10th IEEE International Conference on Information Technology (ICIT), Jordan, July 2021

# Network Intrusion Detection on the IoT Edge Using Adversarial Autoencoders

Fadi Aloul, Imran Zualkernan, Nada Abdalgawad, Lana Hussain, Dara Sakhnini
Department of Computer Science & Engineering
American University of Sharjah, UAE
{faloul, izualkernan, g00068826, g00071496, g00068368}@aus.edu

***Abstract*— Network intrusion detection systems have received a lot of attention in the computer security literature. As the number of IoT devices grows exponentially, intrusion detection on the back-end servers or indeed even the fog will become intractable. Consequently, there is a need to move intrusion detection closer to the IoT edge. Doing so will have a significant impact on the network as well as the compute required on the server-side. In this paper, we show how deep learning can be used to build state-of-the intrusion detection algorithms that can be executed on small routers near the IoT edge. Adversarial autoencoders with the K nearest neighbor algorithm were trained on the NSL-KDD intrusion data set to yield state-of-the-art results. The model had an accuracy of 99.991% and an F1-Score of 0.9990. On a Raspberry PI 3B (RPI) device, using TensorFlow Lite, the model achieved an average per-packet latency of less than 16ms which is sufficient for many IoT sensors on the edge giving a worst-case bandwidth of 3kibts/second.**

***Keywords—Network Intrusion Detection, Adversarial Autoencoder, K Nearest Neighbor, SMOTE, IoT, edge computing.***

## I. Introduction

Cyberattacks pose a significant threat in the digital world today and effective cybersecurity solutions are needed to detect such cyberattacks. Fortunately, with the continuous advancements in machine learning and especially deep learning, automated classification techniques to detect intruders have become more and more accurate. Internet of Things (IoT) represents unique challenges to intrusion detection because given the billions of IoT devices, intrusion detection in the back-end servers will require a large amount of network and computing resources. An obvious solution is to do intrusion detection near the edge. This will distribute the required resources towards the edge and save precious bandwidth and server's computing resources.

This paper presents the design and implementation of a deep learning-based intrusion detection system that can be deployed on small routers on the IoT edge. The feasibility of such a system is demonstrated by using the NSL-KDD [1]. dataset that contains around 100,000 training samples and 10,000 testing samples for intrusion data. The data has 5 classes including normal packets and for 4 types of common cyber-security attacks. This paper used an enhanced version of KDD99 dataset and eliminates the redundant records problem where models tend to learn the more frequent records than the non-frequent ones [2].

The paper used Adversarial Autoencoders (AAE) to train a deep learning model for intrusion detection. An Adversarial autoencoder is a probabilistic autoencoder based on the Generative Adversarial Networks (GAN); AAE's can reduce the probability of overfitting because it can influence the distribution approximated by the hidden layer [3] [4]. The encoder tries to generate samples based on the chosen distribution, while the decoder tries to recreate the original data from the latent space. The discriminator tries to know if the sample, which was generated by the encoder, is in fact generated or from the chosen distribution. For example, in [4], an adversarial autoencoder was applied to a different intrusion dataset and the accuracy achieved was higher than that of normal autoencoder.

The paper contributes by using adversarial autoencoders to train the NSL-KDD data set and achieves a state-of-the-art performance. The paper also evaluates the deployment of this intrusion detection model on a small Raspberry PI class of computers as a proxy for a small router. Background research on best prior performance on NSL-KDD is presented next. This is followed by a discussion about the methodology in Section III. The proposed architecture is presented in Section IV and the conclusion in Section V.

## II. Background

This section explores the various intrusion detection models built for the NSL-KDD intrusion dataset. The model development pipeline involves pre-processing the network data and then applying a classification model.

### *A. Preprocessing of Data*

In terms of preprocessing, study Olusola *et al.* [5] found the most relevant features in the dataset, the dependency ratio and the most discriminating features of each class. Others used one-hot encoding for non-numeric data. For example, Ieracitano *et al.* [6], Wu *et al.* [7], and Xiao *et al.* [8] used one-hot encoding to encode the non-numeric data, whereas Zhang *et al.* [9] used dummy variable encoding. Sezari *et al.* [10], Wu *et al.* [7] and Xiao *et al.* [8] normalized the data. Xiao et al. [8] also applied data reduction using Principal Component Analysis (PCA) and Auto-Encoders (AE) then ran a convolutional neural network (CNN) on the processed data.

