Deep Learning for Intrusion Detection in IoT Networks

Mehdi Selem
University of Sousse
Farah Jemili
jmili_farah@yahoo.fr
University of Sousse
Ouajdi Korbaa
University of Sousse

Research Article

Keywords: Deep learning, IoT network, Intrusion detection

Posted Date: May 7th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4306367/v1

Version of Record: A version of this preprint was published at Peer-to-Peer Networking and Applications on January 9th, 2025. See the published version at https://doi.org/10.1007/s12083-024-01819-3.

Abstract

The rapid proliferation of Internet of Things (IoT) devices has transformed our daily lives, introducing innovations like smart homes, wearables, and advanced industrial automation. While these interconnected systems offer convenience and efficiency, they also present significant security challenges. With the expansion of the IoT network comes an increased risk of malicious attacks, making safeguarding these networks a pressing concern. Intrusion detection serves as a crucial defense mechanism, detecting abnormal activities and triggering appropriate responses. In our study, we harness the power of ensemble learning through a technique known as bagging. By combining the strengths of Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs), we aim to capitalize on their unique advantages and enhance the overall capability of intrusion detection systems.

I. Introduction

The pervasive integration of Internet of Things (IoT) devices into our daily lives has heralded a transformative era of connectivity, impacting industries ranging from smart homes to industrial automation. However, this exponential growth in IoT networks has brought not only unparalleled convenience but also unveiled new and complex security challenges. Safeguarding these networks against unauthorized access and potential threats has emerged as a paramount concern in the digital landscape.

Intrusion detection stands as a cornerstone of network security, tasked with identifying and thwarting unauthorized access, thereby ensuring the integrity and confidentiality of sensitive information traversing IoT networks. While traditional intrusion detection methods have shown efficacy to a degree, the dynamic and intricate nature of IoT environments necessitates more sophisticated and adaptable solutions.

Enter deep learning, a subset of artificial intelligence renowned for its ability to discern intricate patterns and extract high-level features from vast and complex datasets. The application of deep learning techniques holds immense promise in bolstering the efficacy of intrusion detection systems within IoT networks. By harnessing the innate capabilities of neural networks to detect subtle anomalies and deviations from normal network behavior, deep learning models have exhibited a remarkable capacity to fortify the security posture of IoT ecosystems.

This research proposal endeavors to explore the convergence of IoT network security and deep learning, with a specific emphasis on the effectiveness of deep learning methodologies in ensuring robust intrusion detection. Through a comprehensive analysis of the strengths and limitations of various deep learning architectures and their suitability in the context of IoT security, this study aims to offer valuable insights into the design and implementation of effective intrusion detection systems tailored to the unique challenges posed by IoT networks.

Through an exhaustive comparative study, our goal is to determine which deep learning paradigm, whether Convolutional Neural Networks (CNNs), Deep Neural Networks (DNNs), or Artificial Neural Networks (ANNs), exhibits the highest potential for accurately and efficiently identifying intrusions in IoT networks. By doing so, we aspire not only to fortify immediate security measures but also to lay the groundwork for adaptive and resilient security protocols in the ever-evolving landscape of IoT technology.

The anticipated findings of this research are poised to make substantial contributions to the discourse on enhancing IoT network security, ultimately paving the way for more secure and dependable IoT-driven operations across diverse industries and domains.

II. RELATED WORK

The paper [1] presents an explainable deep learning framework for intrusion detection systems (IDS) in Internet of Things (IoT) networks. The proposed framework combines the use of a deep neural network (DNN) with a rule-based expert system to achieve both good accuracy and explain ability. The DNN is trained on a dataset of network traffic features to classify traffic as either normal or malicious. The rule-based expert system uses domain-specific knowledge to generate explanations for the DNN's decisions. The authors use the UNSW-NB15 dataset, which is a publicly available dataset of network traffic designed for evaluating IDS. The proposed framework is able to achieve good accuracy in detecting intrusions while also providing explanations for its decisions. However, the explain ability provided by the rule-based expert system may be limited by the quality of the domain-specific knowledge used. The paper reports that the proposed framework achieved an accuracy of 98.77% on the UNSW-NB15 dataset for detecting network intrusions.

The paper[2] presents a deep learning-based approach for intrusion detection in Internet of Things (IoT) networks. The proposed approach uses a deep convolutional neural network (CNN) to learn features from network traffic data and classify traffic as either normal or malicious. The authors also apply transfer learning by fine-tuning a pre-trained CNN on the IoT dataset to further improve the accuracy of the model. The authors use the CICIDS2017 dataset, which is a publicly available dataset of network traffic designed for evaluating intrusion detection systems. The proposed approach achieved high accuracy rates on the CICIDS2017 dataset, with an overall accuracy of 99.94% for detecting network intrusions. However, the proposed approach is evaluated on a relatively small dataset, so its generalizability to more data is unknown. Additionally, the computational cost of using the transfer learning may be high, which could be a challenge in resource-constrained IoT devices.

The paper [3]provides an overview of the current state-of-the-art in using deep learning for intrusion detection and security in Internet of Things (IoT) networks. The paper discusses the challenges associated with using deep learning in IoT networks and proposes possible solutions to address these challenges. he paper discusses various deep learning algorithms that have been used for intrusion detection in IoT networks, including convolutional neural networks (CNNs), recurrent neural networks

(RNNs), and deep belief networks (DBNs). The authors also discuss the use of transfer learning and unsupervised learning for improving the performance of deep learning models. The paper discusses various datasets that have been used in the literature for evaluating intrusion detection systems in IoT networks. The paper does not present any new experimental results or propose a specific deep learning approach for intrusion detection in IoT networks. Rather, the paper provides a review of the literature and discusses possible solutions to the challenges associated with using deep learning in IoT networks.

The paper [4]proposes a hybrid intrusion detection model for IoT applications that combines machine learning algorithms with rule-based methods. The proposed model aims to improve the accuracy of intrusion detection while reducing false alarms. The proposed model uses the k-Nearest Neighbor (kNN) algorithm and a rule-based method for intrusion detection. The KNN algorithm is used to classify network traffic as normal or abnormal based on the similarity between the current data point and the k nearest neighbors in the training dataset. The proposed model achieves an overall accuracy of 99.18% on the NSL-KDD dataset. However, it is important to note that the accuracy may vary depending on the characteristics of the dataset and the network environment. The proposed model may be limited by the quality and representativeness of the training dataset. The NSL-KDD dataset used in the experiments may not fully capture the diversity of real-world IoT network traffic. Additionally, the proposed model may require significant computational resources, as the KNN algorithm requires the calculation of distances between data points.

The paper [5]proposes a deep learning-based approach for detecting botnet attacks in IoT applications. The proposed approach uses a combination of convolutional neural networks (CNN) and long short-term memory (LSTM) networks to classify network traffic as normal or botnet traffic. The proposed approach uses a CNN-LSTM model for botnet attack detection. The CNN component is used to extract features from the raw network traffic data, while the LSTM component is used to capture the temporal dependencies between the features. The proposed approach achieves high accuracy in detecting botnet attacks in IoT applications, with an overall accuracy of 99.87%. The use of deep learning algorithms allows for automated feature extraction and classification, which can improve the efficiency and effectiveness of intrusion detection. The proposed approach may be limited by the availability and quality of training data. The UNSW-NB15 dataset used in the experiments may not fully capture the diversity of real-world IoT network traffic.

The weaknesses of the previous papers include limitations in the quality and representativeness of the datasets used, the computational cost of some of the proposed approaches, and the potential limitations of rule-based expert systems for providing explain ability. To overcome these weaknesses, we propose a new contribution based on the use of deep learning and bagging techniques for intrusion detection in IoT networks. Bagging can be used to improve the generalizability of deep learning models by aggregating the outputs of multiple models trained on different subsets of the data. This can help address the limitations of the small and potentially biased datasets used in previous studies. Additionally, the use of deep learning models with automated feature extraction can reduce the need for domain-specific knowledge and potentially improve the explain ability of the model. The proposed approach is evaluated on a large and diverse dataset of IoT network traffic to assess its effectiveness and generalizability.

While previous research has made significant strides in utilizing deep learning for intrusion detection in IoT networks, our study introduces several novel contributions aimed at addressing key limitations and advancing the state-of-the-art in this domain.

Firstly, we propose the integration of bagging techniques with deep learning models for intrusion detection in IoT networks. Bagging allows us to enhance the generalizability of deep learning models by aggregating the outputs of multiple models trained on different subsets of the data. This approach mitigates the limitations associated with small and potentially biased datasets used in prior studies, thereby improving the robustness and reliability of intrusion detection systems.

Secondly, our research emphasizes the use of deep learning models with automated feature extraction capabilities. By leveraging the inherent capacity of deep learning models to extract complex features from raw network traffic data, we reduce the reliance on domain-specific knowledge and enhance the explainability of the model. This advancement is crucial for ensuring the effectiveness and interpretability of intrusion detection systems in real-world IoT environments.

Furthermore, our study contributes to the field by evaluating the proposed approach on a large and diverse dataset of IoT network traffic. This comprehensive evaluation enables us to assess the effectiveness and generalizability of our approach across various network environments and attack scenarios, providing valuable insights for practitioners and researchers alike.

In summary, our research introduces novel methodologies and insights aimed at advancing the effectiveness, reliability, and interpretability of intrusion detection systems in IoT networks. By addressing key limitations and leveraging the latest advancements in deep learning and ensemble learning techniques, we aim to propel the field forward and contribute to the development of more secure and resilient IoT ecosystems.

III. BACKGROUND

A. *Convolutional Neural Network Algorithm*

A Convolutional Neural Network (CNN) is a type of neural network that is specifically designed for processing data with a grid-like topology, such as images or time series. The basic idea behind a CNN is to use convolutional layers to extract local features from the input data, and then use pooling layers to down sample the output and make it more robust to translation and distortion [9].

