GOCE DELCEV UNIVERSITY - STIP
FACULTY OF COMPUTER SCIENCE

The journal is indexed in
EBSCO

ISSN 2545-4803 on line
DOI: 10.46763/BJAMI

# BALKAN JOURNAL OF APPLIED MATHEMATICS AND INFORMATICS (BJAMI)

BALKAN JOURNAL
OF APPLIED MATHEMATICS AND INFORMATICS

YEAR 2022
VOLUME V, Number 2

**AIMS AND SCOPE:**
BJAMI publishes original research articles in the areas of applied mathematics and informatics.

**Topics:**
1. Computer science;
2. Computer and software engineering;
3. Information technology;
4. Computer security;
5. Electrical engineering;
6. Telecommunication;
7. Mathematics and its applications;
8. Articles of interdisciplinary of computer and information sciences with education, economics, environmental, health, and engineering.



**Address of the editorial office**
Goce Delcev University – Stip
Faculty of philology
Krste Misirkov 10-A
PO box 201, 2000 Štip,
Republic of North Macedonia

**BALKAN JOURNAL**
**OF APPLIED MATHEMATICS AND INFORMATICS** (BJAMI), Vol 5

**ISSN 2545-4803 on line**
**Vol. 5, No. 2, Year 2022**

# CONTENT

**Sara Aneva, Marija Sterjova and Saso Gelev**
SCADA SYSTEM SIMULATION FOR A PHOTOVOLTAIC ROOFTOP SYSTEM .....................7

**Elena Karamazova Gelova and Mirjana Kocaleva Vitanova**
SOLVING TASKS FROM THE TOPIC PLANE EQUATION USING GEOGEBRA .....................17

**Sadri Alija, Alaa Khalaf Hamoud and Fisnik Morina**
PREDICTING TEXTBOOK MEDIA SELECTION USING
DECISION TREE ALGORITHMS ..........................................................................................27

**Goce Stefanov And Biljana Citkuseva Dimitrovska**
DESIGN OF TFT SWITCH GRID ..........................................................................................35

**Angela Tockova, Zoran Zlatev, Saso Koceski**
GRAPE LEAVES DISEASE RECOGNITION USING AMAZON SAGE MAKER ......................45

**Anastasija Samardziska and Cveta Martinovska Bande**
NETWORK INTRUSION DETECTION BASED ON CLASIFICATION .....................................57

**Aleksandra Risteska-Kamcheski and Vlado Gicev**
ANALYSIS OF THE DEFORMATION DISTRIBUTION IN THE SYSTEM
DEPENDING ON THE YIELD DEFORMATION...........................................................................69

**Aleksandra Risteska-Kamcheski and Vlado Gicev**
DEPENDENCE OF ENERGY ENTERING A BUILDING FROM THE
INCIDENT ANGLE, THE LEVEL OF NONLINEARITY IN SOIL, AND
THE FOUNDATION STIFFNESS ..........................................................................................81

**Sijce Miovska, Aleksandar Krstev, Dejan Krstev, Sasko Dimitrov**
BUSINESS PROCESS MODELING, SYSTEM ENGINEERING AND THEIR
APPROACH TO THEIR APPLICATION IN INDUSTRIAL CAPACITY .....................................89

**Sasko S. Dimitrov, Dejan Krstev, Aleksandar Krstev**
MATRIX METHOD FOR LARGE SCALE SYSTEMS ANALYSIS ............................................99

**Vasko Gerasimovski and Vlatko Chingoski**
SMALL MODULAR NUCLEAR REACTORS – NEW PERSPECTIVES
IN ENERGY TRANSITION ..........................................................................................107

**Vesna Dimitrievska Ristovska and Petar Sekuloski**
TOPOLOGICAL DATA ANALYSIS AS A TOOL FOR THE
CLASSIFICATION OF DIGITAL IMAGES ..........................................................................117

**Sasko Milev And Darko Tasevski and Blagoja Nestorovski**
STRESS DISTRIBUTION ALONG THE CROSS SECTION OF THE
NARROWEST PART OF THE DIAPHRAGM SPRING FINGERS ..............................................127

Balkan Journal of Applied Mathematics and Informatics
Volume V Number 2 Year 2022
Online ISSN 2545-4803
UDC: 004.72.056.5:004.492.3

# NETWORK INTRUSION DETECTION BASED ON CLASIFICATION

ANASTASIJA SAMARDZISKA AND CVETA MARTINOVSKA BANDE

**Abstract.** Network security is a serious concern for information technology users. Intrusion detection systems can detect malicious traffic and suspicious activity looking for signatures of known attacks. This paper describes a network intrusion detection system based on the deep learning approach. The system uses the ability of the neural network to detect attacks for which the system was not explicitly trained. The proposed solution can effectively identify network attacks with the accuracy of 98% tested on the NSL_KDD dataset. The paper analyzes the impact of transformation functions applied to the features of the dataset.

