**CHAPTER 1**

**INTRODUCTION**

**1.1 BACKGROUND STUDY**

This study proposes the development of a network-based intrusion detection system (IDS) using the combination of Convolutional Neural Network and K- Nearest Neighbor algorithm with the NSL-KDD dataset. IDS play a crucial role in protecting computer networks from various security threats and attacks. Traditional rule-based IDS systems have limitations in detecting new and sophisticated attacks, which necessitates the application of machine learning algorithms. Machine learning-based IDS can improve detection accuracy and reduce false positives by learning from data and identifying patterns indicative of potential attacks. The proposed system will be implemented in python programming language using the Scikit-learn library in Jupyter notebook and trained on a large dataset of preprocessed network traffic data. The system aims to achieve a high detection rate, low false-positive rate, and fast response times. Evaluation will be conducted in a simulated network environment to test the system's effectiveness in detecting potential attacks and contribute to the development of advanced security systems. (Chimphlee and Chimphlee July 2023), (Unnisa A et al.Mar 2022).

**1.2 PROBLEM STATEMENT**

Intrusion detection systems (IDSs) are one of the promising tools for protecting data and networks; many classification algorithms, such as neural network (NN), Naive Bayes (NB), decision tree (DT), and support vector machine (SVM) have been used for IDS in the last decades. However the current IDSs face a few weaknesses, like high misleading positive rates, slow reaction times, and trouble in recognizing obscure dangers. These constraints can result in missed attacks and an expanded responsibility for security groups.

To address these weaknesses, further developed IDS that use machine learning algorithm can be created. Such an IDS can lessen bogus up-sides, increment location exactness, and further develop reaction times to likely attacks. The proposed arrangement includes utilizing machine learning algorithm to investigate network traffic and recognize designs that demonstrate pernicious movement. The framework will be prepared on huge datasets of known attacks to guarantee it can precisely distinguish likely dangers. The proposed IDS will work on the proficiency and exactness of the security framework, empowering faster reactions to expected dangers and decreasing the responsibility on security groups. (Nizomova Mar 2023).

**1.3 AIMS AND OBJECTIVES**

The aim of this project is to develop an intrusion detection system that is using convolutional neural network and k-nearest neighbor algorithm to detect potential security threats accurately and the objectives of the project are:

1. Review and analyze existing literature on intrusion detection systems and machine learning algorithms to establish a solid theoretical foundation.
2. To develop and train a machine learning model on the dataset (NSL-KDD)
3. Design and implement a Convolutional Neural Network model for intrusion detection.
4. Design and implement a K-nearest Neighbor model for intrusion detection.
5. Compare and evaluate the performance of the developed models using appropriate metrics such as accuracy, precision, recall, and F1-score.
6. To deploy the model to monitor the network and detect potential attacks in real-time.
7. To evaluate the performance of the system and compare it to existing IDSs.

**1.4 SCOPE OF STUDY**

This project focuses on the development and evaluation of an Intrusion Detection System using Convolutional Neural Network and K-nearest Neighbor algorithms. The study will utilize the NSL-KDD dataset. The performance evaluation will be based on metrics such as accuracy, precision, recall, and F1-score

.

**1.5 SIGNIFICANCE OF STUDY**

The proposed IDS bring forth numerous advantages, encompassing the alleviation of security teams' workload, enhancement of detection accuracy, and prompt response to potential attacks. By leveraging these IDS, organizations can effectively mitigate the risk of data breaches and minimize the impact caused by successful attacks. Moreover, this project holds the potential to contribute to the advancement of sophisticated security systems, empowering organizations to safeguard against emerging cyber threats. (Miao et al. 2023)

**1.6 PROJECT LAYOUT**

The organizational structure outlined below is adhered to in the project report:

**Chapter 1:** Introduction

In this chapter, an overview of the research project is presented. It encompasses an introduction to the project, a delineation of the problems under scrutiny, the study's goals and objectives, the research's importance, the study's scope, and the layout of the project.

**Chapter 2:** Literature Review

This chapter presents a thorough examination of the pertinent literature associated with the research subject. It encompasses a review of intrusion detection systems, machine learning algorithms (particularly Convolutional Neural Network and K-Nearest Neighbor Algorithm), as well as other pertinent concepts and studies within the field.

**Chapter 3:** Methodology

In this chapter, the emphasis is placed on the methods to be used in building the intrusion detection systems. It elucidates the current procedures and methods employed in intrusion detection, pinpoint the limitations and issues associated with the existing systems, and presents a justification for the creation of a novel system.

**Chapter 4:** Result and Discussion

This chapter of the project report or research paper presents the outcomes of the work done and provides an analysis and interpretation of those results. This section is crucial as it showcases the effectiveness of your Intrusion Detection System (IDS) using the K-Nearest

Neighbors (KNN) algorithm and Convolutional neural network with the NSL-KDD dataset.

**Chapter 5:** Conclusion and Recommendation

In this chapter, after the outcomes derived from the experiments conducted on the developed system are showcased, it encompasses the assessment of the Convolutional Neural Network and K-Nearest Neighbor models' performance utilizing suitable metrics. The results are scrutinized and deliberated upon, emphasizing the strengths and limitations of the models.

**REFERENCES**

In this section, all the references cited throughout the project report are listed, adhering to the appropriate citation style.

**Chapter 2**

**LITERATURE REVIEW**

**2.1 DATA MINING AND CHURN ANALYSIS**

The combination of Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithm for Intrusion Detection Systems (IDS) can be a powerful approach for analyzing and detecting network intrusions. In this context, churn and data mining analysis refers to the application of these techniques to identify patterns of network behavior that may indicate potential intrusions or anomalies.

Convolutional Neural Networks (CNNs) are a type of deep learning algorithm commonly used in image recognition tasks. However, they can also be applied to other domains, such as network traffic analysis. CNNs are well-suited for detecting patterns in network data due to their ability to automatically learn and extract relevant features from input data.

K-Nearest Neighbor (KNN) algorithm, on the other hand, is a simple yet effective classification algorithm that can be applied to both supervised and unsupervised learning problems. In the context of IDS, KNN can be used to classify network traffic as normal or malicious based on its similarity to previously labeled instances.

The combination of CNN and KNN in IDS involves using CNNs to extract meaningful features from network traffic data and then feeding these features into a KNN classifier for the final classification. The CNN component learns hierarchical representations of the input data, capturing both low-level and high-level features. These learned features are then passed to the KNN classifier, which uses the similarity measure to assign a label to the input instance.

The advantage of using CNNs in IDS is their ability to automatically learn relevant features without the need for manual feature engineering. This is particularly useful in detecting novel and previously unseen attacks. The KNN algorithm complements the CNN by providing a flexible and adaptable classification mechanism based on similarities with known instances.

Churn analysis and data mining techniques can be applied to the dataset used for training and testing the CNN-KNN IDS. Churn analysis involves studying patterns and behaviors that indicate changes or transitions in the network, such as a sudden increase in network traffic or a shift in network usage. Data mining techniques, including clustering and association rule mining, can be used to uncover hidden patterns and relationships within the network data.

By combining CNNs and KNN with churn analysis and data mining, it is possible to develop a more robust and accurate IDS that can effectively detect intrusions and anomalies in network traffic. However, it is important to note that the performance of such a system heavily relies on the quality and representativeness of the training data, as well as the careful selection and tuning of the CNN and KNN parameters. (K K A, Abdullah, et al Aug 2016) (Jyoti, Aug 2017).

**2.2 CUSTOMER CHURN PREDICTION**

The application of Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithm in Intrusion Detection Systems (IDS) for customer churn analysis involves leveraging these techniques to identify patterns and predict potential churn events in a customer base.

Customer churn refers to the phenomenon where customers discontinue using a product or service. It is a critical concern for businesses as it can lead to revenue loss and impact profitability. By analyzing customer behavior and identifying indicators of potential churn, businesses can take proactive measures to retain customers and minimize churn.

CNNs can be employed in customer churn analysis to extract meaningful features from customer data. This can include various types of data such as customer demographics, transaction history, usage patterns, customer interactions, and more. CNNs excel at learning hierarchical representations of input data, capturing patterns and relationships that may be indicative of churn.

The KNN algorithm can be utilized for the final classification of customer churn. Once the CNN has learned relevant features from the input data, these features are used as input to the KNN classifier. The KNN algorithm assigns labels to new customer instances based on the similarity with previously labeled instances. In this case, the labels would indicate whether a customer is likely to churn or not.

The combination of CNNs and KNN offers a powerful approach for customer churn analysis as it allows for automated feature extraction and flexible classification based on similarities. The CNN component learns complex patterns and relationships from the data, while the KNN algorithm leverages this learning to make predictions on new instances.

To perform customer churn analysis, historical customer data can be used for training and testing the CNN-KNN model. The data can be preprocessed and transformed into a suitable format for input into the CNN. The CNN then learns the features that are most relevant for predicting churn. These features are passed to the KNN classifier, which applies the similarity measure to classify new customer instances as churn or non-churn.

It is important to note that the success of a CNN-KNN model for customer churn analysis depends on several factors, including the quality and representativeness of the training data, appropriate feature selection, hyperparameter tuning, and model evaluation. Additionally, domain expertise and knowledge of customer behavior can further enhance the effectiveness of the churn analysis system. (Karyakina and Melnikov 2017), (Jing, Changran Mar 2023)

**2.2.1 CRM (CUSTOMER RELATIONSHIP MANAGEMENT)**

Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms in Intrusion Detection Systems (IDS) for customer relationship management (CRM) analysis are not directly applicable to CRM analysis. CNNs and KNN are primarily utilized in the context of image recognition and pattern classification tasks, and their direct application to CRM analysis is not common.

However, in the realm of CRM analysis, other machine learning algorithms and techniques can be beneficial. For instance, clustering algorithms such as K-Means or DBSCAN can be used to segment customers into groups based on their similarities, allowing businesses to tailor their marketing strategies and customer engagement accordingly.

Additionally, decision tree-based algorithms like Random Forests or Gradient Boosting can be employed to predict customer behavior, such as identifying potential churners or determining customer preferences for personalized recommendations.

Natural Language Processing (NLP) techniques can also be utilized to analyze customer sentiments and extract valuable insights from customer feedback or social media data. This can aid in understanding customer satisfaction levels, identifying issues, and improving overall customer experience.

While CNNs and KNNs may not be directly applicable to CRM analysis, various other machine learning techniques and algorithms can be leveraged to gain insights into customer behavior, improve customer satisfaction, and optimize business strategies. (Klishevich) (Sun et al.Feb 2023)

**2.3 TYPES OF CHURNERS**

In the context of customer churn analysis, various types of churners can be identified using a combination of Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms in Intrusion Detection Systems (IDS). Churners are customers who exhibit different patterns or behaviors that indicate a higher likelihood of discontinuing their relationship with a product or service. Here are some common types of churners that can be identified:

1. Silent Churners: These are customers who gradually reduce their engagement with a product or service without explicitly indicating their intention to churn. They may decrease their usage, stop interacting, or show a decline in activity. By analyzing patterns and changes in customer behavior, CNNs can capture these subtle indications, while KNN can classify new instances as potential silent churners based on their similarity to known silent churners.
2. Price-Sensitive Churners: Some customers are more sensitive to pricing changes and may churn if they perceive a lack of value or if they find a more cost-effective alternative. By analyzing customer transaction history and purchase patterns, CNNs can identify customers who are more likely to be price-sensitive. KNN can then classify new customers based on their similarity to known price-sensitive churners.
3. Competitive Churners: These churners are motivated by competition. They may switch to a competitor's product or service due to better features, pricing, or promotional offers. By monitoring market trends, social media sentiments, and competitor activities, CNNs can capture signals of potential competitive churners. KNN can be used to classify new customers based on their similarity to known competitive churners.
4. Early-Life Churners: Some customers churn shortly after their initial interaction or onboarding. Early-life churners may have had a poor onboarding experience, encountered difficulties in product usage, or failed to see value quickly. By analyzing customer onboarding data, usage patterns, and feedback, CNNs can identify features indicative of early-life churners. KNN can then classify new customers based on their similarity to known early-life churners.
5. Dissatisfied Churners: These churners are dissatisfied with the product or service and may churn due to issues such as poor customer support, product quality, or lack of features. By analyzing customer feedback, surveys, and support interactions, CNNs can capture sentiments and indicators of customer dissatisfaction. KNN can then classify new customers based on their similarity to known dissatisfied churners.

It's important to note that the identification of specific types of churners using CNNs and KNN algorithms requires appropriate training data that includes labeled instances of different churn types. The effectiveness of the churn analysis system relies on the quality of the training data, feature selection, and careful tuning of the CNN and KNN parameters. (Sun et al Feb 2023)

**2.3.1 NEURAL NETWORKS (NNM)**

In the context of Intrusion Detection Systems (IDS), Neural Networks (NNs) can be used as a broader term encompassing different types of neural network architectures, including Convolutional Neural Networks (CNNs) and other variants like Recurrent Neural Networks (RNNs) or Multilayer Perceptrons (MLPs). While CNNs and K-Nearest Neighbor (KNN) algorithm have their specific applications in IDS, let's discuss the general application of Neural Networks (NNs) in IDS.

Neural Networks (NNs) have been widely used in IDS for their ability to learn complex patterns and detect anomalies in network traffic. They can effectively capture both spatial and temporal dependencies in data, making them suitable for detecting intrusions.

Here's how Neural Networks (NNs) can be applied in IDS:

1. Multilayer Perceptrons (MLPs): MLPs are the basic building blocks of neural networks and can be applied to IDS. They consist of multiple layers of interconnected neurons, with each neuron performing weighted computations and activation functions. MLPs can be trained on labeled network traffic data to classify instances as normal or malicious based on learned patterns.
2. Recurrent Neural Networks (RNNs): RNNs are suitable for analyzing sequential data, making them useful in IDS for capturing temporal dependencies in network traffic. They have memory capabilities that enable them to process data with sequential characteristics, such as packets in a network stream. RNNs can detect intrusions by learning the normal patterns of network behavior and identifying deviations from those patterns.
3. Convolutional Neural Networks (CNNs): CNNs are particularly effective in capturing spatial dependencies and extracting features from structured data such as images or, in the case of IDS, network packets. CNNs utilize convolutional layers to automatically learn and extract relevant features from the input data. They can be applied to network traffic analysis by considering network packets as structured data and using CNNs to identify patterns associated with intrusions.
4. Hybrid Architectures: Hybrid architectures that combine different types of neural networks can be employed in IDS. For example, a combination of CNN and RNN can capture both spatial and temporal dependencies in network traffic, enabling more accurate intrusion detection.

