MSc Computer Science: Data Science

Feature Reduction Methods for Cyber Attack Detection

Anne Liebens
s1838458

Supervisors:
Olga Gadyatskaya & Nele Mentens

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/06/2021

Abstract

Deep learning is a promising technique for the automatic detection of cyber attacks. Deep learning models benefit from feature reduction, which reduces noise in the data and can therefore increase the performance of the model. Comparing the performance of feature reduction methods on attack detection models can be challenging as different works might use different datasets and different preprocessing methods that can act as confounding factors.

In this work the effects of using features selected by Principle Component Analysis (PCA), random forest, correlation and the sequential search algorithm on the performance of a simple deep learning model are investigated using the CICIDS2017 dataset as input. We found that PCA improved the performance the most and features related to IP address, port number and packet information seem to be the most valuable for a cybersecurity dataset when the aim of its use is related to attack detection.

Contents

1 Introduction **1**
- 1.1 Goal and problem . . . 2
- 1.2 Structure . . . 2

2 Background **2**
- 2.1 Intrusion detection . . . 2
- 2.2 Deep learning . . . 3
- 2.3 Feature reduction . . . 3
  - 2.3.1 Principal component analysis . . . 4
  - 2.3.2 Random forest . . . 5
  - 2.3.3 Correlation . . . 5
  - 2.3.4 K-means clustering . . . 6
  - 2.3.5 Search . . . 7

3 Data **8**
- 3.1 In the literature . . . 11

4 Experiments **18**
- 4.1 Performance measures . . . 18
- 4.2 Preprocessing . . . 19
- 4.3 Model . . . 20
- 4.4 Applying PCA . . . 22
- 4.5 Applying random forest . . . 23
- 4.6 Applying correlation . . . 24
- 4.7 Applying K-means clustering . . . 27
- 4.8 Applying search algorithm . . . 28
- 4.9 Experimentation with the full labelset . . . 31

5 Discussion **33**
- 5.1 Results . . . 33
- 5.2 Comparing to other works . . . 36
- 5.3 Limitations . . . 40

6 Conclusion **40**

References **44**

1 Introduction

As digitalization continues to spread and the Internet of Things (IoT) grows, security concerns arise. User information might get compromised, and thus, while a growing IoT brings many opportunities to simplify daily life, the security of data needs to be guaranteed [5]. More and more devices are connected to the internet, not only phones and computers but also microwaves and cars can communicate nowadays. Because of this growing number of internet applications, there are not only more opportunities for malicious actors to gain access to sensitive information that can be abused, but there is also more to gain from this information, passwords might for example be reused for multiple services. Unwanted events such as data breaches and server shutdowns (caused by for example Distributed Denial of Service (DDoS) attacks) inconvenience consumers [10] and cost a lot of money [9]. Therefore it is important that malicious activity is detected before true harm can be done to the system that is under attack.

Cyber attacks are getting more prevalent, especially during the COVID-19 pandemic where online communication is more important than ever before [14]. According to the European Union Agency for Cybersecurity (ENISA), between January 2019 and April 2020 there was a 52% increase in the reported web application attacks compared to the year before [13], the third quarter of 2019 saw a 241% increase in the amount of DDoS attacks compared to the same period the year before [12] and by the midyear, a 54% increase of the amount of data breaches had been observed compared to 2018 [11].

Huyesin et al. [17] mention that cyber attacks do not necessarily need to be complicated. For example, in 2007, the communication architecture of Estonia was paralysed by DDoS attacks due to disagreements among the population regarding a statue of a Russian soldier. As technology evolves, automation of more intricate attacks and attacks of larger scale become a more prominent threat and the prevention of cyber attacks becomes more important.

In order to prevent cyber attacks, it is important to know how they can be detected. For this purpose machine learning can be used. Machine learning is used for a wide range of applications and lends itself for prediction, detection and other uses [38]. Machine learning techniques range from clustering and classification to deep learning and is therefore very diverse. However, in order to train a good machine learning model, a dataset is needed that contains sufficient examples. Over the years, multiple datasets containing cybersecurity data have been developed, such as the DARPA dataset [1], which already dates to 2000 (with earlier versions dating back to 1999 and 1998) and is therefore already quite old. Luckily, more recent datasets such as the NB-UNSW dataset (2015) [26], the LANL dataset (2017) [39] and the CICIDS-2017 dataset (2017) [35] are also publicly available. These datasets are often used to train and test deep learning models, however, comparing the performance of these models should be done carefully. Data preprocessing and feature reduction play an important role in the performance of deep learning methods and these factors are rarely kept constant over different research papers. Additionally, feature reduction methods can reveal what type of features are important, which can be beneficial to keep in mind when new cybersecurity datasets are created. Therefore, this work aims to explore the effects of feature reduction methods on the performance of a deep learning model for cyber attack detection.

1.1 Goal and problem

The aim of this work is to **investigate the effects of different feature reduction methods on the performance of deep learning models for cyber attack detection and to find out what features are the most informational**. This information can be helpful for the creation of new cybersecurity datasets or models that deal with the detection of cyber attacks.

With data from the CICIDS2017 dataset the following research questions will be addressed:

1. How does a simple deep learning model from the literature perform on the CICIDS dataset?
2. What are the effects of different feature reduction methods on the performance of the model?
3. Which features are deemed important by the different feature reduction methods?

1.2 Structure

First we outline relevant background information in Chapter 2. Then we will explore the dataset and present some relevant works that have worked with the dataset in Chapter 3 before moving on to Chapter 4 where our experiments are presented. In Chapter 5 we compare our work to other works executed in the field and in Chapter 6 we close with concluding remarks.

2 Background

2.1 Intrusion detection

Intrusion detection can be subdivided into multiple categories, as described by Husak et al. [16]. Firstly, there is *attack projection*, which is focused on finding out the next steps of the attacker in an attack that is already ongoing. Then, there is *intent recognition*, which is focused on finding out the final goal of the attacker. *Intrusion prediction* deals with finding out what attacks are going to happen in the future (and is thus different from attack projection). Finally, there is *network security situation forecasting*, which looks at, for example, the vulnerabilities in a system. Husak et al. [16] recognize 7 steps that each cyber attack follows and that can be detected: *scanning*, *enumeration*, *intrusion attempt*, *elevation of privilege*, *perform malicious tasks*, *deploy malware or create a backdoor*, and finally *delete the evidence and exit the system*. With this overview it is easier to predict what the next step of an attacker is going to be. Attack detection methods that follow this 7 step model are called *similarity-based methods*, meaning that a model for each attack is needed so that the observed behavior can be compared to this model. *Intent recognition* models follow a similar approach but keep the identification of the end goal more in mind. *Intrusion prediction* deals with the prediction of attacks that have not yet been seen. Methods that do not depend on the detection of pre-set patterns are called *anomaly-based or dynamic methods*. Similarity-based and anomaly-based methods can also be combined to detect both known and unknown attacks [34].

The first methods for attack detection were based on discrete models such as *attack graphs* (in which the nodes represent possible attack events and the edges their predictability) [41, 16]. From attack graphs *Bayesian networks* can be created, they are probabilistic models representing the relationships between the variables, which are the attacker and the different components of the

system [32, 16]. *Markov models* are also often found in the literature, they are based on states and the transition probabilities between them [16]. They also perform well in situations where not all information is available, meaning that some states are hidden [27]. These discrete models are similarity-based methods and therefore their predictive qualities are limited to the attacks that they are modeled after.

Next, there are *machine learning* methods, which are anomaly-based methods and can be subdivided into supervised and unsupervised techniques. *Supervised techniques* require labeled data to evaluate a model while it trains. Methods that fall under the supervised techniques are for example classifiers such as the *decision tree* and *support vector machine*. *Unsupervised techniques* include clustering approaches, such as *hierarchical clustering* or *K-means clustering* [19]. Machine learning methods are more flexible than similarity-based methods as they can learn patterns from the data and might therefore detect behavior that a predefined model has not accounted for. However, this does mean that the dataset that is used to create the machine learning model needs to be of sufficient quality to create a meaningful model. *Deep learning* is also a form of machine learning, centered around mimicking the connections in the human brain with neural networks [38]. Deep learning models therefore provide a multitude of opportunities for solving complicated problems but can also be challenging to work with as they can be quite complex.

2.2 Deep learning

A deep learning model is a *processing model* that is composed of multiple layers [22]. Each layer transforms the data to a new level of abstraction, which allows for the extraction of new features at each layer. For example, in an object recognition system for images the first layer detects the edges that are present in the image while subsequent layers detect more complex arrangements of edges. Another advantage of this architecture is that it allows for generalization over images: the same object can be recognized in different images regardless of it's orientation or the lighting. Each layer is made up of a *set of weights* that transform the input to the layer. During training, these weights are tweaked in order to get the best performance. The gradient vector describes how much the error would increase or decrease based on a small change in the weight that is under consideration. A negative gradient vector shows a decrease in the space of the objective function, which is a generalization of the error over all training examples. This decrease means that the overall error moves closer to a minimum, which is desirable [22].

2.3 Feature reduction

Using features that are not relevant leads to *data redundancy* and thus noise in the data, which might degrade the performance of the model and increase the amount of *storage space* needed [7, 23]. Additionally, as noted by for example Vinayakumar et al. [40], feature reduction can *cut down training and testing time* and Bouktif et al. [6] point out that a deep learning model might *overfit* if too many parameters need to be taken into consideration.

Feature reduction can be subdivided into *feature extraction* and *feature selection*. In feature extraction, the original features are projected to a new feature space with lower dimensionality. In feature selection, a subset of the original features is chosen to represent the dataset [23].

Below we outline the feature reduction methods that will be applied in this work.

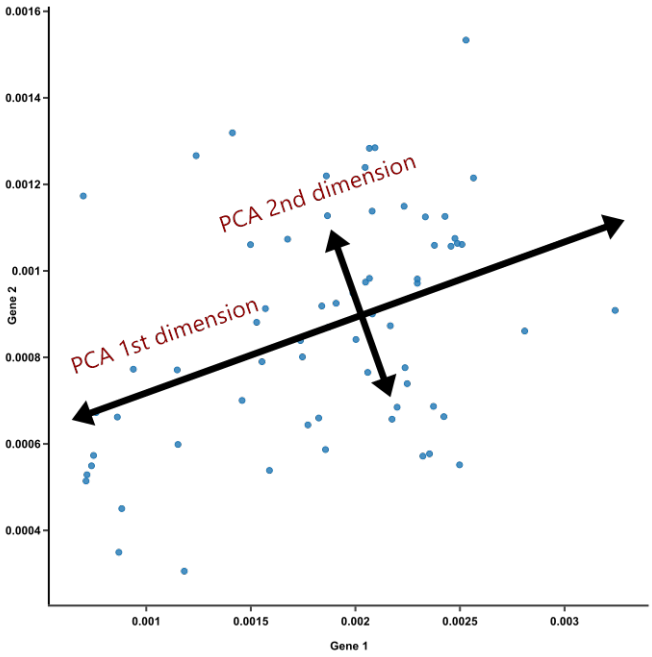

Figure 1: PCA explained. Image taken from https://blog.bioturing.com/2018/06/14/principal-component-analysis-explained-simply/.

2.3.1 Principal component analysis

Principal component analysis (PCA) is a technique that aims to reduce dimensionality while retaining as much information as possible [18] and is therefore a feature extraction technique. It can be helpful in visualizing the data, as the principal components are essentially linear combinations of the variables that have no co-variance with respect to each other and therefore maximize variance. Firstly, the variables are standardized so that they all cover the same range. The data can be represented in the $n \times p$ matrix X, where n represents the number of samples and p the number of variables. The goal is now to find a linear combination of columns of X which maximizes the variance:

$$\sum_{j=1}^{p} a_j x_j = Xa \tag{1}$$

where a is a vector of constants that add up to 1. The variance of these linear combinations is $var(Xa) = a'Sa$ where S is the covariance matrix of X. If we then want to find which linear combination of columns maximizes vectors, we need to solve $Sa = \lambda a$, where λ are eigenvalues and a eigenvectors. The most interesting eigenvector is the one associated to the largest eigenvalue because the eigenvalue represents the variance of the vectors in Xa. The vectors Xa are then the principal components, where the values in a are the loadings of the components [18]. Fig. 1 shows an example of how PCA restructures the data: the first component lies on the axis that maximizes the variance. Then the second component lies on an axis perpendicular to the axis of the first component.

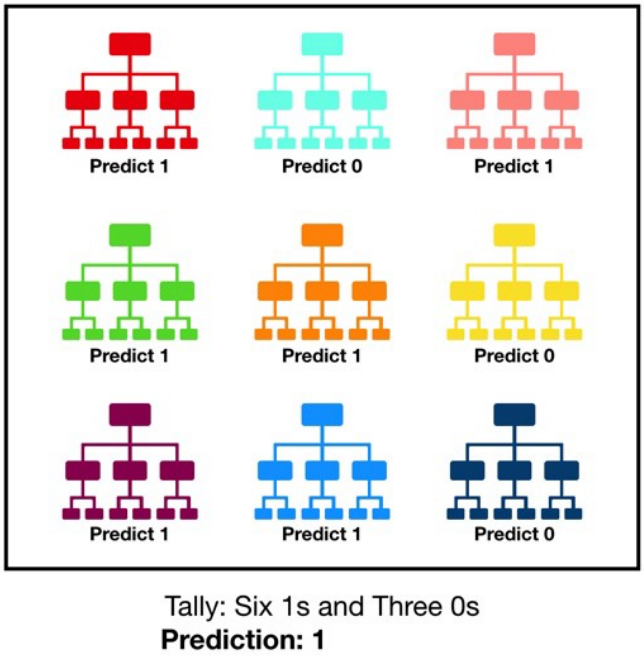

Figure 2: Random forest example. Image taken from https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

2.3.2 Random forest

In [35], the random forest classifier is applied to extract the features that are most important for the detection of different attacks. In a random forest, a collection of decision trees, which are lowly correlated to each other, is created, each with a different subset of features. Each tree will provide a prediction and the class that is chosen by most trees forms the final prediction of this ensemble, as is shown in Fig. 2[1]. Each decision tree in the forest creates its own prediction of the attack class and the prediction that has the majority vote, in this case 1, will be the final prediction for the input that has been fed. Additionally, the SciKit Learn implementation of the random forest classifier that will be used in this work provides the ability to look at feature importance[2].