## 1. **Introduction**

Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) are important for maintaining network security. IDSs analyze network traffic to identify attacks, attempts to gather information about the network or systems, or other malicious activities. IDSs are passive components. If they detect anomalies or deviations from normal activity, they notify the network administrator, for example, by sending an email. Then it is up to the administrator to examine the data and decide whether the network is under attack and, if so, decide on how to proceed. IPSs are active components. They can intercept the direct line of communication between the source and destination and automatically act on detected anomalies. In this sense, IPSs are an improvement to passive IDSs. There are different kinds of IDSs/IPSs and they can be divided into different categories depending either on their location in the network, or the data used to detect attempted breaches. This paper focuses on Network intrusion detection systems (NIDSs). NIDSs examine each packet traversing the network looking for indications of malicious activities in both, the packet header and content payload. NIDSs monitor traffic from the router to the host [1][2]. One way to implement NIDS is to use signatures. NIDS have to maintain a database of known malicious patterns referred to as signatures. Monitored traffic is compared to the signatures in the database. Regular updates of existing attack signatures are important to provide network protection. But this approach cannot detect novel attacks, the so called zero-day attacks. Several researchers propose using machine learning techniques to overcome this drawback of the signature-based NIDSs and to enhance their security [3]. In essence, this approach suggests that if a machine learning model is created that can learn to generalize the characteristics common to attacks, this model should also be able to recognize novel attacks that were not explicitly included in the training dataset. This paper proposes a deep learning approach for NIDS capable to differentiate between normal network connections and malicious network connections. NSL-KDD (Network Security Laboratory-Knowledge Discovery and Data Mining) dataset [4] is used to train the model. Each record in NSL-KDD dataset refers to a particular connection between a source and a destination. The neural network learns to identify a malicious network connection based on the features in the NSL-KDD dataset. The model examines the feature values in the NSL-KDD dataset and looks for indications of malicious activity. If the connection shows characteristics of malicious network traffic, the model returns a number in the interval [0.0, 1.0] denoting how likely a connection is to be malicious. The probabilities that fall below the decision boundary are classified as normal traffic and the probabilities that are above the decision boundary are classified as malicious traffic. The research on using deep learning methods for NIDS is in an early phase and is actively being investigated. The goal of this work is to train a model that will learn to recognize most of the attacks that were included in the training dataset, but also to test how well the model generalizes common characteristics of malicious connections and therefore how well the model performs in recognizing variations of these attacks.

## 2. Related work

Over the last decade, many machine learning and deep learning solutions have been proposed using different methodologies, datasets, and evaluation metrics, to make NIDSs efficient in detecting malicious attacks. Despite the research efforts to preserve the integrity and confidentiality of the network traffic, NIDSs still face challenges in improving detection accuracy, reducing false alarm rates, and detecting novel intrusions. A recent survey of NIDSs is presented in [3]. In Table 1 we compare the model proposed in this paper with several recent approaches to network intrusion detection through traffic classification. The models that we analyze implement deep learning and classic machine learning techniques and use different preprocessing schemes of data delivered to the learning algorithm. Models are created and tested on several available datasets. Both binary and multiclass approaches are proposed. Some models also address class imbalance through sampling or perform feature selection. In [5] authors compare the performance of different DL neural network architectures and conventional ML based models using standardized classification quality metrics: receiver operating characteristics (ROC), area under RoC curve, accuracy, precision-recall curve, and mean average precision. The types of deep neural networks that were compared in this study are: convolutional neural network (CNN), neural network with Long Short-Term Memory (LSTM) layers, and different autoencoders (sparse, denoising, contractive and convolutional). These deep neural network models were trained and tested on the NSL-KDD dataset. All 41 features of the NSL-KDD dataset were used to train the models. Vinayakumar et al. [6] used the KDDCup99 dataset to train a deep neural network (DNN) for classification of network traffic and achieved an accuracy of 92.7%. The resulting model was then applied to the NSL-KDD, UNSW-NB15, Kyoto, WSN-DS and CICIDS2017 datasets. The model trained on the KDDCup99 dataset achieved an accuracy of 93.1% in binary classification when applied to the CICIDS2017 dataset. However, the model performed considerably worse on the NSL-KDD and UNSW-NB15 datasets, achieving an accuracy of 78.9% and 76.1% respectively.

In [5] the authors consider different encoding schemes for categorical features and their impact on the accuracy using the NSL-KDD dataset and theDecision Tree classifier. They analyze several new features created by the encoding algorithm, the training time, and the accuracy of the model, and decide to use LeaveOneOutEncoder for the categorical features in the NSL-KDD dataset. A similar study is described in [7] using the Random Forest classifier. Cao et al. [8] used the LabelEncoder from the scikit-learn library to encode the categorical features. Furthermore, the authors also applied sampling and feature selection in the preprocessing stage. They used a hybrid sampling method to reduce the class size disparity. The majority class is undersampled with the Repeated Edited Nearest Neighbours (RENN) algorithm and the minority class is oversampled with the Adaptive Synthetic Sampling (ADASYN) algorithm. The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is used to remove the noise from the new sampled datasets, and then the sampled datasets are merged to obtain the balanced dataset. To perform feature selection, the Random Forest algorithm is used to calculate the contribution of features, and the Pearson correlation analysis is performed to calculate the correlation between features. The model achieves an accuracy of 99.69% for multiclass classification on the NSL-KDD dataset. The model was also tested using the UNSW-NB15 dataset and the CICIDS2017 dataset and achieved an accuracy of 86.25% and 99.65%, respectively. In [9] the authors present an Adaboost based binary network traffic classifier. The Decision Tree classifier is used as a primary classifier and the Adaboost algorithm is used to perform the weight updates. The UNSW-NB15 dataset is adopted to train and test the model.