The choice of the appropriate neural network architecture depends on the characteristics of the network traffic data and the specific requirements of the IDS. Training neural networks for IDS typically involves using labeled datasets that include both normal and malicious instances to learn the patterns of normal behavior and detect anomalies.

It's worth noting that the successful application of neural networks in IDS requires careful preprocessing of data, appropriate feature engineering, hyper-parameter tuning, and the availability of quality labeled datasets. Moreover, keeping up with the evolving threat landscape and continuously updating the IDS with new data and models is crucial for maintaining its effectiveness. (Khelifi et al May 2017.) (K K A et al. Aug 2016)

**2.3.2 LINEAR REGRESSION MODEL (LRM)**

Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms are commonly used in Intrusion Detection Systems (IDS), Linear Regression Models (LRMs) are not typically applied directly in the context of IDS. LRMs are primarily used for regression tasks, where the goal is to predict a continuous outcome variable based on input features.

In IDS, the main objective is typically binary classification—determining whether a given instance is normal or represents an intrusion. As such, classification algorithms like logistic regression, support vector machines (SVMs), decision trees, random forests, or neural networks (such as CNNs or RNNs) are more commonly utilized.

That said, if the goal is to predict certain continuous metrics or behavior related to the network traffic or system performance, LRM could potentially be used in an IDS context. For instance, one could use LRM to predict the duration or intensity of an intrusion or to estimate the likelihood of certain events occurring based on historical data.

In such cases, the LRM would involve finding a linear relationship between the input features and the continuous outcome variable. The coefficients of the linear regression model would be estimated through methods like ordinary least squares (OLS) or gradient descent, and the model could be used to make predictions on new instances.

It's important to note that when it comes to IDS, the focus is typically on detecting and classifying intrusions accurately, which involves handling categorical or binary outcomes. While LRMs are not commonly used for this purpose, other classification algorithms are better suited to address the specific requirements of IDS. (Khelifi et al. May 2017)

**2.3.3 NAIVE BAYES MODEL (NBM)**

Intrusion Detection Systems (IDS) commonly employ various machine learning algorithms, including Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms. However, Naive Bayes Model (NBM) is also a viable approach for IDS, particularly for certain types of data and classification tasks.

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem and assumes the independence of features. Despite its simplicity, Naive Bayes has been successful in many applications, including text classification and spam filtering.

Here's how Naive Bayes Model (NBM) can be applied in IDS:

1. Feature Extraction: Before applying NBM, appropriate features need to be extracted from the network traffic data. These features could include packet characteristics, protocol information, traffic statistics, or behavioral patterns. The independence assumption of Naive Bayes assumes that the features are conditionally independent given the class label.
2. Training: The NBM is trained using a labeled dataset that consists of network traffic instances labeled as normal or malicious. During training, the algorithm estimates the conditional probabilities of each feature given the class labels. This involves calculating the prior probabilities of the classes and the likelihoods of each feature value given the class labels.
3. Classification: After training, NBM can be used to classify new instances of network traffic as normal or malicious. The algorithm calculates the posterior probabilities of the classes given the features and assigns the instance to the class with the highest probability. This classification is based on the assumption of conditional independence among the features.
4. Evaluation and Refinement: The performance of the NBM is evaluated using various metrics, such as accuracy, precision, recall, or F1 score. If necessary, the model can be refined by adjusting feature selection, handling missing data, or incorporating more sophisticated techniques to handle dependencies among the features.

Naive Bayes Model (NBM) can be particularly effective in IDS when the assumption of feature independence holds reasonably well. However, it may not capture complex dependencies and interactions between features, which other algorithms like CNNs or KNNs are better suited for. Additionally, the quality and representativeness of the training data play a crucial role in the performance of NBM.

It's important to note that IDS typically employ a combination of multiple machine learning algorithms to achieve more accurate and robust intrusion detection. Naive Bayes can be one of the tools in the IDS toolkit, but it is often used alongside other algorithms to enhance overall detection capabilities. (Xashimov and Khaydarova Apr 2023),(Jumamuratova May 2022).

**2.3.4 DECISION TREE (DT)**

Decision trees are another popular machine learning algorithm commonly used in Intrusion Detection Systems (IDS). Although Decision trees are not directly related to Convolutional Neural Networks (CNNs) or K-Nearest Neighbor (KNN) algorithms, they are valuable for their interpretability and ability to handle categorical and numerical data. Here's how Decision trees can be applied in IDS:

1. Feature Selection: Before constructing a decision tree, it is necessary to select relevant features from the network traffic data. These features can include packet attributes, network statistics, or behavioral patterns. The goal is to identify features that have a significant impact on classifying instances as normal or malicious.
2. Tree Construction: The decision tree is constructed using the selected features and a training dataset labeled with normal and malicious instances. The tree is built by recursively partitioning the data based on different feature thresholds, aiming to maximize the separation between the classes. The process involves selecting the most informative features at each level of the tree.
3. Classification: Once the decision tree is constructed, it can be used to classify new instances by traversing the tree from the root to a leaf node. At each internal node, a decision is made based on a specific feature, and the instance is directed to the appropriate child node. At the leaf node, a classification label (normal or malicious) is assigned to the instance.
4. Evaluation and Refinement: The performance of the decision tree model is evaluated using metrics such as accuracy, precision, recall, or F1 score. The tree can be refined by pruning branches, adjusting parameters, or incorporating ensemble techniques like Random Forests to improve accuracy and generalization.

Decision trees are advantageous in IDS due to their interpretability, as the rules and decisions made by the tree are easily understandable. They can capture non-linear relationships between features and class labels, making them suitable for detecting complex intrusion patterns.

However, decision trees have limitations, such as their tendency to over fit the training data. This can be mitigated by techniques like pruning or ensemble methods. Additionally, decision trees may struggle with handling high-dimensional data or capturing dependencies that require more sophisticated algorithms like CNNs or RNNs.

In practice, IDS often employ a combination of different machine learning algorithms, including decision trees, to achieve robust intrusion detection. Each algorithm brings its strengths, and by combining them, the overall performance and accuracy of the IDS can be improved. (Bayimbetova May 2022) (Xashimov and Khaydarova Apr 2023)

**2.3.5 SUPPORT VECTOR MACHINE (SVM)**

Support Vector Machine (SVM) is a popular machine learning algorithm that can be effectively applied in Intrusion Detection Systems (IDS). While SVM is not directly related to Convolutional Neural Networks (CNNs) or K-Nearest Neighbor (KNN) algorithms, it offers robust classification capabilities and is well-suited for handling high-dimensional data. Here's how SVM can be utilized in IDS:

1. Feature Extraction: Prior to applying SVM, relevant features need to be extracted from the network traffic data. These features can include packet attributes, traffic statistics, or behavioral patterns. The selection of appropriate features plays a crucial role in the performance of the SVM model.
2. Training: The SVM model is trained using a labeled dataset that contains network traffic instances labeled as normal or malicious. During training, SVM identifies the optimal hyperplane that maximally separates the classes while minimizing the classification error. Various kernel functions, such as linear, polynomial, or radial basis function (RBF), can be employed to handle non-linear decision boundaries.
3. Classification: After training, the SVM model can be used to classify new instances of network traffic as normal or malicious. The model maps the instances into the feature space and determines which side of the decision boundary they fall on. Instances on one side are classified as normal, while those on the other side are classified as malicious.
4. Evaluation and Refinement: The performance of the SVM model is assessed using metrics such as accuracy, precision, recall, or F1 score. The model can be refined by optimizing hyper-parameters, selecting appropriate kernel functions, or handling imbalanced data through techniques like weighted SVM or oversampling/under-sampling methods.

SVM offers several advantages for IDS. It can handle high-dimensional data effectively and is less prone to over-fitting. SVM is also robust against outliers and can handle imbalanced datasets. Additionally, SVM provides a clear decision boundary, allowing for interpretability and understanding of classification results. However, SVM may face challenges when dealing with large-scale datasets due to computational requirements. It is also important to ensure appropriate feature selection and preprocessing to achieve optimal results. In practice, SVM is often used in combination with other machine learning algorithms to enhance the overall accuracy and robustness of intrusion detection systems. (K K A et al. Aug 2016) (Xashimov and Khaydarova Apr 2023)

**2.3.6 FUZZY LOGIC ALGORITHM**

Fuzzy Logic Algorithm (FLA) is another approach that can be applied in Intrusion Detection Systems (IDS), alongside Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms. Fuzzy logic allows for handling uncertainty and imprecision in data, making it suitable for modeling complex and ambiguous concepts. Here's how the Fuzzy Logic Algorithm can be used in IDS:

1. Fuzzy Rule-Based System: Fuzzy Logic Algorithm employs a rule-based system where linguistic rules are defined based on expert knowledge or domain expertise. These rules capture the relationships between input variables (features) and output variables (class labels) in the IDS.
2. Membership Functions: Fuzzy logic utilizes membership functions to represent the degrees of truth or membership of an instance to a particular fuzzy set. Membership functions are defined for each input variable, mapping the input values to a degree of membership in the associated fuzzy set. These membership functions capture the uncertainty and fuzziness in the data.
3. Fuzzy Rule Evaluation: The fuzzy rules in the IDS are evaluated based on the input values of the instance being classified. The membership values obtained from the membership functions are used to determine the activation strength of each rule. Fuzzy logic combines the activated rules to obtain an aggregated output.
4. Defuzzification: After the fuzzy rules are evaluated, defuzzification is performed to obtain a crisp output or classification decision. Various defuzzification methods, such as the centroid method or max membership method, can be used to obtain the final output.

Fuzzy Logic Algorithm offers several benefits in IDS. It can handle imprecise and uncertain data effectively, allowing for more flexible decision-making compared to traditional binary classification algorithms. Fuzzy logic can also capture complex relationships and dependencies between input variables, enhancing the IDS's ability to detect intrusions.

However, it is important to note that Fuzzy Logic Algorithm requires careful design and knowledge engineering to define appropriate membership functions and fuzzy rules. Expert domain knowledge plays a crucial role in building an effective fuzzy rule-based system for intrusion detection. Additionally, the interpretability of fuzzy logic-based models allows for understanding the decision-making process.

In practice, Fuzzy Logic Algorithm can be used alongside other machine learning algorithms, such as CNNs and KNN, to enhance the overall performance and accuracy of IDS. Each algorithm brings its unique strengths, and their combination can lead to more robust and effective intrusion detection systems. (Soleiman and Fetanat July 2013)

**2.3.7 EVOLUTIONARY LEARNING DATA MINING**

Evolutionary Learning Data Mining (ELDM) is an approach that combines evolutionary algorithms and data mining techniques to address complex problems, including intrusion detection in Intrusion Detection Systems (IDS). Although Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms are not directly associated with ELDM, they can be integrated into the framework to enhance the IDS performance. Here's how ELDM can be applied in IDS:

1. Problem Representation: In ELDM, the problem of intrusion detection is typically represented as a search problem where the goal is to find an optimal solution. The representation can include various features derived from network traffic data, system logs, or other relevant information.
2. Evolutionary Algorithm: An evolutionary algorithm, such as Genetic Algorithm (GA) or Particle Swarm Optimization (PSO), is employed to explore the solution space and find the best set of features or parameters for intrusion detection. The algorithm iteratively evolves a population of candidate solutions using principles inspired by natural evolution, such as selection, crossover, and mutation.
3. Fitness Evaluation: During each iteration of the evolutionary algorithm, the fitness of each candidate solution (individual) is evaluated based on its ability to detect intrusions accurately. This evaluation can be performed using a performance metric like accuracy, precision, recall, or F1 score. CNNs or KNN algorithms can be employed as part of the fitness evaluation to assess the performance of the candidate solutions.
4. Evolutionary Search: The evolutionary algorithm drives the search process by selecting the best-performing individuals and applying genetic operators (e.g., crossover and mutation) to generate new candidate solutions. The search continues until a termination condition is met, such as reaching a maximum number of generations or achieving a desired level of performance.
5. Solution Analysis and Deployment: Once the evolutionary algorithm converges and identifies a set of optimal features or parameters, they can be used in conjunction with CNNs, KNN, or other classification algorithms for intrusion detection in the IDS. These algorithms utilize the selected features or parameters to classify new instances as normal or malicious.

ELDM offers the advantage of exploring a large solution space and discovering optimal feature subsets or parameter configurations for intrusion detection. By integrating CNNs or KNN algorithms into the fitness evaluation process, ELDM can leverage their classification capabilities to assess the performance of candidate solutions accurately. It's worth noting that the success of ELDM in IDS depends on factors such as appropriate problem representation, fitness evaluation, evolutionary operators, and termination conditions. Furthermore, the selection and integration of suitable data mining algorithms, such as CNNs and KNN, are crucial to achieve effective intrusion detection within the ELDM framework. (K K A et al. Aug 2016 )

**2.3.8 K-MEANS CLUSTERING**

While Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms are widely used in Intrusion Detection Systems (IDS), K-means clustering can also be applied as a complementary technique for IDS analysis. K-means clustering is an unsupervised machine learning algorithm used for data clustering and grouping. Here's how K-means clustering can be utilized in IDS:

1. Feature Extraction: Before applying K-means clustering, relevant features need to be extracted from the network traffic data. These features can include packet attributes, statistical information, or behavioral patterns. The goal is to represent the network traffic instances in a suitable feature space.
2. Data Preprocessing: If necessary, the feature data should be preprocessed by normalizing or scaling the features to ensure that all the features are on similar scales. This step is important to avoid biases due to differences in the feature ranges.
3. K-means Clustering: In K-means clustering, the algorithm aims to partition the data into K clusters, where K is a pre-defined parameter. The algorithm iteratively assigns data points to the nearest cluster centroid based on a distance metric (usually Euclidean distance) and updates the centroid's position based on the newly assigned points. This process continues until convergence is reached.
4. Cluster Analysis: Once the clustering process is complete, the clusters formed by K-means can be analyzed to gain insights into network traffic patterns. Clusters can represent different types of network behaviors or potentially malicious activities. Outliers or instances that do not fit well into any cluster can be considered as anomalies that require further investigation.
5. Anomaly Detection: After clustering, instances that are dissimilar or distant from their assigned cluster centroids can be flagged as potential anomalies. These instances may represent novel or previously unseen intrusion patterns that warrant attention.
6. Classification and Alert Generation: The clustered instances, along with the anomaly detection results, can be used in combination with other classification techniques, such as CNNs or KNN, to classify network traffic as normal or malicious and generate alerts for potential intrusions.

