

AE-LSTM: Autoencoder with LSTM-Based Intrusion Detection in IoT

1st Mohamed Mahmoud

College of Electrical and Computer Engineering
Chungbuk National University
Cheongju, South Korea
mohammed1994hamdyy@gmail.com

2nd Mahmoud Kasem

Multimedia Systems
Assiut University
Assiut, Egypt
mahmoud.salah@aun.edu.eg

3rd Abdelrahman Abdallah

Information Technology
Assiut University
Assiut, Egypt
abdelrahmanelsayed@aun.edu.eg

4th Hyun Soo Kang

College of Electrical and Computer Engineering
Chungbuk National University
Cheongju, South Korea
hskang@cbnu.ac.kr

Abstract—The acceleration in the field of the Internet of things had increased security problems, so we find ourselves in need of effective ways to protect IoT systems from intrusions. Recently Machine learning plays an active role in network security and detecting attacks. In this research, we propose a Machine learning method (AE-LSTM) for intrusion detection which uses Autoencoder with LSTM. Our method has 6-layer Autoencoder (AE) model with LSTM that is effective in anomaly detection. To avoid the bias in our model which occur from imbalanced data in the NSL-KDD dataset, we use Standard Scaler in our AE-LSTM model to delete the outliers from the input. AE-LSTM uses the best reconstruction function. It is critical in discovering whether network traffic is normal or abnormal. We use the NSL-KDD test dataset to evaluate our proposed model. Our Model achieved the highest accuracy over other methods with f1-score micro and weight at 98.69% and 98.70% for 5 classes in detection methods (Dos, Probe, R2L, U2R, Normal). Also, we evaluated it with two classes (Malicious, Normal) with f1-score micro and weight at 98.78% and 98.78%.

Index Terms—Long short-term memory, Machine learning, Internet of Things, Intrusion Detection, Deep Learning, Autoencoder

I. INTRODUCTION

IoT is a network of interconnected computing devices which had unique IDs and transfer data between each other without human interaction. These devices (Things) can be any physical object that has an ID and connects to the internet. These devices connected to the internet are constantly increasing. According to a new Statistic report, by 2022, there will be approximately 16.4 billion connected things on the planet. By 2025, this number is predicted to reach 30.9 billion. These numbers indicate that IoT is one of the major markets that could become a cornerstone of the growing digital economy. Smart cities, smart homes, smart healthcare, autonomous vehicles, smart farming, smart education, smart grid, and other

IoT systems are examples of existent IoT systems. In light of the complexity and development of IoT systems, many new security challenges have emerged.

In the last year, deep learning approaches have significantly improved the results of several computer vision challenges, including medical applications such as cancer diagnosis, object detection [1]–[3], classification [4], and medical question answers [5]–[7], as well as applications in software engineering field like Time optimization and schedule of software projects [8], [9], and handwritten recognition for various languages [10]–[14].

Recently, there have been many academic researchers interested in the field of IoT security. We can divide the network traffic into two classes (normal and abnormal) traffics, OR into five classes: Normal, DoS (Denial of Service), R2L (Root to Local), U2R (User to Root), and Probe (Probing) attacks.

In this work, we proposed an autoencoder (AE) model with LSTM that determines which network traffic is normal or not. We train and test our Method on the NSL-KDD dataset [15] which achieves the best results among other models.

The following section defines the related work on intrusion detection using Machine Learning (ML) and Deep Learning (DL). Section III proposed Method. Section IV provides Experiment Results and the conclusion is given in Section V.