Table 1. Comparison of several related works listed in the reference section [5]-[10]

| | [4] | [5] | [6] | [7] | [8] | [9] |
|---|---|---|---|---|---|---|
| Dataset | NSL-KDD | KDDCup99 | NSL-KDD | UNSW-NB15<br>NSL-KDD<br>CICIDS2017 | UNSW-ND15 | NSL-KDD |
| Methods | Autoencoder (AE)<br>Contractive AE (C_AE)<br>Sparse AE (S_AE)<br>Denoising AE (D_AE)<br>LSTM<br>CNN | DNN | DCNN | CNN-GRU | ANN<br>SVM<br>AdaBoost based on decision trees | ANN that combines BLSTM, multiple convolutional layers and attention mechanism |
| Evaluation Metrics | ROC Curve<br>Area under RoC curve<br>precision-recall Curve<br>mean average precision<br>accuracy | accuracy<br>precision<br>F1-Score<br>True Positive Rate (=Recall)<br>False Positive Rate | ROC Curve<br>Area under RoC curve<br>precision-recall curve<br>mean average precision<br>accuracy | accuracy<br>precision<br>recall<br>F1-Score | accuracy<br>precision<br>recall<br>F1-score | accuracy confusion matrix |
| Data Preprocessing | LeaveOneOut Encoder | no description in article | LeaveOneOutEncoder | LabelEncoder | LabelEncoder | One-hot encoding |
| | Remove mean and scale according to IQR (*Interquartile Range)* | L2 Normalization | Remove median and scale according to IQR | min-max normalization | no description in article | min-max normalization |
| Accuracy (%) | AE 81<br>C_AE 81<br>S_AE 79<br>D_AE 77<br>LSTM 89<br>CNN 85 | BIN – 92.7<br>MC – 92.5 | 85.22 | UNSW-NB15 86.25<br>NSL-KDD 99.69<br>CICIDS2017 99.65 | ANN 89.54<br>SVM 94.7<br>Ada boost 99.3 | BIN – 82.56<br>MC – 84.25 |
| BIN \ MC | BIN | model1 BIN<br>model2 MC<br>model3 MC | MC | MC | BIN | BIN and MC |
| FS | No | Yes on model3 (MC) | No | Yes | Yes | No |
| Software | Keras with Theano backend | Keras with Tensorflow backend | Keras with Tensorflow backend | Keras with Tensorflow backend | Keras with Tensorflow backend | Keras with Tensorflow backend |

Feature selection is performed in the data preprocessing stage. The Adaboost based model achieved higher accuracy than the artificial neural network and the Support Vector Machine classifier that are used for comparison. The authors in [10] combine a Bidirectional Long Short-Term Memory layer, multiple convolutional layers, and an attention mechanism to create the BAT-MC model. BAT-MC is trained using the NSL-KDD dataset and has better performances compared to classic machine learning techniques in both binary and multiclass classification.

## 3. **Working environment**

The prototype of the intrusion detection system was developed and tested using several Python libraries. The Anaconda3 Individual Edition was installed and used to create the Python virtual environment mlearn where all the necessary libraries were installed [11]. The Tensorflow open-source machine learning platform is installed to provide effective execution of low-level tensor operations and computing of the gradient of arbitrary differentiable expressions. The Keras library is integrated in TensorFlow. Keras is a machine learning library that allows the creation of deep learning algorithms. The Keras API is very efficient, the core structures are layers and models which are the building blocks used to create neural networks that take advantage of the low-level computational capabilities of tensorflow [12][13]. The scikit-learn library is an open-source machine learning library used for preprocessing the data before it being forwarded to a neural network [14]. Matplotlib [15] and pandas [16] are used as auxiliary libraries for drawing histograms and data analysis, respectively. The Jupyter Notebook [17] web application is used to create files that contain Python scripts and interpretation results, LaTeX equations, HTML markup and images.

## 4. Description of the NSL-KDD dataset

The dataset that is used to train and test the model is the NSL-KDD dataset. NSL-KDD is created by examining and improving the 1999 KDD Cup dataset [18]. The data in the 1999 KDD Cup dataset is used in the International Knowledge Discovery and Data Mining Tools Competition that was held alongside the International Conference on Knowledge Discovery and Data Mining in 1999, the so-called KDD-99. The goal of the competition was to design a machine learning model that will be able to differentiate between malicious network connections and normal network traffic. The data in the 1999 KDD Cup dataset was generated from network traffic collected and stored in raw tcpdump format for the DARPA Intrusion Detection Evaluation Program by the MIT Lincoln Laboratory [19]. The generated traffic was preprocessed, and features that convey useful information were extracted. Based on these features, a machine learning model can learn to classify a network connection as either normal traffic or as an attack. The raw tcpdump data was used to create CSV data where each feature was placed in a separate column. For the purposes of the dataset, the term "network connection" was defined as a sequence of TCP packets exchanged between two hosts, starting and ending at well-defined times with well-defined application level protocols. Then to each record a label was added, either "normal" or an attack, with exactly one particular attack type. The original 1999 KDD Cup dataset was widely used in intrusion detection research, but several drawbacks of this dataset were pointed out [20]. Consequently, [4] described and published a new dataset, NSL-KDD that addressed some of the drawbacks of the 1999 KDD Cup dataset. The NSL-KDD dataset was created from the 1999 KDD Cup dataset by removing all duplicate records from the training and testing datasets. Afterwards, a subset of records that showed better statistical distribution was chosen from the remaining unique records. The resulting NSL-KDD dataset has a smaller number of records compared to the 1999 KDD Cup dataset. NSL-KDD is already split into a training and testing dataset. The dataset consists of two files KDDTrain+.txt and KDDTest+.txt. The KDDTrain+.txt file has 125,973 records and the KDDTest+.txt file has 22.544 records. Each record is about 100 bytes in one line of the CSV file. It is important to notice that these two files have a different statistical distribution of attack labels: the KDDTest+.txt dataset includes types of attacks that were not introduced in KDDTrain+.txt. The dataset was designed in this way to allow researchers to test how well a trained classifier generalized the training data. The hypothesis is that new network attacks very often show similarities to known attacks. This means that a classifier could successfully learn some generalizable properties of several attack categories that allow it to correctly classify attack types that were not introduced during the training process.