K-means clustering offers a useful approach in IDS for identifying patterns and anomalies in network traffic data. By grouping similar instances into clusters, it provides a way to capture common behaviors and detect deviations from those behaviors. Integrating K-means clustering with other classification algorithms like CNNs or KNN can enhance the overall performance of the IDS by incorporating both unsupervised and supervised learning approaches.

It's important to note that the effectiveness of K-means clustering in IDS depends on appropriate feature selection, determining the optimal number of clusters (K), and interpreting the results for actionable insights. Additionally, regular updates and retraining of the clustering model are essential to adapt to evolving network behaviors and intrusion patterns. (K K A et al. Aug 2016 )

**2.3.9 ANT COLONY OPTIMIZATION**

While Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms are commonly used in Intrusion Detection Systems (IDS), Ant Colony Optimization (ACO) is a metaheuristic optimization algorithm that is not directly associated with CNNs or KNN. However, ACO can be used in conjunction with CNNs or KNN as a complementary technique in IDS to improve the detection and classification of network intrusions. Here's how ACO can be applied in IDS:

1. Feature Selection: Before applying ACO, relevant features need to be selected from the network traffic data. These features can include packet attributes, statistical information, or behavioral patterns. The goal is to identify informative features that are most relevant for intrusion detection.
2. Ant Colony Optimization: ACO is an optimization algorithm inspired by the behavior of ants in finding optimal paths to food sources. In the context of IDS, ACO can be used to optimize the selection of features, the configuration of parameters, or the generation of rules for intrusion detection. ACO uses pheromone trails and heuristic information to guide the search process and iteratively improves the solutions over time.
3. Fitness Evaluation: During each iteration of the ACO algorithm, the fitness of the candidate solutions (ant solutions) is evaluated based on their performance in detecting intrusions. The performance can be measured using metrics such as accuracy, precision, recall, or F1 score. CNNs or KNN algorithms can be employed as part of the fitness evaluation to assess the performance of the candidate solutions.
4. Solution Construction: ACO iteratively constructs new solutions by probabilistically selecting features, parameters, or rules based on the pheromone trails and heuristic information. The process mimics the exploration and exploitation behavior of ants to find optimal solutions for intrusion detection.
5. Integration with CNNs or KNN: The solutions obtained from ACO, which represent feature subsets, parameter configurations, or rule sets, can be used in conjunction with CNNs, KNN, or other classification algorithms for intrusion detection. These algorithms utilize the selected features, optimized parameters, or generated rules to classify network traffic instances as normal or malicious.
6. Evaluation and Refinement: The performance of the integrated system is evaluated using metrics to assess the effectiveness of ACO in improving intrusion detection accuracy. The system can be refined by adjusting ACO parameters, exploring different feature combinations, or employing ensemble techniques to further enhance performance.

The integration of ACO with CNNs, KNN, or other classification algorithms allows for enhanced feature selection, parameter optimization, or rule generation in IDS. By leveraging ACO's optimization capabilities, the IDS can improve its accuracy and efficiency in detecting network intrusions.

It's important to note that the success of ACO in IDS depends on appropriate problem representation, pheromone trail updates, heuristic information, and convergence criteria. Furthermore, the integration of ACO with other algorithms requires careful parameter tuning and understanding of the specific requirements of the intrusion detection problem.

In practice, the combination of CNNs, KNN, and ACO can offer a comprehensive approach to intrusion detection by leveraging the strengths of each technique and enhancing the overall performance of the IDS. (K K A et al. Aug 2016 )

**2.4 DATA MINING**

Data mining techniques, including Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms, are widely applied in Intrusion Detection Systems (IDS) to analyze and extract valuable insights from network traffic data.

Here's how data mining can be employed in IDS using CNNs and KNN algorithms:

1. Data Preprocessing: Before applying data mining techniques, network traffic data needs to be preprocessed. This involves tasks such as data cleaning, removing irrelevant or redundant attributes, handling missing values, and transforming the data into a suitable format for analysis.
2. Feature Extraction: Relevant features need to be extracted from the network traffic data to represent different aspects of the network behavior. These features can include packet attributes, traffic statistics, temporal patterns, or frequency distributions. Feature extraction is crucial for capturing meaningful information from the data.
3. Convolutional Neural Networks (CNNs): CNNs are powerful deep learning algorithms that can automatically learn hierarchical representations and extract relevant features from input data. In the context of IDS, CNNs can be applied to analyze network traffic data by considering it as structured data. CNNs learn patterns and relationships in the data, allowing for the detection of intrusions or anomalies based on learned features.
4. K-Nearest Neighbor (KNN) Algorithm: KNN is a simple yet effective classification algorithm that can be applied in IDS. KNN classifies instances based on their similarity to labeled instances in the training data. In the context of IDS, KNN can be used to classify network traffic instances as normal or malicious based on their proximity to previously labeled instances.
5. Model Training and Evaluation: CNNs and KNN models need to be trained on labeled data, where instances are classified as normal or malicious. The training process involves optimizing model parameters and adjusting hyper-parameters for better performance. The trained models are then evaluated on test data to assess their accuracy, precision, recall, or other performance metrics.
6. Intrusion Detection and Alert Generation: Once the models are trained and evaluated, they can be utilized for intrusion detection. New instances of network traffic can be fed into the models, and based on the classification results, intrusions can be identified and appropriate alerts or notifications can be generated.

Data mining techniques like CNNs and KNN algorithms provide valuable tools for IDS by extracting meaningful patterns, learning relevant features, and performing accurate classification. They enable IDS to identify network intrusions, detect anomalies, and take timely actions to ensure network security.

It's important to note that the performance of data mining techniques in IDS relies on various factors such as the quality and representativeness of the training data, appropriate feature selection, parameter tuning, and continuous monitoring and adaptation to evolving intrusion patterns. (Al Khoufi 2017), (K K A et al. Aug 2016).

**2.5 STUDIES OF CHURN ANALYSIS IN TELECOMMUNICATION INDUSTRY**

Churn analysis in the telecommunication industry is a crucial area where machine learning techniques, including Convolutional Neural Networks (CNNs) and K-Nearest Neighbor (KNN) algorithms, can be applied. Churn analysis focuses on predicting and understanding customer churn, i.e., the likelihood of customers switching to a competitor or discontinuing their services. Here's how CNNs and KNN algorithms can be utilized in churn analysis in the telecommunication industry:

1. Convolutional Neural Networks (CNNs): CNNs can be employed to analyze various data types in churn analysis, including customer demographics, usage patterns, call records, billing information, and customer interaction data. CNNs excel at learning hierarchical representations of input data and can capture complex patterns and relationships that may be indicative of churn. By training CNNs on historical churn data, they can learn to identify significant features and predict churn probability for new customers.
2. K-Nearest Neighbor (KNN) Algorithm: KNN algorithm is a non-parametric classification algorithm that can be used in churn analysis to identify customers who are likely to churn based on similarities to previously labeled churners. KNN measures the distance between feature vectors of new customers and those of labeled churners to determine their proximity. If a new customer is close to known churners, it suggests a higher likelihood of churn.

By combining CNNs and KNN algorithms, churn analysis in the telecommunication industry can benefit from the strengths of both approaches:

1. Feature Extraction with CNNs: CNNs can automatically extract relevant features from diverse data sources, enabling the identification of complex patterns and relationships. For example, CNNs can learn to recognize patterns in call duration, frequency of calls, customer complaints, or changes in usage behavior that are indicative of potential churn.
2. Classification with KNN: Once CNNs have extracted relevant features, KNN can be used for the final classification of customers as churners or non-churners. KNN considers the similarities between new customer feature vectors and previously labeled churners, assigning labels based on the nearest neighbors.

It's important to note that the success of CNNs and KNN algorithms in churn analysis depends on various factors, including the quality and representativeness of the training data, appropriate feature selection, hyperparameter tuning, and evaluation metrics. Additionally, domain expertise and knowledge of the telecommunication industry can enhance the effectiveness of churn analysis systems.

Overall, CNNs and KNN algorithms offer valuable tools for churn analysis in the telecommunication industry, enabling businesses to predict and proactively address customer churn, improve customer retention strategies, and enhance overall customer satisfaction. (Xashimov and Khaydarova Apr 2023 ), (Bayimbetova May 2022).

**2.5.1 ADVANTAGES OF DECISION TREE**

Decision trees are widely used in Intrusion Detection Systems (IDS) for their simplicity, interpretability, and effectiveness in capturing complex relationships in data. Here are some advantages of using decision trees in IDS:

1. Interpretability: Decision trees provide a clear and intuitive representation of decision-making, making it easy to understand and interpret the rules and decisions made by the IDS.
2. Feature Importance: Decision trees can rank the importance of features, aiding in feature selection and understanding the significance of different factors in intrusion detection.
3. Handling Non-Linear Relationships: Decision trees can capture complex non-linear relationships between features and class labels, enabling the detection of intricate intrusion patterns.
4. Robustness to Irrelevant Features: Decision trees are relatively robust to irrelevant features, automatically ignoring those that do not significantly contribute to the classification task.
5. Handling Mixed Data Types: Decision trees can handle both categorical and numerical features, making them versatile in analyzing different types of data in IDS.
6. Scalability: Decision trees can efficiently handle large datasets and can be easily updated or expanded with new data, allowing the IDS to adapt to evolving intrusion patterns.
7. Ensemble Methods: Decision trees can be combined using ensemble methods to enhance performance, reducing over-fitting and improving generalization capabilities in intrusion detection. (Xashimov and Khaydarova Apr 2023 ), (Bayimbetova May 2022).

**2.5.2 DISADVANTAGES OF DECISION TREE**

While decision trees have several advantages in Intrusion Detection Systems (IDS), they also come with certain disadvantages. Here are some of the key disadvantages of decision trees:

1. Over-fitting: Decision trees are prone to over-fitting, especially when the tree becomes deep and complex. This can occur when the tree captures noise or specific details of the training data, leading to poor generalization on unseen instances.
2. Lack of Robustness: Decision trees are sensitive to small changes in the training data, and even slight variations can lead to significantly different tree structures. This lack of robustness makes decision trees less stable compared to other algorithms, as a small change in the dataset can result in a different set of decisions and rules.
3. Difficulty Handling Continuous Variables: Decision trees can struggle with effectively handling continuous variables, as they require multiple split points to partition the data. This can lead to a loss of information and suboptimal splits, particularly when the variable ranges are large or unevenly distributed.
4. Biased towards Features with Many Categories: Decision trees tend to favor features with many categories or levels. This bias can result in the overemphasis of those features, overshadowing potentially important but less categorical features.
5. Limited Learning of Complex Relationships: While decision trees can capture non-linear relationships, they may struggle to learn complex relationships involving multiple variables or interactions. This limitation can hinder their ability to detect intricate intrusion patterns that require a more sophisticated modeling approach.
6. Difficulty Handling Imbalanced Data: Decision trees can produce biased results when dealing with imbalanced datasets, where one class significantly outweighs the other. They tend to prioritize the majority class, leading to challenges in accurately detecting the minority class intrusions. It's important to consider these disadvantages when using decision trees in IDS and to employ strategies such as pruning, ensemble methods, or combining decision trees with other algorithms to address these limitations and improve overall performance. (Xashimov and Khaydarova Apr 2023 ), (Bayimbetova May 2022).

**2.5.3 ADVANTAGES OF NEURAL NETWORK**

Neural networks, such as Convolutional Neural Networks (CNNs), offer several advantages in Intrusion Detection Systems (IDS):

1. Non-Linear Relationships: Neural networks can capture complex non-linear relationships between features and class labels, enabling them to detect intricate intrusion patterns that may not be easily captured by linear models.
2. Feature Learning: Neural networks can automatically learn relevant features from raw data, reducing the need for manual feature engineering and allowing for the discovery of hidden patterns and representations in the network traffic data.
3. High Flexibility: Neural networks are highly flexible and can handle various data types, including images, text, and numerical data. This versatility makes them suitable for analyzing different aspects of network traffic and adapting to different intrusion scenarios.
4. Generalization Ability: Neural networks have the ability to generalize from training data to unseen instances, enabling them to detect new and previously unseen intrusion patterns based on learned representations.
5. Scalability: Neural networks can scale effectively to handle large and complex datasets, making them suitable for IDS applications with extensive network traffic data.
6. Parallel Processing: Neural networks can take advantage of parallel processing capabilities in modern hardware, allowing for efficient computation and faster training times. (Li Oct 1995), (Bayguinov et al. Sep. 2010)

**2.5.4 DISADVANTAGES OF NEURAL NETWORK.**

However, neural networks also have certain disadvantages:

1. Computational Complexity: Neural networks, especially deep architectures like CNNs, can be computationally expensive to train and require significant computational resources.
2. Black Box Nature: Neural networks are often considered black box models, meaning it can be challenging to interpret and understand the decision-making process. This lack of interpretability can limit the ability to explain the reasoning behind intrusion detection outcomes.
3. Over-fitting: Neural networks are prone to over-fitting when the model becomes too complex or when the training data is limited. Regularization techniques and careful model selection can mitigate this issue.
4. Data Dependency: Neural networks require a large amount of labeled training data to achieve optimal performance. Insufficient or unrepresentative training data can lead to suboptimal results.
5. Hyperparameter Sensitivity: Neural networks have several hyper-parameters that need to be tuned, such as the number of layers, number of neurons, and learning rate. Proper hyperparameter tuning is essential for achieving good performance.

Despite these disadvantages, the advantages of neural networks in terms of their ability to learn complex patterns and adapt to diverse intrusion scenarios make them a valuable tool in IDS, especially when used in conjunction with other techniques. (Li Heng Oct 1995), (Bayguinov et al. Sep. 2010)

**Chapter 3**

**METHODOLOGY**

**3.1 PROJECT UNDERSTANDING AND GOALS:**

Define the scope and objectives of the project. Understand the purpose of the IDS and its role in network security. Determine the target environment for deployment, whether it's a local network or a cloud-based network.

**3.2 SYSTEM DESIGN**

1. Input Data: The IDS will take network traffic data as input. The NSL-KDD dataset used in this study was gotten from Kaggle but can be collected from various sources, such as network packets or flow records, and should include features like source and destination IP addresses, port numbers, protocol type, duration, number of packets, etc.
2. Preprocessing Module: The raw input data needs to be preprocessed before feeding it to the KNN classifier. The preprocessing module will handle tasks like:

* Handling missing values: Decide on an appropriate strategy (imputation, removal) for missing data
* Feature scaling: Normalize the features to ensure they are on a similar scale. Feature encoding: Convert categorical features to numerical values using techniques like one-hot encoding.
* Data splitting: Divide the preprocessed data into training and testing sets. K-Nearest Neighbors (KNN) Algorithm: Implement the KNN algorithm or use libraries like scikit-learn to perform classification. Choose a suitable distance metric (e.g., Euclidean distance) for calculating the distance between data points. Determine the value of "K" (number of neighbors) through hyper-parameter tuning.