II. RELATED WORK

W. Xu, J. Jang-Jaccard, et al [16] used Autoencoder (AE)-based model and a new Method for deleting the outliers to avoid the bias that occurred by imbalanced data for each input sample. They got an accuracy was 90.61% and an F-score of 92.26% on the NSL-KDD test dataset. T. Su, H. Sun, J. Zhu, et al [17] proposed the BAT

which consists of BLSTM (Bidirectional-LSTM) and Attention Methodology which achieves an accuracy reach of 84.25% on the NSL-KDD test dataset. Wang, H., and Li, W. [18] developed a hybrid neural network DosTC structure that combines efficient and scalable transformers with a CNN (convolutional neural network) to detect DDoS (distributed denial-of-service) assaults on SDN, which was evaluated on the CICDDoS2019 dataset. M. S. Elsayed, et al [19] proposed DDoSNet to detect the DDoS (distributed denial-of-service) attacks in SDN (Software Defined Network). They depend on Deep Learning (DL) to build their method which consists of a Recurrent Neural Network(RNN) with an autoencoder. They evaluate their model on the CICDDoS2019 dataset. S. M. Taghavinejad, M. Taghavinejad, et al [20] developed a method consisting of a mixture of three decision trees To be used in intrusion detection. They evaluate their model on the NSL-KDD dataset.

III. PROPOSED METHOD

In this section, we explain the method of Auto-Encoder and LSTM. We describe Our Proposed model called Autoencoder methods with the LSTM model (AE-LSTM) for IoT Intrusion Detection. In the following section, we will describe how each component of our proposed models works. The Autoencoder (AE), and LSTM will be discussed. for the network Intrusion detection tasks, The reconstruction error is used by the Autoencoder model to identify whether or not a network traffic sample is uncommon [16]. In this study, We present an autoencoder with three encoder layers and three decoder layers. in the encoder and decoder, Dense layer followed by batch normalization [21] and the LeakyReLU [22] activation function before passing the decoder output features to the LSTM model for Intrusion Detection. Figure 1 depicts our Autoencoder (AE), while Figure 5 depicts our LSTM model.

A. Auto-Encoder Method

To reconstruct data, A form of unsupervised neural net especially feed-forward called an autoencoder (AE) . An AE is made up of three levels: input, output , and hidden layers levels. First two have the exact number of neurons, but any buried layers have fewer neurons.

Generic autoencoder architecture have the following operations: encoding in addition to decoding. $[x_1; x_2; x_3; \dots; x_m]$ is an m-dimensional vector for every input sample x. In the encoding operation, linked to buried layer representation (y), as the following formula.

$$y = f_1(wx + b) \quad (1)$$

Activation function of the encoder symbolized by f_1 . Bias vector is symbolized by b, while the weight matrix is symbolized by w.

Inside the decoding procedure, hidden representation of (y) is transformed into a new rebuild \bar{x} , as shown in the equation

$$\bar{x} = f_2(w'y + b') \quad (2)$$

f_2 is the activation function of the decoder. The output layer's weights in addition to bias are denoted by w' and b' . During these two methods, neural network's parameters D ($w; w'; b; b'$) are constantly modified by minimizing the reconstruction error. Loss reconstruction (L) is calculated using non-linear functions to minimize the reconstruction error of x.

$$L(x, \bar{x}) = \frac{1}{m} \sum_{j=1}^m (x_j - \bar{x}_j)^2 \quad (3)$$

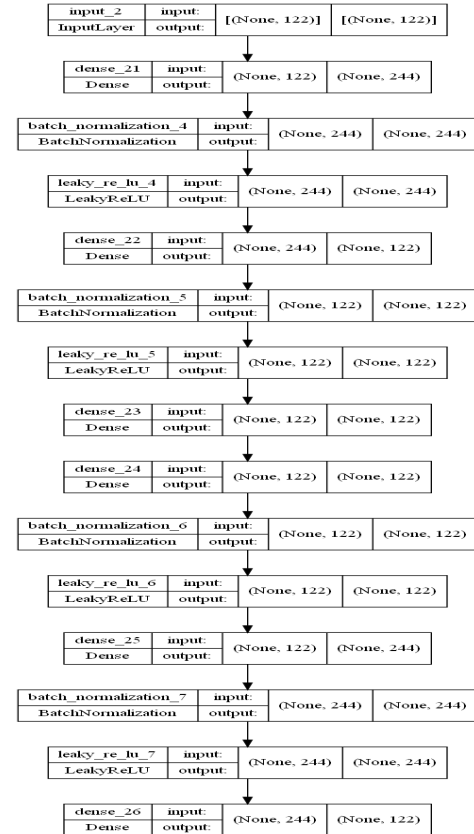


Fig. 1. Auto-Encoder model with 6 layers.