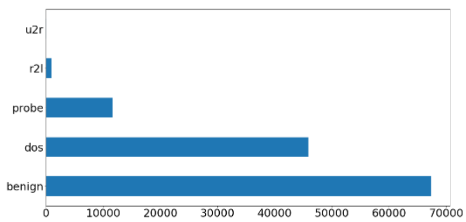


Figure 1. *KDDTrain+.txt class distribution*

| protocol_type_tcp | protocol_type_udp |
|---|---|
| 1 | 0 |
| 0 | 0 |
| 1 | 0 |
| 1 | 0 |
| 1 | 0 |

Figure 2. *Example of one-hot-encoding*

The attack types included in the dataset are listed in [21]. The attacks can be classified in one of five classes: benign, dos, r2l, u2r, and probe. One important observation is that NSL-KDD shows a notable imbalance between the numbers of records in each of the classes [1]. Figure 1 shows the distribution of records in the KDDTrain++.txt file across the five classes. So, for example, the dos class includes three times more samples than the probe class. Instead of dealing with multiclass data, the task of the network traffic classification was approached as a binary classification problem: all network traffic must be attributed to exactly one of two categories – either normal or malicious traffic, as this is the approach taken by the majority of intrusion detection systems. After mapping each record in the training and testing datasets as either "normal" or as an "attack", the number of samples in these two classes is comparable. The last value in each CSV record is the 'success_pred' column. The ‘success_pred’ column was excluded from the analysis. This feature is not a property of a connection and is added to the original 1999 KDD Cup dataset by [4] as part of the evaluation procedure. After that, it is used to create the improved NSL-KDD dataset. The column ‘attack_type’ is the label of the category to which the sample belongs. The rest of the features in NSL-KDD can be divided into three groups: basic, content and traffic features [4]. The features in the first group (basic) contain aggregated packet header data from packets associated with the same connection. Although the packet header data provides valuable information that should be considered when analyzing network traffic, it is not sufficient to identify all types of attacks that are included in NSL-KDD [22]. The attacks in the categories ‘r2l’ and ‘u2r’ can only be identified by inspecting the data portion of network packets. For instance, attacks such as buffer and heap overflow and SQL injection, most commonly occur over one legitimate network connection and can only be detected by examining the content of network packets. To detect such content-based attacks, the analysis must also consider:

- application level protocols (e.g., Telnet, HTTP, FTP, or SMTP)
- failed login attempts
- successful login attempts
- attempts to gain root access (check if the command su root was issued)
- whether root access was granted
- attempts to create files, etc.

Attacks in the ‘dos’ and ‘probe’ categories involve many connections to the same host/hosts over a very short period of time. To detect these types of attacks, the data about more than one network connection must be considered. The data in the “time-based traffic features” group considers connections from the last 2 seconds. However, there are slow probing attacks that can scan hosts (or ports) over a time interval longer than 2 seconds, for example every minute. To get a model capable of identifying slow probing attacks as well, values in the “time-based traffic features” were recalculated, this time based on a fixed number of connections instead of a fixed time interval and “connection-based traffic features” were created [4][1]. Table 2 lists all 41 features in the NSL-KDD dataset and shows which features belong to groups: basic, content, and traffic. The distinction between “same host” and “same service” features is also represented in the table. According to the authors of [23], a disadvantage of the NSL-KDD dataset is that this dataset has been created two decades ago and therefore does not represent a realistic situation of recently encountered network and application-level attacks. However, the dataset continues to be used in research, for models training and for comparison. There is a huge amount of previous work incorporated in the NSL-KDD dataset that can be used for learning and comparison. Several new datasets for network intrusion detection have been proposed, such as the UNSW-NB15 dataset [24][25] and the CICIDS2017 dataset [23]. Different types of network security attacks are evenly distributed between the UNSW-NB15 training and testing sets. All attack types included in the UNSW-NB15 test set have previously been introduced in the UNSW-NB15 train set. The NSL-KDD and CICIDS2017 datasets are created by capturing both normal network traffic and attacks in a simulated environment. UNSW-NB15 is generated using a combination of normal activities and synthetic attack behaviors created using IXIA Perfect Storm. The field of network security is dynamic, and attack strategies evolve continuously. Any machine learning model applicable to network security would need to continuously be retrained on new datasets that are representative of current attacks.