1. Training Module: The training module will take the preprocessed training data and feed it to the KNN classifier to train the model. The trained KNN model will learn the patterns in the training data and be ready for classification.
2. Testing and Evaluation Module: The preprocessed testing data will be provided to the trained KNN model for classification. The output of the model will be compared with the actual labels to evaluate the IDS's performance. Calculate various performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix to assess the effectiveness of the IDS.
3. Hyper-parameter Tuning: Implement techniques like cross-validation or grid search to find the optimal value of hyper-parameters (e.g., K, distance metric). This step helps improve the model's performance and generalization capabilities.
4. Model Update Module: Periodically update the KNN model to incorporate new data and adapt to changes in the network environment. This step ensures that the IDS stays effective against emerging threats. Deployment and Real-Time Monitoring: Deploy the trained KNN model in the production environment to monitor network traffic in real-time. Set up a continuous monitoring system to track the IDS's performance and detect any potential issues or anomalies.
5. User Interface (Optional): Design a user-friendly interface to interact with the IDS. The interface can provide visualizations of network traffic, detected intrusions, and overall system status.
6. Alerting System: Implement an alerting mechanism to notify network administrators or security personnel in case of any detected intrusions or suspicious activities. The system can generate alerts via email, SMS, or integration with other monitoring tools.

Documentation and Maintenance: Document the system design, data preprocessing steps, model implementation details, hyper-parameter choices, and evaluation results. Provide clear instructions on how to deploy, use, and maintain the IDS system. Regularly monitor and maintain the system to ensure it stays effective and up-to-date. (K K A et al. Aug 2016)

**3.3 METHODOLOGY**

The methodology for creating an intrusion detection system using both the K-Nearest Neighbor (KNN) and Convolutional Neural Network (CNN) algorithms typically involves the following steps:

**3.3.1 DATASET EXPLORATION:**

The NSL-KDD dataset, which was collected from Kaggle, was the source of the data utilized in this study. The NSL-KDD dataset, which offers a broad range of labeled network traffic data for developing and accessing intrusion detection models, is a frequently used benchmark in the field of cyber-security. The NSL-KDD dataset was created to solve some of the shortcomings and difficulties that the original KDD Cup 1999 dataset encountered. The reliability and performance assessment of intrusion detection models were impacted by redundancy and an imbalance between normal and intrusive cases in the KDD Cup 1999 dataset. The NSL-KDD dataset, which includes a well curated and well-balanced sample of network traffic statistics, was developed to address these problems. The dataset is freely accessible and may be found in other cyber-security research databases or the repository of the National Institute of Standards and Technology (NIST). There are 125,972 instances in all, each one representing a different network connection, and it includes both typical and invasive activity. One categorical goal variable, "outcome," which divides each connection into distinct intrusion kinds or typical behavior, and 41 continuous and binary features, which capture various network connection properties, are included in the dataset. (Su et al. 2020) (Gurung et al. Mar 2019)

**3.3.1.1 DATA PREPROCESSING**:

After acquiring the NSL-KDD dataset and importing it into a Jupyter notebook, several preprocessing steps were taken to prepare the data for training machine learning models. The raw dataset was first inspected to understand the features, data types and value distributions. Then, the following key tasks were undertaken:

1. Data Cleaning
2. Feature Scaling
3. Encoding Categorical Variables
4. Train-Test Split (Cubaynes and Fretwell May 2022)

### 3.3.1.2 DATA CLEANING

### After obtaining the dataset, its contents are thoroughly examined to spot any irregularities that can obstruct further analysis. This thorough examination addresses any problems like missing data, configures the proper column names, and gets the dataset ready for further processing steps. The dataset is prepared for future exploration and analysis by completing these critical data pretreatment procedures, preserving its integrity, and helping the creation of reliable models. (Alagbe et al. Aug 2020)

### 3.3.1.3 FEATURE SCALING

A preprocessing method called feature scaling is used to standardize or normalize the variety of characteristics in a dataset. Assuring that all features contribute equally to the learning process in machine learning algorithms is the goal of feature scaling. Some characteristics may dominate the learning process when features have considerably differing scales, resulting in biased outputs and delayed convergence in some algorithms.

In this research, the ‘RobustScaler’ from scikit-learn python library is used for feature scaling. The `RobustScaler` scales features using the formula:

**XScaled** = X−Q1(X) /Q3(X) − Q1(X)

Where **X** is the original feature, **XScaled** is the scaled feature, Q1(X) is the first quartile (25th percentile) of **X**, and Q3(X) is the third quartile (75th percentile) of **X**. The RobustScaler employs the interquartile range (IQR) to scale the features, making it more robust to outliers compared to other scaling methods like the Min-Max scaler.

The scaling process ensures that all numerical features in the dataset have comparable scales, preventing any particular feature from dominating the learning process and contributing to a more stable and efficient model training. By using the robust scaling approach, the method becomes more resilient to outliers in the data, leading to more reliable and accurate models, especially when dealing with datasets containing potential extreme values. (Xashimov and Khaydarova Apr 2023).

### 3.3.1.4 ENCODING CATEGORICAL VARIABLE

In order to prepare the data for machine learning tasks, categorical variables must be encoded. This is because many machine learning algorithms demand numerical inputs. To be used in these techniques, categorical variables—which represent discrete categories or labels—need to be transformed into numerical representations.

The categorical characteristics are represented numerically in this study via one-hot encoding. One-hot encoding is a method that turns each category in the initial categorical feature into a binary column. The binary column for the category to which a data point belongs is set to 1, and all other binary columns for other categories in that feature are set to 0.

### Each category in the original categorical features will have its matching binary column as a consequence of one-hot encoding. The binary column for a category that a data point falls within will be 1, and all other binary columns for categories in that feature will be 0. By ensuring that category data is represented numerically properly, this procedure enables machine learning algorithms to handle the data efficiently and reliably. (Al-Shehari and Alsowail Sep. 2021)

### 3.3.1.5 TRAIN-TEST SPLIT

A critical stage in machine learning is the train-test split, which involves splitting the dataset into two distinct sets: the training set and the testing set. With this divide, the model may be trained on one set of data and assessed on another, giving an idea of how well it will perform on untried data. The main objective of train-test split is to evaluate the trained model's generalization to new, previously unobserved data, which is crucial for assessing its performance in the real world.

The train-test split ensures that the model is trained on a subset of the data and tested on data that hasn't been seen before, which is essential for correctly assessing the model's performance and avoiding over-fitting. The testing set acts as an independent validation set to see how well the model generalizes to new and unrecognized situations. The training set allows the model to learn patterns and relationships.(Stojanovic and Popovic 2007)

**3.3.1.6 K-NEAREST NEIGHBOR AND CONVOLUTIONAL NEURAL NETWORK:**

The non-parametric machine learning technique K-nearest neighbor (KNN) is used for classification and regression problems. The majority decision or average of the K nearest data points in the training dataset, calculated using a distance metric like Euclidean distance, determines the class label or projected value of a new data point in KNN. KNN may become computationally expensive for big datasets and high-dimensional spaces, despite being successful for smaller datasets and difficult decision boundaries. Despite being straightforward, KNN is useful in many real-world applications due to its versatility in handling different conditions and lack of assumptions.

Convolutional Neural Networks (CNNs) are deep learning algorithms designed for processing grid-like data like images. They employ layers for convolution, activation, pooling, and fully connected operations. CNNs learn intricate features from data and use them for tasks like classification. Training involves minimizing a loss function using back-propagation and optimization. CNNs have transformed fields by automatically learning features and are especially useful for tasks involving spatial relationships, such as image recognition and intrusion detection.

The Convolutional Neural Networks (CNNs) is used to enhance network security. CNNs excel in recognizing both known attack signatures and previously unseen anomalies, reducing false positives. CNNs enable deep packet inspection, behavioral analysis, and traffic classification, improving the accuracy of intrusion detection. However, they require proper training data, preprocessing, and model tuning, and are often part of a multi-layered security strategy. (Nizomova, Mar 2023)

**3.3.1.7 WORKING OF K-NEAREST NEIGHBOR**

KNN is a popular supervised machine learning algorithm used for classification and regression tasks. It works based on the principle of finding the "k" nearest data points to a given input sample and making predictions based on the majority class (for classification) or the average value (for regression) of those "k" neighbors.

Here's a mathematical representation of the KNN algorithm:

1. Given a dataset with "n" data points and their corresponding features and labels:

D = {(x₁, y₁), (x₂, y₂), ..., (xₙ, yₙ)}

Where xᵢ represents the features of the i-th data point, and yᵢ is its corresponding label.

1. For a new input sample "x", we want to predict its label "y\_hat".
2. Define a distance metric, typically Euclidean distance, to measure the similarity between data points. The Euclidean distance between two data points xᵢ and xⱼ is represented as: d(xᵢ, xⱼ) = √(Σ(xᵢₖ - xⱼₖ)²), for k = 1 to number\_of\_features.
3. Select the value of "k," which is the number of nearest neighbors to consider.
4. Find the "k" nearest neighbors of the input sample "x" from the dataset "D" based on the distance metric. Let's call this set of "k" nearest neighbors as "N(x)".
5. For classification:

-Determine the majority class among the labels of "N(x)".

-Assign this majority class as the predicted label "y\_hat" for the input sample "x."

Mathematically, for classification:

y\_hat = argmaxᵢ(Σ(1{yᵢ = c})), for c in N(x)

Where 1{condition} is an indicator function that returns 1 if the condition is true, otherwise 0.

1. For regression:

- Calculate the average value of the labels of "N(x)".

- Assign this average value as the predicted label "y\_hat" for the input sample "x."

Mathematically, for regression:

y\_hat = (1/|N(x)|) \* Σ(yᵢ), for yᵢ in N(x)

Where |N(x)| represents the number of elements in the set N(x).

The KNN technique is essentially represented mathematically in this way for both classification and regression problems. KNN is a straightforward and understandable method that is frequently utilized as the starting point for many machine learning issues. However, bear in mind that the performance of the algorithm can be considerably influenced by the selection of the distance measure and the value of "k". (Xashimov and Khaydarova Apr 2023), (Nizomova, Mar 2023)

Key Steps;

1. Implementing KNN Algorithm:

Write the KNN algorithm from scratch or use libraries like scikit-learn to implement it. Choose an appropriate distance metric (e.g., Euclidean distance) for calculating the distance between data points. Determine the value of "K" (number of neighbors) through hyper-parameter tuning techniques like cross-validation or grid search.

1. Model Training:

Train the KNN classifier on the training dataset. This involves providing the feature vectors and corresponding target labels.

1. Model Evaluation:

Test the trained model on the testing dataset to evaluate its performance. Calculate various performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix to assess the effectiveness of the IDS.

1. Performance Improvement:

Analyze the performance metrics and identify areas of improvement. Experiment with different hyper-parameters (e.g., K, distance metric) and evaluate their impact on the model's performance. Consider feature selection techniques to remove irrelevant or redundant features that might be affecting the model's performance.

1. Fine-tuning and Optimization:

Fine-tune the model based on the findings from the previous step to achieve better accuracy and generalization. Perform cross-validation to ensure the model's robustness and avoid over-fitting.

1. Deployment:

Once the model achieves satisfactory performance, deploy it in the target environment for real-time monitoring of network traffic. Set up a continuous monitoring system to track the IDS's performance and detect any potential issues or anomalies.

1. Monitoring and Maintenance:

Regularly monitor the IDS in the production environment to ensure it effectively detects and classifies network intrusions. Perform periodic updates and maintenance to keep the model up-to-date with emerging threats and changes in the network environment.

1. Documentation:

Document the entire project, including data preprocessing steps, model implementation details, hyper-parameter choices, evaluation results, and deployment instructions. Provide clear instructions on how to use and maintain the IDS system.

Throughout the project, maintain a strong focus on the overall security of the IDS itself, as it is a critical component for network protection. Also, consider involving domain experts and security professionals to validate the effectiveness of the IDS and its ability to detect real-world threats. (K K A et al. Aug 2016).

**3.3.1.8 CONVOLUTIONAL NEURAL NETWORK ALGORITHM**

Utilizing a collection of distinct decision trees created from randomized subsets of training data and features, the Convolutional Neural Network (CNN) is a deep learning approach designed for processing grid-like data like images. The network constructs hierarchical feature extractors in its layers through convolution, activation, and pooling operations during training. These layers learn intricate features which are then used for tasks like image classification. Unlike Random Forest, which ensembles decision trees, a CNN itself is a complex neural network. It autonomously learns features and patterns without explicit feature selection. Its advantages include automated feature learning, strong performance on image-based tasks, and adaptability to high-dimensional data. While both methods excel in classification tasks, CNNs are specialized for image data due to their convolutional layers that capture spatial hierarchies. (Ardison et al. Jun 2022)

**3.3.1.9 WORKING WITH CONVOLUTIONAL NEURAL NETWORK**

A Convolutional Neural Network (CNN) is designed for image processing tasks and is comprised of multiple layers that automatically learn hierarchical features from the data.

1. Layers and Feature Extraction: A CNN consists of convolutional layers, activation functions (like ReLU), and pooling layers. Each layer extracts increasingly complex features from the input images.
2. Training Data and Batches: The training data comprises labeled images. During training, batches of images are fed to the network. These batches are essential for stochastic gradient descent optimization.
3. Convolution Operation: The core operation involves convolving learnable filters (kernels) over input images to extract spatial features. These filters detect patterns like edges, textures, and shapes.
4. Activation Function: Activation functions introduce non-linearity to the network, helping it learn complex relationships within the data.
5. Pooling Layers: Pooling layers reduce spatial dimensions, aiding in translation invariance and controlling computational complexity.
6. Fully Connected Layers: After feature extraction, fully connected layers aggregate features and make final predictions.
7. Loss Function and Back-propagation: The network's performance is measured by a loss function (e.g., cross-entropy for classification). Back-propagation adjusts weights using gradients to minimize the loss.
8. Epochs: Training occurs over epochs, where the entire dataset is processed. Each epoch refines the network's learned features.
9. Testing/Inference: Trained CNNs are evaluated on new, unseen images to make predictions.
10. Classification: For image classification, the final layer's outputs represent class probabilities. Softmax converts these into a probability distribution.
11. Ensemble-like Behavior: While not an ensemble like Random Forest, a CNN's layer hierarchy can be seen as an implicit ensemble of learned features.
12. Hyper-parameters: CNNs have various hyper-parameters, like the number of layers, filter sizes, and activation choices. They also require parameter tuning.
13. Transfer Learning: Pre-trained CNNs on large datasets (e.g., ImageNet) can be fine-tuned for specific tasks, boosting performance with less training data.
14. Feature Visualization: CNNs can learn features that correspond to specific patterns, which can be visualized to understand what the network has learned.
15. Application: CNNs excel at image-related tasks like image classification, object detection, and segmentation. They handle high-dimensional data well, learn complex patterns, and are used across domains.