B. Recurrent neural network

The Neural Network is an extremely strong model, yet it is not without restrictions. One advantage is that it is a memoryless model, which means that for each input, the network has no knowledge of prior inputs. This characteristic, on the other hand, can be accomplished by employing a "Recurrent Neural Network" (RNN).

RNNs are similar to NNs with the exception that each layer l will not only transfer its output to the following layer $l + 1$, but will also pass it to itself, resulting in a "recurrent connection." This is depicted in Figure 2.

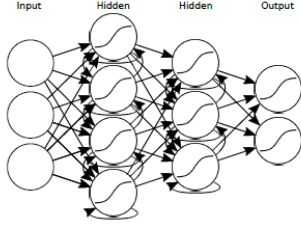


Fig. 2. A 3-layer recurrent neural network with input and output is visualised. the input is $x \in \mathbb{R}^3$ and output is $y \in \mathbb{R}^2$.

A popular method for visualising RNNs is to "unfold" the network and openly show the recurrent connections between each time-step t . Figure 2 shows an unfolded version of the network illustrated in Figure 3.

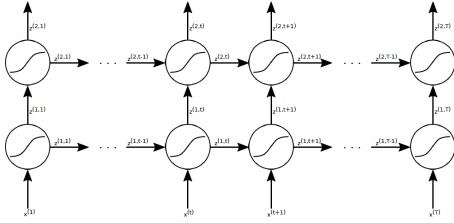


Fig. 3. An unfolded 3-layer recurrent neural network (2 recurrent layers and 1 output layer) with input sequences of length T is shown, with $x(1)$ representing the initial input at time-step $t = 1$ of a given observation x and $z(2, T)$ representing the final output at time-step $t = T$.

Let the input variables be sequences of length T , e.g. $x = x(1), x(2), \dots, x(T)$. For example, designate each character in a sentence with length T .

$$u = \{U^{(1)}, U^{(2)}, \dots, U^{(L-1)}\} \quad (4)$$

the set of weights for the recurrent connections (the output layer has no recurrent connections) with $U^{(l)} = [u_1^{(l)}, u_2^{(l)}, \dots, u_{N^{(l)}}^{(l)}]$ and $u_k^{(l)} \in \mathbb{R}^{N^{(l)}}$. Then, at a given time-step t , represent the activation of neuron k in layer l by expanding (5) with a term for the recurrent connection.

$$z_k^{(l,t)} = \sigma(a_k^{(l,t)}) = \sigma\left(\sum_{k'=1}^{N^{(l+1)}} \delta_{k',t}^{l+1} w_{k',k}^{(l+1)} \underbrace{\sum_{k'=1}^{N^{(l)}} \delta_{k',t+1}^{l+1} u_{k',k}^{(l)}}_{\text{recurrent term}}\right) \quad (5)$$

with $\delta_k^{l,T+1} \forall k, l$ for a complete definition of the recurrence. Finally the BPTT algorithm defines the gradients

as summing the error terms for each time-step t

$$\frac{\partial L(x, y)}{\partial w_{k,j}^{(l)}} = \sum_{t=1}^T \delta_k^{(l,t)} z_k^{(l,t)} \quad (6)$$

C. Long short-term memory

As indicated in the preceding section, the typical RNN architecture has several issues with processing recurrent information in the network. Learning long-term dependencies, in particular, has proven to be a serious challenge for conventional RNNs in reality. Furthermore, the fact that information is transmitted back into its own hidden layer on each time-step has been proven to either blow up or degrade the outputs exponentially, making training extremely difficult. This is referred to as the "exploding gradient problem" or "vanishing gradient problem."

To address these issues, the widely utilised "Long Short-Term Memory" (LSTM) architecture might be adopted. The LSTM design handles long-term dependencies and keeps gradient information over time better than normal RNN architectures, making it the favoured RNN architecture over regular RNN architectures.