2.3.3 Correlation

Looking at correlation is already a way of eliminating variables. Features that highly depend on other features do not convey a lot of new information and instead only contribute noise. These features can therefore be removed [7]. Additionally, if a variable is not correlated to the target, it can also be considered as redundant [7]. Correlation can be evaluated with several metrics, such as the Pearson correlation coefficient or the ANOVA f-score. In this work, the ANOVA f-score is used, as this metric is suitable to test numerical data and categorical targets, to test the dependency of the target on the data and then use the chosen features as input for a deep learning model. For

[1]For more information on random forests see https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[2]See https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

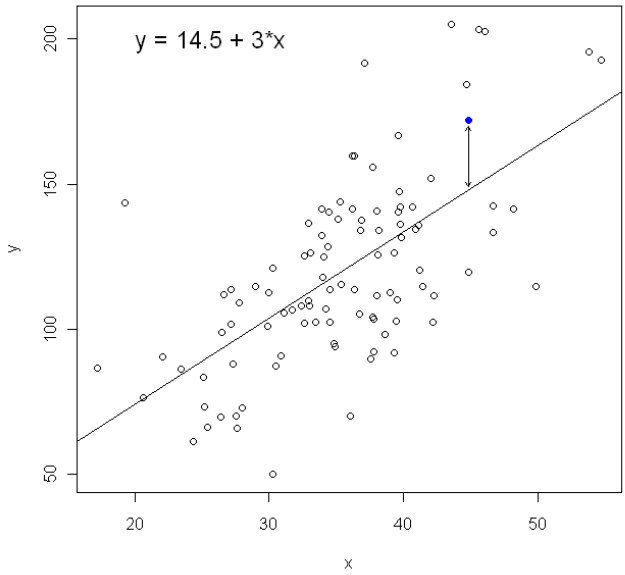

Figure 3: Regression line example. Image taken from https://stats.stackexchange.com/questions/22718/what-is-the-difference-between-linear-regression-on-y-with-x-and-x-with-y.

each feature, a regression line can be plotted to estimate the relationship of the feature to the target, an example of such a regression line is shown in Fig. 3. The f-score is defined as follows:

$$f = \frac{SS_{mean} - SS_{fit}/(p_{fit} - p_{mean})}{SS_{fit}/n - p_{fit}} \tag{2}$$

SS_{mean} is the sum of squares around the mean, while SS_fit refers to the sum of squares around a fitted line. p_{mean} refers to the number of parameters in the equation for the mean, which is thus 1. p_fit refers to the number of parameters in the equation for the fitted line and n stands for the number of samples.
It should be noted that the f-score can only convey information about the effect of individual features on the target, information that is conveyed by the combination of features is not taken into account [8].

2.3.4 K-means clustering

In K-means clustering, the data are clustered into k parts. k is set to the number of attacks found in the dataset and then each feature is used separately to create clusters [15]. This method is unsupervised, meaning that it does not look at the labels of the data and therefore is suitable in cases where it is hard to obtain labels for a dataset. The goal of the algorithm is to choose a mean (centroid) for each cluster that minimizes the inertia within each cluster, which is the variance of all points belonging to that cluster. The initial centroids can be chosen at random after which each point in the dataset is assigned to the cluster which centroid lies closest. After this assignment, the centroids are shifted to the point that represents the mean of all datapoints that were assigned

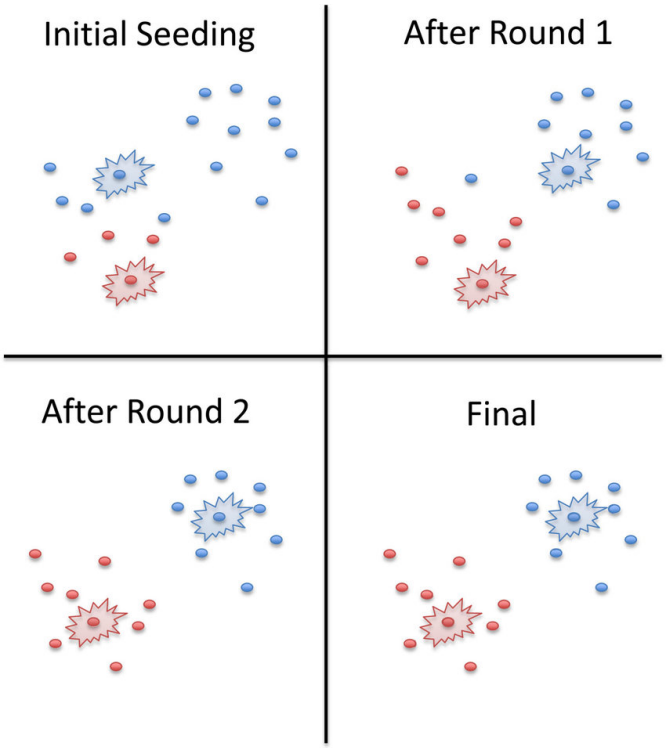

Figure 4: K-means clustering example. Image taken from https://www.researchgate.net/figure/K-means-clustering-algorithm-An-example-2-cluster-run-is-shown-with-the-clusters_fig3_268880805.

to the cluster. The procedure restarts until the centroid locations converge or until a prespecified number of iterations has completed. An example is shown in Fig. 4. The clusters are disjoint and the desired number of clusters needs to be pre-specified before running the algorithm. This means that experimentation might be needed to find the optimal number of clusters if the data are unlabeled. As the algorithm depends on randomization it will return different clustering results each time it is run (these issues can be alleviated by setting a random seed)[3].

2.3.5 Search

Sequential feature selection can also be used to look into the important features and as far as we know it has not yet been applied for feature reduction for the CICIDS dataset. This algorithm first considers each feature individually and chooses the feature that performs best on the classification task. Then it will add more features to the subset based on which feature maximizes performance until the feature subset has the desired size [7]. However, applying the search algorithm with a deep learning model can be very time consuming depending on the number of features in the dataset and the training time allocated to the deep learning model. How these constraints affect this work is discussed further in Chapter 4.

[3]For more information on K-means clustering see https://scikit-learn.org/stable/modules/clustering.html#k-means.

3 Data

There are many datasets for cyber security analysis, such as the KDD Cup 1999 dataset [2] and its improved version, the NSL-KDD dataset [3]. Many of these datasets are now outdated, as new attack patterns are not included. This work makes use of the *CICIDS2017 dataset* developed by Sharafaldin et al. [35]. The dataset contains synthetic data but according to Maseer et al. [25] this dataset is the preferred one for training machine learning methods for anomaly detection systems due to its use of user profiling as this makes the dataset more realistic. The dataset was created by employing a *victim network* and an *attack network* and recording traffic between them. The victim network contains servers, switches, PC's with Windows, Linux and MacOS operating systems, and a firewall. The attack network consists of a router, a switch and 4 PC's running Kali Linux and Windows as operating systems. The IP addresses of the different components can be found in Tab 1. The benign traffic mimicks the behavior of 25 users using HTTP, HTTPS, FTP, SSH, and email protocols. The data are recorded over a period of seven days, where on the first day (Monday) only benign traffic is detected and on the other days different attacks are recorded. Tab. 2 shows the different types of attacks that are found in the dataset[4].

IP Address	Network	Role
192.168.10.3	Victim	Server
192.168.10.50	Victim	Server
205.174.165.68	Victim	Server
192.168.10.51	Victim	Server
205.174.165.66	Victim	Server
192.168.10.19	Victim	PC
192168.10.17	Victim	PC
192.168.10.16	Victim	PC
192.168.10.12	Victim	PC
192.168.10.9	Victim	PC
192.168.10.5	Victim	PC
192.168.10.8	Victim	PC
192.168.10.14	Victim	PC
192.168.10.15	Victim	PC
192.168.10.25	Victim	PC
205.174.165.73	Attacker	PC
205.174.165.69	Attacker	PC
205.174.165.70	Attacker	PC
205.174.165.71	Attacker	PC

Table 1: IP addresses in CICIDS dataset.

All network traffic is labeled, however, there is a huge *imbalance* in the data as can be seen in Tab. 3 and Fig. 5, the number of records describing benign traffic is not balanced with the number

[4]All data exploration presented in this section is done on a version of the full CICIDS dataset from which all records that contain one or more null values have been removed.

Brute force	Trial and error until success, for example to crack passwords or discover hidden pages.
Heartbleed	Based on a bug in the OpenSSL library used to implement the Transport Layer Security protocol. Sends a request with a large length field hoping the receiver responds.
Botnet	Multiple devices are controlled to complete tasks such as data theft.
DoS attack	Flooding a machine with requests to render it unavailable for legitimate requests.
DDoS attack	Like a DoS attack but by employing multiple machines.
Web attack	In this dataset SQL injection (to gain unauthorized access to a database), Cross-Site Scripting (XSS, injection of malicious scripts into a website), and brute force over HTTP (used to find the administrator's password) are considered.
Infiltration attack	Exploit a software vulnerability to create a backdoor into the victim network.

Table 2: Attacks in CICIDS dataset.

of attack records. This is also noted by Panigrahi et al. [28], who describe further limitations of the dataset, such as its high count of *missing values* and the fact that the data, which is high in volume when combined, comes in *different files* and thus needs preprocessing before it can be used as one dataset. They suggest relabeling the data to overcome the limitation of class imbalance, for example by splitting the Benign class or merging the attack classes. This last option could however hinder the detection of different attack types and should be used with care. If too many classes of very small sizes are present in the dataset, the model might not be able to learn how to detect them properly and merging classes can be beneficial. However, once classes are merged, the distinction between attack types disappears.

The dataset contains 85 attributes and 1385330 records, the most interesting attributes have been investigated and visualized where possible. The attributes include IP addresses, port numbers, and statistics about packets sent such as the average packet length of packets sent over the connection between the IP addresses in the record. Figs. 6 and 7 show the most found Source and Destination IPs, in total there are 14946 unique Source IPs and 18764 unique Destination IPs. 14940 IP addresses are used as both Source and Destination IPs. This is more than the IP addresses that are listed by [35] but their existence is not elaborated upon. After looking into some of the IP addresses, we assume that they are introduced into the dataset as traffic is generated. Removing records that contain unexplained IP addresses reduces the dataset to a dataset that only contains two attack types, therefore that option for preprocessing is not desirable.

There are 63613 unique Source Ports to be found in the dataset, and 49378 unique Destination Ports. The most popular ones are shown in Figs. 8 and 9. It can be seen that the most popular ports are for HTTP(S), FTP and SSH services.

The dataset contains 25073 unique timestamps, meaning that multiple events are recorded at the

Attack name	Nr. of occurrences
Benign	950588
DoS Hulk	163854
Portscan	158740
DDoS	81478
DoS GoldenEye	7709
FTP-Patator	6439
SSH-Patator	5883
DoS Slowloris	4154
DoS SlowHTTPTest	2327
Bot	1956
Web attack - brute force	1507
Web attack - XSS	635
Infiltration	32
Web attack - SQL injection	21
Heartbleed	7

Table 3: Attack types in CICIDS dataset.

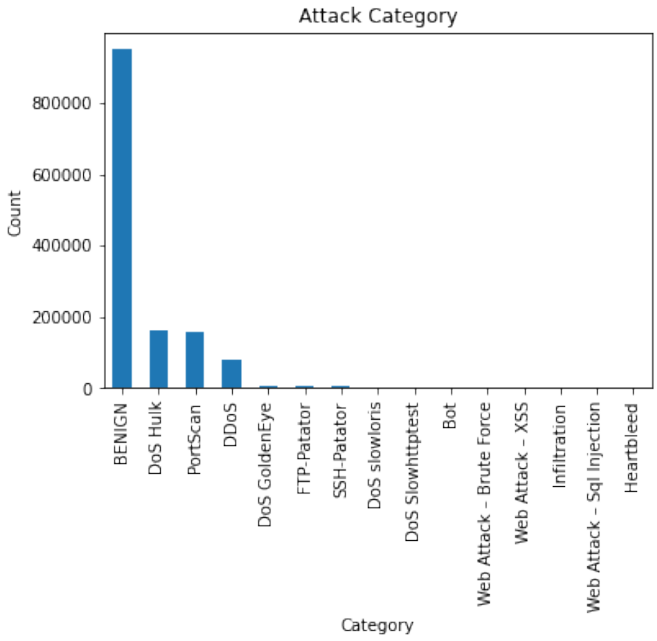

Figure 5: Distribution of labels.

same timestamps. Fig. 10 shows how the events are distributed over the week and Figs. 11, 12, 13, 14 and 15 show the distribution of events for each day. The red dots indicate times at which attack events occurred (on Monday there were no attack events). While it seems that on some days (for example Thursday and Friday) high spikes in activity could be related to malicious activity, this potential correlation cannot be found on other days and it can therefore not automatically be

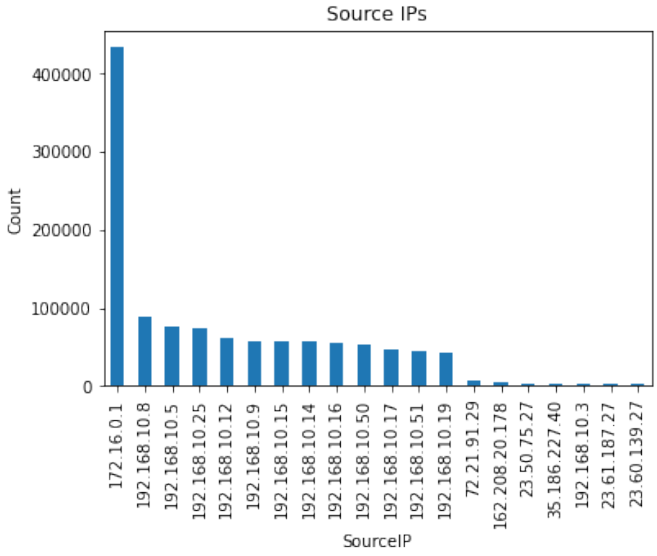

Figure 6: 20 most found Source IPs.

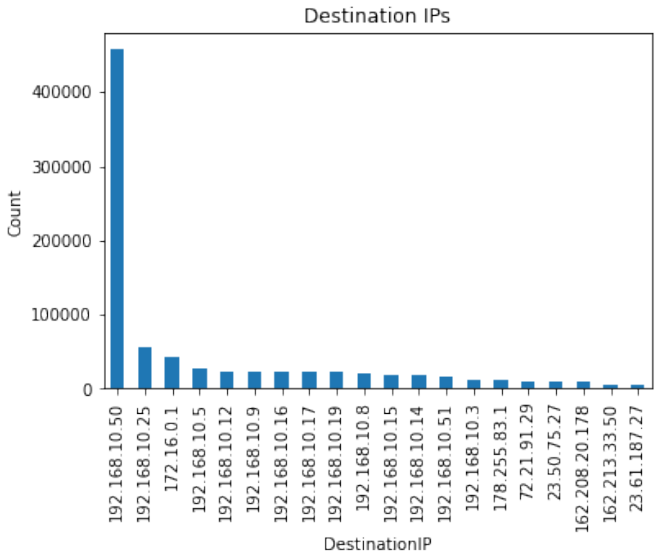

Figure 7: 20 most found Destination IPs.

concluded that an increase in activity is an indicator for attack behavior.

3.1 In the literature

The CICIDS dataset has been used in a plethora of different ways, below several examples of its usage are explored.