Intrusion datasets like NSL-KDD tend to be unbalanced leading to either eliminating some features or attempts to balance the datasets. For example, Chuang *et al.* [11] used variational autoencoders to balance the data and to avoid overfitting. Sezari *et al.* [10] removed 5 features out of 41

input features based on regressions analysis. Similarly, the system proposed by Ieracitano *et al.* [6] first pre-processed the data by eliminating outliers and null values and 20 out of the resulting 38 features were discarded. Finally, Tesfahun *et al.* [12] used Synthetic Minority Oversampling Technique (SMOTE) to balance the data set.

### B. Classification Models

Tang *et al.* [13] used a deep neural network (DNN) model proposed with 3 hidden layers with three, six and three neurons respectively. The hyper-parameters chosen were the following: a batch size of 10, 100 epochs and a learning rate of 0.001. The DNN achieved an accuracy of 75.75%.

Ieracitano *et. al* [6] used a deep autoencoder (AE) and a multilayer perceptron (MLP) that classified the dataset into 4 classes; normal, DoS, R2L and Probe where the U2R class was removed due to the unbalanced nature of the dataset. The deep autoencoder consisted of one input layer, one output layer and one hidden layer. The shallow MLP architecture one the other hand had a single hidden layer with 50 hidden neurons followed by a SoftMax output layer. Their model achieved an accuracy of up to 87%. The F1-score results for the class Normal, DoSm Probe and R2L were 90.27%, 97.61%, 80.14% and 56.83% respectively, which showed that class R2L still performed poorly when compared with other classes.

Javaid *et al.* [14] proposed using Self-taught Learning technique based on sparse autoencoder evaluated on the NSL-KDD dataset. The system was implemented on three different types of classifications, the first being anomaly detection, second being normal versus 4 different attack categories, and the third was normal and 22 different attacks. Evaluating the performance of all the two first models, the 2-class model achieved an accuracy of 88.39% which outperformed previous work at that time.

Shone *et al.* [15] proposed a novel deep learning approach of non-symmetric deep autoencoder (NDAE) for unsupervised feature learning to enable Network intrusion detection system operation with modern networks. The model proposed used a combination of deep and shallow learning by stacking Non-symmetric deep Auto-Encoder. Their model had an average accuracy of 89% with the results for "R2L" and "U2L" attack classes being anomalous.

Wu *et al.* [7] converted the data into an 11×11 array and used a CNN to achieve best accuracy of 79%.

Resende *et al.* [16] surveyed different Random Forest approaches in network intrusions systems using machine learning. Cleetus *et al.* [17], Tesfahun *et al.* [12] and Kim *et al.* [18] used Random Forest to achieve accuracies of 91%, 96%, and 96% respectively. Chauhan *et al.* [19], Hota *et al.* [20] achieved accuracies of 99.7% using Random Forest as well. Using 15 features and Random Forest and forward selection ranking, Al-Jarrah *et al.* [21] were able to achieve an accuracy of 99.8%. The same accuracy was achieved by Tama *et al.* [22] using feature selection and Random Forest.

Support Vector Machine (SVM) and Random Forest were used by Chand *et al.* [23] to achieve an accuracy of 97.5%. Panda *et al.* [24] used Intelligent Decision Technologies to classify 2 classes (i.e., attack or normal), achieving an accuracy of 99.5%. Farnaaz *et al.* [25] used feature selection and Random Forest to classify only the attacks (vs. no attack) achieving 99.6% accuracy.

Semi-Booted Network (SBN) model was proposed by Mikhail et al. [26] to classify 5 classes and achieved 99.8% accuracy. However, the metrics achieved the class U2R were not sufficient to detect anomalies at all times.

Table I summarizes the previous state-of-the-art for the NSL-KDD dataset.

