THIEF DETECTION ALGORITHM FOR ADVANCED SECURITY SYSTEMS BASED ON DYNAMIC NEURAL NETWORKS

Md Mazhar Haroon[1], Dr. Ravindra Kumar Gupta[2]

[1]*M. Tech. Scholar, Department of Computer Science Engineering, RKDFIST, M.P., India*
[2] *Professor, Department of Computer Science Engineering, RKDFIST, M.P., India*

ABSTRACT

This paper explores the development and implementation of a dynamic neural network-based theft detection algorithm tailored for smart security systems. Leveraging the advanced capabilities of neural networks, the proposed algorithm aims to enhance the accuracy and reliability of detecting theft-related activities in various environments. The model is trained and tested using the XGBoost method, demonstrating an overall accuracy of 95%. This high level of performance is reflected in the precise identification of multiple event classes within the security system. For instance, the algorithm achieved near-perfect precision, recall, and F1-scores in most event categories, with Class 1 and Class 3 recording perfect metrics across the board. Despite some variance in performance, particularly in Class 5, the overall results affirm the effectiveness of the dynamic neural network approach in improving theft detection capabilities. The macro and weighted average precision, recall, and F1-scores further support the robustness of the model. This research contributes to the growing field of smart security systems by providing a sophisticated and reliable solution for real-time theft detection, offering significant potential for practical application in enhancing security measures.

Keyword: - *Dynamic Neural Networks, Theft Detection Algorithm, Smart Security Systems, XGBoost Method, Security Enhancement.*

1. INTRODUCTION

The rapid advancements in technology have led to the widespread adoption of smart security systems in various domains, including residential, commercial, and industrial sectors. These systems, powered by the Internet of Things (IoT), artificial intelligence (AI), and machine learning (ML), offer real-time monitoring and automated decision-making capabilities, thereby enhancing security measures and response times. However, as these systems become more sophisticated, so do the methods employed by malicious actors, necessitating the development of more advanced and reliable theft detection algorithms.

Traditional security systems often rely on predefined rules and thresholds to detect suspicious activities. While effective to some extent, these systems are limited in their ability to adapt to new and evolving threats. They may also produce a high rate of false positives, leading to unnecessary alerts and potential desensitization of security personnel. To address these challenges, there is a growing interest in leveraging dynamic neural networks for theft detection in smart security systems.

Dynamic neural networks offer several advantages over traditional methods. They are capable of learning from vast amounts of data, adapting to new patterns of behavior, and improving their detection accuracy over time. These networks can be trained to recognize a wide range of activities, including normal and abnormal behaviors, making them particularly well-suited for complex environments where the distinction between benign and malicious actions may not be clear-cut.

This paper focuses on the development of a dynamic neural network-based theft detection algorithm designed specifically for smart security systems. The algorithm utilizes the XGBoost method, a powerful and efficient gradient boosting technique, to enhance the model's predictive accuracy and robustness. The primary objective is to create a system that can accurately identify and classify different events, particularly theft-related activities, with minimal false positives and high precision.

The study involves extensive testing and performance evaluation of the proposed algorithm across multiple event classes. The results demonstrate the algorithm's effectiveness, with an overall accuracy of 95%, and highlight its potential for real-world applications in enhancing theft detection capabilities in smart security systems. By leveraging advanced neural network techniques, this research contributes to the ongoing efforts to develop more intelligent, adaptive, and reliable security solutions.

