DNN. They explored the optimal network parameters and the network topologies for DNNs with the method: that is, all the experiments of DNNs should run till 1,000 epochs with the learning rate varying in the range [0.01-0.5] on KDD'99 dataset. Liu *et al.* [8] introduced a Difficult Set Sampling Technique (DSSTE) algorithm to tackle the class imbalance problem and enhance the performance of deep learning in IDS. Gupta *et al.* [3] introduced CSE-IDS, a three-layer IDS, based on Cost-Sensitive Deep Learning and Ensemble algorithms. More related research on deep learning / artificial neural networks and relevant applications in IDS can refer to several surveys [4], [16].

V. CONCLUSION

Artificial neural networks have been widely studied in intrusion detection, by transferring data across numerous hidden layers and learning the abstract and high-dimensional attribute representation of the data. The ideal parameters and typologies for the neural network models can be found using hyperparameter selection approaches. In this work, we propose a hyperparameter tuning process with tuner search loop to optimize the hyperparameters for three variants of ANNs including DNN, RNN and CNN. To investigate the performance, we considered four datasets in the evaluation such as KDDCUP'99, UNSW15, CICIDS2017 and CSECICIDS2018. As compared with similar studies and several typical learning algorithms, our approach was shown to greatly enhance the detection performance across different datasets, i.e., maintaining an overall accuracy higher than 91% by averaging all dataset-results.

ACKNOWLEDGMENT

This research was partially supported by H2020 DataVaults.

REFERENCES

[1] Safeguarding the Internet of Things. [Online] https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Risk/gx-ra-safeguarding\%20the\%20IoT.pdf
[2] M. Caudill, "Neural Networks Primer, Part I," *AI Expert*, 2(12), pp 46-52, 1987.
[3] N. Gupta, V. Jindal, and P. Bedi, "CSE-IDS: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems," *Comput. Secur.* 112, 102499, 2022
[4] D. Gumusbas, T. Yildirim, A. Genovese, and F. Scotti, "A Comprehensive Survey of Databases and Deep Learning Methods for Cybersecurity and Intrusion Detection Systems," *IEEE Syst. J.* 15(2), pp. 1717-1731, 2021.
[5] F. Khan, S. Kanwal, S. Alamri, and B. Mumtaz, "Hyper-Parameter Optimization of Classifiers, Using an Artificial Immune Network and Its Application to Software Bug Prediction," *IEEE Access* 8, pp. 20954-20964, 2020.
[6] H. Mliki, A.H. Kaceam, and L. Chaari, "A Comprehensive Survey on Intrusion Detection based Machine Learning for IoT Networks," *EAI Endorsed Trans. Security Safety* 8(29), e3, 2021.
[7] L. Liu, Y. Wang, W. Meng, Z. Xu, W. Gao, and Z. Ma, "Towards Efficient and Energy-aware Query Processing for Industrial Internet of Things," *Peer-to-Peer Networking and Applications*, vol. 14, pp. 3895-3914, 2021.
[8] L. Liu, P. Wang, J. Lin, and L. Liu, "Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning," *IEEE Access* 9, pp. 7550-7563, 2021.
[9] V.S. Lalapura, J. Amudha, and H.S. Satheesh, "Recurrent Neural Networks for Edge Intelligence: A Survey," *ACM Comput. Surv.* 54(4), pp. 91:1-91:38, 2021.
[10] W. Meng, W. Li, and L.F. Kwok, "Towards Effective Trust-based Packet Filtering in Collaborative Network Environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 233-245, 2017.
[11] W. Meng, W. Li, and L.F. Kwok, "EFM: Enhancing the Performance of Signature-based Network Intrusion Detection Systems Using Enhanced Filter Mechanism," *Computers & Security*, vol. 43, pp. 189-204, 2014.
[12] Y. Meng, "The practice on using machine learning for network anomaly intrusion detection," *In: Proc. the 2011 International Conference on Machine Learning and Cybernetics*, pp. 576-581, 2011.
[13] Significant Cyber Incidents. [Online]. https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents
[14] M. Roesch, "Snort Lightweight Intrusion Detection for Networks," *In: Proc. the 13th Conference on Systems Administration (LISA)*, pp. 229-238, 1999.
[15] Suricata. https://suricata.io/
[16] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw. Comput. Appl.* 169, 102767, 2020.
[17] R. Vinayakumar, M. Alazab, K.P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Access* 7, pp. 41525-41550, 2019.
[18] S. Mittal, A survey of FPGA-based accelerators for convolutional neural networks. *Neural Comput. Appl.* 32(4), pp. 1109-1139, 2020.
[19] TensorBorad. https://www.tensorflow.org/tensorboard
[20] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/
[21] UNSW15. https://research.unsw.edu.au/projects/unsw-nb15-dataset
[22] CICIDS2017. https://www.unb.ca/cic/datasets/ids-2017.html
[23] CSECICIDS2018. https://www.unb.ca/cic/datasets/ids-2018.html