TABLE I. PREVIOUS STATE-OF-THE-ART FOR NSL-KDD.

| Paper | Model | Accuracy (%) |
|---|---|---|
| Tang *et al.* [13] | Deep Learning | 75.75 |
| Wu *et al.* [7] | Convolutional Neural Network | 79 |
| Ieracitano *et al.* [6] | Autoencoders | 87 |
| Javaid *et al.* [14] | Self-Taught Learning | 88.39 |
| Shone *et al.* [15] | Non-symmetric Autoencoders | 89 |
| Cleetus and A [17] | Random Forests | 91 |
| Tesfahun and Bhaskari [12] | Random Forests | 96 |
| Kim and Kim [18] | Randim Forests | 96 |
| Chand *et al.* [23] | Support Vector Machine + Random Forest | 97.5 |
| Panda *et al.* [24] | Intelligent Decision Technologies | 99.5 |
| Farnaaz and Jabbar [25] | Random Forest | 99.6 |
| Chauhan *et al.* [19] | Random Forest | 99.7 |
| Stefanova and Ramachandran [27] | 2 step classification process | 99.7 |
| Hota and Shrivas [20] | Random Forest | 99.7 |
| Mikhail *et al.* [26] | Semi-Boosted Nested model | 99.8 |
| Al-Jarrah *et al.* [21] | Random Forest | 99.8 |
| Tama and Rhee [22] | Random Forest | 99.8 |

## III. METHODOLOGY

### A. Dataset description

This section describes some aspects of the NSL-KDD data set.

*1) Dataset Classes:* The main task of this dataset was to classify network traffic into 5 classes, 4 of which represent traffic generated due to malicious network attacks and one representing normal network traffic. The four attacks include:

*a) Denial of Service (DoS)* is an attack where legitimate users are denied network resources. This attack disrupts the network by reducing its efficiency and its services [28]. DoS attacks targets services such as emails and websites by flooding the network server with traffic and overloading it deeming it unresponsive. This can be done in various ways. Some examples of DoS attacks are:

- Smurf Attacks where an attacker sends an Internet Control Message Protocol ICMP broadcast packet to several hosts using a spoofed IP address which is the IP target of the victim. The victim will then be flooded with ICMP replies and becomes unresponsive.
- SYN flood attacks involve the attacker sending a request to connect to the target using the TCP/IP three

way handshake method without completing the connection. As can be seen, detecting DoS attacks detection depends thoroughly on network monitoring and analyzing.

*b) Probe attacks* are generally carried out to gain more information, or to perform reconnaissance about the target through surveillance and finding weak points. Probing a computer typically involves issuing commands from a local computer to computers on a network shared with that computer. These commands are used to learn information about other computers on the same network. Learning information about a computer can be done through port scanning, for instance [29].

*c) User to Root attack (U2R* is an attack where the attacker tries to get root's privilege when accessing the machine. These include buffer overflow, rootkits and SQL attacks [30].

*d) Remote to Local attack (R2L)* is an attack where the attacker gets unauthorized access to a machine. Such an attack sends packets to the machine over a network, then exploits the machine's vulnerability to illegally gain local access as a user. Those attacks exploit vulnerabilities in misconfigured systems [31]. Below are some examples of R2L attacks:

- IMAP attack, which causes buffer overflow through exploiting a bug in the authentication procedure of the IMAP server on some versions of Linux giving the attacker root privileges.
- FTP server attack, which is done when an attacker uploads an illegal software that can be downloaded by others to gain access to their machines

Table II shows a breakdown of the number of training and testing samples per class. As can be seen, the dataset is imbalanced with more than 60 thousand data points belonging to class Normal, in comparison to only 52 data points for the U2R class.

TABLE II. NUMBER OF TRAINING SAMPLES PER CLASS IN NSL-KDD.

| Class | Training | Testing |
|---|---|---|
| Normal | 67,343 | 4,329 |
| DoS | 45,927 | 3,332 |
| Probe | 11,656 | 1,053 |
| U2R | 52 | 87 |
| R2L | 995 | 1,199 |
| Total | 125,973 | 10,000 |

*2) Dataset Attributes:* Each row in the dataset is described by a total of 42 traffic attributes. Those attributes contain information such as the protocol type where it could be either TCP, UDP, or ICMP. The dataset also contains the number of failed logins, number of sudo attempts, how many files were accessed, and many others. Table IV shows the full list of the attributes in the dataset, where xAttack is the class label to be classified. As can be seen, all attributes are numerical values except for the protocol type and Attack.

## B. Dataset Visualization

Fig. 1 shows the distribution of the attack classes. As mentioned earlier, the dataset is imbalanced. Such imbalanced dataset can cause the model to be biased towards the classes with higher number of records, also known as majority classes; therefore, classifying most of the incoming requests as Normal or DoS.