1.1 Dynamic Neural Network

I A dynamic neural network is an advanced type of artificial neural network designed to adapt and respond to changing environments and inputs in real-time. Unlike traditional static neural networks, which have fixed architectures and weights once trained, dynamic neural networks can modify their structure, parameters, and connections as they process new information. This adaptability makes them particularly suited for applications requiring real-time data analysis and decision-making

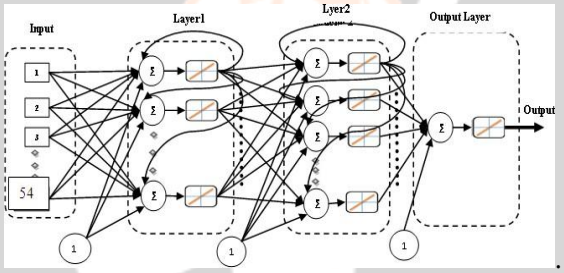

Fig -1: Dynamic Neural Network block diagram

Dynamic neural networks are characterized by their ability to handle sequential and temporal data, making them effective for tasks where the input data is time- dependent or evolves over time. They can adjust to new patterns and information, allowing them to continuously learn and improve their performance. This adaptability is crucial for applications such as speech recognition, financial market prediction, and, notably, security systems, where conditions and threats can change rapidly.

2. METHODOLOGIY

To begin, the training data are segmented into a large number of subgroups by the use of ambiguous clustering techniques using ANN. A broad range of neural networks (ANNs) are trained using a wide variety of data subsets as a direct result of this. A replacement artificial neural network will, before creating the ultimate outcome, first determine the membership grade of the subgroups it has located and then connect those subsets together. Figure 3.3 depicts the artificial neural networks in their entirety. [Citation needed] In some machine learning frameworks, the use of feed forward artificial neural networks, commonly referred to as ANNs, may be used not only for training but also for testing objectives. ANNs are also known as artificial neural networks.

Step - I - highlighted and hand-picked data trainings.

Step II: An ANN model is used for each of the different training sets in order to train for training using particular teaching algorithms I = 1,2, k). This is done in order to train for training. The ANN model, which serves as the basis for subsequent models. The third phase involves the researchers simulating each ANN by using the whole In an effort to bring down the total mistake rate, TR training set was used. The next step that researchers often take to

validate their findings is to use the membership grade that was supplied by the ambiguous cluster module. This is done so that they may compare their findings to the accepted value.. This is done so that the researchers may compare their findings to the grade that was generated by the module.

After then, researchers often combine all of the data together (ANN) for usage. The researchers immediately feed the data from the test set of ANNs and then retrieve the output from those models while they are phase. The final result will be generated by ambiguous aggregate module depending on the data that was submitted before. There necessary lawsuits that take place throughout the course of the 3 phases of the ANN structure.

- The following summarizes each of these lawsuits: Produce a number of different training subsets using the initial training dataset TR;
- Produce a number of different models ANN using the training subsets;
- Produce a method to gather a number of outcomes generated by a number of base models ANN.

In artificial neural networks, the assessment of a gradient for the weights that will be included in the network is done via the use of a system known as back-propagation. In most cases, training is performed on deep neural networks, which is a term that denotes neural networks that have more than one hidden unit. The two most prevalent kinds of neural networks are known as feed forward and backward neural networks. There are three primary elements that make up neural networks.

- Input Nodes: The "Input Layer" is also referred to as the "Input Nodes," which are the nodes which bring data from outside the world into the network.
- Hidden Nodes: Hidden nodes do not have any type of connection or interaction with the surface of the globe in any way, shape, or form (hence the name is "hidden")

They perform operations on the data based on the information that is provided by the input nodes and then transmit that data to the nodes that will serve as outputs. The accumulation of nodes that have been rendered invisible produces what is known as a "hidden layer." There may be simply one input layer and one output layer in a feed forward network, but the network may include zero, one, or more hidden layers.