Because the LSTM extends the ordinary RNN neuron to include several operations, each neuron in the RNN is referred to as a "block." Figure 4 shows a simple illustration of a typical RNN block.

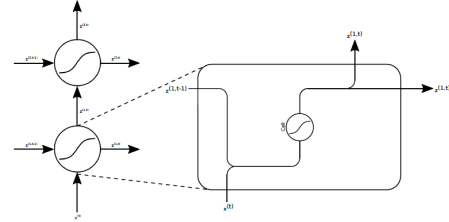


Fig. 4. A typical RNN block with $x(t)$ as the input and $z(1, t)$ as the output, and $z(1, t - 1)$ as the output from the previous time-step.

The LSTM extends the simple block depicted in Figure 4 in a variety of ways. First, a "cell state" $c(t)$ in \mathbb{R}^C is introduced, with C denoting the state's dimensionality, that is capable of storing information throughout time if necessary. This can be thought of as the LSTM block's "memory." Second, a variety of "gates" are established, each of which takes the concatenated input $x(t)$ and previous output.

$z(t - 1)$ as input and returns a C -dimensional vector with values in the range $[0, 1]$ with the sigmoid function with learning weights and biases W_f, W_i, W_o , and b_f, b_i, b_o . Using g and h to represent non-linear functions, which are frequently used as \tanh , and \oplus and \otimes to represent element-wise addition and multiplication, respectively. A visualization of the LSTM block can be seen in Figure 5

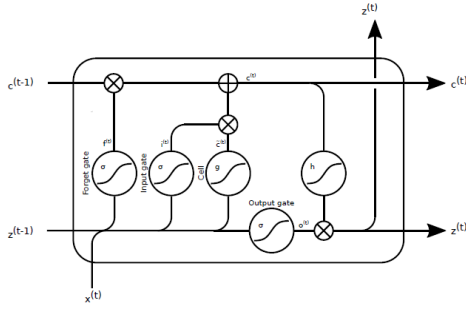


Fig. 5. LSTM block with $x(t)$ and $z(t)$ as input and output, and $c(t)$ as cell state at time step t . σ represents sigmoid functions, whereas g and h represent non-linear functions. *bigoplus* represents element-wise addition, whereas *bigotimes* represents element-wise multiplication.

IV. EXPERIMENT RESULTS

A. Dataset

In our research, the NSL-KDD database is usually used to test and determine strategies. Many researchers use NSL-KDD dataset, which is considered one among the most popular databases in intrusion detection that's available.

Because of concerns regarding numerous train and test data samples, as well as relevant issues, the NSL-KDD database has been updated from the KDD'99 database. Compared to the KDD'99 data collection, the NSLKDD data set offers a few benefits. As a result, in the train and test parts, A collection of duplicate records is not included in NSL-KDD. This is because if you don't follow this guideline, the classifier will likely repeat records.

Total number of records in this collection from every set, on the other hand, has the same difficulty level as the proportion of records in the primary data collection.

Furthermore, inside the train and test parts, the number of recordings is realistic and sufficient for testing, allowing the entire data set will be utilized instead of a random sample of a tiny portion of the dataset. As a result, the outcomes of numerous studies based on this data will be consistent [23], [24].

Each attribute vector in the database comprises 43 components, and each class has 41 attributes. Element 42 set the assault's form, whereas element 43 set a difficulty's level. Handful of DOS, Probe, R26 and U2R attacks are also included.

Figure 6 displays how many assaults and how many typical situations there are in the dataset. Detecting an attack or non-attack as well as determining the sort of assault has two steps. Detecting an assault or not for every data sample described in this study.