## 5. Data preprocessing

NSL-KDD is already split into a training (KDDTrain+.txt) and testing dataset (KDDTest+.txt). Additionally, the training dataset is split into a training and validation dataset. The validation set is necessary to estimate the generalizing capacity of the model on new previously unseen data.

Table 2. *NSL-KDD feature categories*

<table>
<tr><th colspan="2">Basic (K1-K9)</th></tr>
<tr><td colspan="2">(aggregated packet header data from packets associated with one connection)<br>1 duration<br>2 protocol_type<br>3 service<br>4 flag<br>5 src_bytes<br>6 dst_bytes<br>7 land<br>8 wrong_fragment<br>9 urgent</td></tr>
<tr><th colspan="2">Content (K10-K22)</th></tr>
<tr><td colspan="2">(information extracted from the data portion of the packets)</td></tr>
<tr><td>10 hot<br>11 num_failed_logins<br>12 logged_in<br>13 num_compromised<br>14 root_shell<br>15 su_attempted<br>16 num_root</td><td>17 num_file_creations<br>18 num_shells<br>19 num_access_files<br>20 num_outbound_cmds<br>21 is_host_login<br>22 is_guest_login</td></tr>
<tr><th colspan="2">Traffic (K23-K41)</th></tr>
<tr><td colspan="2">(statistics about previous connections)</td></tr>
<tr><td>time-based traffic features<br>(connections from the last 2 seconds are considered)<br>23 count<br>24 srv_count<br>25 serror_rate<br>26 srv_serror_rate<br>27 rerror_rate<br>28 srv_rerror_rate<br>29 same_srv_rate<br>30 diff_srv_rate<br>31 srv_diff_host_rate<br>“same host” features:<br>K23, K25, K27, K29, K20<br>same destination IP as in the current connection record<br>“same service” features:<br>K24, K26, K28, K31<br>same destination port (same service) as in the current connection record</td><td>connection-based traffic features<br>(the last 100 connections are considered)<br>32 dst_host_count<br>33 dst_host_srv_count<br>34 dst_host_same_srv_rate<br>35 dst_host_diff_srv_rate<br>36 dst_host_same_src_port_rate<br>37 dst_host_srv_diff_host_rate<br>38 dst_host_serror_rate<br>39 dst_host_srv_serror_rate<br>40 dst_host_rerror_rate<br>41 dst_host_srv_rerror_rate<br>“same host” features:<br>K32, K34, K35, K36, K38, K40<br>same destination IP as in the current connection record<br>“same service” features:<br>K33, K37, K39, K41<br>same destination port (same service) as in the current connection record</td></tr>
</table>

The features of the NSL-KDD dataset are divided into three groups: Numeric, Nominal/Categorical, and Binary [1]. Table 3 shows how the features in the NSL-KDD dataset are grouped into categories according to the data type. NSL-KDD dataset contains five binary, three categorical features and the remaining are numerical features. Binary features can take a value of either 0 or 1 (i.e., true, or false), depending on whether the condition was met during the connection or not. Categorical features are of type string and take values from a discrete, unordered set.

Table 3. *NSL-KDD feature categories (according to data type)*

| | |
|---|---|
| Numeric | 1, 5, 6, 8, 9, 10, 11, 13, 16, 17, 18, 19, 20, 22-41 |
| Categorical | 2, 3, 4 |
| Binary | 7, 12, 14, 15, 21 |

We refer to these features as “categorical” because the value they take indicates a specific category to which a sample belongs. For instance, the protocol_type feature can take one of three possible values, ['tcp', 'udp', 'icmp'], meaning that one of these protocols was used during the connection. However, neural networks cannot be trained on string values. There are several techniques available to transform categorical features into a form that is suitable for training a neural network. For this project, categorical features were prepared using the technique one-hot encoding (Fig. 2). For each possible value of the feature, a new binary feature is created (this feature can take a value of either 0 or 1 – true or false) [1]. For each sample in the dataset, exactly one of these binary features is assigned a value of 1 - the feature referring to the category this sample belongs to. Features created using this technique are strongly correlated, which is not suitable for applying machine learning algorithms, so an additional common practice of discarding one of the newly added binary features is applied, so that the features are not strongly correlated, and the model can achieve better results. As explained in [1], we usually have full knowledge of all categorical features in the dataset, either because we defined them or because the dataset provided this information. NSL-KDD does not include a detailed list of values for categorical features, so they are obtained using available methods from the libraries. The protocol_type feature has only three distinct values (Fig. 2). However, the service feature has many distinct values. All application-level protocols recognized by tcpdump can be found in /etc/protocols on a Linux system. The one-hot encoding technique could consume a lot of system memory and will increase the number of features considerably.

## 6. Implementation details

The generate_model() function creates a Sequential model with five Dense hidden layers. Each of these Dense hidden layers has the units parameter set to 128. The activation function parameter in each of the Dense hidden layers is set to activation='relu'. The output layer has the activation function parameter set to activation='sigmoid'. The value 'relu' is used to set the activation function of the hidden layers to the rectified linear function ReLu, while the activation function of the output layer is to sigmoid function. The sigmoid function always returns a value between 0 and 1, meaning that the output of the model will be a probability score. The compile() method is called before model training.