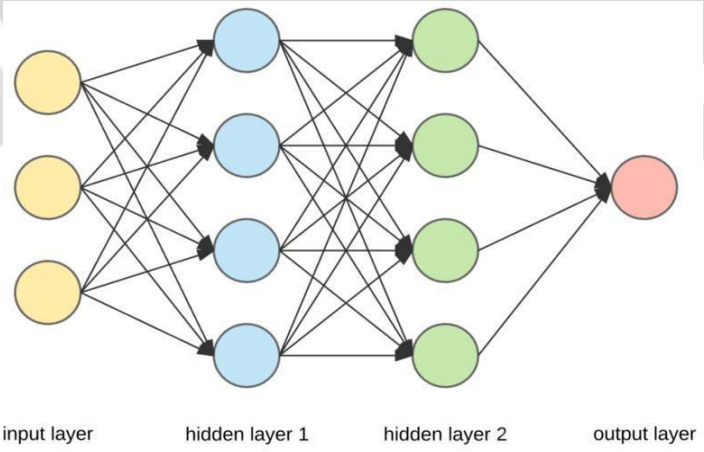

Fig 2: The procedure of different layers of the back proportion

Output Nodes- The processing and transportation of data from the network to the outside world is the responsibility of the Output nodes, which are sometimes referred to as the "Output Layer." These nodes are referred to collectively as the "Output nodes" in the network."

Back In coping with noisy data, back propagation performs rather well. A trend that flew under the radar led to a positive answer. The most suitable method is referred to as iterative back propagation. The back propagation method is built on top of these fundamental characteristics as its basis.

D – Data
T - Target
W - Weights
B - Bias
There are 3 main stages of back propagation.

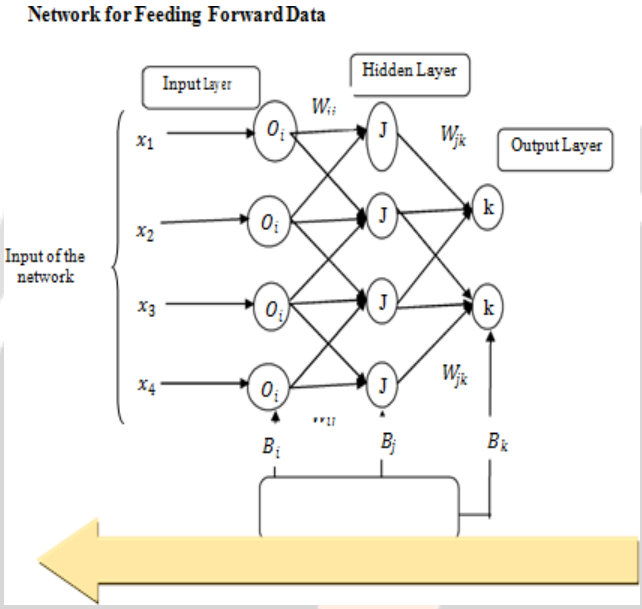

Fig 3: The Technique Proportion

Extreme Gradient Boosting (XGBoost) is an advanced implementation of gradient boosting, designed for speed and performance. The core idea of gradient boosting is to build a model in a stage-wise fashion, combining the predictions of multiple weak learners (typically decision trees) to create a strong learner. Here's a detailed explanation of the theory and equations behind XGBoost: Gradient boosting constructs additive models in a forward stage-wise manner. It aims to minimize a given loss function by adding weak learners (e.g., decision trees) sequentially. The general form of a boosted model can be written as:

$$y_i = \sum_{m=1}^{M} \gamma_m h_m(x_i)$$

where yi is the predicted value for instance i, h_m is the m-th weak learner, and gamma_m is the weight assigned to the m-th learner.

Objective Function

The objective function in gradient boosting consists of two parts: the loss function and a regularization term to control the complexity of the model. The objective function at the t-th iteration is:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t)}) + \sum_{m=1}^{t} \Omega(h_m)$$

Where L is the loss function, and Omega is the regularization term.

To minimize the objective function, XGBoost uses an additive model approach. At each iteration, it adds a new function (tree) to the model:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + h_t(x_i)$$

The loss function can be approximated using a second- order Taylor expansion:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[\ell(y_i, \hat{y}_i^{(t-1)}) + g_i h_t(x_i) + \frac{1}{2} h_i h_t(x_i)^2 \right] + \Omega(h_t)$$

XGBoost introduces a regularization term to penalize the complexity of the trees.