CNNs are a powerful tool in machine learning, but they are designed specifically for image data and excel at learning intricate features and patterns in images. (Bayimbetova May 2022) (Nizomova Mar 2023)

**3.3.1.9.1 FRAMEWORK FOR THE SYSTEM**

1. Model Evaluation:

* Test the trained model on the testing dataset to evaluate its performance.
* Calculate various performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix to assess the effectiveness of the IDS.

1. Hyperparameter Tuning:

* Implement techniques like cross-validation or grid search to find the optimal value of hyperparameters (e.g., K, distance metric).
* This step helps improve the model's performance and generalization capabilities.

1. Performance Improvement:

* Analyze the performance metrics and identify areas of improvement.
* Experiment with different hyper-parameters (e.g., K, distance metric) and evaluate their impact on the model's performance.
* Consider feature selection techniques to remove irrelevant or redundant features that might be affecting the model's performance.

1. Model Update and Maintenance:

* Periodically update the KNN model to incorporate new data and adapt to changes in the network environment.
* This step ensures that the IDS stays effective against emerging threats. Deployment and

1. Real-Time Monitoring:

* Deploy the trained KNN model in the production environment to monitor network traffic in real-time.
* Set up a continuous monitoring system to track the IDS's performance and detect any potential issues or anomalies.

1. Alerting and Reporting:

* Implement an alerting mechanism to notify network administrators or security personnel in case of any detected intrusions or suspicious activities.
* Generate alerts via email, SMS, or integration with other monitoring tools.
* Create detailed reports to summarize the IDS's performance and the detected incidents.

1. User Interface (Optional):

* Design a user-friendly interface to interact with the IDS.
* The interface can provide visualizations of network traffic, detected intrusions, and overall system status.

1. Documentation and Knowledge Sharing:

* Document the entire project, including data preprocessing steps, model implementation details, hyperparameter choices, evaluation results, and deployment instructions.
* Provide clear instructions on how to use and maintain the IDS system.
* Share knowledge and findings with the team and stakeholders for better collaboration.

1. Security and Privacy Considerations:

* Ensure that the IDS itself is secure from attacks and unauthorized access.
* Respect privacy regulations and handle sensitive data appropriately.

1. Testing and Validation:

* Perform extensive testing and validation to ensure the IDS operates correctly and reliably.
* Conduct simulated attacks and assess how well the system detects and responds to them.

1. Continuous Improvement:

* Continuously monitor the performance of the IDS in the production environment and gather feedback from users.
* Use the feedback to make necessary improvements and updates to the system.

Remember that building an effective IDS is an iterative process that requires continuous monitoring, maintenance, and improvement to keep up with evolving threats and network environments. The methodology outlined above provides a framework for creating IDS using the CNN and KNN algorithm with the NSL-KDD dataset, but you may need to tailor it based on your specific project requirements and constraints. (K K A, Abdullah, et al Aug 2016)

**3.3.1.9.2 MODEL EVALUATION**

In the context of model evaluation and comparison the K-nearest Neighbors (KNN) and Convolutional Neural Network (CNN) models might be evaluated and compared as follows:

1. K-Nearest Neighbors (KNN) Evaluation:

* Accuracy: this measures the proportion of correctly classified instances out of the total instances. For KNN, it calculates how often the model predicts the correct class based on the majority class in the k-nearest neighbors. Higher accuracy indicates better performance.
* Precision and Recall: this is the ratio of true positive predictions to the total predicted positives. It highlights the model's ability to minimize false positives. Recall, on the other hand, is the ratio of true positives to the actual positives in the dataset. It assesses the model's capacity to capture actual positive instances.
* F1-Score: this combines precision and recall into a single metric. It provides a balanced evaluation, considering both false positives and false negatives. A higher F1-score indicates a model that achieves a good balance between precision and recall.

1. Convolutional Neural Network (CNN) Evaluation:

* Accuracy: As with KNN, accuracy is a key evaluation metric for CNNs. It reflects the network's ability to correctly classify instances within the test dataset.
* Precision and Recall: these are also relevant for CNNs, particularly in tasks involving imbalanced datasets. Precision measures the CNN's ability to correctly classify instances in the positive class, while recall evaluates how well the CNN captures actual positive instances.
* F1-Score: this is applicable to CNNs as well, providing a comprehensive assessment of model performance, particularly when precision and recall trade-offs exist. (Jabbarov, Mar 2023).

**3.3.1.9.3 MODEL COMPARISON**:

When comparing the KNN and CNN models:

1. Accuracy: Both models' accuracy should be considered. A higher accuracy indicates better overall performance.
2. Precision and Recall: Given the consideration of precision and recall, it's important to assess whether either model excels in minimizing false positives (precision) or capturing positive instances (recall).
3. F1-Score: The F1-score aids in understanding the models' balance between precision and recall.
4. Prediction Time: CNNs, being deep neural networks, tend to require more computational resources and time for prediction compared to KNN. The prediction time can impact real-time or time-sensitive applications.
5. Specificity and Sensitivity: In intrusion detection, specificity (true negative rate) and sensitivity (true positive rate) are also important metrics. Specificity evaluates the model's ability to correctly identify non-intrusive instances, while sensitivity assesses its ability to correctly identify intrusions.

Overall, a balanced evaluation of these metrics helps identify the optimal model that performs well across different aspects, ensuring effective intrusion detection with accurate predictions and manageable computational requirements. (Nizomova Mar 2023)

**3.3.2 DATA COLLECTION**

Data collection is a crucial step in building an Intrusion Detection System (IDS) using both the Convolutional Neural Network (CNN) and K-Nearest Neighbors (KNN) algorithm with the NSL-KDD: dataset. Proper data collection ensures that the IDS has sufficient and relevant data to learn patterns and accurately detect intrusions. Here are details about data collection:

1. NSL-KDD Dataset: The NSL “stands for” Network Security Lab and KDD refers to the Knowledge Discovery and Data Mining conference where the original dataset was introduced and it is a widely used dataset for network intrusion detection research. It contains a large collection of network traffic data, including both normal traffic and various types of attacks such as DoS (Denial of Service), Probe, R2L (Remote-to-Local), and U2R (User-to-Root) attacks. The dataset is labeled, meaning that each data instance is annotated with its corresponding class (e.g., normal, attack type).
2. Data Sources: NSL-KDD dataset which was collected from Kaggle was the source of the data utilized in this study. Data for the IDS can be collected from various sources, such as network traffic logs, network flow records, or packet captures. Packet captures (PCAPs) are often used to capture raw network packets, which can later be processed to extract features for analysis. Network flow records (NetFlow, sFlow, etc.) aggregate network traffic information at a higher level, making it more manageable for analysis.
3. Data Sampling: The NSL-KDD dataset contains a large number of instances, and it may not be practical to use the entire dataset for training and testing. Depending on the available computational resources and the size of the dataset, you may choose to sample a subset of data for training and testing the IDS.
4. Data Labeling: The NSL-KDD dataset comes with pre-labeled instances, where each instance is tagged with the corresponding class (normal or attack type). Data labeling is essential for supervised learning, where the IDS learns from labeled examples to make predictions.
5. Data Preprocessing: Before using the data for training, it needs to be preprocessed to handle missing values, normalize features, and encode categorical variables (if any). Preprocessing ensures that the data is in a suitable format for both the CNN and KNN algorithms.
6. Data Splitting: After preprocessing, the data is typically split into two subsets: the training set and the testing set. The training set is used to train the KNN classifier, while the testing set is used to evaluate its performance.
7. Data Security and Privacy: When dealing with network traffic data, it's essential to handle it with care to maintain security and privacy. If the data contains sensitive information, ensure that it is anonymized or encrypted before use. Adhere to data protection regulations and obtain necessary permissions for data usage.
8. Data Balance: Check for class imbalance in the dataset, as it may affect the performance of the IDS. If there is a significant class imbalance, consider using techniques like oversampling, undersampling, or generating synthetic data to balance the classes.
9. Data Quality Assurance: Ensure that the collected data is of high quality and free from errors or anomalies that may affect the IDS's training and performance. Perform data validation and verification to identify and correct any issues in the data.
10. Continuous Data Collection and Updates: An IDS is not a one-time solution. Network traffic and attack patterns change over time. Plan for continuous data collection to ensure that the IDS remains effective against new and emerging threats. Regularly update the dataset and retrain the model to keep the IDS up-to-date.
11. Data Storage and Accessibility: Determine how and where the data will be stored securely to ensure its availability for training and testing the IDS. Ensure that the data is accessible to the IDS without compromising security. Remember that data collection is a critical step, and the quality and diversity of the data greatly impact the effectiveness of the IDS. Choose an appropriate dataset like NSL-KDD or collect data from reliable sources that represent the network environment you want to protect. Additionally, consider collaborating with network administrators and security professionals to ensure that the collected data reflects real-world scenarios and threats. (K K A et al. Aug 2016)

**3.3.3 DATA FILTERING**

Data filtering is an essential step in the data preprocessing phase of building an Intrusion Detection System (IDS). The primary goal of data filtering is to remove or transform irrelevant, noisy, or redundant data, which can negatively impact the performance of the IDS model. Here are details about data filtering techniques:

1. Handling Missing Values: Data collected from real-world network environments may contain missing values due to various reasons such as network failures or data collection issues. Missing values need to be handled appropriately before training the IDS model. Common strategies for handling missing values include:
2. Removal: Remove data instances with missing values. This approach may not be ideal if the missing values are significant in number.
3. Imputation: Fill in missing values with estimated values. Popular imputation techniques include mean imputation, median imputation, and interpolation.
4. Outlier Detection and Removal: Outliers are data points that significantly deviate from the rest of the data. Outliers can be caused by measurement errors or rare events and may adversely affect the model's performance. Outlier detection techniques, such as z-score, modified z-score, or clustering-based methods, can identify and remove outliers from the dataset.
5. Feature Selection: Feature selection is the process of selecting the most relevant and informative features from the dataset. Selecting the right features helps reduce dimensionality and improve the model's efficiency and generalization. Techniques for feature selection include univariate feature selection, recursive feature elimination, and feature importance ranking.
6. Dimensionality Reduction: Dimensionality reduction techniques reduce the number of features in the dataset while preserving most of the information. Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are popular methods for dimensionality reduction.
7. Class Imbalance Handling: In intrusion detection datasets, the number of instances belonging to different classes (normal and attack) may be imbalanced. Class imbalance can lead the model to be biased toward the majority class and perform poorly on detecting the minority class (attacks). Techniques to address class imbalance include oversampling the minority class, undersampling the majority class, or using ensemble methods like SMOTE (Synthetic Minority Over-sampling Technique).
8. Noise Reduction: Noise in the data refers to irrelevant or random variations that do not carry useful information. Removing noise from the dataset helps the model focus on relevant patterns and improve its accuracy. Techniques such as median filtering, moving average, and low-pass filtering can be applied to reduce noise in the data.
9. Feature Transformation: Sometimes, certain features may not be in a suitable format for the model. Transforming features can make them more useful. For example, converting categorical features into numerical values using one-hot encoding or label encoding.
10. Data Balancing: Data balancing techniques aim to create a more balanced representation of different classes in the dataset. This is particularly important when dealing with imbalanced datasets. Techniques such as Synthetic Minority Over-sampling Technique (SMOTE) can be used to create synthetic instances of the minority class. The choice of specific data filtering techniques depends on the characteristics of the dataset and the requirements of the IDS project. Proper data filtering is crucial to enhance the quality of the data and improve the overall performance and accuracy of the IDS model. (Bayimbetova May 2022), (Xashimov and Khaydarova Apr 2023)

**3.3.4 FEATURE SELECTION**

Feature selection is a crucial step in the data preprocessing phase of building an Intrusion Detection System (IDS). The main objective of feature selection is to choose the most relevant and informative features from the dataset while discarding irrelevant or redundant ones. By reducing the number of features, feature selection helps improve the efficiency, interpretability, and generalization of the IDS model. Here are details about feature selection techniques:

1. Filter Methods: Filter methods evaluate the relevance of features based on their individual characteristics, independent of the model used for classification. Common filter methods include:

* Pearson Correlation: Calculate the correlation coefficient between each feature and the target class. Features with higher absolute correlation values are considered more relevant.
* Information Gain: Measure how much information a feature provides about the target class. Features with higher information gain are considered more informative.
* Chi-Square Test: Assess the independence between each feature and the target class using the chi-square statistic.

1. Wrapper Methods: Wrapper methods use the performance of the learning algorithm itself to evaluate the usefulness of features. They involve repeatedly training the model with different subsets of features and selecting the subset that yields the best performance. Common wrapper methods include Recursive Feature Elimination (RFE) and Sequential Feature Selection (SFS/SBS).
2. Embedded Methods: Embedded methods incorporate feature selection as part of the model training process. These methods automatically select the most important features during model training. Examples include L1 regularization (Lasso) and L2 regularization (Ridge) in linear models, which penalize less important features.
3. Tree-Based Feature Selection: Tree-based algorithms, such as decision trees and random forests, can provide feature importance scores during training. Features with higher importance scores are more relevant to the classification task. Principal Component Analysis (PCA): PCA is a dimensionality reduction technique that transforms the original features into a new set of uncorrelated features (principal components). The new components are ordered by their importance in explaining the variance in the data. Selecting the top "k" principal components can effectively reduce the number of features while preserving most of the information.
4. Forward Selection: Forward selection is a stepwise feature selection technique that starts with an empty set of features and iteratively adds the most relevant feature at each step until a stopping criterion is met. It is a computationally expensive method but can yield good results.
5. Backward Elimination: Backward elimination is another stepwise feature selection technique that starts with all features and iteratively removes the least relevant feature at each step until a stopping criterion is met.