To explain CNNs, we can use the following notation:

• X: the input data, represented as a 3D tensor (height x width x channels)

• Y: the output of the network, represented as a 3D tensor (height' x width' x channels')

• W: the weights of the network, represented as a set of 4D tensors (filter_height x filter_width x input_channels x output_channels)

• b: the biases of the network, represented as a set of 1D tensors (output_channels)

Here's a brief overview of the main operations involved in a typical CNN:

• Convolution: The convolutional layer applies a set of filters to the input data X, producing a set of feature maps. The output of the k-th filter at position (i,j) can be computed as follows:

$$Y_{i,j,k} = b_k + \sum_{r,s,c} W_{r,s,c,k} * X_{i+r,j+s,c} \quad (1)$$

where b_k is the bias term for the k-th filter, $W_{r,s,c,k}$ is the weight of the k-th filter at position (r,s) for input channel c, and $X_{i+r,j+s,c}$ is the input value at position $(i+r,j+s)$ for channel c.

• Nonlinearity: After each convolutional layer, a nonlinear activation function is applied to the feature maps to introduce nonlinearity into the network. The most common activation function used in CNNs is the Rectified Linear Unit (ReLU), defined as follows:

$$f(x) = \max(0,x) \quad (2)$$

• Pooling: The pooling layer downsamples the output of the convolutional layer by taking the maximum or average value within a local region. The most common type of pooling is max pooling, which computes the maximum value within a window of size (pool_height x pool_width) and stride (pool_stride, pool_stride). The output of the pooling layer at position (i,j,k) can be computed as follows:

$$Y_{i,j,k}= \max_{r,s} X_{ipool\_stride + r,jpool\_stride + s,k}$$

where X is the input to the pooling layer.

These operations are typically repeated multiple times, with multiple convolutional layers followed by pooling layers, and finally one or more fully connected layers. The specific architecture of the network, including the number of layers, the size of the filters and pooling windows, and the number of output channels, can be tuned to optimize performance on a given task [10].

B. *Deep Neural Network Algorithm*

A Deep Neural Network (DNN) is a type of neural network that consists of multiple layers of interconnected nodes, with each layer processing the output of the previous layer. The basic idea behind a DNN is to learn a hierarchy of features, with each layer capturing increasingly complex and abstract representations of the input data [11].

To explain DNNs, we can use the following notation [16]:

• X: the input data, represented as a vector of size n

• Y: the output of the network, represented as a vector of size m

• W: the weights of the network, represented as a set of matrices (W_1, W_2, ..., W_l)

• b: the biases of the network, represented as a set of vectors (b_1, b_2, ..., b_l)

• f: the activation function used in the network, such as the sigmoid function or the Rectified Linear Unit (ReLU)

Here's a brief overview of the main operations involved in a typical DNN:

• Forward propagation [17]: The input data X is fed into the first layer of the network, which applies a linear transformation followed by a nonlinear activation function. The output of the k-th layer can be computed as follows:

Z_k = W_k X + b_k

A_k = f(Z_k)

where Z_k is the pre-activation vector of the k-th layer, W_k is the weight matrix connecting the (k-1)-th layer to the k-th layer, b_k is the bias vector of the k-th layer, and f is the activation function.

• Backward propagation [18]: The output of the network is compared to the desired output, and the error is propagated backwards through the layers of the network using the chain rule of calculus. The goal is to update the weights and biases in order to minimize the error. The update rule for the k-th layer can be computed as follows:

dZ_k = dA_k * f'(Z_k)

dW_k = (1/m) dZ_k A_{k-1}^T

db_k = (1/m) sum_{i = 1}^m dZ_k^{(i)}

dA_{k-1} = W_k^TdZ_k

where dZ_k is the error signal for the k-th layer, dA_k is the gradient of the activation function with respect to the pre-activation vector, f' is the derivative of the activation function, and dW_k and db_k are the gradients of the weight matrix and bias vector, respectively. The gradient dA_{k-1} is then used to compute the gradients for the previous layer, and the process is repeated until the first layer is reached.

• Optimization [19]: The weights and biases are updated using an optimization algorithm, such as gradient descent, that minimizes the cost function of the network. The update rule for the k-th layer can be written as follows :

W_k = W_k - alpha * dW_k

b_k = b_k - alpha * db_k

where alpha is the learning rate of the algorithm.

These operations are typically repeated multiple times, with multiple layers and multiple passes through the training data. The specific architecture of the network, including the number of layers, the number of nodes in each layer, and the activation function, can be tuned to optimize performance on a given task [12]

C. *Ensemble Learning*

Ensemble learning involves training multiple models and then combining their predictions to improve overall performance. There are several types of ensemble learning methods such as Bagging, Boosting, and Stacking. Bagging, Boosting, and Stacking are specific methods of ensemble learning, each with its own unique approach to combining the predictions of multiple models. Bagging involves training multiple independent models on different subsets of the training data and then aggregating their predictions. Boosting, on the other hand, involves training models sequentially, with each subsequent model attempting to correct the errors of the previous models. Stacking involves training multiple models and then using their predictions as input to a "meta-model" that learns how to combine them optimally [13].

Each ensemble learning technique has its own strengths and weaknesses, and the choice of technique depends on the specific problem at hand. In our contribution, we used bagging to combine the CNN and DNN predictions. Here some bagging advantages compared to the other ensemble techniques:

• Bagging can reduce variance [20]: Bagging can be particularly effective in reducing the variance of individual models, especially when there is high variance in the training data. By creating multiple instances of each model and training them on different subsets of the training data, bagging can reduce the overall variance of the ensemble and lead to more stable predictions.

• Bagging is relatively simple to implement: Bagging is a relatively simple ensemble technique to implement, and it doesn't require significant changes to the training procedure or architecture of the individual models [21].

• Bagging can work well with diverse models: Bagging can be effective even if the individual models in the ensemble are quite different from each other, as long as they are independently trained and can produce reliable predictions [22].

On the other hand, boosting and stacking can be more powerful ensemble techniques in certain situations. Boosting can be particularly effective when there are systematic biases in the training data, as it can help the ensemble learn to correct for these biases. Stacking can be effective when the individual models in the ensemble are complementary, as it can learn to combine their strengths and overcome their weaknesses. However, both boosting and stacking can be more complex to implement and require more careful tuning of hyperparameters [14].

Bagging consists in these main steps [15]:

• For each model in the ensemble, randomly sample a subset of the training data using "bootstrap sampling". Bootstrap sampling involves sampling the training data with replacement, meaning that each sample can be selected multiple times. Let's denote the bootstrap sample for model k as D_k*.

• ,Train each model in the ensemble independently on its respective bootstrap sample, using the same training procedure and hyperparameters as before. Let's denote the parameters (weights and biases) of model k as theta_k and the loss function as L.

• After training, combine the predictions of the models using a weighted voting scheme or averaging their predictions. Let's denote the prediction of model k on input x as f_k(x). Then, the final prediction of the bagged ensemble for input x can be computed as:

• f_ensemble(x) = (1/K) * sum_{k = 1}^K f_k(x)

where K is the number of models in the ensemble.

This formula computes the average of the predictions of each model in the ensemble, weighted equally.

The main idea of bagging is to train multiple models independently on different subsets of the training data, and then combine their predictions in some way to form a final prediction. This can help reduce the variance of the ensemble and lead to more stable and accurate predictions [23].

IV. CONTRIBUTION

Our approach is based on many steps; we will define them one by one. So, we have proposed a deep learning model for intrusion detection in the IoT network based on ensemble learning techniques by combining CNN and DNN models using Edge-IIoTset for training those models and comparing the obtained CNN and DNN models with each other and how to deal with an unbalanced dataset. So in the first part, we are going to do some preprocessing on the used dataset, then in the second part, we will create some CNN and DNN models with different architectures, and we are going to focus on the differences between those architectures (activation function and layer numbers) and compare them (train them without bootstrapping technique for the first time to ensure the comparative study). After this we are going to apply the class weights technique so we will give for each class a weight to make her more detectable in the training phase and compare the obtained results. In the final phase, having assigned weights to each class, we aim to enhance accuracy and prediction by proposing an ensemble learning model. This involves employing Bagging techniques, training models through bootstrapping, aggregating predictions, and utilizing majority voting for the final prediction. This comprehensive approach integrates preprocessing, individual model analysis, and ensemble strategies to achieve improved intrusion detection performance in IoT networks.[6]

A. *Dataset*

In our research, we utilized the Edge-IIoTset, which is a cyber-security dataset designed for intrusion detection using machine learning in the context of IoT and IIoT systems. This dataset was generated using a purpose-built IoT/IIoT tested that includes a large representative set of devices, sensors, protocols, and cloud-edge configurations. The Edge-IIoTset proposes 61 features that can be used for training and testing intrusion detection models. The dataset was created specifically for industrial IoT systems and includes network traffic collected from a diverse set of IoT devices and digital sensors. The dataset is intended to be used for detecting various types of attacks, including denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks, information gathering attacks, man-in-the-middle (MitM) attacks, injection attacks, and malware attacks []. By using this dataset, we aimed to improve the accuracy and effectiveness of intrusion detection systems in industrial IoT environments. The dataset itself is quite large, with dimensions of 157800×63, containing approximately 10 million elements.

The edge-IIoTset dataset is often large-scale, enabling comprehensive analysis and facilitating the development of scalable solutions that can handle the volume of data generated in IIoT deployments.

This dataset contains 15 classes, but this dataset has unequal data for each class. For example, the normal class contain the most data, it represents 71.43% of the dataset, while some classes don’t pass 0.03%, like “fingerprints”, which can make some classes invisible for the model or be detected as other attack. So, this is what pushes us to think of methods to make our model consider this class, and detect it correctly and know its effect on the accuracy of our model.

B. *Preprocessing*

In the initial phase of data preparation, we engaged in several essential data transformations. Specifically, we addressed the columns 'http.request.method,' 'http.referer,' "http.request.version," "dns.qry.name.len," "mqtt.conack.flags," "mqtt.protoname," and "mqtt.topic." To accomplish this, we employed the encode_text_dummy technique, specifically designed for handling qualitative or categorical data that cannot be directly interpreted by statistical algorithms. This technique effectively converts such data into binary variables, typically taking values of 0 or 1, signifying the presence or absence of a particular category within the respective variable. In the preprocessing phase, the raw data undergoes various transformations to make it suitable for deep learning algorithms. These steps encompass feature selection, encoding categorical features, and scaling the features.