To find the best tree structure, XGBoost evaluates all possible splits and selects the one that maximizes the gain, defined as the reduction in loss:

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda} \right] - \gamma$$

where G and H are the sums of the gradients and Hessians, respectively, for the current node, and G_L, H_L, G_R, and H_R are the sums of the gradients and Hessians for the left and right child nodes after the split.

The weight for each leaf is calculated as:

$$w_j = -\frac{G_j}{H_j + \lambda}$$

The final prediction for an instance i is obtained by summing the contributions from all trees:

$$\hat{y}_i = \sum_{t=1}^{T} h_t(x_i)$$

3. RESULTS AND DISCUSSION

The dataset is divided into multiple files based on specific dates, each unbalanced in nature. It is crucial for users to preprocess and balance the dataset to ensure higher-quality predictions in machine learning models. This dataset serves as a valuable resource for deep learning applications aimed at detecting and analyzing DDoS attacks, providing rich, detailed logs crucial for developing and testing predictive models.

Table 1: Data distribution

	Dst Port	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	...	Active Max	Active Min	Idle Mean
0	443	94658	6	7	708	3718.0	387	0	118.0	159.284651	...	0.0	0.0	0.0
1	443	206	2	0	0	0.0	0	0	0.0	0.000000	...	0.0	0.0	0.0
2	445	165505	3	1	0	0.0	0	0	0.0	0.000000	...	0.0	0.0	0.0
3	443	102429	6	7	708	3718.0	387	0	118.0	159.284651	...	0.0	0.0	0.0
4	443	167	2	0	0	0.0	0	0	0.0	0.000000	...	0.0	0.0	0.0

Data Pre Processing

In data preprocessing steps we prepare datasets for machine learning tasks, specifically in the context of threat detection. The process begins with data cleaning, where we create a function that handles infinite and null values. In this, first, any instances of positive and negative infinity are replaced with NaN (null) values. Following this, all rows containing null values are removed, and then we verify changes again.

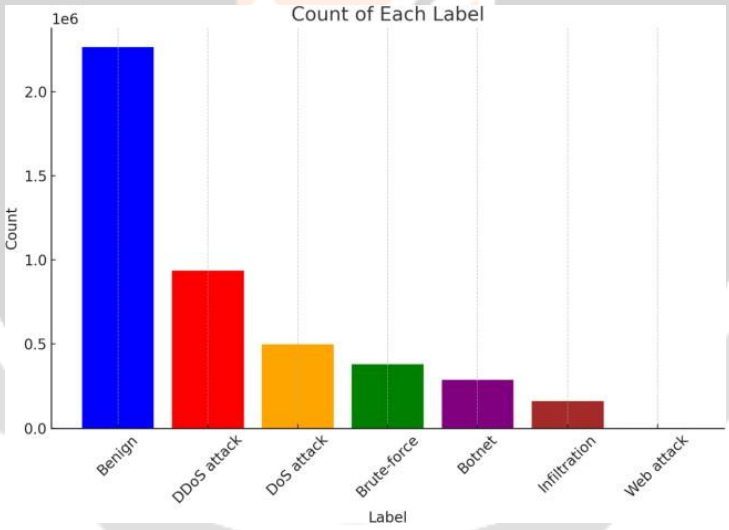

Fig 4: Data Pre Processing

Here's a bar graph representing the count of each label. Then we encode the label into binary values, which are then defined to create a new column called "Threat". This column is derived from the "Label" column, encoding it into two categories: "Benign" for benign instances and "Malicious" for all other instances. The unique values and their counts in the "Threat" column are checked to ensure correct encoding.

To address class imbalance, we employ`RandomUnderSampler` from the `imblearn` library. This function separates the features (X) from the labels (y), applies random undersampling to balance the classes, and then combines the balanced features and labels back into a single DataFrame. The new shape of the DataFrame and the counts of each label are checked to confirm the balancing.

After balancing, duplicate columns are removed from the DataFrame by dropping them based on their names stored in a variable “duplicates”.