## C. Data Pre-processing

*1) Balancing:* In order to balance the dataset Synthetic Minority Over-sampling Technique (SMOTE) was used. This technique duplicates the records of the minority classes so that they are equal to that of the majority classes. Normal, has a total of 67,343 records. SMOTE was used so all classes had equal number of records.

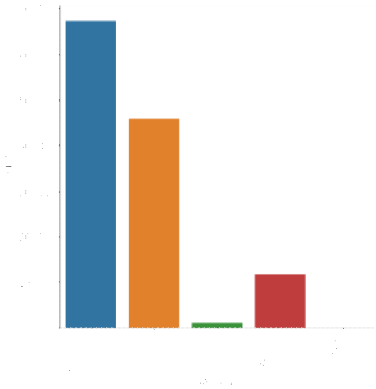

Fig. 1. Distribution of the attack classes.

TABLE III. ATTRIBUTES IN THE NSL-KDD DATA SET.

| # | Column | Dtype |
|---|---|---|
| 0 | duration | int64 |
| 1 | protocol_type | object |
| 2 | service | int64 |
| 3 | flag | int64 |
| 4 | src_bytes | int64 |
| 5 | dst_bytes | int64 |
| 6 | land | int64 |
| 7 | wrong_fragment | int64 |
| 8 | urgent | int64 |
| 9 | hot | int64 |
| 10 | num_failed_logins | int64 |
| 11 | logged_in | int64 |
| 12 | num_compromised | int64 |
| 13 | root_shell | int64 |
| 14 | su_attempted | int64 |
| 15 | num_root | int64 |
| 16 | num_file_creations | int64 |
| 17 | num_shells | int64 |
| 18 | num_access_files | int64 |
| 19 | num_outbound_cmds | int64 |
| 20 | is_host_login | int64 |
| 21 | is_guest_login | int64 |
| 22 | count | int64 |
| 23 | srv_count | int64 |
| 24 | serror_rate | float64 |
| 25 | srv_serror_rate | float64 |
| 26 | rerror_rate | float64 |
| 27 | srv_rerror_rate | float64 |
| 28 | same_srv_rate | float64 |
| 29 | diff_srv_rate | float64 |
| 30 | srv_diff_host_rate | float64 |
| 31 | dst_host_count | int64 |
| 32 | dst_host_srv_count | int64 |
| 33 | dst_host_same_srv_rate | float64 |
| 34 | dst_host_diff_srv_rate | float64 |
| 35 | dst_host_same_src_port_rate | float64 |
| 36 | dst_host_srv_diff_host_rate | float64 |

| | | |
|---|---|---|
| 37 | dst_host_serror_rate | float64 |
| 38 | dst_host_srv_serror_rate | float64 |
| 39 | dst_host_rerror_rate | float64 |
| 40 | dst_host_srv_rerror_rate | float64 |
| 41 | xAttack | object |

*2) Normalization:* the dataset has been normalized between 0 and 1 using the MinMaxScalar from the sklearn library.

### D. Model

*1) Adversarial Autoencoder:* As adversarial autoencoder [3] was used for creating a latent representation of the input data. All the 41 features are used and reduced to a latent dimension of size 16 using the autoencoder. 1000 epochs were run with batch size of 10. Adam optimizer was used with a learning rate of $10^4$. Table V, Table VI and Table VII show the characteristics of the layers of the encoder, decoder and discriminator respectively. On the 1000th epoch, the loss of the autoencoder was 0.00092866.

TABLE IV. ENCODER'S LAYERS CHARACTERISTICS.

| Layer Number | Neurons | Input Dimension | Activation Function |
|---|---|---|---|
| 1 | 128 | 41 | Relu |
| 2 | 128 | - | Relu |
| 3 | 16 | - | - |

TABLE V. DECODER'S LAYERS CHARACTERISTICS.

| Layer Number | Neurons | Input Dimension | Activation Function |
|---|---|---|---|
| 1 | 128 | 16 | Relu |
| 2 | 128 | - | Relu |
| 3 | 41 | - | Sigmoid |

TABLE VI. DISCRIMINATOR'S LAYERS CHARACTERISTICS.

| Layer Number | Neurons | Input Dimension | Activation Function |
|---|---|---|---|
| 1 | 128 | 16 | Relu |
| 2 | 128 | - | Relu |
| 3 | 1 | - | Sigmoid |

*2) Classifier:* To classify the test dataset, the trained encoder from the Adversarial Autoencoder was used to predict the sample and produce a representation in the latent space. As shown in Fig. 2, this latent space was then input to a classifier like the Nearest Neighbor (KNN) algorithm to predict the attack type.