1) *Feature selection*

Feature selection is a crucial step in the data preprocessing phase, aiming to enhance the efficiency and performance of machine learning models by focusing on the most relevant and informative features. In the provided example.

• Drop Function:

- The removal of these features was executed using the $drop$ function from the pandas library. The $drop$ function is a pandas method that allows the removal of specified columns or rows from a Data Frame. In this context, it was employed to eliminate the mentioned features from the dataset.

• Streamlining the Dataset:

- Feature selection is performed to streamline the dataset, ensuring that it contains only the most pertinent information for model training. This process helps reduce dimensionality, mitigate the risk of over fitting, and enhance the model's generalization to new, unseen data.

• Removing Irrelevant Information:

- By excluding certain features, the goal is to eliminate irrelevant or less informative data points that may not significantly contribute to the model's ability to make accurate predictions. This results in a more focused and efficient dataset.

2) *Splitting Data into Features and Labels*

In this stage we are going to split data into features and labels, which is a crucial preparatory step. Using this code "X **= df.drop ([label_col], axis = 1)"** and "**y = df [label_col]"**to achieves this separation, with **X** representing the input features and **y** representing the output labels.

This separation is essential for supervised learning, where the model is trained to understand the relationship between the input features and the corresponding output labels. The process involves utilizing existing data to teach the model patterns and associations, enabling it to make predictions on new, unseen data

The reason for splitting the data into features and labels is rooted in the foundational principle of supervised learning. It acknowledges that the model needs to learn from historical data where the input features (X) are associated with known output labels (y). This separation facilitates the training process, allowing the model to understand the underlying patterns in the data and develop the ability to generalize its predictions to new, unseen examples.

3) *Train-Test Split*

This is a critical step in preparing the dataset for machine learning. This process involves dividing the dataset into two subsets: one for training the machine learning model and another for evaluating its performance.

• For training models (no ensemble learning)

Using the **train_test_split** function from scikit-learn to achieve this purpose. It takes the input features (**X**) and corresponding labels (**y**) as its parameters.

test_size = 0.2

This parameter specifies the proportion of the dataset that should be reserved for testing. In this case, 20% of the data is set aside for evaluation, while the remaining 80% is used for training.

random_state = 1

This parameter ensures reproducibility by seeding the random number generator. The same random seed guarantees that the data split remains consistent across multiple runs of the code.

Stratify = y

This parameter ensures that the class distribution in the original dataset is preserved in both the training and testing sets. Stratified sampling is crucial when dealing with imbalanced datasets, where certain classes may be underrepresented.

4) *Label Encoding*

Encoding Categorical Features: Subsequently, in the second step, we employed the LabelEncoder class from the sklearn.preprocessing library to encode categorical features within the resnet dataset. This transformation is crucial since it converts categorical data into numerical form, a prerequisite for efficient functioning of deep learning algorithms

5) *Min-Max Scaling*

In this stage, the Min-Max scaling technique is applied to normalize the feature values in the dataset. Min-Max scaling is a preprocessing method that transforms numerical features to a specific range, typically between 0 and 1. This is achieved by adjusting the values proportionally based on the minimum and maximum values observed in the original dataset.

The $M \in MaxSca \leq r$ from scikit-learn is employed, creating an instance called $\min_{\max} \_ sca \leq r$. The $fit_t ran$ or m method is then used on the training set ($X_t ra \in$), which calculates the minimum and maximum values for each feature in the training data and scales the values accordingly. The same scalar is then applied to the test set ($X_t est$) using the $tran$ or m method, ensuring that both the training and test sets are scaled consistently.

Min-Max scaling is particularly beneficial for machine learning models, especially those based on distance metrics or optimization algorithms, as it helps prevent features with larger scales from dominating the learning process. By bringing all features within a uniform numeric range, Min-Max scaling contributes to improved model convergence and performance.

6) *One-Hot Encoding for Labels*

In this stage, one-hot encoding is applied to transform categorical labels into a binary vector representation, enhancing the model's ability to understand and learn from multiple classes efficiently. This technique is particularly essential when dealing with classification tasks involving more than two categories.

The to_categorical function from TensorFlow's Keras utility is employed for this purpose. It automates the process of converting numerical categorical labels into their one-hot encoded equivalents. This binary representation facilitates better model convergence and performance, especially in scenarios where the classes are not inherently ordered, and their relationships are categorical rather than ordinal.

C. *Deep learning*

• CNN 1

In first place we created CNN model, which takes an input_shape parameter representing the shape of the input data which is 92. The model is then built layer by layer. The model then applies two 1D convolutional layers with 32 and 64 filters respectively, using a 'ReLU' activation function and maintaining the input size with padding. Each convolutional layer is followed by max-pooling to reduce spatial dimensions. After this, the output is flattened to prepare for fully connected layers. The network then consists of four dense layers with progressively decreasing units (512, 256, 128, 64) and 'ReLU' activation functions. The final output layer employs a SoftMax activation function with 15 units, indicating a multi-class classification task. The model is trained using categorical cross-entropy loss and optimized with the Adam optimizer. Overall, this architecture is well suited for tasks involving sequential data, such as time series or signal processing applications.

• CNN 2

Also, we have created another CNN model alternative, which take some input shape, then It employs two convolutional layers with 64 and 128 filters, using 'ReLU' activation functions followed by Leaky ReLU activations with an alpha value of 0.1. Max-pooling operations reduce spatial dimensions. The output is flattened for fully connected layers. Three dense layers with 256, 128, and 64 units, along with Leaky ReLU activations, follow. The final layer uses a SoftMax activation with 15 units for multi-class classification. The model is trained with categorical crossentropy loss and the Adam optimizer. This architecture excels at capturing intricate patterns in sequential data, particularly in tasks like time series analysis or signal processing.

• DNN 1

A first DNN model starts with an input layer, taking in data of this dimension. The network then contains four hidden layers with 512, 256, 128, and 64 units respectively, all using ReLU activation functions. Finally, there's an output layer with 15 units, employing a SoftMax activation for multi-class classification tasks. This architecture is suitable for tasks that involve transforming data with 92 features into predictions across 15 different classes.

• **DNN2**

The second model is structured with eight dense layers. The first layer takes input with 92 features, using a ReLU activation. Subsequent layers contain 512, 256, 128, 64, and 32 units, all employing ReLU activations. The final output layer has 15 units with a sigmoid activation

• **DNN3**

The third DNN model takes the same input, and has the same architecture as the second DNN, just adding one layer after the input contains 1028 units.

Finally, the model is compiled using "binary_crossentropy," commonly used for solving binary classification problems. The Adam optimizer is employed to improve the model's weights and biases during training. Model performance is evaluated based on accuracy metrics.

D. *Class weights*

Class imbalance is a typical difficulty in binary classification tasks, as it frequently impedes accurate predictions in deep learning algorithms. When one class substantially dominates the other in terms of data samples, class imbalance arises, which skews prediction. In this case the clever use of class weights is one method for resolving class imbalance. By giving the minority class greater weights, class weights lessen bias towards the majority class and enable the model to focus more on its patterns.

Accordingly we have chosen this technique to deal with our imbalanced dataset, So we gave the minority class a higher weight so they can be more detectable by ours models in the training, such as “Fingerprinting”, “XSS” and “Port_Scanning” which are the lowest classes in our dataset.

In this stage we assigned weight to each class. These weights are calculated using scikit-learn's compute_class_weight function with balanced parameters. The purpose of these weights is to compensate for imbalances in the dataset when some classes have fewer examples than others. By assigning higher weights to underrepresented classes and lower weights to overrepresented classes, the training process can be adjusted to give more weight to less common classes. This helps the model learn and predict better from imbalanced data, ultimately improving its overall performance and fairness across different categories.

E. *Bagging*:

In our approach, we utilize ensemble learning represented as Bagging (B), wherein we combine Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs). Initially, we partition the dataset (D) into five (N = 5) random subsets (D1, D2, D3, D4, D5) using bootstrapping, denoted as {Di}, and each subset is employed to train individual models represented as M1, M2, M3, M4, M5.

Subsequently, we aggregate predictions (Pi) from these models using a majority voting mechanism (MV), which selects the class most frequently predicted (C*) to enhance the accuracy of our intrusion detection system (IDS):

IDS = MV(P1, P2, P3, P4, P5) = C*

Where C* represents the selected class, based on the majority vote, ultimately enhancing the accuracy of our intrusion detection system.

• Bootstraping

The provided code snippet illustrates the process of generating bootstrapped samples for training multiple models. Bootstrapping is a resampling technique where random samples, known as bootstrap samples, are drawn with replacement from the original dataset. These samples are then used to train individual models, enabling an ensemble approach to machine learning. Let's break down the code in a paragraph:

In this code, the goal is to create bootstrapped samples for training a specified number of models (**num_models**). The size of each bootstrapped sample is determined by the variable **sample_size**, which is set to 80% of the total number of data points (**num_data_points**). The loop iterates through the number of models to be trained, and for each iteration, a random set of indices is drawn with replacement using NumPy's **np.random.choice** function. This set of indices represents the indices of the data points to be included in the current bootstrapped sample.

• Majority Voting

The Majority Voting function employs the Counter class to tally the occurrences of each class in a given list of predictions. Utilizing $Vote_counts.most_common$, the function identifies and returns the class with the highest count, representing the majority vote. This mechanism ensures that the final prediction is determined by the most frequently predicted class across the contributing models, thereby consolidating diverse predictions into a single, robust outcome

V. EXPERIMENTATION & DISCUSSION

In our initial step, we divided the dataset into two parts: 80% of the data was allocated for training purposes, and the remaining 20% was set aside for testing. We employed the "fit" function from the Keras library to train our models. These models were compiled using the Adam optimizer, which is a popular optimization algorithm. As part of our evaluation process, we computed essential metrics including precision, recall, and the F1 score. It's worth noting that we maintained consistency by training all the models with identical hyperparamètres, which included running for 15 epochs (complete iterations through the training data) and using a batch size of 256 (the number of data samples processed in each training step). This standardization ensures a fair and comparable evaluation of the models' performance.