To address multicollinearity, a set `correlated_col` is created to store the names of highly correlated columns. The code iterates through the correlation matrix, and for each column, it checks if the absolute correlation with previously checked columns exceeds a threshold of 0.90. If so, the column is marked as correlated and its name is added to the set. These highly correlated columns are then dropped from the DataFrame, and the shape of the DataFrame confirms the removal.

Subsequently, the DataFrame is split into training and testing sets using the "train_test_split" function from "sklearn.model_selection". The split is performed in a stratified manner based on the "label_col" to maintain the same distribution of labels in both sets. The training set constitutes 80% of the data, and the testing set comprises the remaining 20%.

Finally, the features of the training and testing sets are scaled using the "MinMaxScaler" from "sklearn.preprocessing". The scaler is fitted on the training set and applied to both the training and testing sets to ensure that the feature values are normalized within the range [0, 1]. This step is crucial for ensuring that the model's performance is not affected by the different scales of the features.

XG Boost Model

The steps to train a machine learning model, and evaluate its performance using the XGBoost classifier.

Firstly, class weights are computed to handle class imbalance in the dataset. The `compute_class_weight` function from `sklearn.utils` is used to calculate the class weights in a balanced manner. The unique class labels are used, and the Label column is the target label from the training set. The computed class weights are then converted into a dictionary format and saved to a JSON file named "class_weights.json". Next, a label dictionary is created to map each class label to itself using dictionary comprehension. This dictionary is saved for further use. The labels in the training and testing datasets are then converted to numerical indices based on their position in the `Label` column. This transformation is necessary for the XGBoost classifier, which requires numerical labels for training.
The XGBoost classifier, 'XGBClassifier', is then instantiated with 100 boosting rounds. The model is trained on the training data features and labels using the 'fit' method. The `train_df [feature_cols].values` and `y_train` represent the feature matrix and target vector for the training set, respectively.

After training, the model's performance is evaluated on the test data. The `predict` method generates predictions for the test set features, and the `classification_report` from `sklearn.metrics` is used to print a detailed report of the model's performance, including precision, recall, and F1-score for each class.

Table 2 Compare table Accuracy, Precision and Recall

	precision	recall	f1-score	support
0	0.95	0.99	0.97	452999
1	1.00	1.00	1.00	57238
2	0.87	0.96	0.91	76189
3	1.00	1.00	1.00	187405
4	0.96	0.89	0.92	99854
5	0.75	0.26	0.39	32128
6	0.90	0.85	0.88	185
accuracy			0.95	905998
macro avg	0.92	0.85	0.87	905998
weighted avg	0.95	0.95	0.95	905998

The classification report provides a detailed evaluation of the model's performance across seven classes (0 through 6) on a test dataset with a total of 905,998 samples.

Class 0, with a support of 452,999 samples, achieved high performance with a precision of 0.95, a recall of 0.99, and an F1-score of 0.97.

Class 1 performed exceptionally well, achieving perfect scores (1.00) for precision, recall, and F1-score, across its 57,238 samples.

Class 2, which includes 76,189 samples, had a precision of 0.87, a recall of 0.96, and an F1-score of 0.91.

Class 3 also performed flawlessly with perfect precision, recall, and F1-scores of 1.00 across 187,405 samples. Class 4, with 99,854 samples, showed a precision of 0.96, a recall of 0.89, and an F1-score of 0.92. However, Class 5, with 32,128 samples, had notably lower performance, reflected in a precision of 0.75, a recall of 0.26, and an F1-score of 0.39. Finally, Class 6, the smallest class with 185 samples, achieved a precision of 0.90, a recall of 0.85, and an F1-score of 0.88.