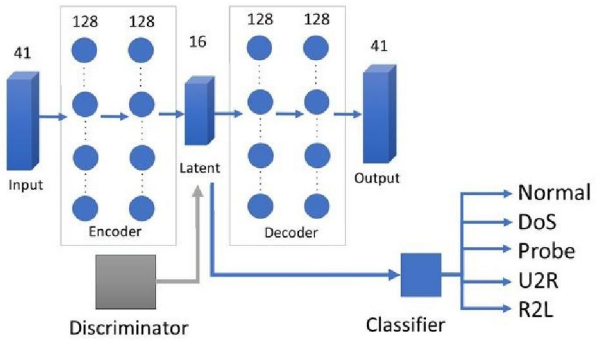


Fig. 2. Adversarial Autoencoder and classifier (e.g., kNN Classifier).

Fig. 3 shows the t-SNE [32] represention of the manifolds in the original 41 dimensions. As the Figure shows, the normal data represnted by the blue dots is split into various islands of data. The corresponding t-SNE representation of the 16-dimensional encoded space is give in Fig. 4. While the representations looks different, it is difficult to tell if there is any fundamental difference between the original and the reduced space except that now the dimensionality has been reduced from 41 to 16 dimensions.

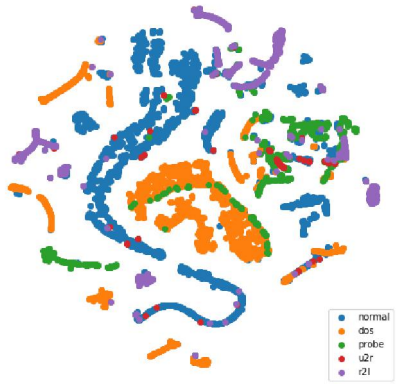


Fig. 3. t-SNE plot of the test data in the original 40 feature space.

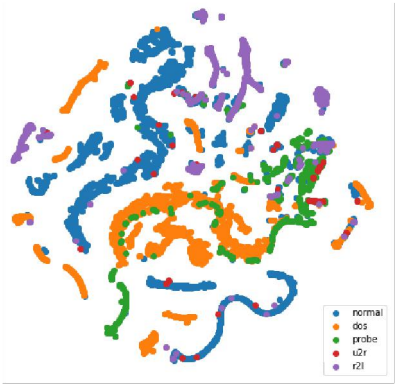


Fig. 4. t-SNE plot of the latent space of the test data.

## IV. PROPOSED IOT ARCHITECTURE

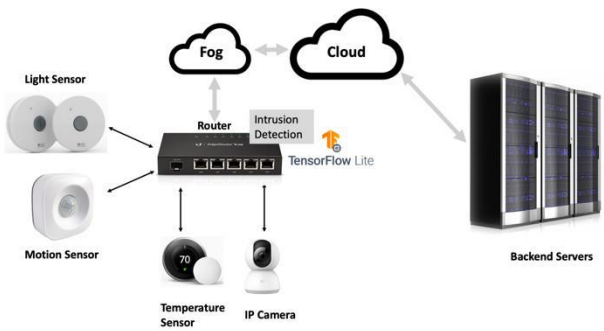


Fig. 5. IoT Architecture Diagram.

Fig. 5 shows the proposed IoT architecture. The intrusion detection is carried out on the edge router hence obviating the need for doing intrusion detection on the back-end servers or in the fog layer. A router is connected to several sensors such as temperature, light, motion and to an IP camera. Such devices can be targeted by network attacks.

### A. Evaluation

The encoder model trained on the data was saved as a TensorFlow model and a Raspberry Pi 3 B+ (RPI) was used as a proxy for a small edge router to run this model. RPI uses the Cortex-A53 processor which is used in several small routers. This means that it is reasonable to assume that the results achieved on RPI will generalize to similar smallish edge routers.

As shown in Fig. 2, once the encoder is trained, the output of the encoder (latent space) can then be used as input to train various types of classification models. Based on previously successful models shown in Table I, kNN [33], XGBoost [34], SVM [23] and Random Forest [25] were evaluated as alternative classifiers. The models were evaluated on non-SMOTED test data of the NSL-KDD which consisted of 10,000 points.