In first place, we start by training the 5 models by the whole dataset (edge-IIoTset) so with that we can observe the results and compare each model to each other to know his strength point and weak point. When analyzing the outcomes, we focused on key evaluation metrics, including accuracy, precision, recall, and the F1 score. Interestingly, both the DNN and CNN models yielded remarkably similar accuracy scores, reaching a value of 0.95. However, it's important to note that while accuracy provides an overall assessment of model performance, metrics such as the F1 score, precision, and recall provide a more granular perspective, offering insights into how each model performs across different classes or categories.

The result of each model showing in those Confusion matrix bellows:

Table 1
Models Accuracy

Class	**CNN1**	**CNN2**	**DNN1**	**DNN2**	**DNN3**
Backdoor	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**
DDoS_HTTP	**0.83**	**0.83**	**0.83**	**0.83**	**0.83**
DDoS_ICMP	**1**	**1**	**1**	**1**	**1**
DDoS_TCP	**1**	**1**	**1**	**1**	**1**
DDoS_UDP	**1**	**1**	**1**	**1**	**1**
Fingerprinting	**0.71**	**0.71**	**0.71**	**0.7**	**0.7**
MITM	**1**	**1**	**1**	**1**	**0.89**
Normal	**1**	**1**	**1**	**0.19**	**0**
Password	**0.27**	**0.43**	**0.91**	**0.18**	**0.17**
Port_Scanning	**0.61**	**0.61**	**0.61**	**0.61**	**0.59**
Ransomware	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
SQL_injection	**0.83**	**0.67**	**0.18**	**0.92**	**0.92**
Uploading	**0.55**	**0.55**	**0.55**	**0.55**	**0.49**
Vulnerability_scanner	**0.84**	**0.84**	**0.85**	**0.84**	**0.85**
XSS	**0.92**	**0.92**	**0.83**	**0.92**	**0.75**
Total accuracy	**0.951**	**0.95**	**0.95**	**0.37**	**0.25**

The provided table presents a comparative analysis of different classification models (CNN1, CNN2, DNN1, DNN2, DNN3) across various classes of cyber threats. The table includes accuracy scores for each class and an overall total accuracy.

In the "Backdoor" category, all models (CNN1, CNN2, DNN1, DNN2, DNN3) demonstrate a high accuracy of 0.98, indicating a consistent and robust performance in identifying backdoor threats. Similarly, classes such as DDoS_ICMP, DDoS_TCP, and DDoS_UDP exhibit perfect accuracy scores of 1 for all models, suggesting an effective detection capability against these types of denial-of-service attacks.

For "MITM" (Man-in-the-Middle) attacks, CNN1, CNN2, DNN1, and DNN2 achieve perfect accuracy, while DNN3 slightly lags with a score of 0.89. This variation may imply that DNN3 performs slightly less optimally in detecting Man-in-the-Middle attacks compared to the other models.

In the "Normal" class, CNN1, CNN2, and DNN1 exhibit perfect accuracy, while DNN2 and DNN3 show lower scores, suggesting potential challenges in accurately classifying normal network traffic, particularly for DNN2 and DNN3.

The "Password" class displays varied accuracy scores across models, with DNN1 achieving the highest accuracy of 0.91, while CNN1, CNN2, and DNN2 also demonstrate reasonable scores. DNN3, however, lags behind with an accuracy of 0.17, indicating limitations in accurately detecting password-related threats.

In the "SQL_injection" class, CNN1 and DNN1 show relatively high accuracy, while CNN2, DNN2, and DNN3 exhibit lower scores, suggesting potential challenges in identifying SQL injection attacks for these models.

The "Total accuracy" row summarizes the overall performance of each model across all classes. CNN1 and CNN2 achieve the highest total accuracy of 0.951 and 0.95, respectively, indicating their effectiveness in providing an accurate classification across the entire dataset. DNN1 also performs reasonably well with a total accuracy of 0.95. However, DNN2 and DNN3 show lower total accuracy scores of 0.37 and 0.25, respectively, indicating potential limitations in their overall performance compared to the other models.

This weakness pushes us to think of a solution to deal with this imbalanced dataset, so that is why we have proposed to apply the weighted class technique.

A. *Class Weights*

After training our model, despite of the good accuracy, that most of the models has achieved it we observed that several classes are not well detectable by our models. This issue arises due to the imbalanced dataset, leading to confusion between different classes. Therefore, we proposed a weighted class technique to address this problem, aiming to enhance the visibility and detectability of minority classes by our models.

Consequently, for each model, we assigned different weights, with a lower weight for the majority class and a higher weight for the minority class. These weights are represented below:

Table 2
Weight of each class

Class	**Weight**
Backdoor:	5.298843279052425,
DDoS_HTTP	2.622610188403931,
DDoS_ICMP	1.8739133901246834,
DDoS_TCP	2.5430478568456096,
DDoS_UDP	1.047258867764148,
Fingerprinting	149.33880742913001,
MITM	356.1156177156177,
Normal	0.09333692571528418,
Password	2.549668719437908,
Port_Scanning	6.37273599466066,
Ransomware	13.140119554466091,
SQL_injection	2.5048342801865835,
Uploading	3.458842174375693,
Vulnerability_scanner	2.544890599102138,
XSS	8.450100943057054

The result of each model showing in those Confusion matrix bellows:

In our research, we placed a particular emphasis on precision, especially when assessing the performance of each model for individual classes. Precision is a metric that signifies the accuracy of positive predictions made by a model, specifically within each class or category.

Table 3
Comparative of CNNs and DNNs Models

Class	CNN1	CNN2	DNN1	DNN2	DNN3
Backdoor	0.94	0.95	0.95	0.95	0.95
DDoS_HTTP	0.78	0.83	0.83	0.83	0.83
DDoS_ICMP	1	1	1	1	1
DDoS_TCP	0.7	0.7	0.7	0.7	0.68
DDoS_UDP	1	1	1	1	0.11
Fingerprinting	1	1	0.79	0.29	0.37
MITM	1	1	1	0	1
Normal	1	1	1	1	1
Password	0.89	0.76	0.22	0.91	0.91
Port_Scanning	0.82	0.88	0.91	0.84	0.84
Ransomware	0.97	0.98	0.98	0.97	0.98
SQL_injection	0.19	0.34	0.88	0.18	0.18
Uploading	0.55	0.55	0.55	0.18	0.43
Vulnerability_Sacnner	0.84	0.85	0.85	0.55	0.81
XSS	0.88	0.89	0.89	0.85	0.84
Total accuracy	0.953	0.954	0.949	0.948	0.89

After applying class weights, we observed a noticeable improvement in the prediction accuracy for most classes. This adjustment in the model's learning process has led to enhanced performance, particularly for classes that were previously challenging to predict accurately.

The impact of class weights is evident across various threat categories, with increased accuracy observed in classes such as "Normal" and "Password," where the initial predictions were suboptimal. The refined model, incorporating class weights, demonstrates a more balanced and effective approach to handling diverse cyber threats.

• CNN1 and CNN2

Both CNN1 and CNN2 models exhibited outstanding precision across multiple classes, achieving a precision score of 1.00 for the "DDoS_ICMP," "DDoS_UDP," "MITM," Normal," and "fingerprinting" classes. A precision score of 1.00 indicates that these models made no false positive predictions for these specific classes, signifying their exceptional accuracy in correctly identifying instances belonging to these categories.

But we have noticed that the alternative CNN (cnn2), which contains LeackyReLU in the hidden layers instead of ReLU as CNN1, has achieved better accuracy than CNN1 in more classes, as we see in "Port-Scanning," "DDoS_HTTP," and even "SQL_injection," although not with good accuracy, while CNN1 shows good accuracy in the "Password" class.

• DNNs Comparison

The third DNN model (DNN3), which has more layers than the other, has achieved a weakness accuracy of 89% compared to the other, which has fewer layers. For the other models, the two models (DNN1 and DNN2) have achieved almost the same accuracy, but the difference between those two models appears in the accuracy of each class. As we noticed from the table, the dnn1 shows better accuracy in many models: "Uploading," "SQL_injection," "Vulnerability_scanner," "Port_Scanning"," and Fingerprinting." While the DNN2 and DNN3 show good performance on "password" detection.

• CNNs and DNNs Comparison

Comparatively, all the models have achieved 100% accuracy in "DDoS_ICMP" and "MITM" except DNN2, and in "Normal" except DNN3. The DNN model, while generally performing well, exhibited slightly lower recall scores for most classes when contrasted with the CNN model. These minor differences in recall scores highlight distinctions in how these models handle specific classes or categories.

Across all models and various architectures, the CNN models consistently showcased higher accuracy than the DNN models. In several classes, both CNN and DNN models demonstrated similar accuracy. However, disparities were particularly noticeable in the "fingerprinting" class. Here, the DNN models achieved lower performance, while the CNN models excelled, accurately detecting this intrusion class.

DNN models, especially DNN1, demonstrated their strength in handling SQL injection attacks. It achieved an impressive accuracy of 0.88 for this class, surpassing the performance of other models while they achieved a lower performance.

Although, the Imbalanced dataset our models achieved better accuracy, but still there some classes like Uploading and SQL_injection causes some problem, while it cannot be detected correctly by our models.

In summary, both CNN and DNN models achieved similar overall accuracy levels. However, each model exhibited strengths in detecting specific classes, with some models excelling in classes where others performed less effectively. This diversity in strengths contributes to the overall power and effectiveness of these models in the context of intrusion detection

B. *Bagging*

All the models have individually attained almost an accuracy of 93% except DNN3 86%, indicating they are quite accurate in their predictions. However, the fact that each model provides different predictions for each class motivates us to explore ensemble techniques. Ensemble methods can help us combine these diverse predictions and potentially enhance the overall accuracy and reliability of our results by leveraging the collective insights from these models.