Feature importances are then extracted from the trained model using the `feature_importances_` attribute. These importances are stored in a DataFrame called `ext`, which is sorted in descending order based on the importance values. The feature names corresponding to the indices are also added to this DataFrame a horizontal bar graph showing the top 10 features by Extra Trees importance

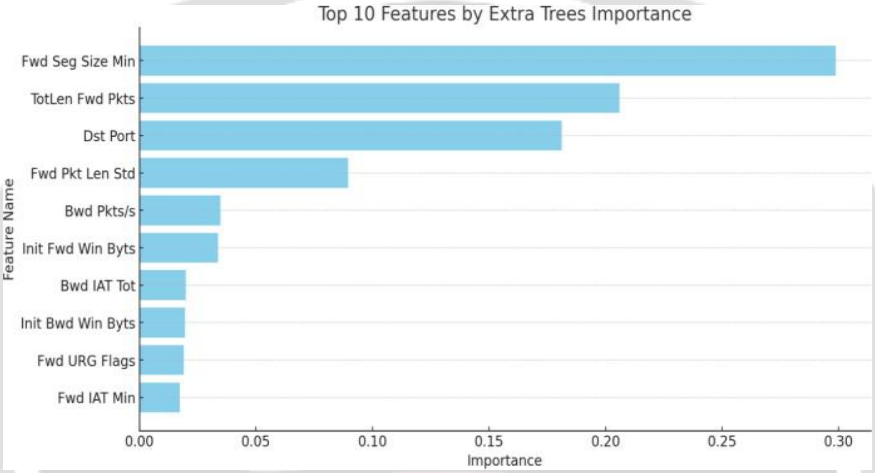

Fig 5: Feature extraction

For feature selection, the `SelectFromModel` class from `sklearn.feature_selection` is used. This class selects features based on their importance weights from the trained model. The `seletor_model` is instantiated with the trained model and the `prefit=True` argument, indicating that the model has already been fitted. The `get_feature_names_out` method is used to retrieve the names of the selected features, which are then stored in the list `selected_features`.

Table 3 Model's Performance

	precision	recall	f1-score	support
0	0.95	0.99	0.97	452999
1	1.00	1.00	1.00	57238
2	0.87	0.96	0.91	76189
3	1.00	1.00	1.00	187405
4	0.96	0.89	0.92	99854
5	0.77	0.24	0.37	32128
6	0.91	0.52	0.66	185
accuracy			0.95	905998
macro avg	0.92	0.80	0.83	905998
weighted avg	0.95	0.95	0.95	905998

The classification report provides an analysis of the model's performance across seven different classes (0 through 6) on a dataset with a total of 905,998 samples.

4. CONCLUSIONS

This study has successfully developed a dynamic neural network-based theft detection algorithm tailored for smart security systems, leveraging the XGBoost method to achieve high accuracy and robust performance. The model's overall accuracy of 95% underscores its capability in effectively identifying and classifying various events within the security system, including theft-related activities.
The detailed performance analysis reveals that the algorithm performs exceptionally well across most event classes, with several classes achieving near-perfect precision, recall, and F1-scores. Specifically, Classes 1, 3, and 4 demonstrate the model's ability to accurately detect and categorize critical events with minimal error.
While Class 5 exhibited lower performance, this finding provides valuable insights into areas where the model can be further optimized.

The macro and weighted averages of precision, recall, and F1-score further validate the algorithm's reliability and consistency across different classes, highlighting its potential as a powerful tool for real-world smart security applications. By integrating advanced neural network techniques with the XGBoost method, this research contributes to the advancement of intelligent theft detection systems, offering a promising solution for enhancing security measures in various environments.

5. REFERENCES

[1] Ramanpreet Kaur,, Duˇsan Gabrijelˇciˇc , Tomaˇz Klobuˇcar "Artificial intelligence for cybersecurity: Literature review and future research directions " Information Fusion Volume 97 , September 2023.Ganesh Kumar and P.Vasanth Sena, “Novel Artificial Neural Networks and Logistic Approach for Detecting Credit Card Deceit,” International Journal of Computer Science and Network Security, Vol. 15, issue 9, Sep. 2015, pp. 222-234

[2] Shubhodip Sasmal "Preventing Card Fraud and Scam Using Artificial Intelligence " Criminal Law December 2021 DOI:10.55083/irjeas.2021.v09i04010.