B. Experiment Settings

All of the models were trained using Tensorflow [25], a deep learning toolkit written in Python applied in

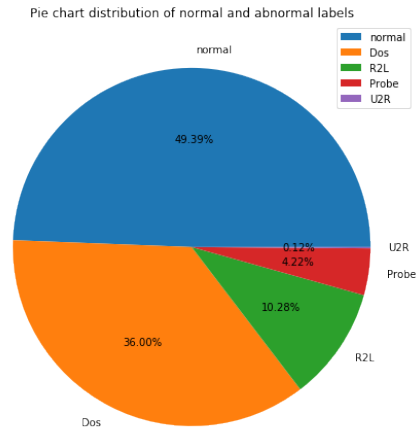


Fig. 6. Percentage of Attacks and Non- Attacks in the dataset.

training all models. Tensorflow enables the use of extremely efficient mathematical processes on GPUs in a transparent manner. Also, all actions necessary for the provided calculations are defined by a computational graph.

A Core i7 CPU, one NVIDIA RTX2060 graphics card, and 16 GB of RAM were used in the testing. The time for the model to train is decreased by third when the GPU is used. However, because speedup was not properly assessed during the project, it is possible that it has altered.

In all AutoEncoder and LSTM models, the value of the validation loss is kept to a minimum. The stochastic gradient descent optimization is accomplished by Adam technique [26] in addition to a learning rate of 0.0001 and 128 mini-batches.

C. Results

64% of the dataset was used to train our AE-LSTM model, verified using the residual 16%, and tested using 20% of the dataset. Our model has been trained for a total of 100 epochs. To avoid overfitting, at the start of every epoch, the training set is randomized. Because there was no pre-trained or transfer model from a previous dataset, we had to train our network from scratch. we trained our model in two experiments.

In the first experiment, we used binary classification to train our model to detect intrusions and categorize them as malicious or benign. We also trained the model to categorize it as Dos, Probe, R2L, U2R, or Normal using multi-classification. In Fig 7 and 8 show the validation dataset's loss and accuracy while training model for binary classification.

In Fig 9 and 10 show the validation dataset's loss and accuracy while training model for multi-classification.

We compared our model's performance to other similar models. Accuracy, precision, recall, and F1 score were

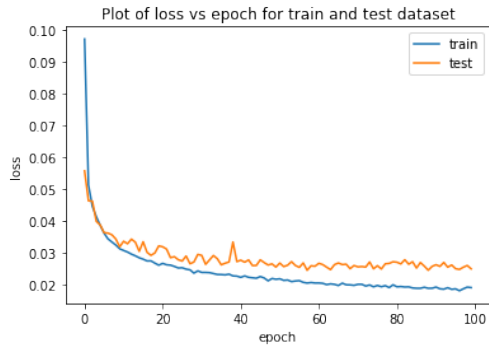


Fig. 7. loss vs epoch plots for binary classification.

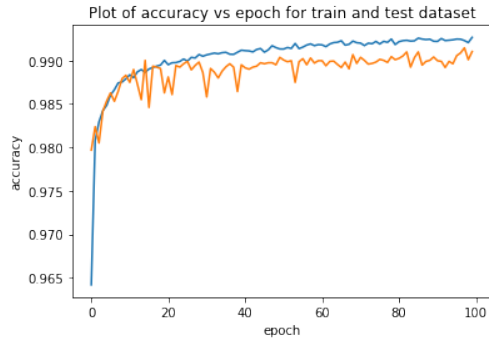


Fig. 8. accuracy vs epoch plots for binary classification.

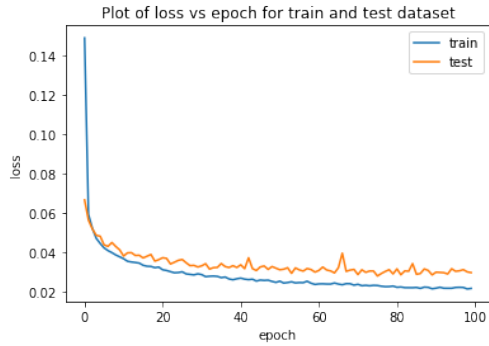


Fig. 9. loss vs epoch plots for multi-classification.