Usage for model-based attack detection. Pivarníková et al. [30] use the dataset for attack detection and projection. Firstly, they employ *alert aggregation* to merge alerts based on criteria

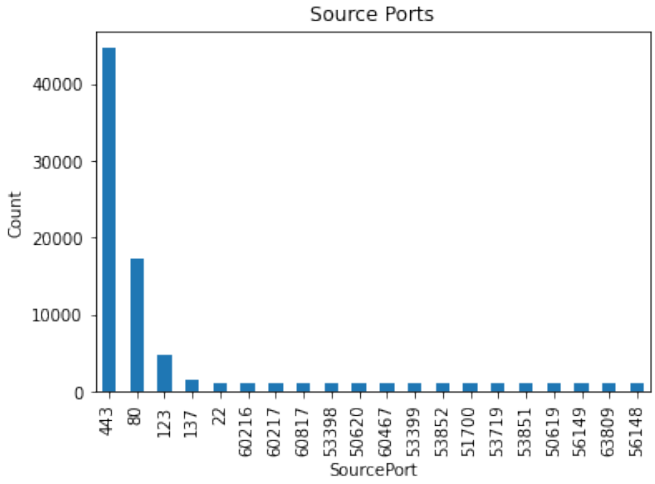

Figure 8: 20 most found Source Ports.

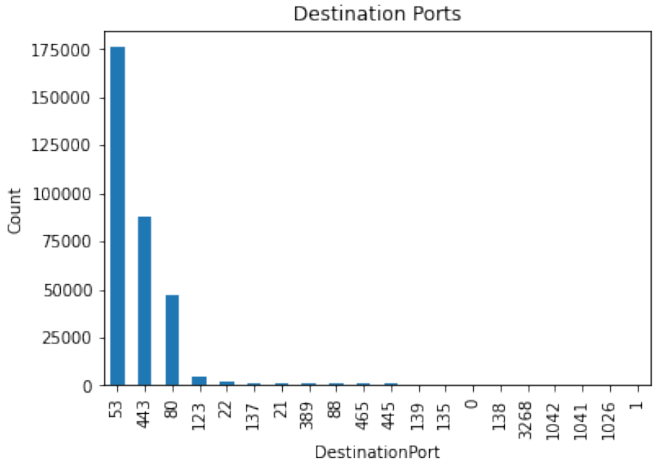

Figure 9: 20 most found Destination Ports.

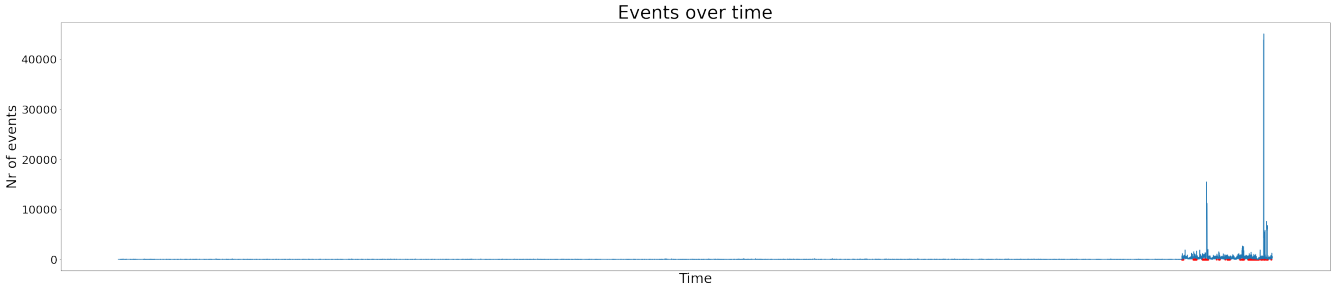

Figure 10: Events over time.

such as time window in order to prevent alert flooding. Alerts are generated by Snort, an IDS that generates alerts based on incoming data and a series of rules[5], by processing the CICIDS dataset.

[5]See https://www.snort.org/.

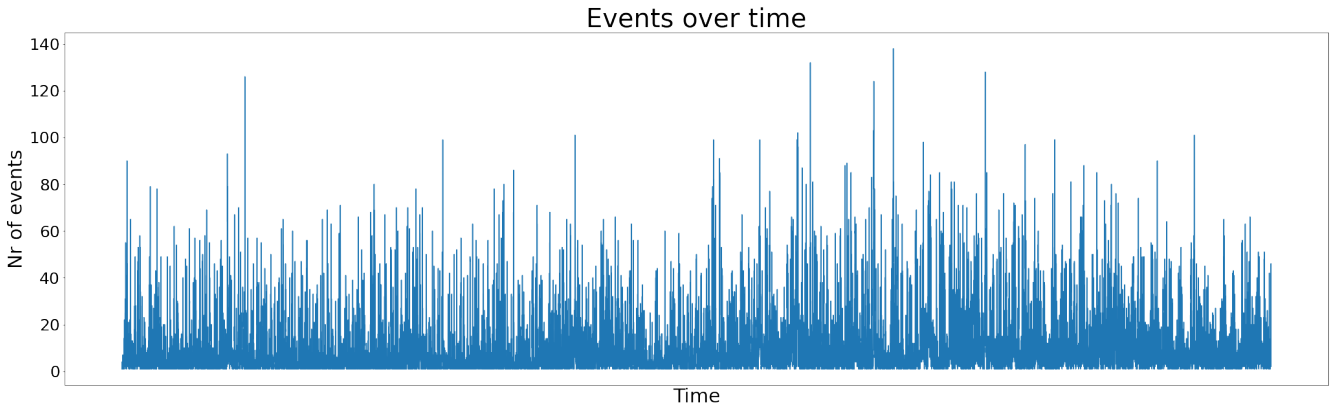

Figure 11: Events on Monday.

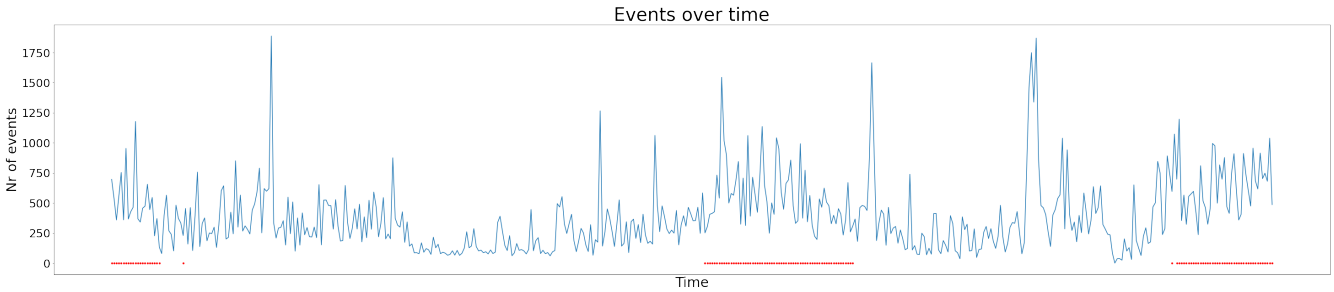

Figure 12: Events on Tuesday.

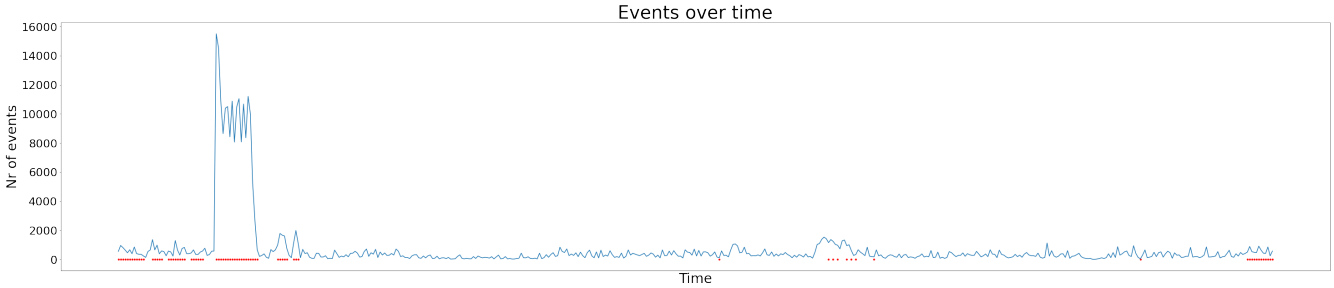

Figure 13: Events on Wednesday.

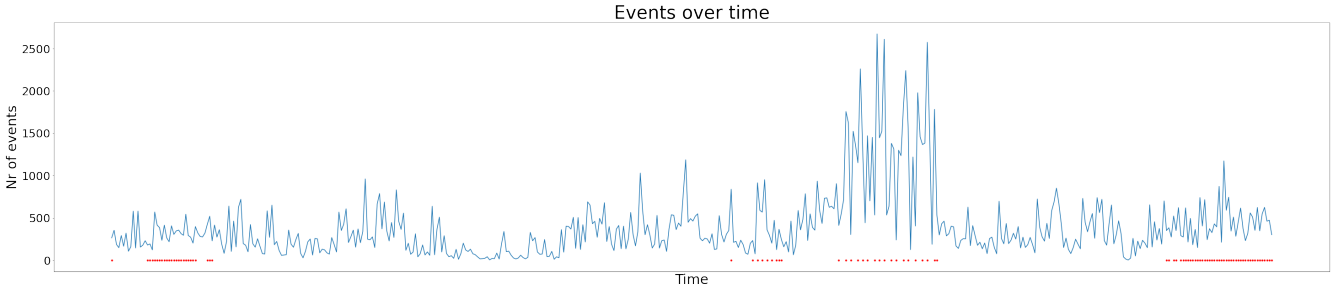

Figure 14: Events on Thursday.

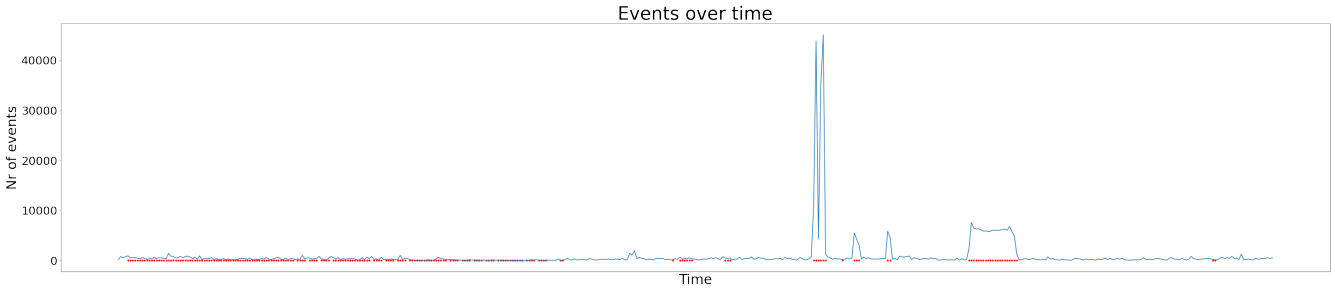

Figure 15: Events on Friday.

Next, *causal relationships* are defined based on the timestamps. With these causal relationships and conditional probabilities, a Bayesian network can be constructed. This network can be used to calculate the probability that a certain alert will occur based on all alerts that have already been observed. The authors only focus on alerts that would be generated in the intrusion attempt and perform malicious tasks stages of an attack. They only look at the web attacks that are found in the Thursday part of the dataset. As the results in this work focus on probabilities, they are not suitable for direct comparison to our work, which produces accuracy, precision, recall and F1-score metrics.

Usage for attack detection with neural networks. Maseer et al. [25] evaluated the performance of *both supervised and unsupervised machine learning algorithms* on the CICIDS2017 dataset and found that overall supervised methods showed better performance when looking at accuracy, precision, recall and F1 score. The best performing algorithm was the random forest algorithm. Other algorithms under evaluation were artificial neural networks, decision trees, naive Bayes, K-nearest neighbors, support vector machine and the convolutional neural network as supervised methods and K-means, expectation-maximization and the self organizing map as unsupervised methods.

Malaija et al. [24] implemented three different types of *neural networks*: a fully connected network, a variational autoencoder, and a sequence-to-sequence model. In a variational autoencoder the input is first reduced to a simplified mapping before being reconstructed again while a sequence-to-sequence model is composed of recurrent neural networks which can use past inputs as well as current inputs. Because each day of the dataset contains a different type of attack, they choose random partitions of the entire dataset for training and testing. They compared the performance of these networks to each other and to the performance of a random forest and a support vector machine and also implemented five-fold crossvalidation, meaning that the tests were repeated five times and averaged for the final results. They found that the variational autoencoder performed very poorly. The sequence-to-sequence model performed the best on all metrics (accuracy, precision, recall and F1 score). The random forest had higher average scores for accuracy and precision than the fully connected network, however, it showed more variation within the scores, implying that the performance of the fully connected network is more robust. The authors also note that the networks they used were quite shallow (consisting of one or three layers) and that the performance of deep learning techniques might be improved by experimenting with layers or using more sophisticated networks such as the convolutional neural network.

Vinayakumar et al. [40] designed *neural networks* for several datasets, including the CICIDS2017 dataset and compared their performance on an IDS task to the performance of classical clasisfiers, including the random forest and decision tree among others. They found that the neural networks outperformed these classifiers.

Feature reduction for better attack detection. We will now look at some works that have applied feature reduction methods to improve classification models for the CICIDS2017 dataset. A common limitation found in the literature is that only accuracy is reported as a performance metric. As the majority class is so big, this metric is not a good representation of the actual performance of a classifier as there is a high chance that the 'BENIGN' class choice is correct even if chosen by chance. Additionally, not many authors present which features were chosen exactly by their methods, limiting further comparisons to their work.

Yulianto et al. [43] use the *SMOTE* (Synthetic Minority Oversampling Technique) algorithm to alleviate the issue of data imbalance before applying AdaBoost, which is a classifier based on an ensemble of multiple classifiers that weights each classifier in the ensemble based on their performance to gain maximum performance. Imbalanced data can have a negative effect on the model under training as it might favor the majority class [42], which is why attempting to alleviate this problem could be of interest. Yulianto et al. [43] also use *ensemble feature selection (EFS)* and *PCA* and find that combining SMOTE with EFS boosts performance the most. EFS is an R package that uses an ensemble of feature selection methods to determine feature importance. Using SMOTE in combination with PCA lowers the performance scores in comparison to only using PCA, suggesting that attempting to eliminate the data imbalance problem is not always advantageous.

Faker et al. [15] use *K-means clustering* as a feature selection technique before applying a deep learning model. They find that the performance of their neural network improves when feature selection is applied. Faker et al. [15] are include a detailed description of data preprocessing methods in their work, which is not found in all works that employ the CICIDS2017 dataset. For example, they choose to remove IP addresses and port numbers and normalize the data. Additionally, all records labelled as normal traffic are removed as the authors are only interested in attack traffic. They find that a (relatively simple) deep learning model scores an accuracy of 0.9956 on both the full dataset and a dataset consisting of 67 features (which was the optimal number of features found during the reduction process). They also implemented a random forest classifier, which scores 0.9254 when using all features and an accuracy of 0.9271 when using an optimal subset of 6 features.

Abdulhammed et al. [4] used an *autoencoder* to reduce the number of features before feeding the data to several classifiers: random forest, Bayesian network, Linear Discriminant Analysis (LDA) and Quadratic Disriminant Analysis (QDA). In an autoencoder, the input is first compressed to a smaller representation that is then mapped to the output layer. In addition to an autoencoder, they also tested principal component analysis for feature reduction. They map IP addresses to integer numbers and normalize the data as preprocessing steps. They list `Subflow Fwd Bytes, Flow Duration, Flow IAT, PSH Flag Count, SYN Flag Count, Average Packet Size, Total Len Fwd Packets, Active Mean and Min, ACK Flag Count`, and `Init_Win_bytes_fwd` as important features. In this work, Uniform Distribution Based Balancing (UDBB) is applied to alleviate the problem of data imbalance. This technique samples from a distribution learned from each feature-label pair.