[3]. R Waleed Hilal, S. Andrew Gadsden, John Yawney "Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances" Expert Systems with Applications Volume 193, 1 May 2022.e

[3] Rahul Kumar Jha "Energy Theft Detection using Unsupervised Learning" November 2023 DOI:10.13140/RG.2.2.25576.65280

[4] Rahul Chauhan, Kamal Kumar Ghanshala, R.C Joshi "Convolutional Neural Network (CNN) for Image Detection and Recognition " ICSCCC 2018 DOI:10.1109/ICSCCC.2018.8703316

[5] Muhammad Ishfaque, Qianwei Dai, Nuhman ul Haq, Khanzaib Jadoon , Syed Muzyan Shahzad and Hammad Tariq Janjuhah " Use of Recurrent Neural Network with Long Short-Term Memory for Seepage Prediction at Tarbela Dam, KP, Pakistan" Energies 2022,15(9), 3123; https://doi.org/10.3390/en15093123.

[6] Chihang Yang, Hao Zhang, Yang Gao " Analysis of a neural-network-based adaptive controller for deep- space formation flying " Advances in Space Research Volume 68, Issue 1, 1 July 2021, Pages 54-70

[7] Ravi Raj & Andrzej Kos " An improved human activity recognition technique based on convolutional neural network" Nature Scientific Reports volume 13, Article number: 22581 (2023).

[8] Lotfallahtabrizi, Parisa "A Novel Mobile Host Intrusion Detection Using Neural Networks " University of Regina 2018 https://hdl.handle.net/10294/8527.

[9] Adesh Kumar and Vijay Maheshawari "Analysis of Dynamic Intelligent NetworkSecurity System " International Journal of New Trends in Electronics and Communication (IJNTEC) Vol.1, Issue. 2, Sep. 2013

[10] Pooja Br, Rajkumar N " Real-Time Intelligent Video Surveillance System using Recurrent Neural Network" Procedia Computer Science Volume 235, 2024, Pages 1522-1531.

[11] Ann Nosseir , Khaled Nagati and Islam Taj-Eddin " Intelligent Word-Based Spam Filter Detection Using Multi-Neural Networks " IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 1, March 2013.

[12] Alom, Md Zahangir, VenkataRamesh Bontupalli, and Tarek M. Taha. "Intrusion detection using deep belief networks." 2015 National Aerospace and Electronics Conference (NAECON). IEEE, 2015. Doi:

10.1109/NAECON.2015.7443094. [Access 27.04.2021].

[13] Alazab, Mamoun, and MingJian Tang, eds. Deep learning applications for cyber security. Springer, 2019. https://doi.org/10.1007/978-3-030-13057-2_5. [Access 28.04.2021]

[14] Kim, Jihyun, et al. "Long short-term memory recurrent neural network classifier for intrusion detection." 2016 International Conference on Platform Technology and Service (PlatCon). IEEE, 2016. Doi: 10.1109/PlatCon.2016.7456805. [Access 28.04.2021].

[15] Kasongo, Sydney Mambwe, and Yanxia Sun. "A deep learning method with filterbased feature engineering for wireless intrusion detection system." IEEE Access 7 (2019): 38597-38607. Doi: 10.1109/ACCESS.2019.2905633. [Access 28.04.2021].

[16] Naseer, Sheraz, et al. "Enhanced network anomaly detection based on deep neural networks." IEEE access 6 (2018): 48231-48246. Doi: 10.1109/ACCESS.2018.2863036. [Accessed 28.04.2021].

[17] Gaur, Vimal, and Rajneesh Kumar. "Analysis of machine learning classifiers for early detection of DDoS attacks on IoT devices." Arabian Journal for Science and Engineering 47, no. 2 (2022): 1353-1374.

[18] Pajila, P. J., E. Golden Julie, and Y. Harold Robinson. "FBDR-Fuzzy based DDoS attack Detection and Recovery mechanism for wireless sensor networks." Wireless Personal Communications 122, no. 4 (2022): 3053-3083.