In our research, our primary emphasis was on optimizing precision for individual classes. Precision measures the accuracy of positive predictions for a specific class. In this context, the CNN model excelled in achieving outstanding precision, with a perfect score (1.00) for several classes, including "Backdoor," "DDoS_ICMP," "MITM," "Normal," and "Ransomware." Likewise, the DNN model demonstrated strong precision scores, notably 0.97 for the "Backdoor" class and 0.98 for the "Vulnerability_scanner" class. This indicates that both models excelled in making accurate positive predictions for these particular classes, highlighting their effectiveness in classifying these specific types of data.. In line with the principle of bagging, we trained the models concurrently. Each base model underwent independent training using a distinct subset of the data.

These subsets were generated through a process called bootstrap sampling, where data points are randomly selected with replacement. Consequently, each base model was exposed to a unique portion of the dataset, enabling them to capture and learn different facets of the underlying patterns present in the data. This diversity in training helps the ensemble of models collectively make more accurate predictions and generalize better to new, unseen data.

Once each individual base model has completed its training, it independently generates its own prediction. These individual predictions from all base models are then combined through a voting process. In this process, the class that is predicted most frequently among the base models is chosen as the final prediction of the ensemble. This decision-making method, known as majority voting, ensures that the collective wisdom of the models is used to make a final prediction, making it a robust and reliable choice for classification tasks.

The combined model showed improved performance in categorizing data into multiple classes. By harnessing the capabilities of both the Convolutional Neural Network (CNN) and Deep Neural Network (DNN) models, the averaged predictions resulted in enhanced accuracy, precision, recall, and F1 scores for a range of different classes. This indicates that by leveraging the strengths of both models, the combined approach was able to make more accurate and well-rounded predictions across various categories or classes in the dataset, ultimately enhancing its overall performance in classification tasks.

Table 4
Final prediction

Model	CNN1	CNN2	DNN1	DNN2	DNN3	Bagging
Accuracy	0.953	0.954	0.95	0.95	0.89	0.982

The combined model showed improved performance in categorizing data into multiple classes. By harnessing the capabilities of both the Convolutional Neural Network (CNN) and Deep Neural Network (DNN) models, the averaged predictions resulted in enhanced accuracy, precision, recall, and F1 scores for a range of different classes. This indicates that by leveraging the strengths of both models, the combined approach was able to make more accurate and well-rounded predictions across various categories or classes in the dataset, ultimately enhancing its overall performance in classification tasks.

Insights gleaned from this improvement suggest that the fusion of CNNs and DNNs can effectively mitigate the limitations inherent in individual models. CNNs excel in capturing spatial dependencies in data, making them particularly suited for tasks such as image recognition. On the other hand, DNNs demonstrate proficiency in learning intricate patterns and relationships within complex datasets, making them valuable for tasks requiring deeper levels of abstraction.

By integrating these complementary strengths through ensemble learning techniques like bagging, we create a hybrid model capable of leveraging the collective wisdom of diverse architectures. This not only enhances the model's ability to discern subtle nuances in different classes but also promotes greater resilience against overfitting and data biases.

Furthermore, the observed improvements in precision, recall, and F1 scores underscore the combined model's effectiveness in achieving a more balanced performance across multiple evaluation metrics. This balanced performance is crucial in real-world intrusion detection scenarios, where false positives and false negatives can have significant consequences.

Overall, the success of the combined CNN-DNN model highlights the importance of synergy in model design and the potential for ensemble learning techniques to advance the state-of-the-art in intrusion detection systems. Moving forward, further exploration of hybrid architectures and ensemble methodologies holds promise for addressing the evolving challenges posed by IoT security threats.

Conclusion

In our research, we proposed leveraging Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) to bolster the detection of IoT attacks, aiming to forge a comprehensive intrusion detection system that fortifies the security of IoT ecosystems. This endeavor marks a significant stride in the realm of IoT security, laying the groundwork for further strides in safeguarding interconnected devices and the critical data they handle.

While our research contributes substantially to advancing IoT security, it is not without its limitations. One such limitation lies in the complexity of IoT environments, where factors like device heterogeneity and dynamic network conditions can pose challenges to intrusion detection. Additionally, the efficacy of our proposed models may vary across different IoT deployment scenarios, necessitating further validation and fine-tuning.

Future research endeavors can explore several avenues to address these limitations and enhance the efficacy of intrusion detection systems in IoT networks. Firstly, delving deeper into transfer learning techniques and model generalization could bolster the adaptability and robustness of intrusion detection systems, enabling them to discern emerging threats more effectively.

Moreover, integrating real-time monitoring and response mechanisms into intrusion detection systems can augment their proactive defense capabilities, enabling swift and targeted responses to evolving cyber threats. Collaborating with industry stakeholders can provide invaluable insights into real-world IoT security challenges, fostering the development of more practical and effective solutions.

Furthermore, exploring emerging technologies such as blockchain and decentralized architectures holds promise for enhancing the security and data integrity of IoT systems. By leveraging the inherent properties of blockchain, such as immutability and decentralized consensus, IoT security frameworks can mitigate single points of failure and enhance resilience against attacks.

In summary, while our research represents a significant step forward in IoT security, there remains ample room for further exploration and innovation. By addressing the aforementioned limitations and embracing emerging technologies, we can collectively advance the state-of-the-art in protecting IoT ecosystems against an ever-evolving landscape of cyber threats.

Declarations

Ethical statements

1. The authors declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere.
2. The authors declare that they have no funding.
3. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
4. The authors have read and agree to the Terms and Conditions of Peer-to-Peer Networking and Applications journal.
5. The authors give their consent for the publication of identifiable details to be published in Peer-to-Peer Networking and Applications journal.

Availability of Data and Materials

The datasets generated during and/or analyzed during the current study are available from the authors.

References

1. Abou El Houda, Z., Brik, B., &Khoukhi, L. (2022). "Why Should I Trust Your IDS?" An Explainable Deep Learning Framework for Intrusion Detection Systems in Internet of Things Networks. IEEE

Transactions on Dependable and Secure Computing, 1-1. doi: 10.1109/TDSC.2022.3190187

2. Banaamah, A. M., & Ahmad, I. (2022). Intrusion Detection in IoT Using Deep Learning. In Advances in Cybersecurity: Principles, Techniques, and Applications (pp. 23-38). Springer. doi: 10.1007/978-3-030-90435-1_2.
3. Khan, A. R., Kashif, M., Jhaveri, R. H., Raut, R., Saba, T., &Bahaj, S. A. (2022). Deep Learning for Intrusion Detection and Security of Internet of Things (IoT): Current Analysis, Challenges, and Possible Solutions. Electronics, 11(3), 272. doi: 10.3390/electronics11030272.
4. Alghamdi, M. I. (2022). A Hybrid Model for Intrusion Detection in IoT Applications. Electronics, 11(1), 92. doi: 10.3390/electronics11010092.
5. Alkahtani, H., &Aldhyani, T. H. H. (2021). Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications. IEEE Access, 9, 73446-73456. doi: 10.1109/ACCESS.2021.3086677.
6. M.A. Ferrag, O. Friha, D. Hamouda, L .Maglaras And H. Janicke (2022). Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning. IEEE Digital Object Identifier 10.1109/ACCESS.2022.3165809
7. J. Vitorino(B) , R.Andrade , I.Praça(B) , O .Sousa , and E. Maia (2021) .A Comparative Analysis of Machine Learning Techniques for IoT Intrusion Detection.. https://doi.org/10.1007/978-3-031-08147-7_13
8. Walid I. Khedr, Ameer E. Gouda, And Ehab R. Mohamed (2023). A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks . IEEE 10.1109/ACCESS.2023.3260256
9. M.Radhi Hadi, A .Saher, M. David, and C. Wyld (2022). A NOVEL APPROACH TO NETWORK INTRUSION DETECTION SYSTEM USING DEEP LEARNING FOR SDN:. DOI: 10.5121/csit.2022.121106
10. Mahmoud Said El Sayed, Nhien-An Le-Khac, Marwan Ali Albahar, Anca Jurcut (2021). A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique https://doi.org/10.1016/j.jnca.2021.103160 2021
11. Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke (2021) Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study
12. Ziadoon K. Maseer, Robiah Yusof, Salama A. Mostafa, Nazrulazhar Bahaman, O .Musa and Bander Ali Saleh Al-rimy (2021) DeepIoT.IDS: Hybrid Deep Learning for Enhancing IoT Network Intrusion Detection DOI:10.32604/cmc.2021.016074
13. Thomas Lagkas, Konstantinos Rantos, (2022). Deep Learning in IoT Intrusion Detection Stefanos Tsimenidis1 ·https://doi.org/10.1007/s10922-021-09621-9
14. Sahba Baniasadi, Omid Rostami, Diego Martín , and Mehrdad Kaveh (2022). A Novel Deep Supervised Learning-Based Approach for Intrusion Detection in IoT Systems https://doi.org/10.3390/s22124459
15. Baraa I. Farhan, Ammar D. Jasim (2022) Survey of Intrusion Detection Using Deep Learning in the Internet of Things DOI: https://doi.org/10.52866/ijcsm.2022.01.01.009