## 7. Metrics used to evaluate the classification model

First we define two classes using labels consistent with the terminology in NSL-KDD:

- „attack“ – (positive) this network connection is malicious
- „normal“ – (negative) this is normal network traffic

Precision and recall are common metrics used to evaluate the classifier:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

To evaluate the binary classification model, a confusion matrix [26] is used. This matrix is a 2x2 matrix that depicts all four possible outcomes when evaluating binary classification models:

| | |
|---|---|
| ***True negative*** A true negative occurs when the model correctly predicts the negative class (normal network traffic was correctly classified as "normal"). | ***False positive*** A False positive occurs when the model incorrectly predicts the positive class (normal network traffic was incorrectly classified as an "attack"). |
| ***False negative*** A False negative occurs when the model incorrectly predicts the negative class (a malicious network connection was classified as „normal“ traffic). | ***True positive*** A True positive occurs when the model correctly predicts the positive class (a malicious network connection was correctly classified as an „attack“). |

Other metric is accuracy computed as the percentage of examples correctly classified. Accuracy is not considered to be a helpful and comprehensive metric for this task and will not be used to evaluate the classification model during training. Despite achieving high accuracy, the model could still return a high number of false positives and false negatives that are considered as incorrect predictions. Additionally, the time needed to train the model was tracked with the purpose of choosing a model that can run in a reasonable amount of time on all data in the large training set.

## 8. Results and discussion

The proposed prototype takes the features of a network connection as input data and returns a probability of a connection to be malicious (a number between 0.0 and 1.0). A decision boundary is introduced to determine if the returned probability is going to be interpreted as a normal or malicious connection. In the three training runs discussed in this paper, a decision boundary of 80% is used. If the model returns a probability score greater than 0.8, the connection is classified as an attack. The objective of this research is to choose the model parameters and the format and representation of the data for the learning algorithm that give as a result a low number of incorrect predictions. Given that the application area is network intrusion detection, the aim is to create a prototype that results in a lower number of false negatives despite the cost of increasing the number of false positives. This tradeoff is preferable because many false negative alerts mean that the model would fail to detect attacks more frequently [27].

Since recall [28] is the percentage of actual positives that were correctly classified, a conclusion can be drawn that the larger the value for recall is, the fewer false negatives are returned. For this reason, during the model training process, recall is calculated on the validation dataset and used as a validation metric. To determine the optimal number of epochs for model training, EarlyStopping from the Keras Callbacks API is used. Since the goal of training is to maximize the value of the recall, the training loop will check at the end of every epoch whether recall is increasing. Once recall is found to no longer increase for several consecutive epochs, the training procedure terminates. The resulting model is the one that has the best value for recall. Three separate training runs were performed, each attempting to improve the results obtained from the previous training run. During the first training run, the model (model_init) was trained using the NSL-KDD dataset, with data preprocessed as described in Section 5. Figure 3 shows the confusion matrix generated from this training run. As confusion matrix illustrates the initial training run resulted in a relatively low number of false positives. The number of false negatives returned by this model is used as a base value. The following two variations of the model attempt to improve the result for false negatives by applying transformation functions to the data. To make the three training runs more comparable, the weights of the initial model were kept in a checkpoint file and loaded before the training of the next variation of the model. To improve the performance of the model, the second training run considers the characteristics of the numeric features. An attempt was made to apply transformation functions that would result in values that lead to more representative characteristics of the two classes (normal and malicious traffic) and to better classification results.

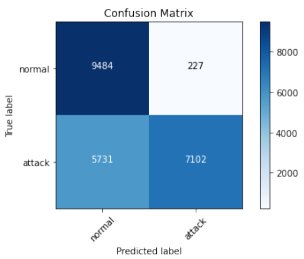


Figure 3. *model_init*

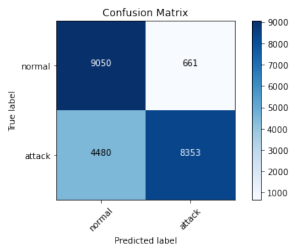


Figure5. *model_log*

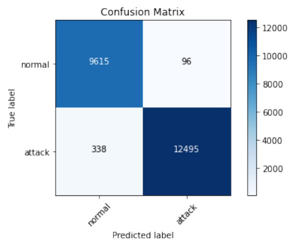


Figure 6. *model_sscaler*

The features in the „numeric" category can be divided into two subcategories: rates and integers. Due to the nature of the rates (they represent rates, percentages), the values in these columns are already in the range of [0.0, 1.0], which is suitable for training a neural network. Hence, the data in these columns is kept in its original form, as is provided in the dataset. The results of transformations of the columns src_bytes and dest_bytes which belong to the integers are presented in Figure 4. There is a significant range of values between the min and max values in the samples and furthermore many samples have a zero value for these features. To make the range of values smaller, a logarithmic function to the values in these columns was applied. The histograms on the right represent the distribution of the src_bytes and dist_bytes columns after the transformations were applied. After converting the columns src_bytes and dest_bytes to log space, the second training run was performed. The resulting model is referred to as model_log (Figure 5). The confusion matrix corresponding to model_log shows some decrease in the number of false negatives compared to model_init. However, this decrease in false negatives comes at the cost of increasing the number of false positives.