Stiawan et al. [37] use *information gain* to select features for classification algorithms but find that it is still vital to use human intervention to decide what threshold is used for choosing the features to feed to the classifiers. The researchers only use 20% of the original dataset meaning that not all attack types are present. Information gain is a metric that can be calculated for each feature with respect to the class labels. Feature groups are created as well, which are used as input for random forest, Bayesian network, decision tree, random tree and naive Bayes. They find that random forest is able to outperform the other methods with relatively small feature sets while the decision tree has superior performance for bigger feature sets.

Pelletier et al. [29] have applied the *Boruta algorithm* from a package from the R programming language with the same name, to determine feature importance. In this work, a section on data preprocessing is present and attention is drawn to the duplicate presence of the Fwd Header Length column. Additionally, they replaced `NaN` values with double the maximum value found in the respective column of the `NaN` value (to represent infinity). Boruta is an algorithm that uses random forests on 'shadow features', which are duplicates of the features of the dataset but the data in each column are randomized. If the original version of a feature performs better than its shadow version, this feature is marked as important. The algorithm can look at all features or run for a set number of iterations if saving computation time is desired[6]. Pelletier et al. [29] completed 20 iterations. As the dataset needs to be duplicated, this algorithm can be considered to be expensive for the storage space. This work presents classification accuracy of a neural network created with R packages for each individual class rather than overall accuracy, which complicates comparison of the results as finding an overall accuracy requires knowing the number of samples in each class[7] and that information is not disclosed.

Powell et al. [31] evaluated the effect of several feature reduction options provided by SciKit Learn on several classifier algorithms, namely logistic regression, K-nearest neighbors, decision tree and random forest. Overall they found that the number of features could be reduced from 80 to between 26 and 10 features, depending on which feature reduction method was chosen. The methods under investigation were *VarianceThreshold*, which removes low variance features (features that have the same value in all samples have no variance), *SelectKBest* and *SelectPercentile*, which both determine statistical scores for all features and retains only a specified percentage of or the k highest scoring ones, and the *Extra-Tree Classifier*, which fits randomized decision trees to subsets of the dataset, from which feature importance can be determined. While a loss of accuracy was recorded, the computation time decreased drastically. Especially for systems that are supposed to perform in real time, a decrease in computation time is interesting. They found that the decision tree algorithm, when combined with VarianceThreshold, was able to produce an accuracy of 1 while reducing the featureset from 79 to 23.

Krishna et al. [20] introduce *LVFE*, Least Variance Feature Elimination, and evaluate its performance with the Fast K-Nearest Neighbors algorithm for classification. The method is based on the variance of each feature and iteratively reducing the feature subset by removing the feature that has the least amount of variance. It was found that the feature set could be reduced to 11 features while maintaining an accuracy of 0.9994. No other performance metrics are presented. The features found were `ACK`

[6]For more information on the Boruta algorithm, see https://www.datacamp.com/community/tutorials/feature-selection-R-boruta.

[7]See https://www.indeed.com/career-advice/career-development/how-to-calculate-average-percentage.

`Flag Count`, `PSH Flag Count`, `URG Flag Count`, `Destination Port`, `Flow Duration`, `Fwd IAT Total`, `Bwd IAT Total`, `Init_Win_bytes_forward` `min_seg_size_forward`, `Bwd Packet Length Std` and `Down/Up Ratio`.

Reis et al. [33] use decision trees and random forest to evaluate the performance of *gini importance* (used in decision trees to decide how to split the data), *permutation importance* (which measures the importance of a feature after the model is trained), and *drop-column importance* (which looks at the performance of the model when one column is excluded). They include a section about preprocessing and for example mention how some columns were dropped as they only contained constant values. They found that reducing the dataset form 69 to 10 features did not lead to a significant loss in performance and that the most important features found by the three methods included `Destination Port`, `Fwd IAT Min`, `Init_Win_bytes_forward`, `Init_Win_bytes_backward` and `Flow IAT Min`. They compare the performance of a random forest with 69 features to the performance of a random forest with 10 features. Although not mentioned specifically, it is assumed that for the feature reduction permutation importance is used as this method finds a balance between reasonable computation time (the best method, drop-column importance, is quite expensive time-wise) and accuracy (gini importance has low accuracy when used for a random forest).

Singh Panwar et al. [36] implement *correlation-based feature selection* (CFS) and the classifier subset evaluator (which considers the performance of a subset of features) with Naive Bayes, J48 and decision tree as available in Weka. The feature sets that are found by these methods are then evaluated with a one-tier decision tree and a reduced error pruned tree (REPtree). They evaluated the performance for each attack seperately and found that processing time and dataset size were reduced significantly without affecting the performance of the classifiers too much. As the authors consider the performance on each attack class individually, it is hard to compare performance. This problem is also found when looking at the work presented by Pelletier et al. [29].

In Tab. 4 the known feature reduction methods for the CICIDS dataset have been collected. However, each method has been evaluated for different models, making it hard to judge if there is one method that works better than the others. Additionally, not all sources specify which features were chosen. In this work we would like to investigate the performance of several feature reduction methods and the effect of the chosen features on the performance of a relatively simple deep learning model. Additionally, we want to see if there are features that are found to be important by multiple feature reduction methods; these features might be interesting to keep in mind for the creation of new cybersecurity datasets.

We will compare the performance of our models to the performance of other deep learning models, namely the ones created by Maseer et al. [25], Malaija et al. [24] and Vinayakumar et al. [40]. We will also consider the performance of classification models which have been combined with feature reduction, namely the ones outlined in [43, 4, 15, 31, 20] and [33]. Similar to [43, 15] and [31] we want to look into the effects of PCA, K-means clustering and correlation as feature reduction methods. Additionally, we want to investigate the effects of the use of random forest and the sequential feature selection algorithm as feature selection methods. As opposed to these works, which all use different data preprocessing techniques and models for their evaluation, we want to work with the same dataset and same deep learning model to evaluate each feature reduction technique to provide a fair comparison of the techniques.

Source	Feature reduction method
[43], [4]	PCA
[43]	EFS
[15]	K-means clustering
[37]	Information gain
[29]	R package Boruta
[31]	(Unspecified) statistical scores
[31]	Variance based feature removal
[31]	Decision tree
[20]	LVFE
[33]	Gini importance
[33]	Permutation importance
[33]	Drop-column importance
[36]	Weka correlation-based feature selection
[36]	Weka classifier subset evaluator

Table 4: Feature reduction methods applied to models for CICIDS.

4 Experiments

4.1 Performance measures

For the experiments, several metrics will be calculated. Firstly, the *true positives* (correctly identified attacks), *true negatives* (correctly identified benign traffic), *false positives* (benign traffic labeled as attack) and *false negatives* (attacks labeled as benign traffic) will be taken into account. With these quantities the following metrics can be calculated:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{6}$$

These metrics are first calculated for each attack separately and then aggregated into one score covering all attacks. For all these metrics, both the *weighted* and *macro* scores were calculated[8], where the weighted score takes class imbalance into account and the macro score does not. We will also look at the *identification score* [21], which is a metric based on the weighted ($F1_w$) and macro ($F1_m$) F1 scores and thus takes class imbalance into account by design:

[8] In text references to scores are macro scores unless stated otherwise.

$$IS = \frac{2 \cdot F1_w \cdot F1_m}{F1_w + F1_m} \tag{7}$$

Attack name	Nr. of occurrences	Label
Benign	950588	0
DoS	178044	1
Portscan	158740	2
DDoS	81478	3
FTP-Patator	6439	4
SSH-Patator	5883	5
Web attack	2163	6
Bot	1956	7
Infiltration	32	8
Heartbleed	7	9

Table 5: Attack types in CICIDS dataset after preprocessing.

4.2 Preprocessing

The dataset consists of separate files for each day, some days are even split into morning and afternoon parts. These files first need to be concatenated into one dataset. In order to make the data suitable for a deep learning model all rows with `NaN` values need to be removed as a deep learning model cannot train properly when it is fed such values. Next, to combat data imbalance, the three web attack types (brute force, XSS and SQL injection) are merged into one label. The DoS attack types (named after different tools for DoS attacks), DoS Hulk, DoS GoldenEye, DoS slowloris, and DoS Slowhttptest are also merged. This means that the only labels, which are categorized with {0-9} labels, are respectively benign, DoS, PortScan, DDoS, FTP-Patator, SSH-Patator [9], Web Attack, Bot, Infiltration and Heartbleed. The new division of labels is found in Tab. 5. The port columns are transformed to integers rather than objects and the IP addresses are mapped to integers as well, as the model will not be able to handle non-numerical input. In order to combat data skewedness, the columns are log-normalized. The data are split according to a 75/25 train-test split[10] using stratification to alleviate class imbalance between the datasets. The columns FlowID and Timestamp are removed from the dataset as the ID is not interesting for predictive purposes and we are not interested in time-based detection of attacks. After removing all records with one or more `NaN` values from the dataset, it becomes apparent that some columns now only contain constant values. Therefore, the columns `Protocol, Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk` and `Bwd Avg Bulk Rate` are also removed. The column `Fwd Header Length` was present in duplicate so `Fwd Header Length.1` was removed from the dataset as well. After preprocessing the dataset contains 1385325 records and 72 features. The training part of the dataset consisted of 1038993 records, 326056 (31%) of which were attack records. The testing part of the dataset contained 346332 records with 108686 (31%) labeled as attacks.

[9] Patator is a Kali Linux penetration testing tool.

[10] Cross validation is not applied.

4.3 Model

The deep learning model that is used for the experiments is relatively simple and based on an already existing model [40]. The model contains five layers, with 1024, 768, 512, 256 and 128 nodes per layer respectively and each layer uses the Relu activation function, except for the output layer, which has 10 nodes and uses the softmax activation function. The weights are initialized with random numbers and between each layer batch-normalization and dropout (with a 0.01 rate) are applied. The loss function that is used is sparse categorical cross-entropy, which is suitable for multi-class classification. The SGD optimizer is used with a learning rate of 0.001. The model is trained for 200 epochs with a batch size of 256 and implemented using the Tensorflow[11] and Keras[12] libraries in Python.

First we evaluate the performance of this baseline model, where all columns are included. The model reaches an accuracy of 0.9877, a precision of 0.7554, recall of 0.6278 and an F1-score of 0.6459. The original paper that implemented this model [40] reports an accuracy of 0.956, a precision of 0.962, a recall of 0.956 and an F1-score of 0.957. Details about the preprocessing of the data are not specified and several details regarding the model, such as the optimizer, are not mentioned, therefore it is not possible to create a true replicate of the experiments and these details could explain the differences in performance. Another important detail that is not clarified in [40] is whether or not the reported scores take class imbalance into account. The weighted precision, recall and F1-score for our model are 0.9875, 0.9877 and 0.9869 respectively and seem to indicate a much better performance.

As the baseline does not perform very well when looking at the macro scores, several changes were applied to the model to see if the performance would improve. For the first improvement the data were *lognormalized* (as the distribution of values was skewed, regular normalization was not deemed fit to equalize the distribution of the data). In the lognormalization all values are first incremented by 1 (to get rid of any zeros), then divided by 2 and then the log is calculated for each value in the dataset. Fig. 16 shows the effect of lognormalization on the data in the Fwd Packet Length Mean column. It can be seen that the values spread out more over the scale. Another option that was tried was to *cut out part of the benign records*, reducing the size of the majority class from 950588 to 515841, which is the size of all attack classes combined. Originally, the learning rate for the optimizer was set at 0.001, but a new experiment was run with the *learning rate set at 0.01*. For the fourth option, an *extra layer* with 56 nodes was added after the layer of 128 nodes and finally, an experiment was run with *RMSProp* as the optimizer instead of SGD. Tab. 6 shows the performance of the baseline model after implementing the different options for improvement. It also shows which classes each version of the model is able to predict. The lognormalized data and larger learning rate seem to improve the performance the most, therefore another experiment was run where both improvements were implemented, which increased the performance even more (see the last column of Tab. 6 for the results). Therefore it was decided to run all experiments with the lognormalized data and adjust the models learning rate from 0.001 to 0.01. The final model used for further experiments is described in Tab. 7. In this table it is also indicated what elements of the model could be reused from the model created Vinayakumar et al. [40] and what elements were self-derived, either because they were not specified by Vinayakumar et al. [40] or because these

[11] https://www.tensorflow.org/
[12] https://keras.io/

	Regular	Log data	Smaller dataset	LR=0.01	Extra layer	RMSProp	Log+LR=0.01
Accuracy	0.9877	0.9995	0.8975	0.9720	0.9728	0.9152	**0.9995**
Precision (macro)	0.7554	**0.9958**	0.6767	0.7537	0.7476	0.3872	0.9956
Recall (macro)	0.6278	0.9348	0.7330	0.6451	0.6147	0.3370	**0.9477**
F1 score (macro)	0.6459	0.9593	0.6822	0.6635	0.6381	0.3517	**0.9682**
Precision (weighted)	0.9875	**0.9995**	0.9475	0.9747	0.9724	0.9122	**0.9995**
Recall (weighted)	0.9877	**0.9995**	0.8975	0.9720	0.9728	0.9152	**0.9995**
F1 score (weighted)	0.9869	**0.9995**	0.8974	0.9720	0.9714	0.9001	**0.9995**
Identification score	0.7808	0.9789	0.7750	0.7886	0.7702	0.5058	**0.9836**
Predicted classes	0-7	**0-9**	0-7	0-7, 9	0-7	0-3	**0-9**

Table 6: Performance of the baseline model with modifications.

elements improved the performance of the baseline model. Fig. 17 shows the confusion matrix for the predictions of the original model. While we found that predictions are made for each class, it can be seen that the attacks for classes 8 and 9 are not recognized. This is understandable as there are not many instances of these attacks in the dataset. Mainly the majority classes get the highest proportion of correct predictions, but as there is a lot of data to train on for these classes this could be expected. However, the improved baseline model is able to predict cases for the smallest classes as can be seen in the confusion matrix shown in Fig. 18.

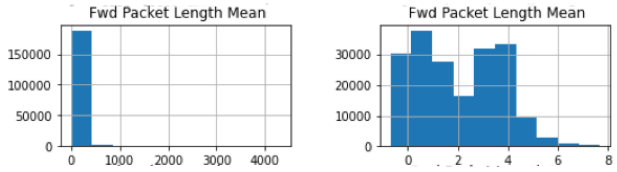

Figure 16: Data before lognormalization (left) and after lognormalization (right).