16. Elena Fedorchenko , Evgenia Novikova, and Anton Shulepov (2022). Comparative Review of the Intrusion Detection Systems Based on Federated Learning: Advantages and Open Challenges https:// doi.org/10.3390/a15070247
17. Abeer Ali Alnuaim, Mohammed Zakariah, Chitra Shashidhar, Wesam Atef Hatamleh , Hussam Tarazi, Prashant Kumar Shukla, and Rajnish Ratna (2022). Speaker Gender Recognition Based on Deep Neural Networks and ResNet50 https://doi.org/10.1155/2022/4444388
18. J. Toldinas, A. Venckauskas, A. Liutkevicius and N. Morkevicius (2022) .Framing Network Flow for Anomaly Detection Using Image Recognition and Federated Learning https://doi.org/ 10.3390/electronics11193138
19. Tatsuya Yasui, Takuro Tanaka, Asad Malik and Minoru Kuribayashi (2022). Coded DNN Watermark: Robustness against Pruning Models Using Constant Weight Code. https://doi.org/10.3390/ jimaging8060152
20. Safi Ullah, Muazzam A. Khan, Jawad Ahmad, Sajjad Shaukat Jamal, Zil e Huma, Muhammad Tahir Hassan, Nikolaos Pitropakis, Arshad and William J. Buchanan. (2022) HDL-IDS: A Hybrid Deep Learning Architecture for Intrusion Detection in the Internet of Vehicles https://doi.org/10.3390/s22041340
21. Jawad Yousef Ibrahim Alzamily, Syaiba Balqish Ariffin, Samy S. Abu Naser (2022) Classification Of Encrypted Images Using Deep Learning –Resnet50 E-ISSN: 1817-3195 www.jatit.org
22. Farah Jemili : Intelligent intrusion detection based on fuzzy Big Data classification. *Cluster Comput* (2022). https://doi.org/10.1007/s10586-022-03769-y
23. Ahlem Abid, Farah Jemili and Ouajdi Korbaa : Distributed architecture of an Intrusion Detection System in Industrial Control Systems, *ICCCI 2022 14th International Conference on Computational Collective Intelligence,* 2022-09