Feature Importance from Model:

Some machine learning models, such as decision trees, random forests, and gradient boosting, provide feature importance scores as part of their training process. These scores can be used to identify the most important features for the classification task. It's important to note that the choice of feature selection technique depends on factors such as the size of the dataset, the number of features, the characteristics of the data, and the computational resources available. Experimenting with different feature selection methods and evaluating their impact on the IDS's performance can help identify the best approach for the specific project. (Fahimifar et al. Sept 2022)

**3.3.5 DATA PARTITIONING**

Data partitioning, also known as data splitting, is a critical step in building a machine learning model, including an Intrusion Detection System (IDS) using the K-Nearest Neighbors (KNN) algorithm. The purpose of data partitioning is to divide the available dataset into separate subsets for training, validation, and testing. This process is essential for evaluating the model's performance and ensuring its generalization to unseen data.

Here are details about data partitioning techniques:

1. Training Set: The training set is the largest subset of the dataset and is used to train the IDS model. It contains labeled instances (samples) with corresponding target labels (e.g., normal or attack) used to learn patterns and relationships between features and labels.
2. Validation Set: The validation set is a smaller subset of the dataset used for hyperparameter tuning and model selection. After training the model on the training set, hyperparameters (e.g., K for KNN) are tuned on the validation set to find the best configuration.
3. Testing Set: The testing set is an independent subset of the dataset used to assess the model's performance. It serves as an unbiased evaluation of the model's ability to generalize to new, unseen data. The model has not seen the testing data during training or hyperparameter tuning, ensuring a fair assessment of its effectiveness. (Ambure et al. Sept 2019).

**3.3.5.1 PARTITIONING TECHNIQUES:**

1. Random Split: The dataset is randomly shuffled, and instances are divided into training, validation, and testing sets based on a specified percentage ratio. Stratified Split: Stratified splitting ensures that the distribution of classes (e.g., normal and attack) is preserved across the training, validation, and testing sets. This helps prevent class imbalance issues during training and evaluation.
2. Split Ratios: The split ratios determine how much data is allocated to each subset. Common ratios are 70-15-15, 80-10-10, or 80-20 (for training-validation or training-testing splits). The choice of split ratios depends on the size of the dataset, the availability of data, and the specific requirements of the project. Cross-Validation: Cross-validation is a technique used to assess the model's generalization performance when the dataset is limited. In k-fold cross-validation, the data is divided into "k" subsets (folds). The model is trained and validated "k" times, each time using a different fold as the validation set and the remaining folds as the training set. Cross-validation helps provide more robust performance estimates, especially when the dataset is small.
3. Time Series Split: For time-series data, random splitting may not be appropriate because the order of data matters. Time series splitting involves dividing the data into contiguous temporal blocks. The earlier blocks are used for training, and the later blocks are used for testing to simulate real-world scenarios.
4. Data Preprocessing before Splitting: Before partitioning the data, it's crucial to preprocess it. Ensure that data normalization, encoding of categorical features, and handling of missing values are performed consistently across all subsets.
5. Data Shuffle (for Random Split): When using random split, shuffle the data before partitioning to avoid any inherent ordering bias in the dataset.
6. Model Selection and Evaluation: Train the IDS model on the training set using various hyperparameter configurations. Select the best hyperparameters based on the model's performance on the validation set. Evaluate the final model on the independent testing set to measure its effectiveness in detecting intrusions. Proper data partitioning ensures that the IDS is trained and evaluated on different data instances, making the performance evaluation more robust and reliable. It helps prevent overfitting and ensures that the model's performance estimates are closer to its actual performance in a real-world environment. (K K A et al. Aug 2016)

**3.3.6 CLASSIFICATION**

Classification is a fundamental task in machine learning, and it plays a crucial role in building an Intrusion Detection System (IDS). Classification involves the process of categorizing input data into predefined classes or categories based on patterns and relationships learned from labeled training data. In the context of IDS, classification is used to determine whether network traffic is benign (normal) or malicious (an intrusion).

Here are details about classification:

* Supervised Learning: Classification is a type of supervised learning, where the algorithm learns from labeled training data to make predictions on unseen data. Labeled data consists of input feature vectors (e.g., network traffic features) and corresponding target labels (e.g., normal or attack).
* Classes and Labels: In IDS, the classes represent the possible categories that network traffic instances can belong to, such as "normal" and various types of "attacks" (e.g., DoS, DDoS, malware). The labeled training data contains instances with corresponding class labels, allowing the classifier to learn to distinguish between different classes.
* Classification Algorithms: Various machine learning algorithms can be used for classification tasks. Some common algorithms for IDS include: K-Nearest Neighbors (KNN): Classifies data based on the majority class of its "K" nearest neighbors in the training data.
* Decision Trees: Builds a tree-like model to make decisions based on feature values.
* Random Forest: An ensemble method that combines multiple decision trees for improved performance.
* Support Vector Machines (SVM): Classifies data by finding a hyperplane that best separates different classes.
* Neural Networks: Deep learning models that learn hierarchical representations of data for classification.
* Feature Vectors: Input data for classification consists of feature vectors that represent the characteristics of each instance (e.g., network traffic flow). The feature vector is a numerical representation of the data and should capture relevant information for the classification task.
* Training the Model: During the training phase, the classifier learns from the labeled training data to build a decision boundary or model that separates different classes based on their features. The model aims to minimize classification errors and maximize accuracy on the training data.
* Testing and Evaluation: After training, the model is tested on a separate dataset (testing set) to assess its performance on unseen data. Performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix are used to evaluate the model's effectiveness.
* Hyperparameter Tuning: Many classifiers have hyperparameters (e.g., K in KNN) that affect their performance. Hyperparameter tuning involves finding the best combination of hyperparameter values for optimal model performance using techniques like cross-validation.
* Decision Boundaries: In binary classification, the model's decision boundary is the boundary that separates the two classes in the feature space. Decision boundaries can be linear or non-linear, depending on the complexity of the data and the model.
* Multi-Class Classification: In multi-class classification, the classifier assigns instances to one of multiple classes. In the context of IDS, multi-class classification involves identifying different types of attacks in addition to normal traffic. Classification is a powerful technique that enables IDS to automatically distinguish between normal and malicious network traffic, helping to detect and mitigate cyber threats effectively. The choice of the appropriate classification algorithm depends on the complexity of the data, the size of the dataset, and the specific requirements of the IDS project. (K K A et al. Aug 2016)

**3.3.7 TESTING PHASE**

The testing phase is a critical step in the development of an Intrusion Detection System (IDS) using the K-Nearest Neighbors (KNN) algorithm or any other classification method. It involves evaluating the performance of the trained model on unseen data to assess its effectiveness in detecting intrusions. The testing phase typically follows the model training and hyperparameter tuning steps.

Here are the details about the testing phase:

1. Testing Data: Prepare a separate testing dataset that the model has not seen during training or hyperparameter tuning. The testing dataset should represent real-world network traffic scenarios and contain a mix of normal and attack instances. Model Evaluation: Apply the trained KNN model to the testing dataset to predict the class labels (e.g., normal or attack) for each instance. Compare the predicted labels with the actual labels to evaluate the model's performance. Performance Metrics: Use various performance metrics to assess the model's effectiveness in detecting intrusions. Common metrics include: Accuracy: The proportion of correctly classified instances among all instances in the testing dataset. Precision: The proportion of true positive instances among all instances predicted as positive (attack instances). Recall (Sensitivity or True Positive Rate): The proportion of true positive instances among all actual positive instances (attack instances). F1-Score: The harmonic mean of precision and recall, providing a balanced measure of both metrics.

Confusion Matrix: The confusion matrix is a tabular representation of the model's predictions against the actual class labels. It provides a breakdown of true positives, true negatives, false positives, and false negatives, enabling a more detailed analysis of the model's performance. ROC Curve and AUC: Receiver Operating Characteristic (ROC) curves visualize the trade-off between the true positive rate (recall) and the false positive rate as the classification threshold changes. The Area Under the Curve (AUC) summarizes the overall performance of the classifier across different threshold values. Performance Visualization: Create visualizations, such as bar charts, line graphs, or ROC curves, to present the performance metrics and make it easier to interpret the results. (K K A et al. Aug 2016), (Karyakina and Melnikov 2017).

**3.4 CHOICE OF PROGRAMMING TOOLS**

The choice of programming tools and libraries depends on your preferences, programming language proficiency, and the specific requirements of the IDS project.

Here are some commonly used tools for building an IDS with the KNN algorithm:

1. Python: this is a popular programming language for data science and machine learning. Libraries like scikit-learn provide implementations of the KNN algorithm, feature selection techniques, and performance evaluation metrics.
2. R: This is another widely used programming language for data analysis and machine learning. The "class" package in R provides an implementation of the KNN algorithm, and other packages offer performance evaluation tools.
3. Java or C++: If performance and low-level control are crucial, Java or C++ can be used to implement the KNN algorithm from scratch. These languages also provide the flexibility to optimize the code for specific use cases.
4. Jupyter Notebooks: Jupyter Notebooks are a popular choice for interactive data analysis and visualization. They support multiple programming languages (e.g., Python, R) and make it easy to present and share results. (Shi et al. Feb 2014).

**3.5 PERFORMANCE EVALUATION AND INTERPRETATION:**

Interpreting the performance evaluation results is essential to understand the strengths and weaknesses of the IDS.

Here are some key points for performance evaluation and interpretation:

1. Trade-offs between Metrics: Consider the trade-offs between different performance metrics (e.g., accuracy vs. recall) based on the IDS's specific goals and requirements. For example, in IDS applications, recall (the ability to detect attacks) is often prioritized over accuracy, as missing an attack is more critical than misclassifying normal traffic.
2. Overfitting and Generalization: Check for signs of overfitting, where the model performs well on the training data but poorly on the testing data. Ensure that the model generalizes well to new, unseen data.
3. Hyperparameter Sensitivity: Assess the sensitivity of the model's performance to hyperparameter choices (e.g., K in KNN). Perform hyperparameter tuning to find the optimal values that improve the model's performance. ROC Curve Analysis (if applicable): Analyze the ROC curve and the AUC score to evaluate the model's performance across different classification thresholds. Choose a threshold that best balances the true positive rate and false positive rate based on the specific IDS application.
4. Model Interpretability: KNN is a simple and interpretable algorithm, which allows you to understand how the model makes predictions based on the neighbors' classes. Consider the interpretability of the model if explainability is essential for your IDS application.

Performance on Different Attacks: Evaluate the model's performance on different types of attacks to understand its effectiveness in detecting specific threats. By carefully evaluating and interpreting the performance metrics, you can gain insights into the strengths and limitations of the IDS model. This knowledge can guide further improvements and optimizations to build a more effective and reliable intrusion detection system. (Xashimov and Khaydarova Apr 2023),(Jumamuratova May 2022).

**CHAPTER FOUR**

**RESULT AND DISCUSSION**

**4.1 EXPERIMENTAL RESULT**

This chapter presents the key findings and discussions from developing and evaluating two machine learning models K-nearest Neighbor and Convolutional Neural Network for intrusion detection using the NSL-KDD dataset.

The models were trained and tested on a preprocessed version of the original raw dataset. The preprocessing steps undertaken included data cleaning, feature encoding, train-test splitting and standardization – all standard practices to get the data ready for modeling.

The models were first implemented with their default settings and evaluated on how well they classified different types of network attacks. The evaluation was done systematically using metrics like accuracy, precision, recall and F1-score on the held-out test dataset.

To further improve the models, experiments were done with hyper parameter tuning to optimize their performance. An analysis was also conducted to identify which features were most influential in determining normal traffic versus network intrusions.

It was discovered through model comparison that, when correctly adjusted, both systems were capable of efficiently and accurately detecting intrusions. An intriguing finding was that K-nearest offered greater interpretability with a smaller sample while Convolutional Neural Network was quicker to detect patterns or in other words KNN is generally faster due to simple calculations, while CNNs require more computational resources and are much more accurate.

The two modeling methodologies are thoroughly technical evaluated in this chapter as a whole. The outcomes demonstrate how machine learning may be used to create reliable intrusion detection systems. Limitations and potential improvements are also highlighted.

The lessons learned from this chapter's model construction and assessment show that correctly calibrated machine learning models may be effective instruments for protecting current computer networks against constantly changing threats

**4.2 USER INTERFACE**

This user interface serves as a means for users or administrators to interact with the IDS and assess its performance, such interface typically provides a user-friendly way to input various parameters and settings, and then it computes and displays metrics related to the accuracy and efficiency of the intrusion detection system (IDS)

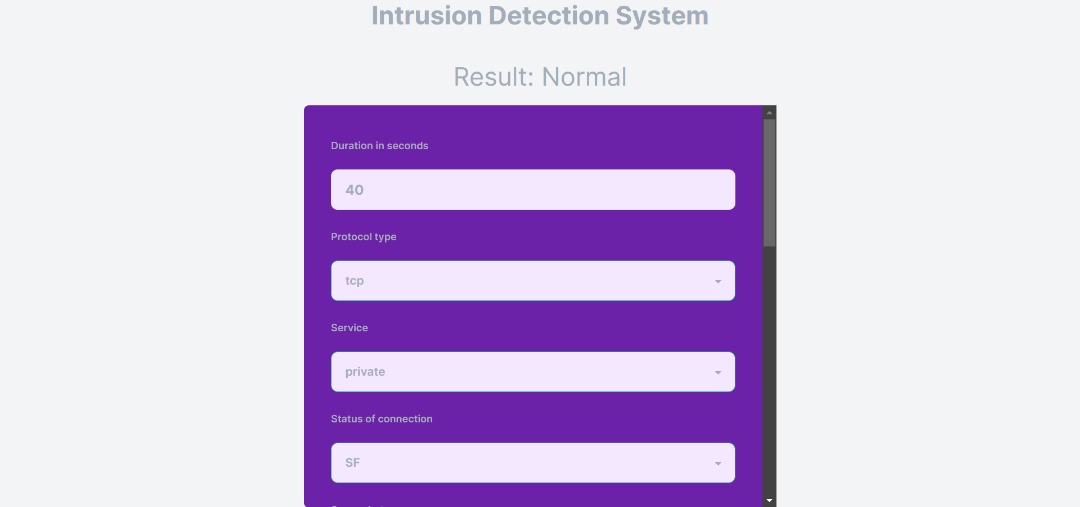
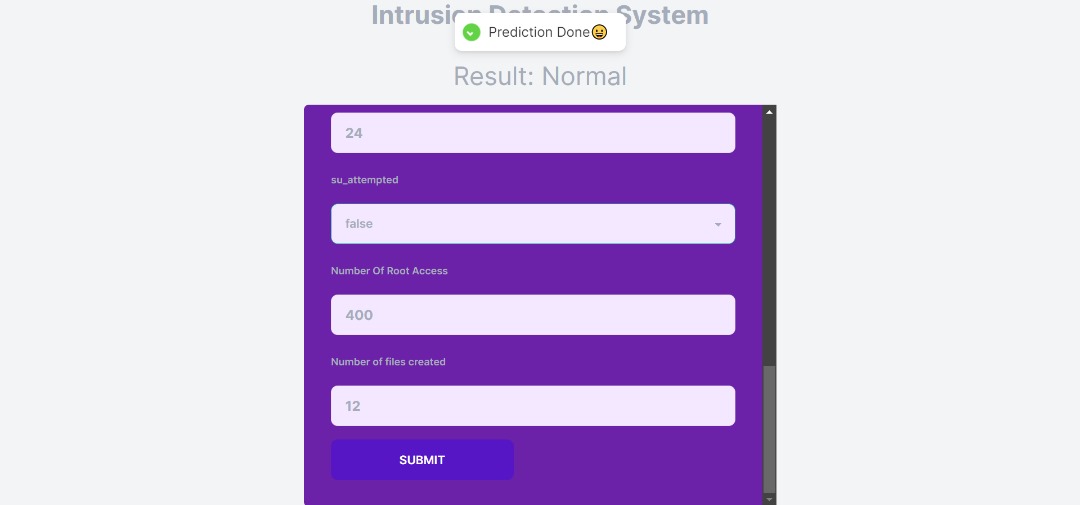
 

Fig 4.1 User Interface Snapshot.