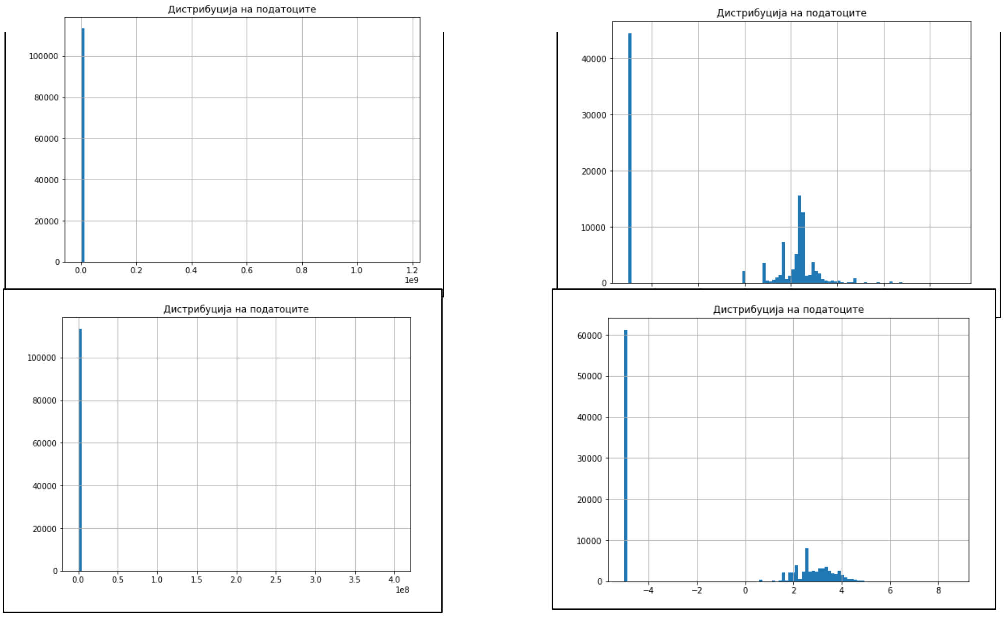


Figure 4. *Histograms of the features src_bytes and dest_bytes before (left) and after (right) the transformation*

The next step in the attempt to improve the model performance was to „standardize" the values in the numeric columns. For src_bytes and dest_bytes, the log space values were standardized. In terms of statistics, „standardization" refers to a data scaling technique that results in the feature having a mean of 0 and standard deviation of 1. Standardization was applied because machine learning models have poor performance when individual features do not have a normal (Gaussian) distribution. To standardize the data StandardScaler from the scikit-learn library [29] was applied. The resulting model is referred to as model_sscaler. Figure 6 shows the confusion matrix for model_sscaler. We can see that model_sscaler results in relatively small values for the numbers off the main diagonal of the confusion matrix, indicating incorrect predictions, which was the aim of the research. Table 4 summarizes the results produced by the three training runs. Converting features with a large range of values into log space was used to create model_log. Model_log showed improved accuracy and recall with a 4% decrease in precision. Scaling selected features further improved the performance of the model. Model_sscaler has an overall accuracy of 98% and recall of 97.366%. This model has the lowest false positive rate achieved during the three training runs. In Table 5 we present the values of the characteristics according to which we compared different approaches to network intrusion detection for the solution proposed in this paper. The main contribution of our approach is in the data preprocessing techniques which enable the selection of the model parameters that give a low number of incorrect predictions.

Table 4. *Results of the data analysis with data preprocessing*

| | accuracy | precision | recall |
|---|---|---|---|
| model_init | 0.73572 | 0.96903 | 0.55342 |
| model_log | 0.77196 | 0.92667 | 0.65090 |
| model_sscaler | 0.98075 | 0.99238 | 0.97366 |

We used the NSL-KDD dataset and our learning algorithms are implemented using Keras and Tensorflow. Categorical features are preprocessed using one-hot encoding and we applied scaling methods for some numerical features. The overall accuracy is 98% which is comparable to current approaches that use machine learning and deep learning techniques.

Table 5. Characteristics of the proposed model relevant for comparison with current models listed in Table 2

| Dataset | Meth. | Eval. Metrics | Data Prepr. | Accur. | BIN or MC | FS | Software |
|---|---|---|---|---|---|---|---|
| NSL-KDD | ANN | confusion matrix accuracy precision recall | One-hot encoding and dropping one generated feature, Standard Scaler from scikit-learn on selected features | 98 | BIN | No | Keras with Tensorflow backend |

## 9. Conclusion

This research described, implemented, and analyzed a deep learning model for a binary classification of network traffic. Network intrusion detection is a possible application area for the proposed model. An artificial neural network architecture was implemented using Keras with a Tensorflow backend. The NSL-KDD dataset was used to train and test the model. The features in the NSL-KDD dataset were grouped according to data type. Categorical features were preprocessed using one-hot encoding. The numeric features were analyzed and data transformations were applied to improve the performance of the model. Three training runs were performed with different transformations of data to improve the classification performance. An overall accuracy of 98% and a recall of 97.366% was achieved by converting features with a large range of values to log space and scaling selected features. The obtained results suggest that it is possible to improve the classification performance of deep learning models by applying transformations to the features in the dataset. For future research, we can examine the effects of data transformations on new network intrusion detection datasets proposed in the literature, such as the UNSW-NB15 dataset and the CICIDS2017 dataset.