Figures

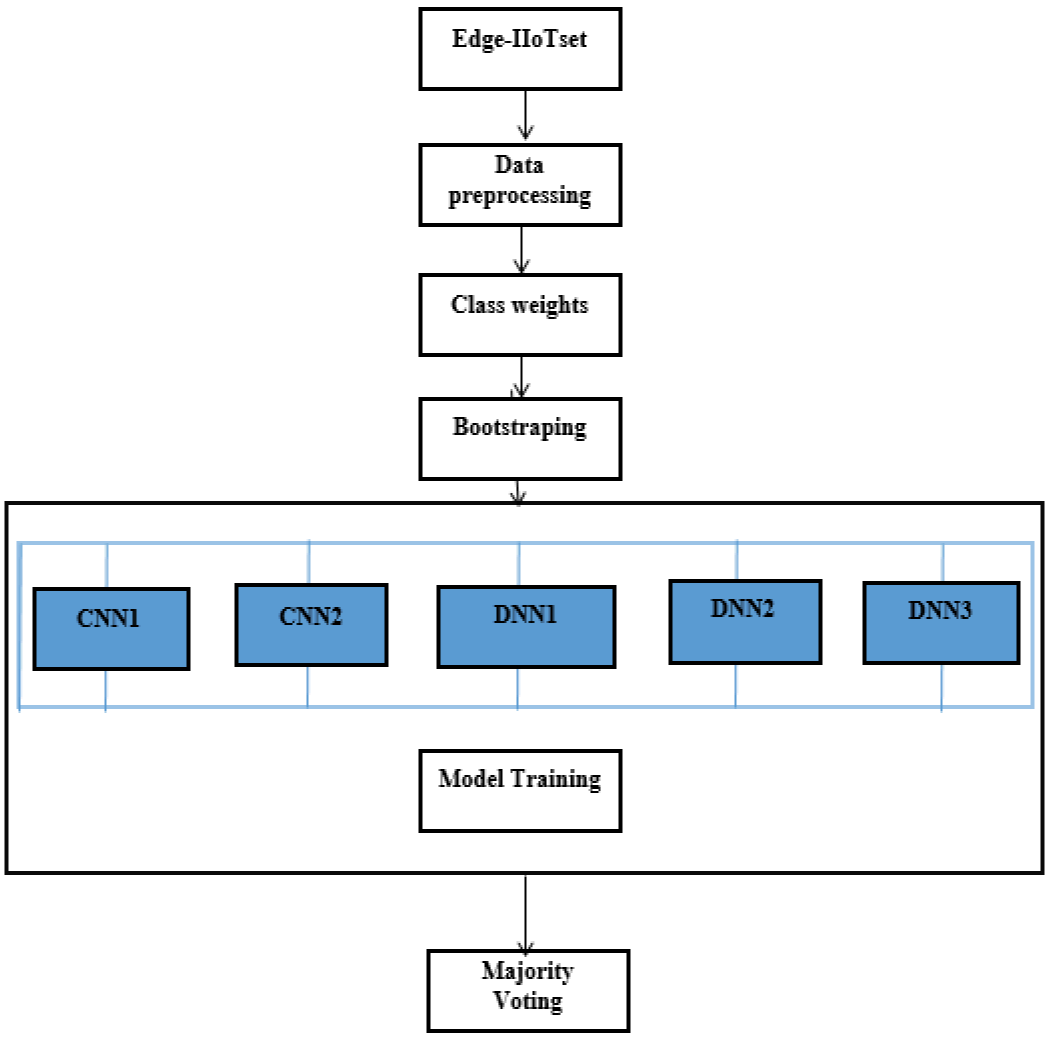

Figure 1

Proposed Contribution

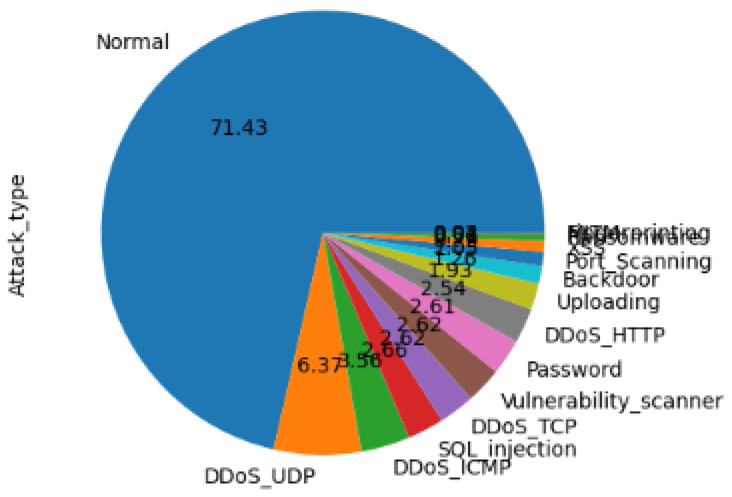

Figure 2

Distribution of Edge-IIoTset classes

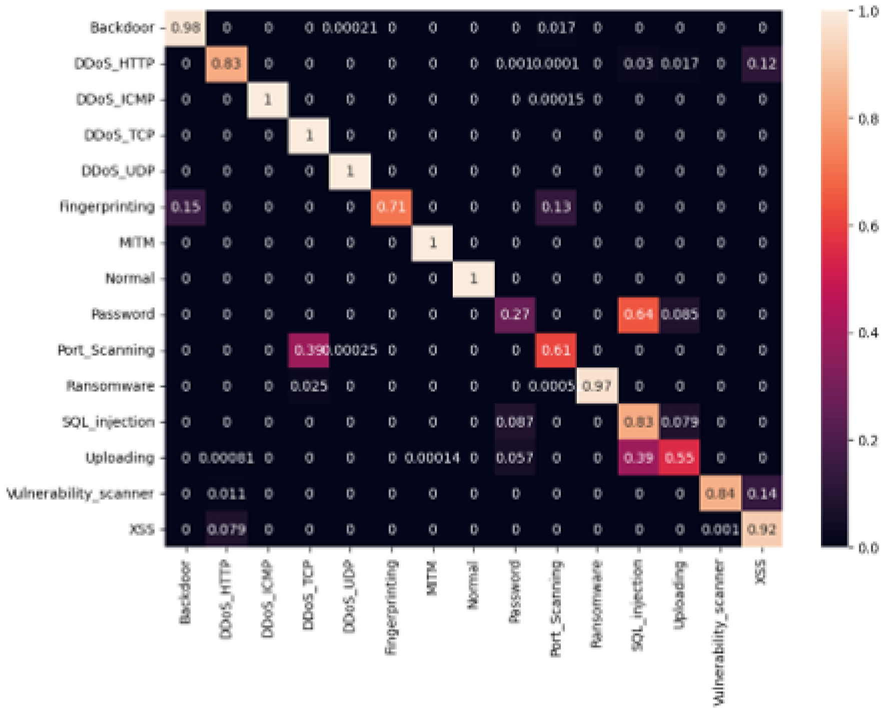

Figure 3

Confusion matrix of DNN1

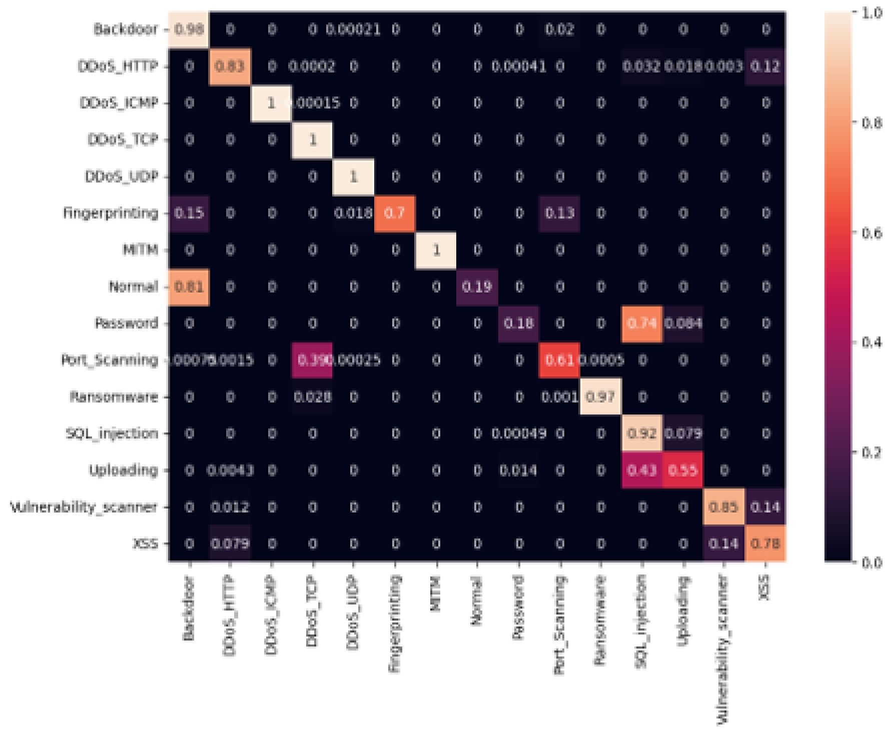

Figure 4

Confusion matrix of DNN2

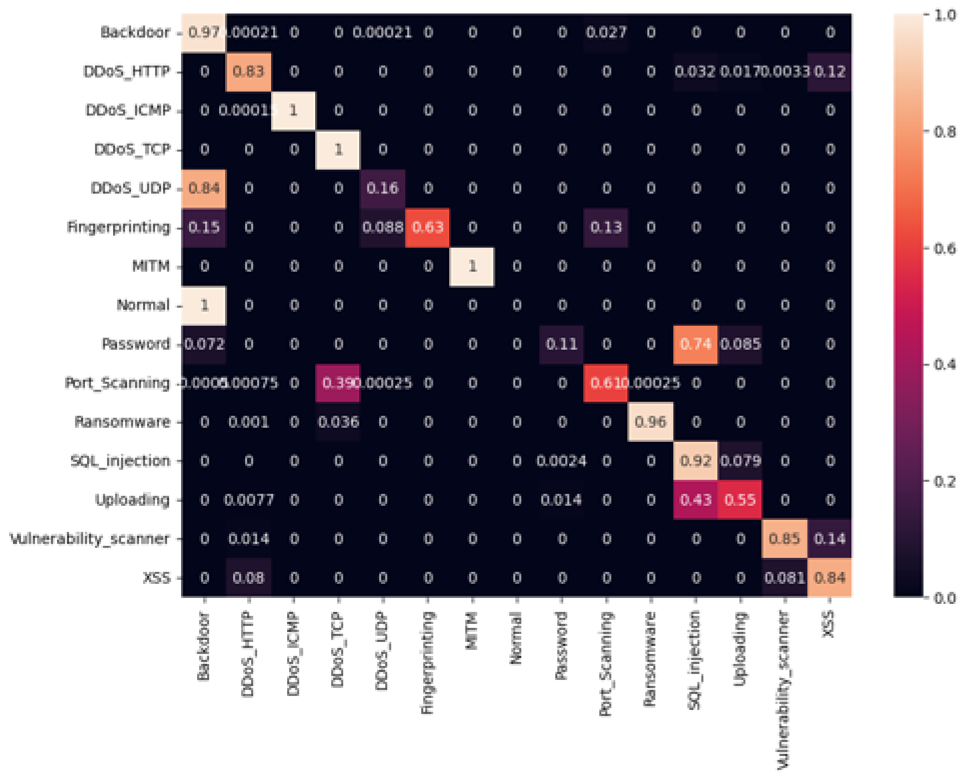

Figure 5

Confusion matrix of DNN3

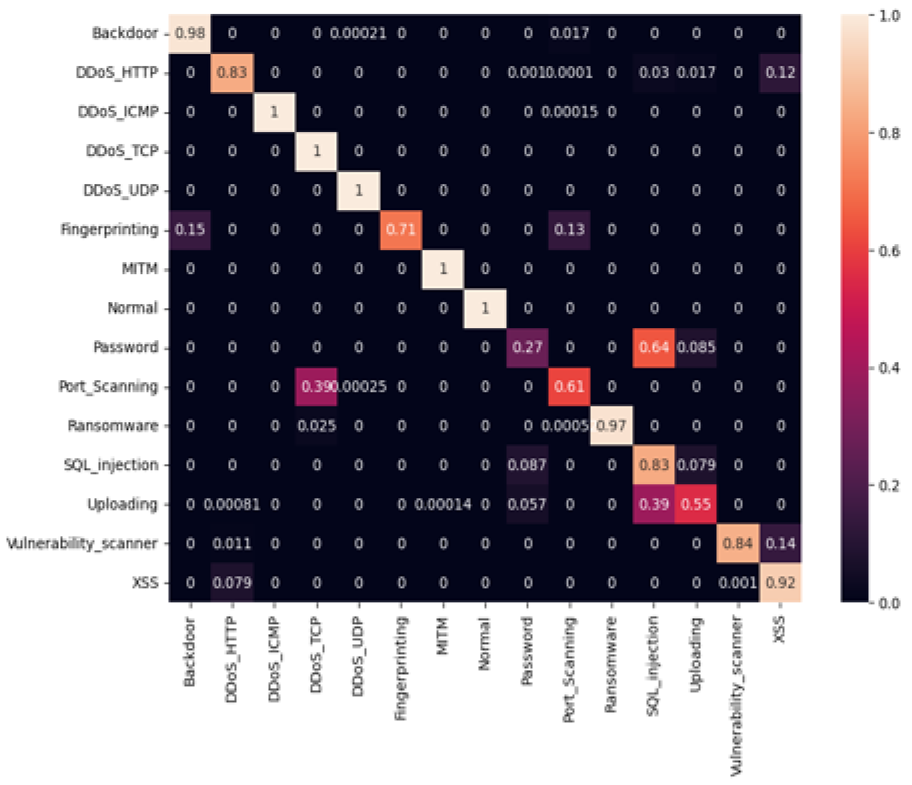

Figure 6

Confusion matrix of CNN1

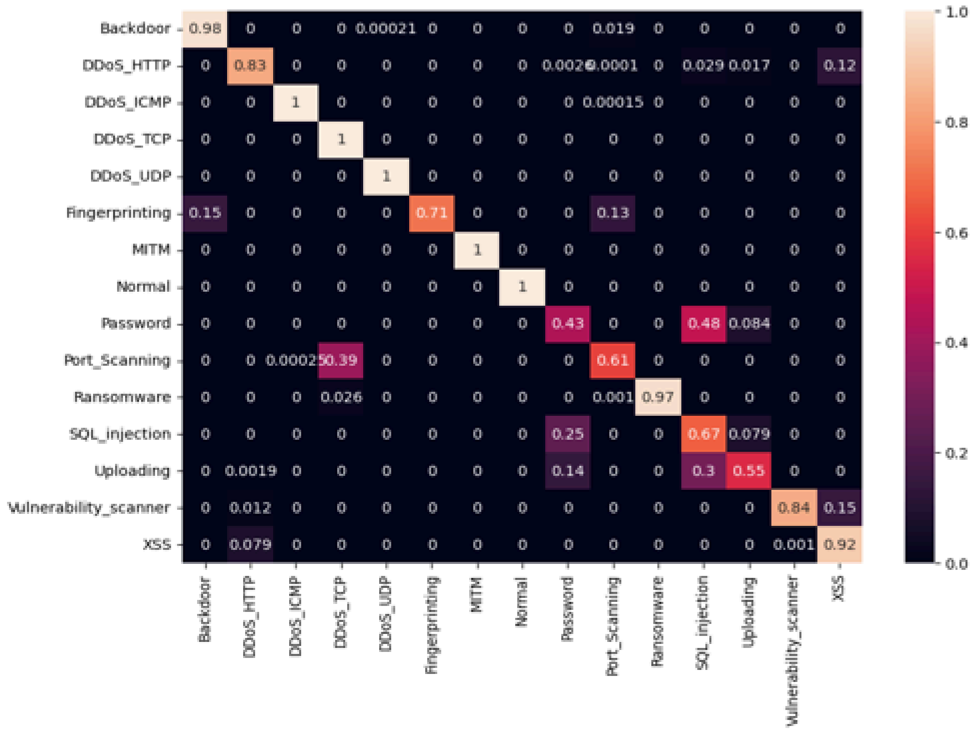

Figure 7