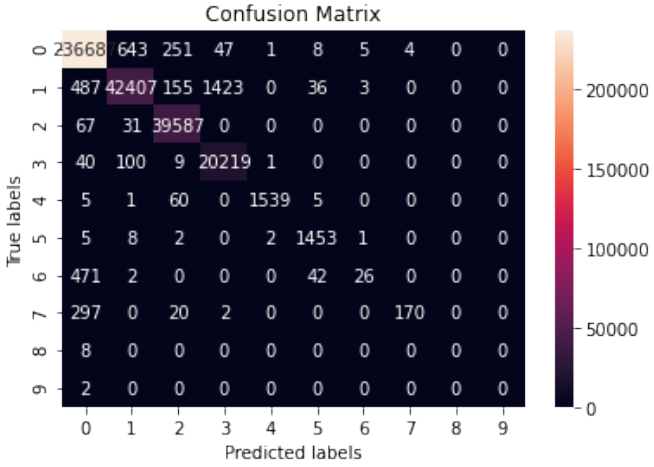

Figure 17: Confusion matrix for the baseline model.

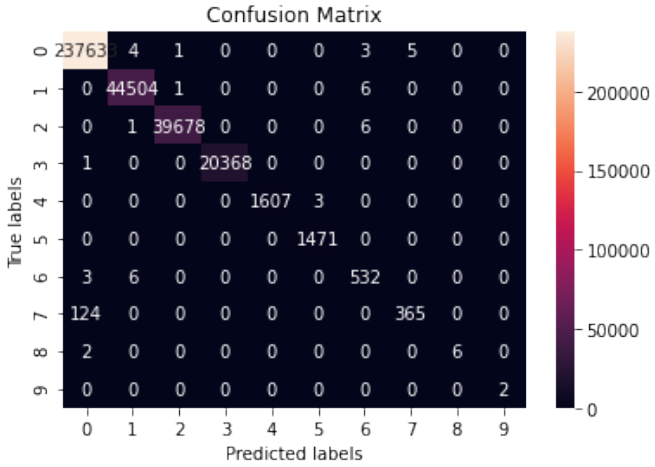

Figure 18: Confusion matrix for the baseline model after lognormalization and increasing the learning rate.

Data	Lognormalized	Self-derived
Dropout	0.01	From [40]
Batch normalization	Yes	From [40]
Optimizer	SGD	Self-derived
Learning rate	0.01	Self-derived
Training epochs	200	Self-derived
Batch size	256	Self-derived
Nodes layer 1	1024	From [40]
Nodes layer 2	768	From [40]
Nodes layer 3	512	From [40]
Nodes layer 4	256	From [40]
Nodes layer 5	128	From [40]
Nodes output layer	10	From [40]
Activation hidden layers	Relu	From [40]
Activation output layer	Softmax	From [40]
Loss function	Sparse categorical crossentropy	Self-derived

Table 7: Model specifications final model.

4.4 Applying PCA

We use SciKit Learn[13] functions to apply PCA to the dataset. First, we standardize the data, this is necessary because PCA depends on variance, which in turn depends on units of measurement, and the covariance matrix changes if a variable undergoes a change in the unit of measurement [18]. The PCA function reduces the dataset to 17 features, which is only a third of the size of the original feature set, while 95% of the variance is retained.

The model was run on the reduced dataset and this resulted in an accuracy of 0.9996, a precision of 0.9945, a recall of 0.9543 and a F1-score of 0.9719. All performance results of this model and all other models that are evaluated in this work are found in Tab. 14. This model performs better than

[13] https://scikit-learn.org/stable/

the baseline model, however, as the baseline model already performs well, the *improvements are marginal*. The small improvement implies that some noise could have been eliminated. Additionally, it is remarkable that the dataset can be transformed in such a way that only *17 features* convey enough information from the original 72 features to achieve good performance. This demonstrates how important the data representation is and that the original representation of the dataset is clearly not efficient. This version of the model is also able to outperform the model created by Vinayakumar et al. [40]. Like the baseline model, this improved version of the model is able to predict all attack types. Fig. 19 shows the confusion matrix for the predictions and the true labels of this model. It can be seen that there are more correct predictions than for the original model, which is in line with the higher performance scores that were found.

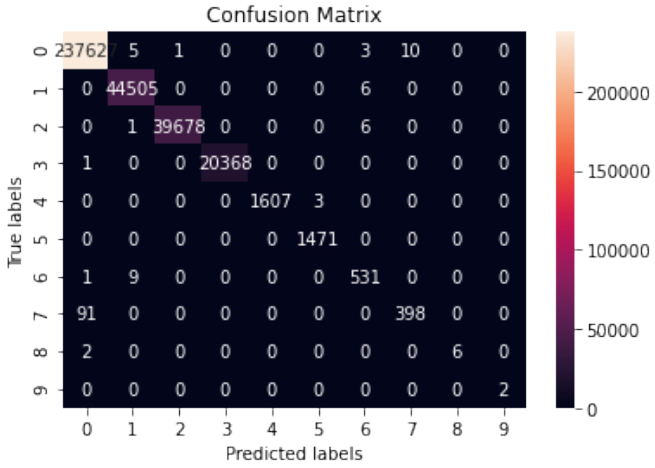

Figure 19: Confusion matrix for the model after PCA has been applied to the dataset.

4.5 Applying random forest

The RandomForestRegressor from Sci-Kit Learn is used to create 1000 decision trees and extract the most important features. Only 9 out of the 72 features were found to be interesting for the prediction, namely `Source IP`, `Destination IP`, `Initial Forward Window Bytes (Init_Win_bytes_forward)`, `Source Port`, `Destination Port`, `Backward Packet Length Mean`, `Backward Packet Length Max`, `Average Backward Segment Size` and `Minimum Forward Segment Size (min_seg_size_forward)`. Sharafaldin et al. [35] investigated which features are important for each attack separately and our list is missing some of those. For the Thursday subset we tried to select features as well, to see if these would match with the ones found in [35]. The features found were `Source IP`, `Destination IP`, `Forward Interarrival Time Minimum (Fwd IAT Min)`, `Initial Forward Window Bytes (Init_Win_bytes_forward)`, `Destination Port`, `Backward Packet Length Mean`, `Average Backward Segment Size` and `Minimum Forward Segment Size (min_seg_size_forward)` and these features are not the same as the ones found in [35], which states that for the web attack and infiltration attack `Initial Forward Window Bytes`, `Forward Subflow Bytes`, `Initial Backward Window Bytes`, `Total Length Forward Packets`, `Flow Duration` and `Active Mean` are important features. This could be due to the fact that we only take part of the dataset into account or the authors of [35]

might have used a different number of trees in their random forest, which is sadly not specified. Additionally, they worked with 80 (unspecified) features in their random forest, while we take 72 features into account.

Next, we train the deep learning model with the features that were selected with the random forest. We find an accuracy of 0.9815, and a precision of 0.9538, but the recall of 0.8689 and the F1-score of 0.8859 show a slight degradation in performance. This model does therefore underperform compared to the baseline model. This could mean that 9 features cannot provide enough information for the deep learning model to train properly, however, the model is able to predict attacks for all classes. The confusion matrix shown in Fig. 20 shows that the performance of the model with features from the random forest does not make better predictions than the baseline model, as also seen in the performance scores.

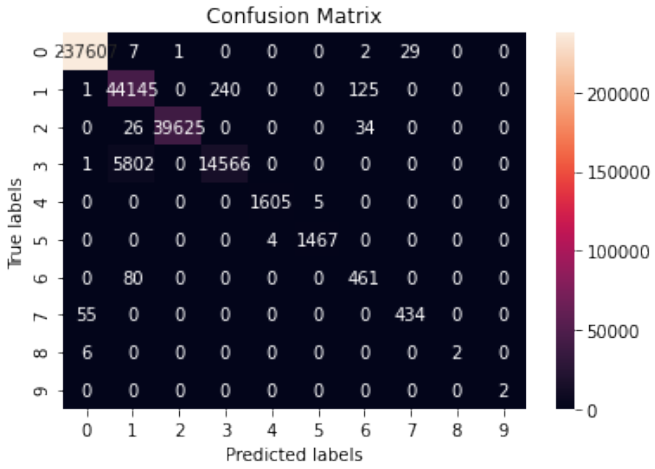

Figure 20: Confusion matrix for the model trained with features extracted by random forest.

4.6 Applying correlation

Correlation has been applied by Powell et al. [31] as a feature reduction method for classification algorithms such as a decision tree with the SelectKBest and SelectPercentile methods in SciKit Learn but they did not specify which statistics were used. Additionally, they do not mention which features were chosen by the algorithms. We will therefore apply these SelectKBest with the *f-score*. Experiments with different values for k were performed, namely $\{9, 17, 36, 54\}$. These values were chosen because 9 features were chosen by the random forest, therefore we would like to see which features are chosen by the correlation method and if the model will perform better or worse with the features chosen by this method. 36 features halves the feature set and 54 features constitutes 75% of the feature set. Tab. 8 shows the performance scores for models with different feature sets and Tab. 9 shows the chosen features for each experiment. It can be seen that when attempting to reduce the features through PCA (which yields 17 new features) the results are more promising than when choosing 17 features based on correlation. The same holds for the 9 features that are chosen by the random forest, they form a more informative set for the model than the 9 features chosen by correlation. The feature set consisting of 54 features provides the best performing model

Number of features:	9	17	36	54
Accuracy	0.9816	0.9736	0.9991	**0.9995**
Precision (macro)	0.9771	0.9428	**0.9776**	0.9771
Recall (macro)	0.8785	0.8664	0.9207	**0.9724**
F1 score (macro)	0.9111	0.8913	0.9433	**0.9719**
Precision (weighted)	0.9824	0.9773	0.9990	**0.9996**
Recall (weighted)	0.9816	0.9736	0.9991	**0.9995**
F1 score (weighted)	0.9807	0.9712	0.9990	**0.9995**
Identification score	0.9446	0.9296	0.9704	**0.9855**
Predicted classes	0-9	0-9	0-9	0-9

Table 8: Performance for {9, 17, 36, 54} features as chosen by correlation.

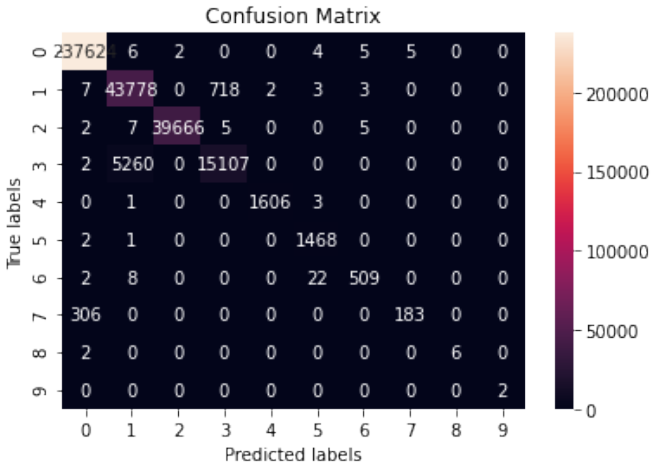

Figure 21: Confusion matrix for model with 9 features chosen by correlation.

compared to the other feature sets chosen by correlation and the baseline and the features chosen by random forest. Figs. 21, 22, 23 and 24 show the confusion matrices for the predictions and the true labels. From these figures it is not directly clear that the models with correlation features are not as good as the baseline model, but the performance scores have shown that only the model with 54 features seems to be able to outperform the baseline with regards to the recall and F1-score.

It is interesting to note that while each set of features is a subset of the bigger feature sets, the featureset of 17 features is not able to outperform the featureset of 9 features, even though it arguably contains more information. This could mean that the features added to this featureset do not convey enough information and add mostly noise. Combined with the new features of the bigger feature sets they might be more informative, which could explain why the bigger subsets perform better than the subset of 9 features. This could mean that the features are correlated amongst each other. This idea is heavily supported by the fact that PCA can reduce the dataset to one with minimal covariance among features that only contains 17 features and is still able to perform very well.

As both PCA and the correlation method with 54 features showed better performance compared to the baseline model, an experiment was conducted in which PCA was applied to the 54 features chosen by correlation. The PCA procedure reduced the dataset to 11 components that explain

Number of features	Features chosen
9	`Source IP, Destination IP, Bwd Packet Length Std, Flow IAT Std, Flow IAT Max, Packet Length Mean, Packet Length Variance, Average Packet Size, Init_Win_bytes_backward`
17	Featureset of size 9 and `Flow Duration, Bwd Packet Length Mean, Flow Packets/s, Flow IAT Mean, Fwd Packets/s, Bwd Packets/s, Packet Length Std, Avg Bwd Segment Size`
36	Featureset of size 9, 17 and `Destination Port, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Bwd IAT Mean, Bwd IAT Max, Max Packet Length, Subflow Fwd Bytes, Subflow Bwd Bytes, Idle Mean, Idle Max, Idle Min`
54	Featureset of size 9, 17, 36 and `Source Port, Total Fwd Packets, Total Backward Packets, Fwd Packet Length Mean, Flow Bytes/s, Bwd IAT Total, Bwd IAT Std, Bwd IAT Min, Fwd Header Length, Bwd Header Length, FIN Flag Count, PSH Flag Count, Avg Fwd Segment Size, Subflow Fwd Packets, Subflow Bwd Packets, Init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward`

Table 9: Features chosen by correlation.

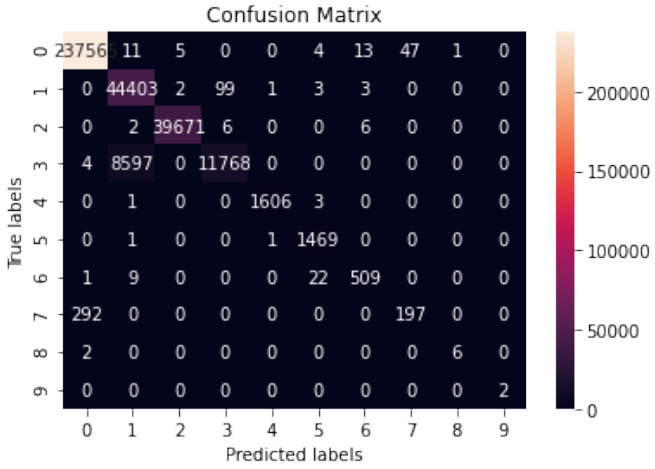

Figure 22: Confusion matrix for model with 17 features chosen by correlation.

95% of the variation in the data. The model is able to reach an accuracy of 0.9996, a precision of 0.9895, a recall of 0.9573, an F1-score of 0.9714 and an identification score of 0.9853. The weighted precision, recall and F1-score is 0.9996 for all metrics. This shows that the PCA procedure does not improve upon the model that uses 54 features while it does improve on the baseline that uses all 72 features. This could mean that the correlation method loses too much information to provide meaningful data for a more successful application of PCA.

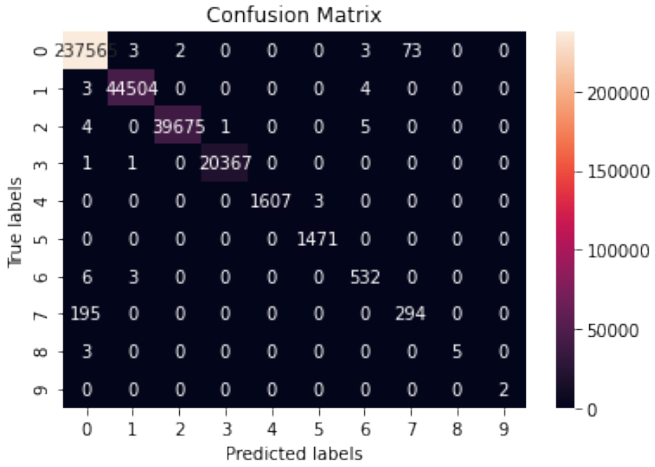

Figure 23: Confusion matrix for model with 36 features chosen by correlation.