References

[1] Chio C., Freeman D.: (2018) Machine Learning and Security, O'Reilly Media, Inc.

[2] Oriyano S.: (2016) Certified Ethical Hacker Version 9 Study Guide, Sybex

[2] Ahmad Z., Khan A. S., Shiang C. W., Abdullah J., Ahmad F. (2021) "Network intrusion detection systems: A systematic study of machine learning and deep learning approaches", Transactions on Emerging Telecommunications Technologies, vol. 32, e4150

[4] Tavallaee M., Bagheri E., Lu W., Ghorbani A., (2009) "A Detailed Analysis of the KDD CUP 99 Data Set", IEEE symposium on computational intelligence for security and defense applications (pp.1-6)

[5] Naseer S., Saleem Y., Khalid S., Bashir M. K., Han J., Iqbal M. M., Han., K. (2018) "Enhanced Network Anomaly Detection Based on Deep Neural Networks", IEEE Access, vol. 6 (pp. 48231-48246)

[6] Vinayakumar R., Alazab M., Soman K. P., Poornachandran P., Al-Nimrat A., Vankatraman S.: (2019) Deep Learning Approach for Intelligent Intrusion Detection System, IEEE Access, vol. 7 (pp. 41525-41550)

[7] Naseer, S., Saleem, Y. (2018) „Enhanced Network Intrusion Detection using Deep Convolutional Neural Networks", KSII Trans. Internet Inf. Syst., vol. 12 (pp. 5159–5178)

[8] Cao, B., Li, C., Song, Y., Qin, Y., Chen, C. (2022) „Network Intrusion Detection Model Based on CNN and GRU", Appl. Sci., 12, 4184.

[9] Ahmad, I., Ul Haq, Q.E., Imran, M., Alassafi, M.O., AlGhamdi, R.A., (2022) „An Efficient Network Intrusion Detection and Classification System", Mathematics, 10, 530

[10] Su T., Sun H., Zhu j., Wang S, Li Y.: (2020) BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset, IEEE Access, vol. 8 (pp. 29575-29585)

[11] Anaconda3 Individual Edition, https://www.anaconda.com/products/individual(accessed: January 2022)

[12] Documentation for Tensorflow 2.5.0, https://www.tensorflow.org/versions/r2.5/api_docs/python/tf/keras (accessed: January 2022)
[13] Documentation for Keras, https://keras.io/api/ (accessed: January 2022)
[14] Documentation for scikit-learn 0.24.2, https://scikit-learn.org/0.24/ (accessed: January 2022)
[15] Documentation for matplotlib, https://matplotlib.org/stable/users/index.html (accessed: January 2022)
[16] Documentation for pandas, https://pandas.pydata.org/pandas-docs/ (accessed: January 2022)
[17] Documentation for Jupyter Notebook, https://docs.jupyter.org/en/latest/ (accessed: January 2022)
[18] UCI Knowledge Discovery in Databases Archive, KDD Cup 1999 Data https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed: January 2022)
[19] MIT Lincoln Laboratory, 1998 DARPA Intrusion Detection evaluation https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset
[20] McHugh J.: (2000) Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, ACM Transactions on Information and System Security, vol. 3, no. 4, (pp. 262–294)
[21] MIT Lincoln Laboratory, DARPA Intrusion Detection Evaluation: Intrusion Detection Attacks Database, https://archive.ll.mit.edu/ideval/docs/attackDB.html (accessed: January 2022)
[22] Carnegie Mellon University Software Engineering Institute Blog, Traffic Analysis for Network Security: Two Approaches for Going Beyond Network Flow Data, https://insights.sei.cmu.edu/blog/traffic-analysis-for-network-security-two-approaches-for-going-beyond-network-flow-data/
[23] Sharafaldin I., Lashkari A. H., Ghorbani A.: (2018) Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization, Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018) (pp. 108-116)
[24] Nour M., Slay J., (2015) „UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), IEEE
[25] Nour M., Slay J., (2016) „The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset." Information Security Journal: A Global Perspective, vol.25, (pp. 1-14)
[26] Classification: True vs. False and Positive vs. Negative, https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative (accessed: January 2022)
[27] TensorFlow Tutorials, https://www.tensorflow.org/tutorials/structured_data/imbalanced_data
[28] Classification: Precision and Recall,https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (accessed: Jan. 2022)
[29] Documentation for scikit-learn: StandardScaler, https://scikit-learn.org/0.24/modules/generated/sklearn.preprocessing.StandardScaler.html (accessed: Jan. 2022)

**Anastasija Samardziska**
Goce Delcev University,
Faculty of Computer Science, "Krste Misirkov" 10-A, North Macedonia
*E-mail: anastasija.102036@student.ugd.edu.mk*

**Cveta Martinovska Bande**
Goce Delcev University,
Faculty of Computer Science, "Krste Misirkov" 10-A, North Macedonia
*E-mail: cveta.martinovska@ugd.edu.mk*