### B. Results

#### 1) Model Comparison

Table VII shows a comparison between different models arranged in the ascending order of the F1-score. The first set of comparison used the input space (41 dimensions) directly and as expected does not perform well. The best performance was the RF with a very bad F1-score of 0.4489. Using AAE followed by a classification technique yielded much better results. The best performer was AAE+KNN which beat the state-of-the-art in the literature as shown in Table I.

TABLE VII. COMPARISON OF THE VARIOUS ALTERNATIVE MODELS.

| Model | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| ***Input Space only (41-dimensional)*** | | | | |
| SVM (RBF) | 43.29 | 0.0865 | 0.2 | 0.1208 |
| KNN | 23.05 | 0.2020 | 0.1854 | 0.1344 |
| XGBoost | 61.81 | 0.4073 | 0.3683 | 0.3598 |
| RF | 73.25 | 0.4731 | 0.4524 | 0.4489 |
| ***Embedded Space (16 dimensions)*** | | | | |
| AAE+SVM | 62.87 | 0.4769 | 0.3482 | 0.3154 |
| AAE + XGBoost | 95.15 | 0.9413 | 0.8598 | 0.8880 |
| AAE + RF | 99.90 | 0.9990 | 0.9987 | 0.9989 |
| **AAE + KNN** | **99.91** | **0.9989** | **0.9991** | **0.9990** |

#### 2) The Best Model

Table VIII shows the confusion matrix for the best model (AAE+KNN). The model performed exceptionally well, and the only errors were with the R2L class which is expected as this was one of the minority classes in the original data with only 52 data points.

TABLE VIII. CONFUSION MATRIX OF THE BEST MODEL AAE + KNN.

| | Classes | Predicted | | | | |
|---|---|---|---|---|---|---|
| | | ***Normal*** | ***DoS*** | ***Probe*** | ***U2R*** | ***R2L*** |
| **Actual** | Normal | 4324 | 0 | 0 | 0 | 5 |
| | DoS | 0 | 3,332 | 0 | 0 | 0 |
| | Probe | 0 | 0 | 1053 | 0 | 0 |
| | U2R | 0 | 0 | 0 | 87 | 0 |
| | R2L | 4 | 0 | 0 | 0 | 1,195 |

The original TensorFlow model was converted to TensorFlow Lite and run on a Model 3B+ RPI. On RPI, the model took a total of 15.75 ms (sd = 0.98 ms) on average per inference. It took 0.51 ms (sd = 0.05 ms) on average for the encoder and 14.86 ms (sd = 1.72 ms) for the KNN to generate a prediction (e.g., U2R, Probe, etc.). Table IX shows the best and worst-case capacity of the model to handle the TCP/UDP traffic. When running the experiment, if every TCP/UDP packet was a maximum size then the total bandwidth an RPI can support is 4.1 Mbits/sec. Similarly, if all the packets were empty while running the experiments, then the worst-case performance would be 3.3-4 kbits/sec.

TABLE IX. BANDWIDTHS OF THE BEST AND WORST CASE SCENARIOS FOR BOTH UDP AND TCP PACKETS.

| Cases | byte/packet | bit/s | kbit/s |
|---|---|---|---|
| Best | 65,507 | 4159174.6 | 4159.1746 |
| Worst (UDP) | 52 | 3301.5873 | 3.3015873 |
| Worst (TCP) | 64 | 4063.49206 | 4.06349206 |

For example, an IP camera with compression of MPEG4 (low quality) and 32 fps, resolutions (640 × 360), HD and Full HD will require network bandwidths of 0.4 Mb/s, 1.5 Mb/s and 3.4 Mb/s respectively. Cameras with such characteristics will be suitable for our best-case scenarios but not the worst scenarios. Other IoT sensors like motion sensors, for example, have maximum 1 or 2 bytes of data, and a typical maximum sampling rate of 200 bytes/min. This means the required bandwidth is about 0.02 kbits/s which is suitable for our worst-case scenarios. While the RPI cannot handle live video or streaming audio (e.g., 256Kbs), Field Programmable Gate Arrays (FPGA) can be used to address this problem. For example, Watanabe *et al.* [35] ran several neural network algorithm on (FPGAs) and found that the FPGA could run at least 1000 inferences per second and as high as 10,000 inferences per second. Similarly, Liu *et al.* [36] showed that neural networks implemented on an FPGA had a speed up of 5.2x compared to a GPU.