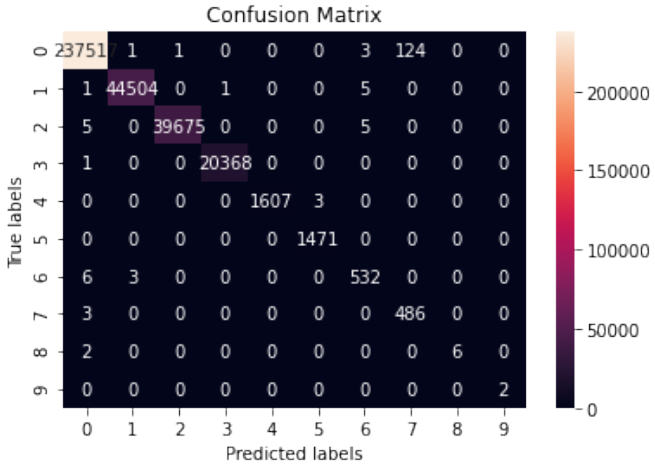

Figure 24: Confusion matrix for model with 54 features chosen by correlation.

4.7 Applying K-means clustering

The K-means method has been applied to the CICIDS2017 dataset by Faker et al. [15] who rank the clusters according to the *homogeneity score*, which can be calculated as labels are known. Features that rank higher are deemed more suitable for classification. SciKit Learn includes a K-means function that was used for this part of the experiments. The entire dataset was fed to the algorithm for each feature and each clustering result was scored with the homogeneity score, which can be defined as follows [15]:

$$h = \frac{H(C,K)}{H(C)} \tag{8}$$

where $H(C,K)$ represents the conditional entropy of the classes given cluster assignments, calculated as

$$H(C,K) = -\sum_{K=1}^{|K|}\sum_{C=1}^{|C|}\frac{n_{c,k}}{n}\log\frac{nc,k}{n_k} \tag{9}$$

and $H(C)$ is the entropy of the classes:

$$H(C) = -\sum_{C=1}^{|C|}\frac{n_c}{n}\log\frac{n_c}{n}. \tag{10}$$

n is the total number of datapoints, n_c represents the number of datapoints belonging to class c and n_k represents the number of datapoints belonging to cluster k. $n_{c,k}$ is then the number of datapoints in class c that is assigned to cluster k.

For each feature the K-means clustering procedure was run with the desired number of clusters set to 10 (the number of classes in the dataset) for 10000 iterations. After evaluating the results, several feature sets were set up to feed to the deep learning model. Sets of size 9, 17, 36, and 54 were chosen as seen in the correlation experiment. Faker et al. [15] find optimal performance with a featureset of size 67, therefore a featureset of this size was created as well.

Source Port is a feature that is included in the featureset of size 67. With this feature an initial run of the K-means algorithm was completed in order to test how it could work for the final experiment. It was found that when the clusters created by K-means based on this feature were used for prediction the results were very bad. The clusters were created based on the full dataset but the labels did not match well at all: the overall precision was 0.0128 and recall was 0.0180. This led to some scepticism regarding the performance of the features chosen by K-means in the deep learning model as this feature did not seem to have a very clear relationship with the target. However, as we were not sure that Source Port would be included in the final feature sets, the method's potential was not fully discarded yet. The results below show that the feature sets chosen by K-means do in fact perform well with the deep learning model and that Source Port is actually found to be of some importance, even though it is only included in the featureset when 67 features are considered.
Tab. 11 shows the results of the experiments with the deep learning model with the features chosen by K-means as input. As the size of the featureset increases, the performance seems to increase at first but shows a dip for the set of 54 features. Figs. 25, 26, 27, 28 and 29 show the confusion matrices for the results. It can be seen that the predictions are generally appropriately divided over all classes, except in the case of the featureset of 9 features which shows some bias towards the majority class.

4.8 Applying search algorithm

The search algorithm bases its choice for the best features on the recall score, meaning that the feature that scores the highest recall after training the deep learning model is added to the growing featureset. For a fair comparison to the other methods presented in this work, each iteration of the search algorithm required a training process of 200 epochs, making the algorithm very time consuming as there are 72 features to consider. This means that the results presented in this section are limited due to time constraints as the algorithm could only finish 3 iterations. The first feature selected by the search algorithm is `Packet Length Std`. A model was trained with

Number of features	Features chosen
9	Avg Bwd Segment Size, Subflow Bwd Bytes, Total Length of Bwd Packets, Fwd Packet Length Max, Bwd Packet Length Max, Average Packet Size, Packet Length Mean, Packet Length Std, Packet Length Variance
17	Featuresubset of size 9 and Destination Port, Fwd IAT Mean, Init_Win_bytes_backward, Source IP, Destination IP, Max Packet Length, Bwd Packet Length Std, Bwd Packet Length Mean
36	Featuresubsets of size 9, 17 and Total Fwd Packets, Subflow Fwd Packets, Subflow Bwd Packets, Total Backward Packets, min_seg_size_forward, Flow Bytes/s, Init_Win_bytes_forward, Avg Fwd Segment Size, Fwd Packet Length Mean, Fwd IAT Total, Fwd IAT Std, Flow IAT Max, Fwd Packet Length Std, Flow IAT Std, Flow IAT Mean, Subflow Fwd Bytes, Total Length of Fwd Packets, Fwd IAT Max, Bwd Header Length
54	Featuresubsets of size 9, 17, 36 and Fwd IAT Min, Active Min, Fwd Packets/s, Bwd IAT Total, Idle Min, Idle Max, Bwd IAT Min, Bwd Packets/s, Active Max, Idle Mean, Flow Packets/s, Active Mean, Bwd IAT Std, Flow Duration, Bwd IAT Mean, Bwd IAT Max, act_data_pkt_fwd, Fwd Header Length
67	Featuresubsets of size 9, 17, 36, 54 and Fwd PSH Flags, Active Std, ACK Flag Count, FIN Flag Count, Idle Std, URG Flag Count, Down/Up Ratio, PSH Flag Count, Fwd Packet Length Min, Min Packet Length, Flow IAT Min, Source Port, Bwd Packet Length Min

Table 10: Features chosen by K-means.

Number of features:	**9**	**17**	**36**	**54**	**67**
Accuracy	0.9727	0.9990	0.9994	0.9993	**0.9996**
Precision (macro)	0.7414	**0.9965**	0.9903	0.9819	0.9921
Recall (macro)	0.6573	0.9113	0.9444	0.9470	**0.9580**
F1 score (macro)	0.6699	0.9386	0.9640	0.9622	**0.9730**
Precision (weighted)	0.9673	0.9990	0.9994	0.9993	**0.9996**
Recall (weighted)	0.9727	0.9990	0.9994	0.9993	**0.9996**
F1 score (weighted)	0.9694	0.9987	0.9994	0.9993	**0.9996**
Identification score	0.7923	0.9677	0.9814	0.9804	**0.9861**
Predicted classes	0-9	0-9	0-9	0-9	0-9

Table 11: Performance for {9, 17, 36, 54, 67} features as chosen by K-means.

only this feature as input. This feature is also chosen in the correlation experiments but as those models did not manage to outperform the baseline unless the featureset was big enough, it was not expected that a model with only this feature as input would perform well. This model scores 0.4091 on precision, 0.4181 on recall, 0.4110 on the F1-score and 0.5664 on the identification score. The accuracy is 0.9158, which is quite high, but as the other scores are very low the high accuracy is most likely due to the large size of the majority class and it can be concluded that this metric is

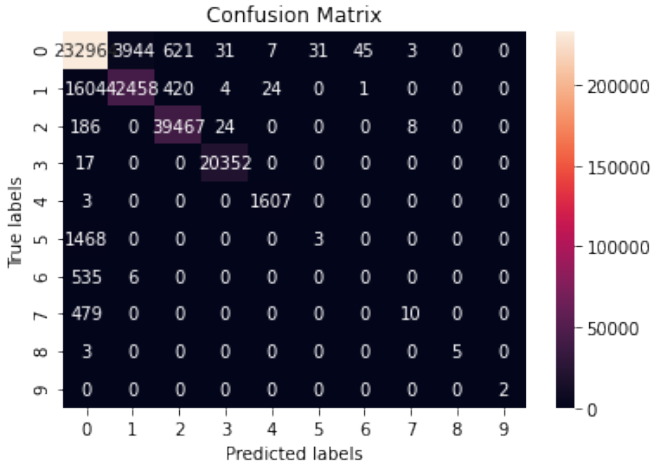

Figure 25: Confusion matrix for model with 9 features chosen by K-means.

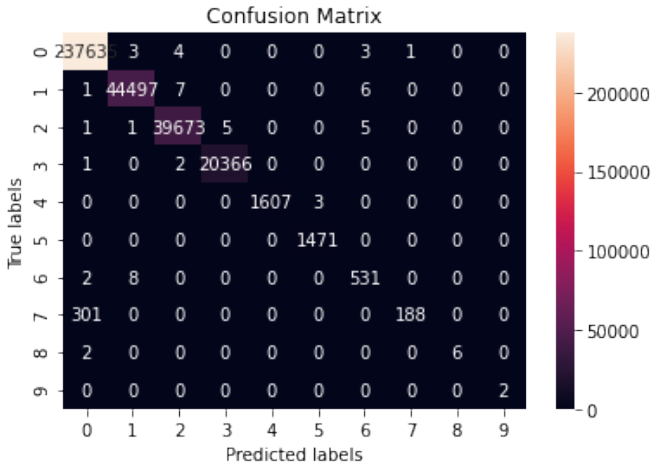

Figure 26: Confusion matrix for model with 17 features chosen by K-means.

not a proper representation of the performance of the model. The weighted scores for precision, recall and F1-score are 0.9076, 0.9158 and 0.9104 respectively. In the confusion matrix, shown in Fig. 30, we see that these metrics are skewed representations of the performance as well as there are many misclassifications and a clear preference for the majority class. We can thus conclude that a model with only one feature does not perform well and that the search algorithm should be run for a sufficient amount of time that will hopefully allow it to create a meaningful subset of features. Additionally, we see that the way performance is reported is important and might not be an honest representation of the true performance of a model.

The second feature that is chosen by the search algorithm `Total Fwd Packets`, which also appears in the correlation results. The accuracy of a model trained with both features found by search is 0.9446. The performance improves quite a bit, with a precision of 0.8439, a recall of 0.6242, and an F1-score of 0.6254. This is a performance that is on the same level as the baseline model's performance before it was enhanced with lognormalization and an increased learning rate. Fig. 31 shows the confusion matrix for the results. Again there seems to be a preference for the majority

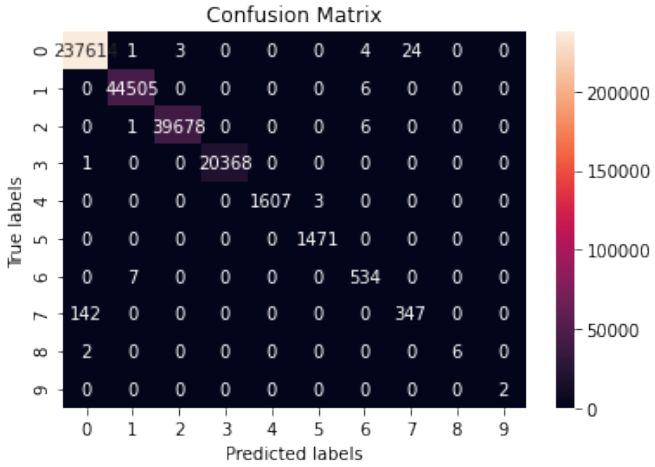

Figure 27: Confusion matrix for model with 36 features chosen by K-means.

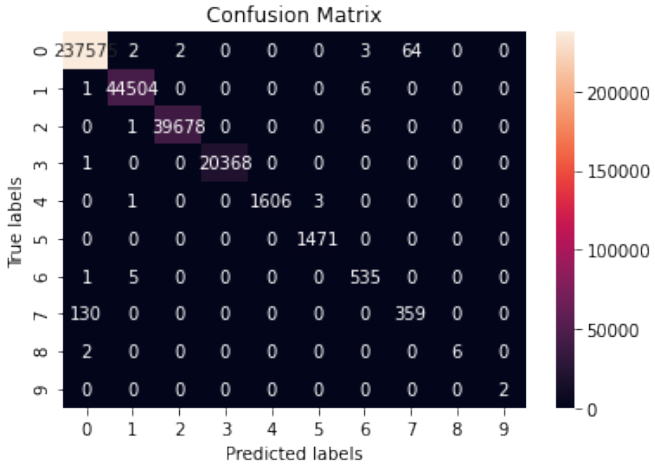

Figure 28: Confusion matrix for model with 54 features chosen by K-means.

class, however, a wider range of classes is included in the predictions.

The last feature that the search algorithm selected is `Source IP`. When this feature was added, the accuracy increased to 0.9709, precision to 0.9132, recall to 0.7543 and the F1-score increased to 0.7861. Overall, the increases in performance that are caused by the addition of new features are unexpected. Fig. 32 shows the confusion matrix of the results, which still shows a rather strong preference for the majority class compared to the other methods discussed in this work. Unfortunately, it is not possible to investigate the effects of additional features as it would be interesting to see if the search algorithm can produce a feature set that outperforms the feature sets chosen by correlation or random forest but is of smaller size.

4.9 Experimentation with the full labelset

As the results from the experiments showed that feature reduction can improve the performance of the baseline model, the experimentation was extended to a dataset in which *no labels were merged*.

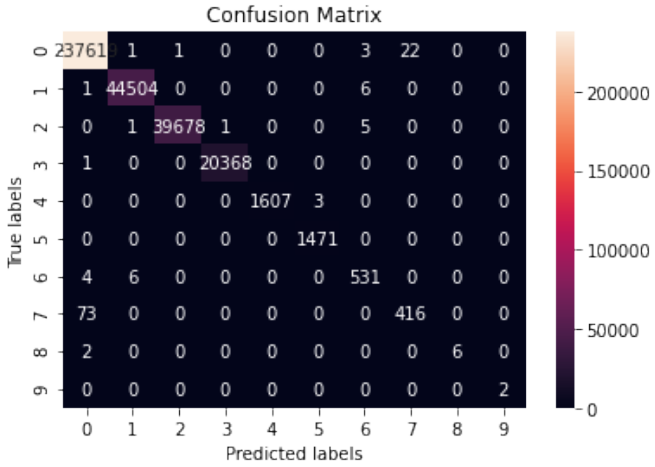

Figure 29: Confusion matrix for model with 67 features chosen by K-means.