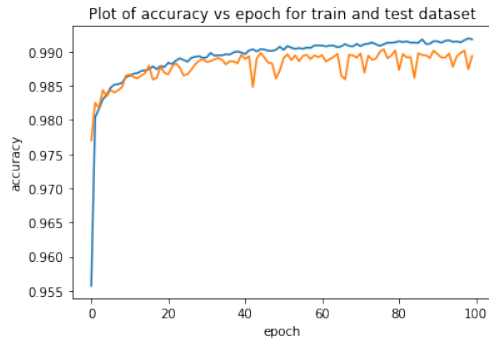


Fig. 10. accuracy vs epoch plots for multi-classification

utilized in comparing findings. Our AE-LSTM model has a maximum f1-score of 98.88% and an accuracy of greater than 98.88%, as shown in Table I.

TABLE I
PERFORMANCE COMPARISON WITH OTHER AE MODELS

Model	Precision	Recall	F1-score	Accuracy
AE-LSTM	98.89%	98.89%	98.88%	98.88%
Al-Qatf et al. [27]	96.23%	76.57%	85.28%	84.96%
Cosimo et al. [28]	87%	80.37%	81.98%	84.21%
Quamar et al. [29]	85.44%	95.95%	90.4%	88.39%
Kishwar et al. [30]	87.92%	93.48%	90.61%	88.98%
WEN XU et al. [16]	86.83%	98.43%	92.26%	90.61%
Yousefi et al. [31]	-	-	-	83.34%

Figure 11 shows our model's ROC curve (curve of receiver operating characteristic). AUC (area under the ROC curve) result (AUCAE D 0.999) demonstrates its excellent performance, with a very high true positive rate in addition to a low false-positive rate.

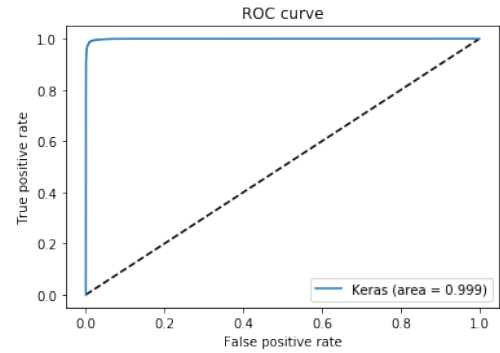


Fig. 11. ROC curves of AE-LSTM

V. CONCLUSION

This work entails, 6-layer autoencoder (AE) model is a novel model with LSTM for intrusion detection. In our AE-LSTM model, we also use a unique data Standard Scaler approach to modify and eliminate the input samples' most affected outliers, decreasing model bias induced by data imbalance in the feature set across distinct data types. On the NSL-KDD test dataset, we assessed our suggested model and compared it with the previous models and our model achieved the highest accuracy over other methods with f1-score macro and weight at 93.13% and 98.92% for 5 classes in detection methods (Dos, Probe, R2L, U2R, Normal). Also, we evaluated it with two classes (Malicious, Normal) with f1-score macro and weight at 98.88% and 98.89%.

ACKNOWLEDGMENT

The Ministry of Education funded the Basic Science Research Program through the National Research Foundation of Korea (NRF) under Grant 2020R1I1A3A04037680, and the Ministry of Oceans and Fisheries funded the Research Projects "Development

of automatic screening and hybrid detection system for hazardous material detection in a port container."