Table X shows the CPU utilization when running the 10,000 test cases on the Raspberry Pi continuously. The maximum %CPU utilization was 55.5% which means that the RPI had most than enough spare computer capacity while running. This capacity is required to handle other router functions.

TABLE X. %CPU UTILIZATION.

| CPU | User% | Sys% | Wait% | Idle% | CPU% |
|---|---|---|---|---|---|
| Avg | 15.0 | 2.1 | 0.0 | 82.9 | 17.1 |
| Max | 49.7 | 6.2 | 0.0 | 99.9 | 55.5 |

## V. CONCLUSION

In this paper, we proposed a new model combining Adversarial Autoencoders and K-Nearest Neighbor to be applied on the NSL-KDD dataset, a famous network intrusion dataset. Before running the model, Synthetic Minority Over-sampling Technique (SMOTE) was used to balance the training dataset. The accuracy achieved was 99.91% and the F1-score was 0.9990 which is state-of-the-art. The trained model was run on a Raspberry Pi 3 B+ where the time for the inference per packet was 15.75 ms/packet which is a reasonable result for IoT sensor processing. These results show that for many IoT applications this model can be run on an edge device instead of a server or the fog to reduce the workload and to scale the network.

## REFERENCES


[1] NSL-KDD Intrusion Detection Dataset. Available at: *https://www.kaggle.com/what0919/intrusion-detection*.

[2] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, July 2009.

[3] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial Autoencoders," *ArXiv151105644 Cs*, May 2016. Available at: *http://arxiv.org/abs/1511.05644*.

[4] S. Puuska, T. Kokkonen, J. Alatalo, and E. Heilimo, "Anomaly-Based Network Intrusion Detection Using Wavelets and Adversarial Autoencoders," in *Innovative Security Solutions for Information Technology and Communications*, Cham, Springer, 234-246, 2019.

[5] A. Olusola, A. Oladele, and D. Abosede, "Analysis of KDD '99 Intrusion Detection Dataset for Selection of Relevance Features," in *World Congress on Engineering and Computer Science*, 20-22, 2010.

[6] C. Ieracitano *et al.*, "Statistical Analysis Driven Optimized Deep Learning System for Intrusion Detection," *ArXiv180805633 Cs*, Aug. 2018. Available at: *http://arxiv.org/abs/1808.05633*.

[7] K. Wu, Z. Chen, and W. Li, "A Novel Intrusion Detection Model for a Massive Network Using Convolutional Neural Networks," *IEEE Access*, vol. 6, 50850–50859, September 2018.

[8] Y. Xiao, C. Xing, T. Zhang, and Z. Zhao, "An Intrusion Detection Model Based on Feature Reduction and Convolutional Neural Networks," *IEEE Access*, vol. 7, 42210–42219, March 2019.

[9] L. Zhang, M. Li, X. Wang, and Y. Huang, "An Improved Network Intrusion Detection Based on Deep Neural Network," in *IOP Conf. Ser. Materials Science and Engineering*, 563 (5), 052019, Aug. 2019.

[10] B. Sezari, D. Möller, and A. Deutschmann, "Anomaly-Based Network Intrusion Detection Model Using Deep Learning in Airports," in *IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 1725–1729, Aug. 2018.

[11] P. Chuang and D. Wu, "Applying Deep Learning to Balancing Network Intrusion Detection Datasets," in *IEEE International Conference on Advanced Infocomm Technology,* 213-217, Oct. 2019.

[12] A. Tesfahun and D. Bhaskari, "Intrusion Detection Using Random Forests Classifier with SMOTE and Feature Reduction," in *International Conference on Cloud Ubiquitous Computing Emerging Technologies*, 127–132, Nov. 2013.

[13] T. Tang, L. Mhamdi, D. McLernon, S. Zaidi, and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," in *International Conference on Wireless Networks and Mobile Communications*, 258-263, Oct. 2016.

[14] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," *Security and Safety*, 3(9), Dec. 2015.

[15] N. Shone, T. Ngoc, V. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1), 41–50, Feb. 2018.