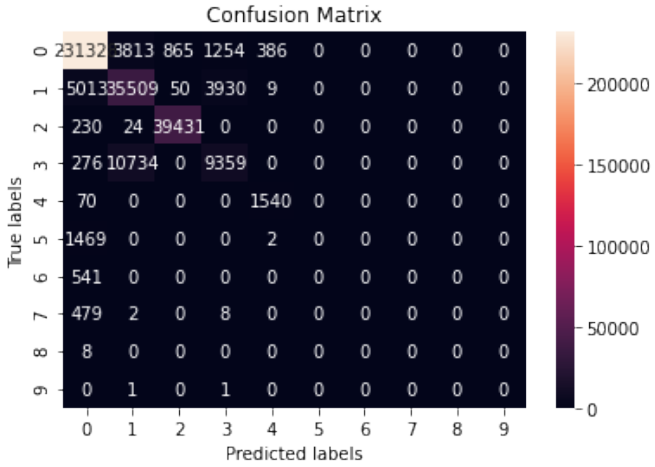

Figure 30: Confusion matrix for model with 1 feature chosen by the search algorithm.

The performance of a baseline model and a model that uses PCA data as input. As the data remained the same and only the labelset changed, there are still 17 features under consideration after the application of PCA. Tab. 12 shows the new labelset.
The results of the experiments with the full labelset can be seen in Tab. 13. The performance actually degrades, probably because the new classes are too small or because the attack types are too similar to separate properly. Figs. 33 and 34 show the confusion matrices of the results of the new models. It can be seen that the web attack classes (10, 11 and 13) have most misclassifications. The prediction of the different DoS types seems to still work well, especially for the PCA model, possibly because these classes were relatively big, but when looking at the web attack classes, these results show that merging the labels was an adequate choice to increase predictive power.

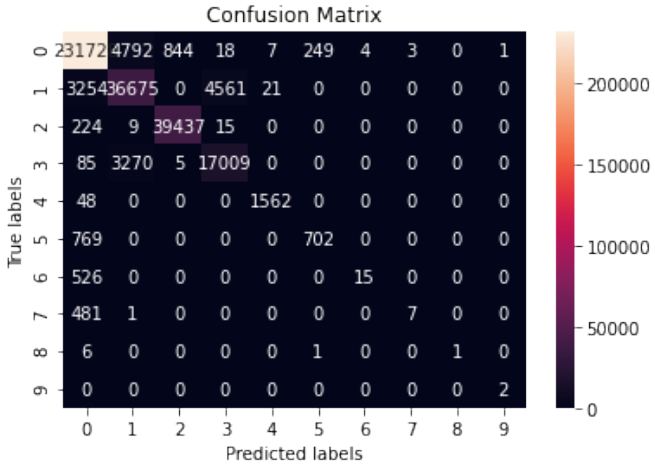

Figure 31: Confusion matrix for model with 2 features chosen by the search algorithm.

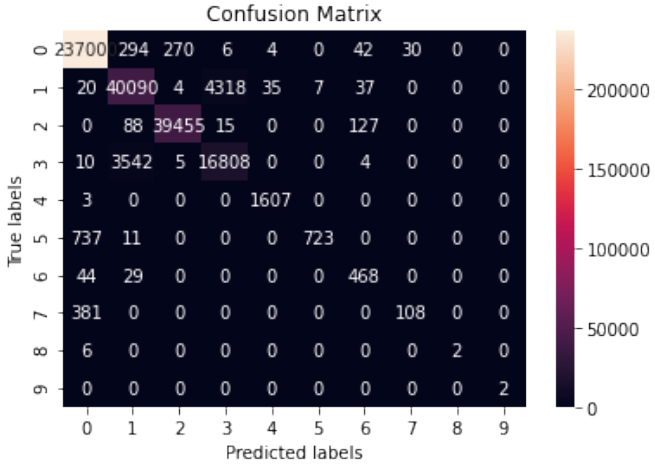

Figure 32: Confusion matrix for model with 3 features chosen by the search algorithm.

5 Discussion

5.1 Results

When looking back at the objectives defined in Chapter 1, we see that a relatively simple deep learning model is able to perform quite well. However, the performance of this model as described by Vinayakumar et al. [40] could not be replicated.

The effects of feature reduction vary per method. Tab. 14 shows the performance of the different models with different feature reduction methods. When looking at the confusion matrices shown in the previous chapter, it can be concluded that there seems to be no favoritism towards the majority classes. This did seem to be the case for the original baseline model but after introducing lognormalized data and an increased learning rate, this problem seems to have been alleviated.

The deep learning model with features chosen by random forest shows the least promising performance, as it is not able to score higher than the baseline model. K-means seems to deliver the

Attack name	Nr. of occurrences	Label
Benign	950588	0
DoS Hulk	163854	1
Portscan	158740	2
DDoS	81478	3
DoS GoldenEye	7709	4
FTP-Patator	6439	5
SSH-Patator	5883	6
DoS slowloris	4154	7
DoS Slowhttptest	2327	8
Bot	1956	9
Web attack - Brute Force	1507	10
Web attack - XSS	635	11
Infiltration	32	12
Web attack - Sql Injection	21	13
Heartbleed	7	14

Table 12: Attack types in CICIDS dataset without merged labels.

	Baseline merged	**PCA merged**	**Baseline full**	**PCA full**
Accuracy	0.9995	0.9996	0.9990	0.9991
Precision (macro)	0.9956	0.9945	0.8820	0.8809
Recall (macro)	0.9477	0.9543	0.8407	0.8348
F1 score (macro)	0.9682	0.9719	0.8426	0.8356
Precision (weighted)	0.9995	0.9996	0.9990	0.9991
Recall (weighted)	0.9995	0.9996	0.9990	0.9991
F1 score (weighted)	0.9995	0.9996	0.9988	0.9989
Identification score	0.9836	0.9855	0.9141	0.9100
Predicted classes	0-9	0-9	0-12, 14	0-12, 14

Table 13: Performance of models with a merged labelset and a full labelset.

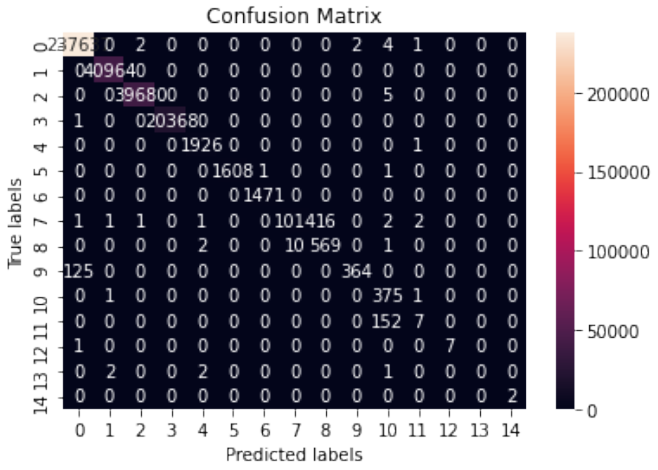

Figure 33: Confusion matrix for baseline model with full labelset.

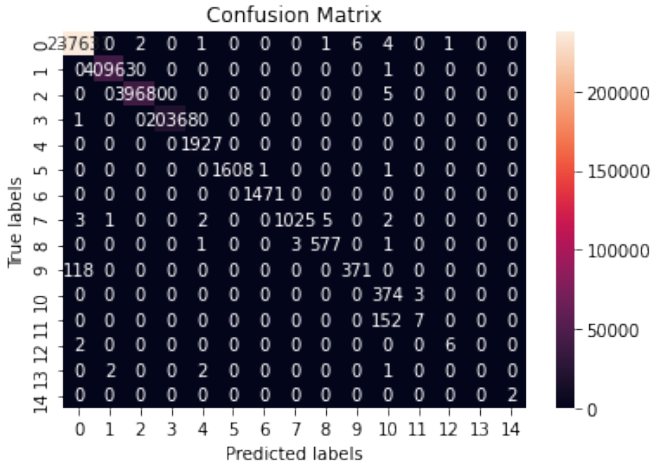

Figure 34: Confusion matrix for PCA model with full labelset.

	Baseline	**PCA**	**Random Forest**	**Correlation (54 features)**	**K-means (67 features)**	**Search (3 features)**
Accuracy	0.9995	**0.9996**	0.9815	0.9995	**0.9996**	0.9709
Precision (macro)	**0.9956**	0.9945	0.9538	0.9770	0.9921	0.9132
Recall (macro)	0.9477	0.9543	0.8689	0.9724	**0.9580**	0.7543
F1 score (macro)	0.9682	0.9719	0.8859	0.9719	**0.9730**	0.6254
Precision (weighted)	0.9995	**0.9996**	0.9832	**0.9996**	**0.9996**	0.9711
Recall (weighted)	0.9995	**0.9996**	0.9815	0.9995	**0.9996**	0.9709
F1 score (weighted)	0.9995	**0.9996**	0.9806	0.9995	**0.9996**	0.9704
Identification score	0.9836	0.9855	0.9308	0.9855	0.9861	0.8686
Predicted classes	0-9	0-9	0-9	0-9	0-9	0-9

Table 14: Performance of models based on different feature reduction methods.

most promising results, though the performance improvements can be considered to be small. PCA also showed promising results and it might be argued that PCA is more suitable than K-means as it requires less features as input. The model with features chosen by correlation also shows some improvement in performance and could be considered to provide a model that is of similar quality of the model with features created by PCA, however, it requires 54 features while PCA only requires 17 and is thus arguably able to include more information in a smaller dataset. It should however be kept in mind that PCA has the advantage of using all the data to create 17 new features while correlation is restricted to using the features as they are and might thus be less efficient. Applying PCA to the 54 features found by the correlation method did not boost performance.

We also found that alleviating data imbalance by combining classes together led to an improvement in performance. This could either be thanks to the fact that the classes that were combined were too small on their own to provide sufficient information during the training process or it might be the case that several attack classes are too similar to be detected separately.

5.2 Comparing to other works

Next, we will look at the performance of other deep learning methods and how their performance compares to the performance of our models. The performance of these neural networks has been summarized in Tab. 15. For all comparisons to other works it should be kept in mind that it is often not clear whether or not macro or weighted scores are presented and many works only present accuracy as performance metric, however, this metric does not fairly represent the true performance of a model due to the problem of class imbalance.

Maseer et al. [25] used a test set that only included traffic of the three types of web attack and benign traffic and therefore performance results cannot be compared directly to the results presented in this work. They implement a neural network model that is able to score 0.9931 on accuracy, 0.9950 on precision, 0.9931 on recall and 0.9922 on the F1-Score. They report that this model is unable to detect the SQL injection attack, of which there are only 21 instances in the entire dataset. As this attack type is so underrepresented it could be possible that the model simply does not have enough data to learn how to detect this attack type. In this work the web attacks were combined into one attack type, which seems to have been a good decision. Maseer et al. [25] also implemented a random forest, which was able to score the best results namely 0.9954 on accuracy, 0.9956 precision, 0.9954 on recall and 0.9955 on the F1-Score. This performance is even better than our baseline and PCA deep learning models, however, this could be due to the limitations of the test set that they used. The confusion matrices for the baseline model and the PCA model (Figs. 18 and 19) show that these models have a small amount of misclassifications and are thus also able to detect the web attack well. Malaija et al. [24] created a fully connected neural network, a variational autoencoder and a sequence-to-sequence model and used a random sample of the dataset for training and testing. The variational autoencoder performed very poorly and the authors chose to not report any results for this model. The fully connected network scored 0.834 on accuracy and precision, 1 on recall and 0.910 on the F1-score. The sequence-to-sequence model scored 1 on all metrics.

Faker et al. [15] also used K-means for feature selection and found that a featureset of 67 features led to optimal performance for a deep learning model. They were able to produce an accuracy of 0.9956, which is lower than the accuracy of 0.9996 found in our work. They do not present any other performance scores, thus comparison is limited. Additionally, their deep learning model is built differently and they left out all benign records, which will have affected the performance as well.

We will also briefly look at the performance of other types of models and how their performance compares to our deep learning model, the results of these models are summarized in Tabs. 16, 17 and 18. However, as the differences between the models and usage of the feature reduction methods are quite stark, comparison of performance could be considered unsuitable and should be done carefully.

Krishna et al. [20] present a variance based feature elimination procedure and report an accuracy of 0.994 on fast K-nearest neighbors, however, this is the only metric that is presented and in this work we have seen that accuracy is not a good measure of performance because of the large majority class. All our models score relatively high on accuracy but their actual performance is reflected by other metrics.

Yulianto et al. [43] present the use of SMOTE (an oversampling technique) in combination with PCA and EFS, a feature selection package available in R, to improve the AdaBoost algorithm for classification. They find an accuracy of 0.8183, a precision of 0.8183, a recall of 1 and an F1-score of 0.9001 when combining SMOTE, EFS and AdaBoost. In the case of EFS, adding SMOTE leads to better performance, however, for PCA SMOTE degrades the performance. Only using PCA with AdaBoost leads to an accuracy of 0.8147, a precision of 0.8169, a recall of 0.9596 and an F1-score of 0.8817. It is not clarified whether these scores are weighted or macro scores, however, in both cases the deep learning models presented in this work show better performance. Yulianto et al. [43] cite a baseline experiment with AdaBoost that reports a precision and F1-score of 0.77 and a recall of 0.84 thus it could be concluded that the effects of PCA on AdaBoost are more impactful than its effects on a deep learning model. Additionally, it should be kept in mind that this work focuses on the detection of DDoS attacks and only uses part of the dataset or maybe even a different dataset, as they claim to use the monday-working-hours part of the dataset, while on Monday no attacks are included. The fact that different data are used complicates the comparison of the works.

Powell et al. [31] implemented SelectKBest with a decision tree and found an accuracy of 0.97 while the size of the optimally sized featureset decreased to 23. They were able to obtain an accuracy of 1 when combining decision tree with VarianceThreshold. In this work we were able to reach an accuracy of 0.9995 when combining SelectKBest with a deep learning model and retaining 54 features. A similar accuracy (0.9736) was reached when the deep learning model got 17 features as input. This could suggest that the deep learning model works better than the decision tree as it can reach a similar performance with less features.

Abdulhammed et al. [4] implemented an autoencoder and PCA for feature reduction for random forest, Bayesian network, LDA and QDA. They found that for the random forest an F1-score of 0.977 and an accuracy of 0.996 could be obtained when 64, 40 or 30 features were taken into account as chosen by PCA. With PCA, the Bayesian network performed best with 40 features, reaching an accuracy of 0.964 and an F1-score of 0.974, LDA scored 0.914 on the F1-score and 0.901 on accuracy with 81 features and QDA scored 0.975 on the F1-score and 0.967 on accuracy with 70, 64 and 59 features. When looking at the results for features chosen by the autoencoder, the random forest scores 0.996 on both accuracy and the F1-score for both 70 and 69 features, the Bayesian network scores 0.996 on the F1-score and 0.953 on the accuracy with 70 features, LDA scores 0.922 on the F1-score and 0.912 on the accuracy for 81 features and scores 0.970 on the F1-score and 0.960 on the accuracy for 70 features. From these results it cannot directly be concluded that either the autoencoder or PCA is the best option for feature detection as superiority of the methods differs per classifier. Our deep learning models are able to outperform these methods, thus it might be concluded that deep learning models are superior to these simple classifiers even when the classifiers are enhanced with feature reduction. As baseline results are not presented by Abdulhammed et al. [4], it is not possible to draw conclusions about for example for which model PCA boosts performance the most. Abdulhammed et al. [4] also show conflicting results after applying UDBB: recall for the random forest decreases from 0.996 to 0.988 but precision increases from 0.965 to 0.989.