REFERENCES

- [1] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun, "Deep learning for image-based cancer detection and diagnosis: a survey," *Pattern Recognition*, vol. 83, pp. 134–149, 2018.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
- [3] A. Abdallah, A. Berendeyev, I. Nuradin, and D. Nurseitov, "Tncr: Table net detection and classification dataset," *Neurocomputing*, vol. 473, pp. 79–97, 2022.
- [4] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," in *Proceedings of the international conference on machine learning*, vol. 28. ACM, New York, USA, 2013, pp. 3937–3949.
- [5] S. Minaee and Z. Liu, "Automatic question-answering using a deep similarity neural network," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 923–927.
- [6] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep learning in software engineering," *arXiv preprint arXiv:1805.04825*, 2018.
- [7] A. Abdallah, M. Kasem, M. A. Hamada, and S. Sdeek, "Automated question-answer medical model based on deep learning technology," in *Proceedings of the 6th International Conference on Engineering & MIS 2020*, 2020, pp. 1–8.
- [8] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 50–59.
- [9] M. A. Hamada, A. Abdallah, M. Kasem, and M. Abokhalil, "Neural network estimation model to optimize timing and schedule of software projects," in *2021 IEEE International Conference on Smart Information Systems and Technologies (SIST)*. IEEE, 2021, pp. 1–7.
- [10] S. A. Mahmoud, I. Ahmad, W. G. Al-Khatib, M. Alshayeb, M. T. Parvez, V. Märgner, and G. A. Fink, "Khatt: An open arabic offline handwritten text database," *Pattern Recognition*, vol. 47, no. 3, pp. 1096–1112, 2014.
- [11] A. Abdallah, M. Hamada, and D. Nurseitov, "Attention-based fully gated cnn-bgru for russian handwritten text," *Journal of Imaging*, vol. 6, no. 12, 2020. [Online]. Available: <https://www.mdpi.com/2313-433X/6/12/141>
- [12] D. Nurseitov, K. Bostanbekov, D. Kurmankhojayev, A. Alimova, A. Abdallah, and R. Tolegenov, "Handwritten kazakh and russian (hkr) database for text recognition," *Multimedia Tools and Applications*, vol. 80, no. 21, pp. 33 075–33 097, 2021.
- [13] N. Toiganbayeva, M. Kasem, G. Abdimanap, K. Bostanbekov, A. Abdallah, A. Alimova, and D. Nurseitov, "Kohtd: Kazakh offline handwritten text dataset," *arXiv preprint arXiv:2110.04075*, 2021.
- [14] D. Nurseitov, K. Bostanbekov, M. Kanatov, A. Alimova, A. Abdallah, and G. Abdimanap, "Classification of handwritten names of cities and handwritten text recognition using various deep learning models," *arXiv preprint arXiv:2102.04816*, 2021.
- [15] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, 2009, pp. 1–6.
- [16] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset," *IEEE Access*, vol. 9, pp. 140 136–140 146, 2021.
- [17] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, "Bat: Deep learning methods on network intrusion detection using nsl-kdd dataset," *IEEE Access*, vol. 8, pp. 29 575–29 585, 2020.
- [18] H. Wang and W. Li, "Ddostc: A transformer-based network attack detection hybrid mechanism in sdn," *Sensors*, vol. 21, no. 15, p. 5047, 2021.
- [19] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Ddosnet: A deep-learning model for detecting network attacks," in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2020, pp. 391–396.
- [20] S. M. Taghavinejad, M. Taghavinejad, L. Shahmiri, M. Zavvar, and M. H. Zavvar, "Intrusion detection in iot-based smart grid using hybrid decision tree," in *2020 6th International Conference on Web Research (ICWR)*. IEEE, 2020, pp. 152–156.
- [21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.
- [22] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.
- [23] S. Hosseini and B. M. H. Zade, "New hybrid method for attack detection using combination of evolutionary algorithms, svm, and ann," *Computer Networks*, vol. 173, p. 107168, 2020.
- [24] Y. Li, Y. Xu, Z. Liu, H. Hou, Y. Zheng, Y. Xin, Y. Zhao, and L. Cui, "Robust detection for network intrusion of industrial iot based on multi-cnn fusion," *Measurement*, vol. 154, p. 107450, 2020.
- [25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: <http://arxiv.org/abs/1603.04467>
- [26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [27] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with svm for network intrusion detection," *Ieee Access*, vol. 6, pp. 52 843–52 856, 2018.
- [28] C. Ieracitano, A. Adeel, F. C. Morabito, and A. Hussain, "A novel statistical analysis and autoencoder driven intelligent intrusion detection approach," *Neurocomputing*, vol. 387, pp. 51–62, 2020.
- [29] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based ddos detection system in software-defined networking (sdn)," *arXiv preprint arXiv:1611.07400*, 2016.
- [30] K. Sadaf and J. Sultana, "Intrusion detection based on autoencoder and isolation forest in fog computing," *IEEE Access*, vol. 8, pp. 167 059–167 068, 2020.
- [31] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3854–3861.