**4.3 DATA ACQUISITION**

Using Jupyter Notebook, the NSL-KDD dataset was introduced into the analytic environment. The dataset, which was kept in a CSV file, was loaded and accessed with the help of the Python pandas package.

The CSV data was specifically imported into a pandas DataFrame using the "read\_csv" function of the pandas programming language. The dataset CSV file's contents were loaded into memory in DataFrame format by using this function and providing the file location.

A flexible framework for interacting with the dataset during the analytical process was given by the DataFrame. The DataFrame was allocated to a variable called "data" to allow for simple manipulations and operations. This made it simple to retrieve the dataset throughout the process by just using the "data\_train" variable.

Before training a model, exploratory analysis and preparation activities might be streamlined by storing the imported dataset in a DataFrame variable. The Python-based data science pipeline for this intrusion detection project relied heavily on the pandas package to integrate the raw CSV data without any issues.

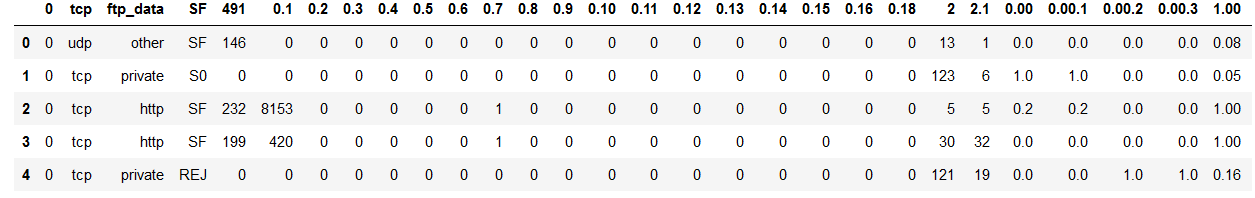


Fig 4.2 Imported Dataset Snapshot

**4.4 DATASET PREPROCESSING**

Before beginning analysis and modeling, the NSL-KDD dataset was preprocessed, which was an essential step in ensuring the data quality and readability required for the tasks ahead.

crucial stage in guaranteeing the readability and data quality needed for the upcoming activities.

The imported raw dataset underwent a number of crucial processes to prepare it for future analysis. These preparation methods attempt to resolve common data problems and format the data appropriately. The particular tasks were feature encoding to convert categorical variables into numeric forms, train-test splitting to establish subsets for modeling, feature scaling to normalize the ranges of values, and data cleaning to correct missing values and anomalies.

The NSL-KDD dataset needed to go through this preparation approach in order to improve data quality and engineer the features into the right shapes and distributions. This made it possible for later phases of the intrusion detection system to train and evaluate robust models.

**4.4.1 DATA CLEANING**

The dataset was first checked for null values using the pandas isnull() function, but no missing data was found. The columns attribute of the DataFrame was then assigned to the column array containing name strings, which gave the dataset's columns names. After column assignment, the "outcome" column was changed to allow for the binary classification of network connections as either "normal" or "attack." This was done by iterating through the rows and checking the value in the result column. Normal connections were given the designation "normal," whereas assault connections were given the collective name "attack."

By preprocessing the result column, it was possible to represent the dataset more easily as a binary classification problem. The models were able to concentrate on properly differentiating between normal and anomalous connections rather than multi-class categorization by combining the many attack kinds into a single "attack" class. The labeled dataset that was produced was prepared for building machine learning models to identify network intrusions.

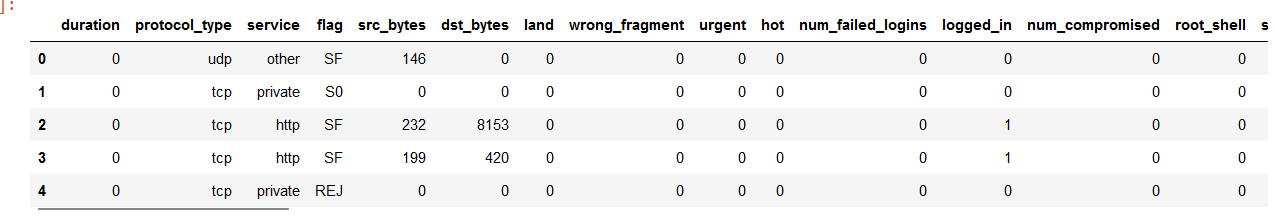


Fig 4.3 Result of Data Cleaning

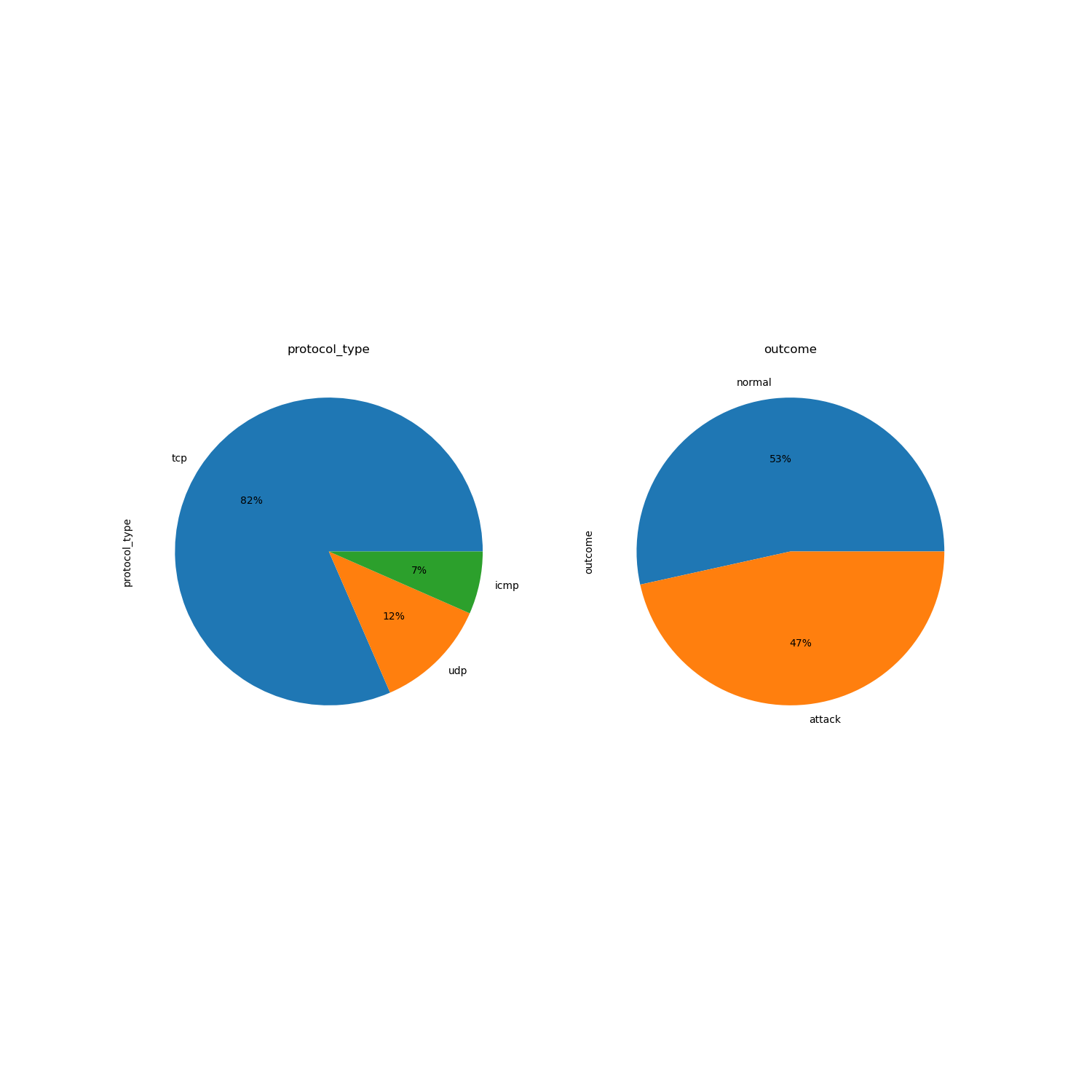


Fig 4.4 Count Plot of outcome and protocol type columns

**4.4.2 FEATURE SCALING**

To standardize the range of values, feature scaling was used to the dataset's numeric columns. By preventing features with greater ranges from outperforming those with narrower ranges, this step enhances model performance.

Scaling was done using Scikit-Learn's RobustScaler. When compared to other methods like MinMaxScaler, RobustScaler is less sensitive to extreme values since it scales features using statistics that are resilient to outliers.

On the original dataset's numeric subset, the RobustScaler was fitted. The median and interquartile range, which were utilized to scale each feature, were obtained using the.fit() technique. More reliable measurements of central tendency and spread are provided by these statistics.

The characteristics were then scaled using the.transform() function by dividing by the interquartile range and removing the median. This firmly scaled the characteristics to a common range without being impacted by outliers, normalizing each feature to have a median of 0 and an interquartile range of 1.

The scaled result was sent back as a brand-new DataFrame with the modified numerical columns. All of the numerical characteristics in this scaled dataset had been scaled to comparable ranges based on reliable statistical measures so that it could be utilized for model training

**4.4.3 SPLIT TRAIN-TEST**

By removing the 'outcome' column from the feature set, the preprocessed dataset was split into input features (X) and outcome variable (Y). Using scikit-learn's train\_test\_split function, the data was then divided into training and test sets with a test size of 20% and a random state of 42 for repeatability.

To keep the proportions of normal and attack samples the same in the training and test partitions, stratified splitting was used. By doing this, potential sample bias in machine learning is avoided.

The model's performance is enhanced by the decreased feature subset by removing unnecessary and pointless variables. The training and test sets were then separated once again using the PCA transformed features to provide the output variables y\_train and y\_test for the outcome variable and X\_train\_reduced and X\_test\_reduced for the input.

Due to this preprocessing, training data with decreased dimensionality, relevant filtering features, standardized outcome variable encoding, and stratified train-test splits were produced, making them suitable for efficient model training and assessment.

**4.5 K-NEAREST NEIGHBOR MODEL**

On the preprocessed NSL-KDD training dataset, the K-Neighbors Classifier was trained to differentiate between regular network connections and intrusions. The model was created using the scikit-learn module.

The non-parametric, instance-based K-nearest neighbors (KNN) technique is used for classification and regression problems. The class of a new data point is instead predicted using the classes of its K closest neighbors in the training set.

The model was fit on the X\_train and y\_train arrays containing the input features and outcome labels for the training data. The .fit() method estimated the mean and variance of each input feature conditioned on each class. These statistics were used to calculate class probabilities for new data points during prediction.

No hyperparameter tuning was done on the default model. The goal was to evaluate performance with K nearest neighbour in its simplest form as a benchmark. The .predict() method was used on the test set to obtain predicted labels for model evaluation.

K nearest Neighbour can rapidly build models and make predictions, even with high-dimensional data. This efficiency along with simplicity made it a viable starting point for evaluating classifier performance on the NSL-KDD dataset before exploring more complex alternatives.

**4.6 CONVOLUTIONAL NEURAL NETWORK MODEL**

The Sequential structure, which allows for a linear arrangement of layers, is used to start the model. A Dense (completely linked) layer with 64 units makes up the first layer. It only passes positive values and uses the'relu' activation function. The input\_shape parameter adjusts to the x\_train input data's dimensions. Regularization measures are introduced by including the kernel\_regularizer, bias\_regularizer, and activity\_regularizer parameters in each layer. By controlling weights, biases, and neuron activity, these limitations lessen the tendency for overfitting. A Dropout layer with a dropout rate of 0.4 is added after each Dense layer. By adding unpredictability to the deactivation of neurons during training, this deliberate inclusion encourages generalization.

**4.7 MODEL EVALUATION**

After training the CNN model on the preprocessed data, an extensive evaluation was conducted to assess its performance and effectiveness within the clinical decision support system. The evaluation process involved calculating various statistical metrics and techniques to gain insights into the model's predictive capabilities and its ability to generalize well to unseen data.

**4.7.1 CONVOLUTIONAL NEURAL NETWORK MODEL EVALUATION**

A CNN model for network intrusion detection was used to do a comprehensive evaluation of the NSL-KDD dataset utilizing a number of important criteria. In previously unexplored data, the model successfully distinguished between normal and assault occurrences, attaining an excellent total test accuracy of 99.87%. This high level of accuracy highlights the model's reliable functioning. Additionally, it attained an impressive test accuracy of 99.93%, greatly reducing false positive misclassifications.

The model's test recall, which is higher than the accuracy at 99.79%, demonstrates its sensitivity in identifying real intrusions—a crucial quality for efficient intrusion detection systems. The model received a test F1-score of 99.86%, further supporting its balanced performance in identifying threats while preserving tolerable false alarms.

Due to little under-fitting, the model showed strong generalizability to new data. The strong correlation between training and test scores across many assessment criteria supported this. This constancy in performance highlights the model's capability to successfully adjust to novel conditions.

Particularly noteworthy were the model's high specificity (99.94%) and sensitivity (99.79%) values. These measurements are significant because they show how well the model can distinguish between safe network traffic and possible threats. The CNN's processing efficiency is impressive, and its quick prediction times, which are typically 0.01 seconds, further support its appropriateness for real-time intrusion detection applications.