[16] P. Resende and A. Drummond, "A Survey of Random Forest Based Methods for Intrusion Detection Systems," *ACM Computing Survey*, 51(3), May 2018.

[17] N. Cleetus and K. Dhanya, "Multi-objective functions in particle swarm optimization for intrusion detection," in *Int'l Conference on Advances in Computing, Communications and Informatics*, Sep. 2014.

[18] E. Kim and S. Kim, "A Novel Hierarchical Detection Method for Enhancing Anomaly Detection Efficiency," in *Int'l Conference on Computational Intelligence and Communication Networks*, Dec. 2015.

[19] H. Chauhan, V. Kumar, S. Pundir, and E. Pilli, "A Comparative Study of Classification Techniques for Intrusion Detection," in *Int'l Symposium on Computational and Business Intelligence*, Aug. 2013.

[20] H. Hota and A. Shrivas, "Data Mining Approach for Developing Various Models Based on Types of Attack and Feature Selection as Intrusion Detection Systems (IDS)," *Intelligent Computing, Networking, and Informatics*, 845–851, 2014.

[21] O. Al-Jarrah, A. Siddiqui, M. Elsalamouny, P. Yoo, S. Muhaidat, and K. Kim, "Machine-Learning-Based Feature Selection Techniques for Large-Scale Network Intrusion Detection," in *IEEE Int'l Conference on Distributed Computing Systems Workshops*, Jul. 2014.

[22] B. Tama and K. Rhee, "A Combination of PSO-Based Feature Selection and Tree-Based Classifiers Ensemble for Intrusion Detection Systems," *Advances in Computer Science and Ubiquitous Computing*, Singapore, 489–495, 2015.

[23] N. Chand, P. Mishra, C. Krishna, E. Pilli, and M. Govil, "A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection," in *Int'l Conference on Advances in Computing, Communication, Automation*, Apr. 2016.

[24] M. Panda, A. Abraham, and M. Patra, "A Hybrid Intelligent Approach for Network Intrusion Detection," *Procedia Eng.*, 30, 1–9, 2012.

[25] N. Farnaaz and M. Jabbar, "Random Forest Modeling for Network Intrusion Detection System," *Procedia Computer Science*, 89, 213–217, 2016.

[26] J. Mikhail, J. Fossaceca, and R. Iammartino, "A Semi-Boosted Nested Model With Sensitivity-Based Weighted Binarization for Multi-Domain Network Intrusion Detection," *ACM Transactions on Intelligent Systems and Technology*, 10(3), Apr. 2019.

[27] Z. Stefanova and K. Ramachandran, "Network attribute selection, classification and accuracy (NASCA) procedure for intrusion detection systems," in *IEEE International Symposium on Technologies for Homeland Security*, Apr. 2017.

[28] R. Singh, A. Prasad, R. Moven, and H. Sarma, "Denial of service attack in wireless data network: A survey," in *Devices for Integrated Circuit (DevIC)*, Mar. 2017.

[29] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Incremental Approaches for Network Anomaly Detection: Existing Solutions and Challenges," *International Journal of Communication Networks and Information Security*, Aug. 2011.

[30] S. Bahl and S. Sharma, "Performance Analysis of User to Root Attack Class Using Correlation Based Feature Selection Model," in *International Joint Conference*, Cham, Springer, 177–187, 2015.

[31] I. Ahmad, A. Abdullah, and A. Alghamdi, "Remote to Local attack detection using supervised neural network," in *Int'l Conference for Internet Technology and Secured Transactions*, Nov. 2010.

[32] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, 9(86), 2579–2605, Nov. 2008.

[33] G. Shakhnarovich, T. Darrell, and P. Indyk, "Nearest-Neighbor Methods in Learning and Vision: Theory and Practice," *MIT Press*, March 2006.

[34] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *ACM SIGKDD International Conference on Knowledge Discovery and Data,* 785–794, Aug. 2016.

[35] D. Watanabe *et al.*, "An Architectural Study for Inference Coprocessor Core at the Edge in IoT Sensing," in *IEEE International Conference on Artificial Intelligence Circuits and Systems*, Sep. 2020.

[36] X. Liu, D. Kim, C. Wu, and D. Chen, "Resource and data optimization for hardware implementation of deep neural networks targeting FPGA-based edge devices," in *ACM/IEEE Int'l Workshop on System Level Interconnect Prediction*, Jun. 2018.