Confusion matrix of CNN2

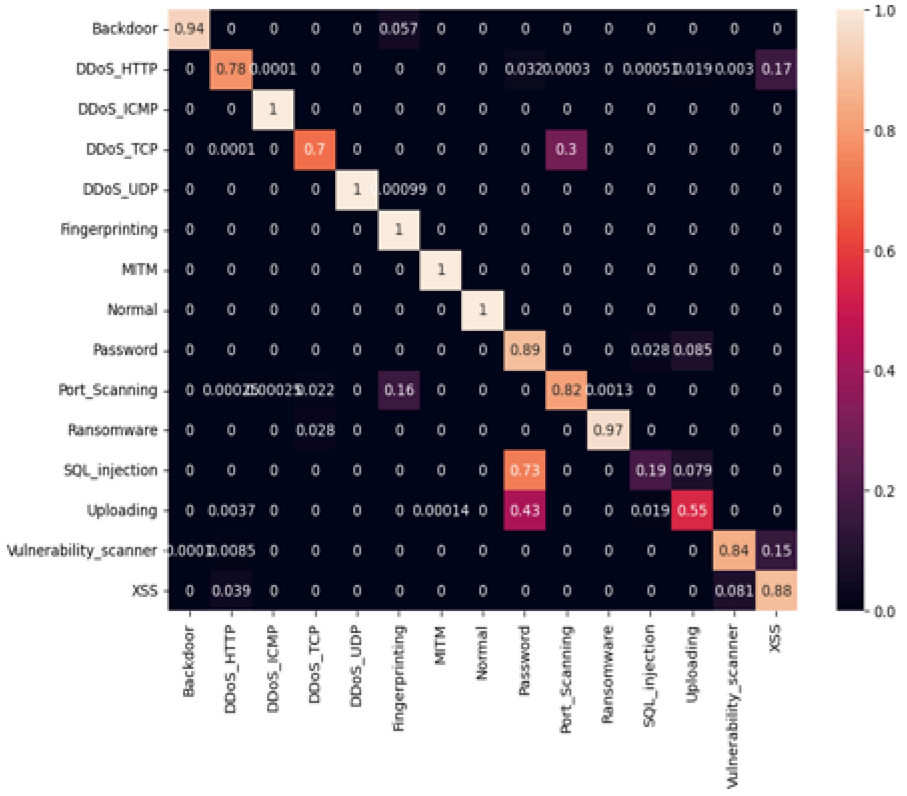

Figure 8

Figure 9:CNN1 confusion matrix after class weight

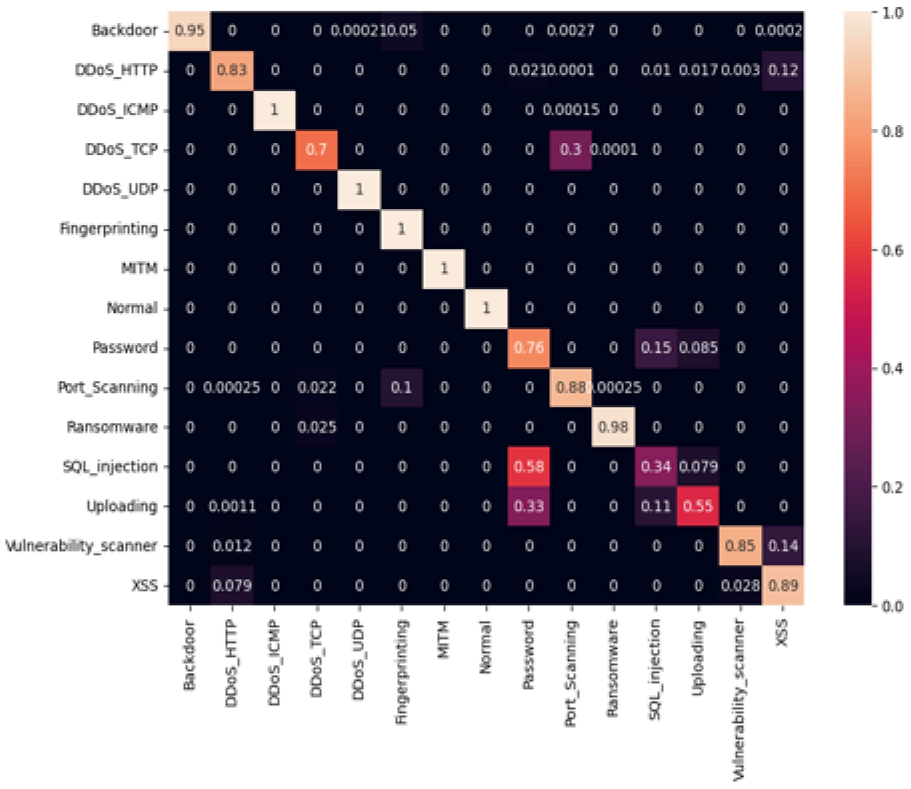

Figure 9

Figure 10: CNN2 confusion matrix after class weights

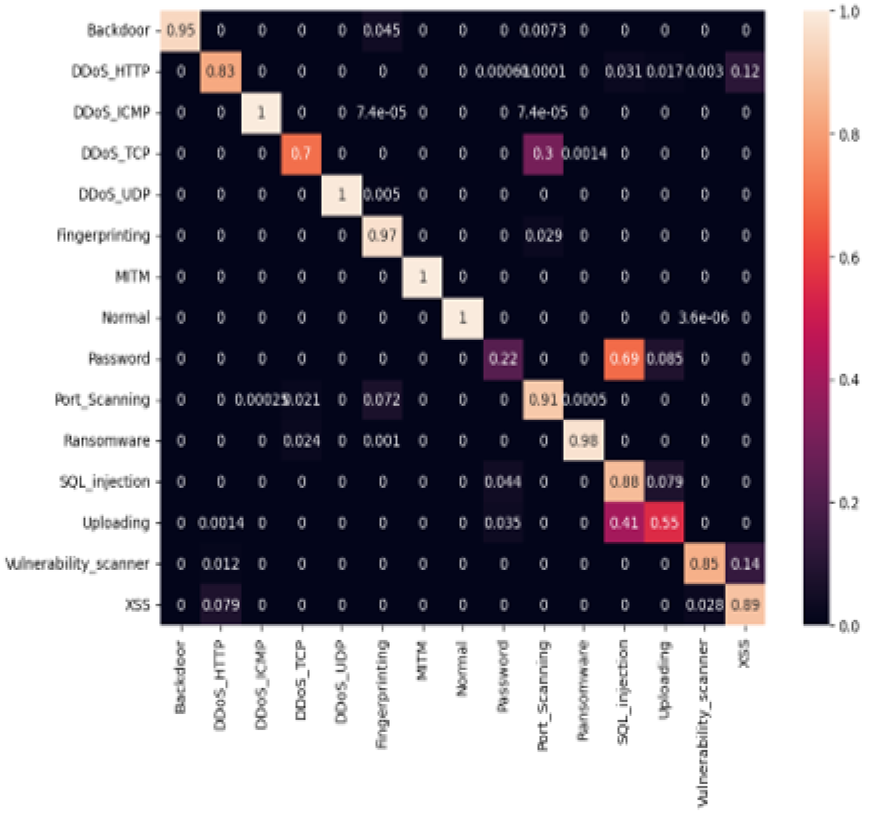

Figure 10

Figure 11: DNN1 confusion matrix after class weights

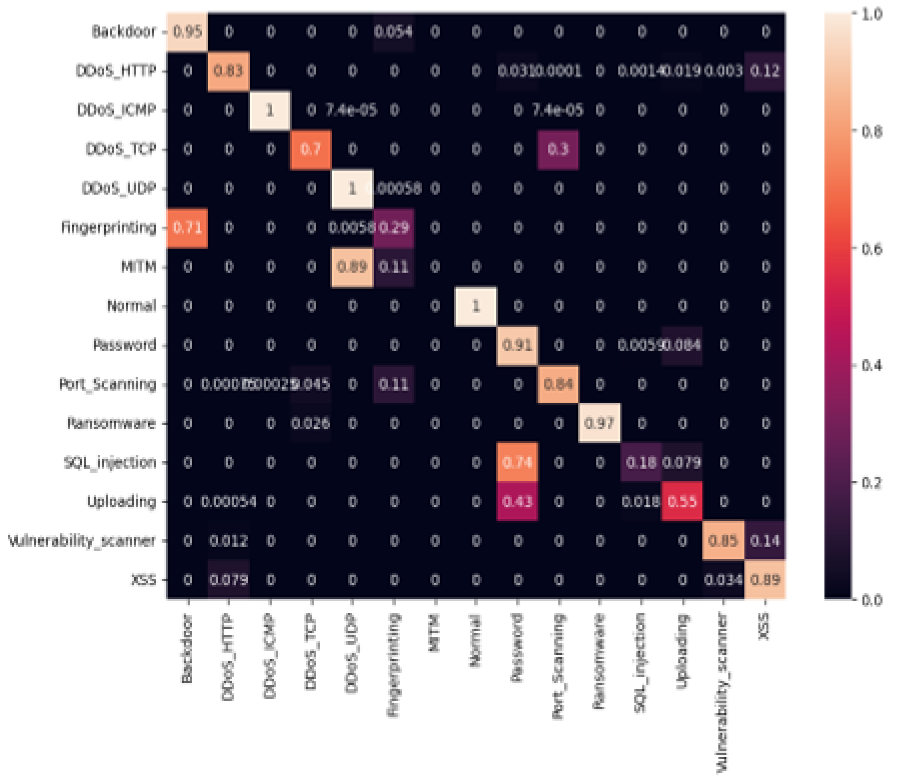

Figure 11

Figure 12: DNN2 confusion matrix after class weights

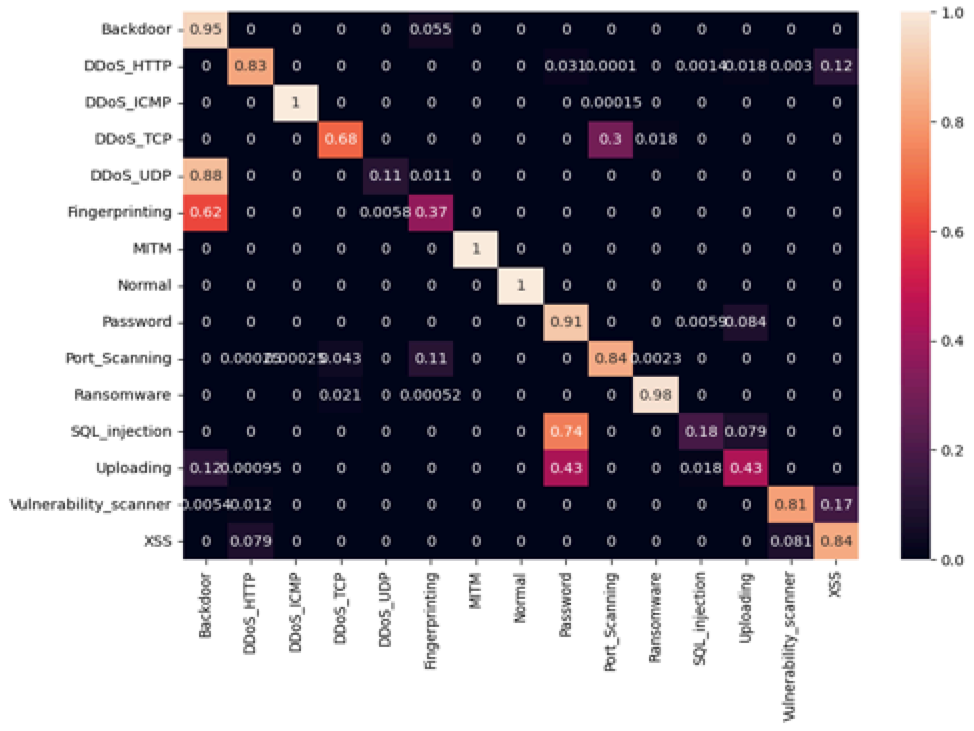

Figure 12

Figure 13: DNN3 confusion matrix after class weight