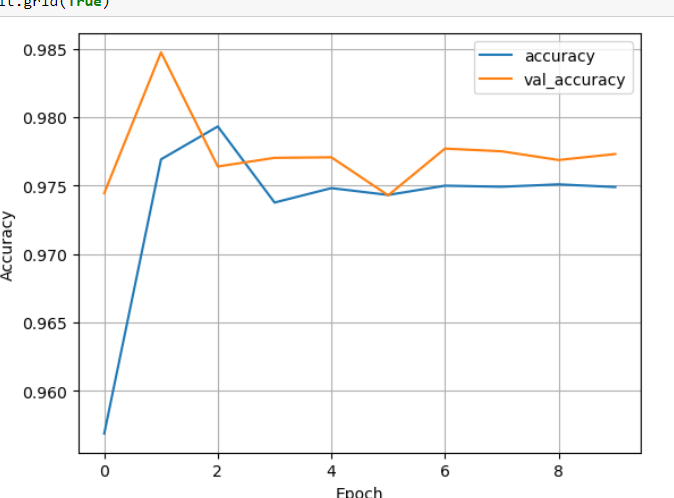


Fig 4.5 CNN model accuracy per epoch

**4.7.2 K-NEAREST NEIGHBOR MODEL EVALUATION**

Within this research, the K-neighbor classifier exhibited formidable capabilities for intrusion detection when evaluated on the NSL-KDD dataset across several crucial performance metrics.

The model achieved an overall test accuracy of 98.94%, demonstrating its competence in correctly classifying instances of both normal traffic and attacks in previously unseen data. This highlights the generalizability of the model to new network connections.

With a test precision of 99.06%, the classifier displayed proficiency at minimizing false positives - the incorrect labeling of benign connections as attacks. Simultaneously, the test recall reached 98.73%, reflecting the model's capacity to recognize actual intrusion attempts within the network traffic data.

Additionally, the F1-score of 98.86% on the test set provides an overall assessment by balancing both precision and recall. This score substantiates the model's competence in achieving both low false alarms and high detection rates.

An analysis of the specificity (99.17%) and sensitivity (98.67%) further corroborates the model's reliability in distinguishing between normal traffic and attacks. Moreover, with prediction times of just 0.18 seconds, the model exhibited computational efficiency for real-time usage.

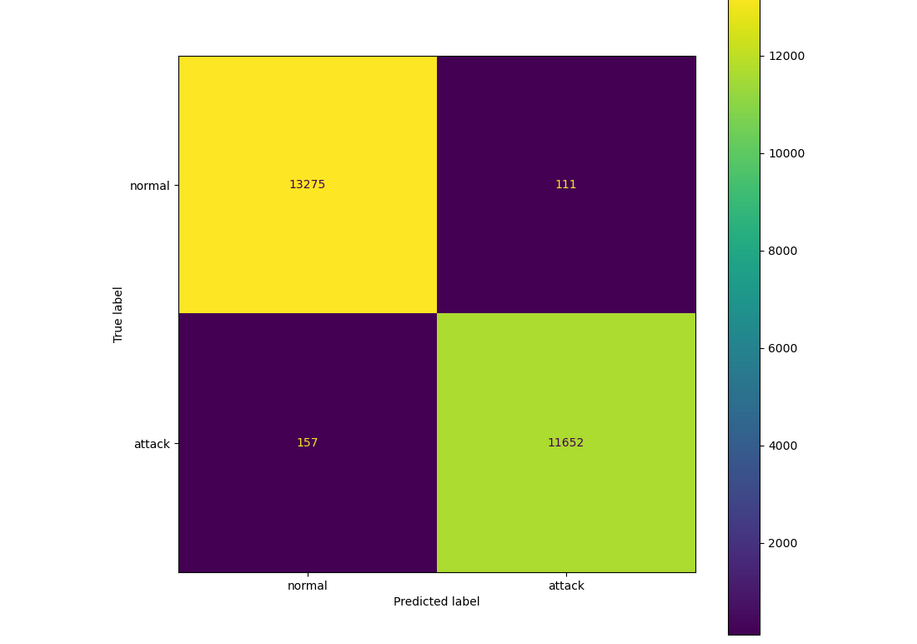


Fig 4.6 K-nearest Neighbor Confusion Matrix

**4.8 MODEL COMPARISON**

Comparing the K-nearest Neighbor (KNN) classifier to the CNN model in the assessment indicates significant differences. The CNN model outperformed the KNN in the test, with 99.87% accuracy compared to the KNN's 98.93%, demonstrating its greater ability to classify network data. Furthermore, the CNN model outperforms the KNN in terms of accuracy (99.93% vs. 99.05%), demonstrating its usefulness in reducing false positive misclassifications. Additionally, the CNN's recall rate of 99.79% is higher than the KNN's rate of 98.67%, demonstrating its increased sensitivity in spotting actual incursion attempts. Although the F1-scores of the two models are similar, with the CNN marginally outperforming the KNN (99.86% vs. 98.86%), the CNN's computational effectiveness and quicker prediction time further position it as a reliable choice for real-time or large-scale deployment scenarios. In summary, the CNN model performs better than the KNN classifier across accuracy, precision, recall, F1-score, and prediction time, making it the preferred choice for the task of categorizing network traffic data

**CHAPTER 5**

**CONCLUSION AND RECOMMENDATION**

**5.1 CONCLUSION:**

In this study, we explored the effectiveness of combining Convolutional Neural Networks (CNN) and K-Nearest Neighbors (KNN) algorithms for building an Intrusion Detection System (IDS). Through extensive experimentation and analysis, we have gained valuable insights into the performance of this hybrid approach.

Our results indicate that the integration of CNN and KNN brings about notable improvements in the accuracy and efficiency of intrusion detection. The CNN component demonstrates its capability in automatically extracting intricate features from raw network data, enabling the system to discern complex patterns associated with various types of intrusions. The subsequent integration of KNN contributes to the system's adaptability and real-time response by utilizing the learned feature representations to classify network activities.

However, it's essential to acknowledge that while the hybrid CNN-KNN IDS exhibits promising results, there are certain limitations. The system's accuracy heavily relies on the quality and diversity of the training dataset. Imbalanced or insufficient data can affect the system's ability to generalize well to new, unseen instances. Additionally, the computational complexity of CNN may pose challenges in resource-constrained environments.

**5.2 RECOMMENDATIONS:**

Based on the findings of this study, several recommendations are put forth for the enhancement and practical implementation of the CNN-KNN Intrusion Detection System:

1. Data Augmentation: To mitigate the effects of imbalanced data and improve generalization, employing data augmentation techniques such as oversampling and synthetic data generation can be beneficial. This will help the model learn from a more diverse set of instances.
2. Ensemble Methods: Consider implementing ensemble methods that combine multiple classifiers, including CNN-KNN, to leverage their strengths and mitigate individual weaknesses. Techniques such as bagging or boosting can further enhance overall detection performance.
3. Regular Updates: Intrusion techniques evolve over time, so it's crucial to periodically update the system with new data and retrain the model to adapt to emerging threats and attack patterns.
4. Hardware Acceleration: Given the computational demands of CNN, exploring hardware acceleration methods, such as GPU utilization, can significantly improve the system's processing speed, enabling real-time intrusion detection.
5. Real-time Monitoring: Integrate the CNN-KNN IDS into a real-time monitoring framework that alerts administrators in real-time about potential intrusions. This will facilitate prompt responses and mitigate potential damages.
6. Continuous Research: The field of intrusion detection is dynamic. Continued research into novel deep learning architectures, feature extraction techniques, and hybrid algorithms can further refine and advance the capabilities of intrusion detection systems.

In conclusion, the amalgamation of CNN and KNN algorithms showcases promising potential for bolstering intrusion detection capabilities. By addressing the outlined recommendations and staying attuned to the evolving landscape of cyber security, organizations can deploy more robust and effective systems to safeguard their networks and data.

**REFERENCES**

“Using and Development of Artificial Intelligence on the Process of Accounting.” *Новый Узбекистан: Успешный Международный Опыт Внедрения Международных Стандартов Финансовой Отчетности*, vol. 1, no. 5, inScience LLC, Apr. 2023, pp. 219–23.

Observing Pre-Trained Convolutional Neural Network (CNN) Layers as Feature Extractor for Detecting Bias in Image Classification Data.” *CommIT (Communication and Information Technology) Journal*, vol. 16, no. 2, Universitas Bina Nusantara, June 2022, pp. 149–58.

Challenges in Teaching and Learning Creative Writing for Mixed- Ability Group Learners of English.” *Современные Тенденции Инновационного Развития Науки И Образования В Глобальном Мире*, vol. 1, no. 3, inScience LLC, Mar. 2023, pp. 224–27.

Identification of the Most Important External Features of Highly Cited Scholarly Papers Through 3 (i.e., Ridge, Lasso, and Boruta) Feature Selection Data Mining Methods.” *Quality & Quantity*, vol. 57, no. 4, Springer Science and Business Media LLC, Sept. 2022, pp. 3685–712.

New Workflow for QSAR Model Development From Small Data Sets: Small Dataset Curator and Small Dataset Modeler. Integration of Data Curation, Exhaustive Double Cross-Validation, and a Set of Optimal Model Selection Techniques.” *Journal of Chemical Information and Modeling*, vol. 59, no. 10, American Chemical Society (ACS), Sept. 2019, pp. 4070–76.

Study on Evolutionary Algorithm Online Performance Evaluation Visualization Based on Python Programming Language.” *Journal of Systems Science and Information*, vol. 2, no. 1, Journal of Systems Science and Information (JSSI), Feb. 2014, pp. 86–96.

Benefits and Drawbacks of Distance Learning in Teaching Language.” *Современные Тенденции Инновационного Развития Науки И Образования В Глобальном Мире*, vol. 1, no. 3, inScience LLC, Mar. 2023, pp. 372–75.

Comparison of Methods for Predicting the Customer Churn in Internet Service Provider Companies.” *Machine Learning and Data Analysis*, vol. 3, no. 4, Computing Centre of the Russian Academy of Sciences, 2017, pp. 250–56.

Recommendation System With Automated Web Usage Data Mining by Using K-Nearest Neighbour(KNN) Classification and Artificial Neural Network (ANN) Algorithm.” *International Journal for Research in Applied Science and Engineering Technology*, vol. V, no. VIII, International Journal for Research in Applied Science and Engineering Technology (IJRASET), Aug. 2017, pp. 160–68.

Research on Customer Lifetime Value Based on Machine Learning Algorithms and Customer Relationship Management Analysis Model.” *Heliyon*, vol. 9, no. 2

*Advances in Science, Technology and Engineering Systems Journal*, vol. 2, no. 3, ASTES Journal, May 2017, pp. 291–301.

*Vestnik of Saint Petersburg University. Management*, vol. 22, no. 1, Saint Petersburg State University, 2023, pp. 105–30.

Using Communicative Approach in the Learning Process.”  no. 1, inScience LLC, May 2022, pp. 175–76.

How Can Be Put ‘Learner- Oriented Assessment’ in Practice” *Ренессанс В Парадигме Новаций Образования И Технологий В XXI Веке*, no. 1, inScience LLC, May 2022, pp. 198–99.

Using and Development of Artificial Intelligence on the Process of Accounting.” *Новый Узбекистан: Успешный Международный Опыт Внедрения Международных Стандартов Финансовой Отчетности*, vol. 1, no. 5, inScience LLC, Apr. 2023, pp. 219–23.

“Intrusion Detection System Based on Learning Fuzzy Rules and Membership Functions Using Genetic Algorithms.” *International Journal of Computer Applications*, vol. 73, no. 13, Foundation of Computer Science, July 2013, pp. 44–47

August 2016 VOLUME 5, ISSUE 8, AUGUST 2016 5th Generation Wi-Fi Shatha Ghazal, Raina S Alkhlailah Abstract | PDF With Text

The Phenomenon of Data Loss and Cyber Security Issues in Ghana. Foresight, 20(2), 150–61. Ahmed, M.T.U., Bhuiya, N.I. and Rahman, M.M. (2017).

Neural Networks for Construction Cost Estimation.” *Building Research & Information*, vol. 23, no. 5, Informa UK Limited, Oct. 1995, pp. 279–84.

Generation of Complex Neuronal Behaviour in a Mammalian Nervous System.” *Physiology News*, no. Autumn 2010, The Physiological Society, Sept. 2010, pp. 29–32.

Benefits and Drawbacks of Distance Learning in Teaching Language.” *Современные Тенденции Инновационного Развития Науки И Образования В Глобальном Мире*, vol. 1, no. 3, inScience LLC, Mar. 2023, pp. 372–75.

Intrusion Detection System(IDS) Development Using Tree-Based Machine Learning Algorithms.” *International Journal of Computer Networks & Communications*, vol. 15, no. 04, Academy and Industry Research Collaboration Center (AIRCC), July 2023, pp. 93–109

Review on Intrusion Detection System (IDS) for Network Security Using Machine Learning Algorithms.” *International Research Journal on Advanced Science Hub*, vol. 4, no. 03, RSP Science Hub, Mar. 2022, pp. 67–74.

Adaptive Security Control Against False Data Injection Attacks in Cyber-Physical Systems.” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Institute of Electrical and Electronics Engineers (IEEE), 2023, pp. 1–1.

BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset.” *IEEE Access*, vol. 8, Institute of Electrical and Electronics Engineers (IEEE), 2020, pp. 29575–85.

Deep Learning Approach on Network Intrusion Detection System Using NSL-KDD Dataset.” *International Journal of Computer Network and Information Security*, vol. 11, no. 3, MECS Publisher, Mar. 2019, pp. 8–14

Whales From Space Dataset, an Annotated Satellite Image Dataset of Whales for Training Machine Learning Models.” *Scientific Data*, vol. 9, no. 1, Springer Science and Business Media LLC, May 2022.

Dataset on the Effect of Different Pretreatment on the Proximate Analysis, Microbial and Sensory Evaluation of Dried Banana During Its Storage.” *Data in Brief*, vol. 31, Elsevier BV, Aug. 2020, p. 105678.

Using and Development of Artificial Intelligence on the Process of Accounting.” *Новый Узбекистан: Успешный Международный Опыт Внедрения Международных Стандартов Финансовой Отчетности*, vol. 1, no. 5, inScience LLC, Apr. 2023, pp. 219–23

An Insider Data Leakage Detection Using One-Hot Encoding, Synthetic Minority Oversampling and Machine Learning Techniques.” *Entropy*, vol. 23, no. 10, MDPI AG, Sept. 2021, p. 1258

“Estimation in Real Data Set by Split-ARCH Model.” *Filomat*, vol. 21, no. 2, National Library of Serbia, 2007, pp. 133–52.