Reis et al. [33] implement a random forest with presumably permutation importance as feature selector. They find that reducing the dataset from 69 features improves precision (from 0.926 to 0.931), does not improve recall (from 0.919 to 0.910), and does not improve F1-score (from 0.921 to 0.919). However, the authors consider these differences negligible. Our deep learning model is able

to reach higher performance scores, as can be expected as we have seen that other comparisons of our deep learning model to random forest models led to the conclusion that the deep learning model performs better. However, deep learning models can be very complex and hard to train so sometimes a simpler model with lower (but still acceptable) performance might be preferred.

Overall, the models created in this work perform on the same level as deep learning models found in the literature, suggesting that feature reduction is a promising technique to investigate further to create even better models. However, comparison with models found in the literature should be done with care as these models were not trained with exactly the same partition of the dataset and different preprocessing techniques could have been applied to the data which could have affected the performance. Additionally, it is not known whether the reported scores are weighted or macro scores which makes comparison increasingly difficult. In this work, different feature reduction methods have been compared under the same conditions, resulting in a tangible comparison that showed that PCA outperforms other feature reduction methods. When looking at the performance of different types of models, it can be concluded that deep learning models perform on a higher level and seem to be most promising for further investigation in relation to attack detection.

	Neural Network [25]	**Random Forest [25]**	**FCN [24]**	**Seq-to-seq [24]**	**K-means [15]**
Accuracy	0.9931	0.9954	0.834	1	0.9956
Precision	0.9950	0.9956	0.834	1	-
Recall	0.9931	0.9954	1	1	-
F1 score	0.9922	0.9955	0.910	1	-

Table 15: Performance of neural networks (with or without feature reduction applied) found in other works.

	LVFE on K-NN [20]	**SMOTE, EFS on AdaBoost [43]**	**PCA on AdaBoost [43]**	**SelectKBest on decision tree [31]**	**VarianceThreshold on decision tree [31]**
Accuracy	0.994	0.8183	0.8147	0.97	1
Precision	-	0.8183	0.8169	-	-
Recall	-	1	0.9596	-	-
F1 score	-	0.9917	0.8817	-	-

Table 16: Performance of other types of models with feature reduction applied found in other works.

	PCA on RF [4]	**PCA on BN [4]**	**PCA on LDA [4]**	**PCA on QDA [4]**
Accuracy	0.996	0.964	0.901	0.967
Precision	-	-	-	-
Recall	-	-	-	-
F1 score	0.977	0.974	0.914	0.914

Table 17: Performance of other types of models with feature reduction applied found in other works.

	Autoencoder on RF [4]	Autoencoder on BN [4]	Autoencoder on LDA [4]	Permutation importance on RF [33]
Accuracy	0.996	0.953	0.960	-
Precision	-	-	-	0.931
Recall	-	-	-	0.910
F1 score	0.996	0.996	0.970	0.919

Table 18: Performance of other types of models with feature reduction applied found in other works.

Not only do we present a fair comparison between different feature reduction methods, we also look at the features selected by each method. While it seems that PCA delivers the most promising results, this method does not extract any explicit features but instead combines features to maximize the variance. For the last objective we can thus only take random forest, correlation and search into account. Tab. 19 shows the features chosen for each method. It can be seen that the features chosen by random forest are also chosen by the correlation method but Packet Length Std is only chosen by K-means and the search method. As the correlation featureset contains 54 features, it can be expected that the random forest features can be found in this set. When looking at the other feature sets chosen by correlation, it can be seen that only when 36 features are chosen the features from random forest are included. Additionally, the featureset of 9 features chosen by correlation outperforms the featureset of 9 features chosen by random forest, which could imply that random forest might not be the most efficient features reduction method. Interestingly, `Source IP` and `Destination IP` are included in smaller subsets for random forest, correlation and search but only included in K-means when 17 features are selected. When looking at the chosen features it seems that the IP addresses and port numbers are amongst the most informative features, as well as information about the packets and interarrival times of the flows. This information might be interesting for researchers who want to create a new cybersecurity dataset (for attack detection or other purposes) and need to know what information they need to save.

Not many works on feature reduction with the CICIDS2017 dataset disclose what features have been marked important. Works that do will be discussed next. Krishna et al. [20] present a list of 11 features that has been chosen by their LVFE framework, namely `ACK Flag Count, PSH Flag Count, URG Flag Count, Destination Port, Flow Duration, Fwd IAT Total, Bwd IAT Total, Init_Win_bytes_forward, min_seg_size_forward, Bwd Packet Length Std` and `Down/Up Ratio`. `PSH Flag Count` is only included in the subset of 54 features found by correlation and in the biggest subset of K-means features, suggesting that these methods do not consider this feature to be of too high importance compared to other features. `Flow Duration` and `Destination Port` are more commonly found features. `Bwd IAT Total` is an example of a feature only found by correlation. Interestingly, Init_Win_bytes_forward is only found in the subset of 54 features chosen by correlation and in the subset of 36 features chosen by K-means, while `Init_Win_bytes_backward` is found in all other feature subsets but is not chosen by LVFE. min_seg_size_forward is only found by random forest and the correlation method for 54 features. `Bwd Packet Length Std` is found by all methods except search. `Down/Up Ratio` is only found by the K-means method.

Abdulhammed et al. [4] do not provide a complete list of important features but note that `Subflow Fwd Bytes, Flow Duration, Flow IAT, PSH Flag Count, SYN Flag Count, Average Packet Size, Total Length Fwd Packets, Active Mean and Min, ACK Flag`

`Count`, and `Init_Win_bytes_fwd` are among the most important ones. Again we see that information about flags, packets and bytes is deemed most important. Most of these features are also found by the feature reduction methods employed in our work.

Reis et al. [33] listed `Destination Port`, `Fwd IAT Min`, `Init_Win_bytes_forward`, `Init_Win_bytes_backward` and `Flow IAT Min` as the most important features found by their feature reduction methods, which were metrics related to decision trees. These features are in line with the features found in our work and other works that have been presented above.

Thus, we can see that overall there is a preference for features that contain information about flow bytes and flow packets and identifying information such as port numbers and IP addresses. Additionally, flag counts are also highlighted by multiple feature reduction methods.

5.3 Limitations

This work only focuses on one dataset and could therefore be considered to be at least *partially biased*. In order to improve the quality of the conclusions regarding feature reduction methods and their effects on deep learning methods for attack detection, the experiments should be repeated with other datasets. As already pointed out, comparing to other works has to be done with care as the nature of the reported scores (weighted or macro) is often not known, only accuracy might be reported or the nature of the model is vastly different. It might for example be interesting to rerun the feature reduction experiments with other models than a deep learning model such as a random forest, AdaBoost or a decision tree.

Especially the experiment considering sequential feature selection was very limited due to the time consuming nature of the algorithm. With a different, fasters classifier this algorithm might be more suitable for investigation. Alternatively, if enough computation time or more computation power is used it might be possible to complete the experiment as it was intended in this work.

In order to create more robust and more reliable experiments, cross validation can be applied to future experiments.

6 Conclusion

Feature reduction is a valuable tool for reducing the amount of noise in a dataset. We found that using K-means to select features shows the most performance gain on an attack detection model when using the CICIDS2017 dataset. Another promising technique seems to be PCA, which combines features in order to reduce covariance but is able to create a smaller input space of less features for a performance that is competitive with the performance of the features selected by K-means. IP addresses, port numbers and packet statistics convey the most useful information for the detection of attack events. Combining classes to combat class imbalance can be beneficial but should be done carefully. Future research could look more into the sequential feature selection method and elaborate more on the features it picks when given more time to run. Additonally, it could be run with alternative evaluation measures such as the identification score. In this work, remarkable performance improvements were found through the addition of single features, and it would therefore be interesting to see if the algorithm can lead to smaller, but better performing feature sets than the feature sets found by the other methods or if the performance improvements

speed up when sequential feature selection uses another metric for choosing new features. Another avenue for future work that might be interesting would be to investigate how to deal with new information, for example when unknown IP addresses are introduced, and how to help the model correctly identify whether or not this new IP is malicious or not or if a new port number relates to a specific attack. The K-means experiment could be extended to the evaluation of clusters based on multiple features. This would however complicate the method a lot and should be done with care.

References

[1] DARPA Dataset. Available on: https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets.

[2] KDD Cup 1999 data. Available on: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[3] NSL-KDD Dataset. Available on: https://www.unb.ca/cic/datasets/nsl.html.

[4] R. Abdulhammed, H. Musafer, A. Alessa, M. Faezipour, and A. Abuzneid. Features Dimensionality Reduction Approaches for Machine Learning-based Network Intrusion Detection. *Electronics*, 8(3):322, 2019.

[5] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi. Internet of Things Security: A Survey. *Journal of Network and Computer Applications*, 88:10–28, 2017.

[6] S. Bouktif, A. Fiaz, A. Ouni, and M. A. Serhani. Optimal Deep Learning LSTM model for Electric Load Forecasting Using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches. *Energies*, 11(7):1636, 2018.

[7] G. Chandrashekar and F. Sahin. A Survey on Feature Selection Methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[8] Y.-W. Chen and C.-J. Lin. Combining SVMs with Various Feature Selection Strategies. In *Feature extraction*, pages 315–324. Springer, 2006.

[9] L. Cheng, F. Liu, and D. Yao. Enterprise Data Breach: Causes, Challenges, Prevention, and Future Directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(5):e1211, 2017.

[10] R. V. Deshmukh and K. K. Devadkar. Understanding DDoS Attack & Its Effect in Cloud Environment. *Procedia Computer Science*, 49:202–210, 2015.

[11] ENISA. Data breach. 2020. Retrieved from https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl2020-data-breach.

[12] ENISA. Distributed denial of service. 2020. Retrieved from https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl-2020-denial-of-service.

[13] ENISA. Web application attacks. 2020. Retrieved from https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl-2020-web-applications.

[14] ENISA. The year in review. 2020. Retrieved from https://www.enisa.europa.eu/publications/year-in-review.

[15] O. Faker and E. Dogdu. Intrusion Detection Using Big Data and Deep Learning Techniques. In *Proceedings of the 2019 ACM Southeast Conference*, pages 86–93, 2019.

[16] M. Husák, J. Komárková, E. Bou-Harb, and P. Čeleda. Survey of Attack Projection, Prediction, and Forecasting in Cyber Security. *IEEE Communications Surveys & Tutorials*, 21(1):640–660, 2018.

[17] K. Huseyin. Evolution of War and Cyber Attacks in the Concept of Conventional Warfare. *Journal of Learning and Teaching in Digital Age*, 3(1):12–20, 2018.

[18] I. T. Jolliffe and J. Cadima. Principal Component Analysis: a Review and Recent Developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[19] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman. Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges. *Cybersecurity*, 2(1):1–22, 2019.

[20] K. V. Krishna, K. Swathi, and B. B. Rao. LVFE: A Feature Selection Approach for an Efficient NIDS on Cloud Environment Using Least Variance Feature Elimination. 2020.

[21] L. Le Jeune, T. Goedemé, and N. Mentens. Machine Learning for Misuse-Based Network Intrusion Detection: Overview, Unified Evaluation and Feature Choice Comparison Framework. *IEEE Access*, 2021.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[23] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature Selection: A Data Perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–45, 2017.

[24] R. K. Malaiya, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim. An Empirical Evaluation of Deep Learning for Network Anomaly Detection. In *2018 International Conference on Computing, Networking and Communications (ICNC)*, pages 893–898. IEEE, 2018.

[25] Z. K. Maseer, R. Yusof, N. Bahaman, S. A. Mostafa, and C. F. M. Foozy. Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset. *IEEE Access*, 9:22351–22370, 2021.

[26] N. Moustafa and J. Slay. UNSW-NB15: a Comprehensive Dataset for Network Intrusion Detection Systems (UNSW-NB15 Network Dataset). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6. IEEE, 2015.

[27] D. Ourston, S. Matzner, W. Stump, and B. Hopkins. Applications of Hidden Markov Models to Detecting Multi-Stage Network Attacks. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 10–pp. IEEE, 2003.

[28] R. Panigrahi and S. Borah. A Detailed Analysis of CICIDS2017 Dataset for Designing Intrusion Detection Systems. *International Journal of Engineering & Technology*, 7(3.24):479–482, 2018.

[29] Z. Pelletier and M. Abualkibash. Evaluating the CICIDS2017 Dataset Using Machine Learning Methods and Creating Multiple Predictive Models in the Statistical Computing Language R. *Int. Research J. of Adv. Engineering and Science*, 5(2):187–191, 2020.

[30] M. Pivarníková, P. Sokol, and T. Bajtoš. Early-stage detection of cyber attacks. *Information*, 11(12):560, 2020.

[31] A. Powell, D. Bates, C. Van Wyk, and D. de Abreu. A Cross-comparison of Feature Selection Algorithms on Multiple Cyber Security Datasets. In *FAIR*, pages 196–207, 2019.

[32] X. Qin and W. Lee. Attack Plan Recognition and Prediction Using Causal Networks. In *20th Annual Computer Security Applications Conference*, pages 370–379. IEEE, 2004.

[33] B. Reis, E. Maia, and I. Praça. Selection and Performance Analysis of CICIDS2017 Features Importance. In *International Symposium on Foundations and Practice of Security*, pages 56–71. Springer, 2019.

[34] I. H. Sarker, A. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng. Cybersecurity Data Science: an Overview from Machine Learning Perspective. *Journal of Big Data*, 7(1):1–29, 2020.

[35] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSp*, pages 108–116, 2018.

[36] S. Singh Panwar, Y. Raiwani, and L. S. Panwar. Evaluation of Network Intrusion Detection with Features Selection and Machine Learning Algorithms on CICIDS2017 Dataset. In *International Conference on Advances in Engineering Science Management & Technology (ICAESMT)-2019, Uttaranchal University, Dehradun, India*, 2019.

[37] D. Stiawan, M. Y. B. Idris, A. M. Bamhdi, R. Budiarto, et al. CICIDS2017 Dataset Feature Analysis with Information Gain for Anomaly Detection. *IEEE Access*, 8:132911–132921, 2020.

[38] J. M. Torres, C. I. Comesaña, and P. J. Garcia-Nieto. Machine Learning Techniques Applied to Cybersecurity. *International Journal of Machine Learning and Cybernetics*, 10(10):2823–2836, 2019.

[39] M. J. Turcotte, A. D. Kent, and C. Hash. Unified Host and Network Dataset. *arXiv preprint arXiv:1708.07518*, 2017.

[40] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman. Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access*, 7:41525–41550, 2019.

[41] L. Wang, A. Liu, and S. Jajodia. Using Attack Graphs for Correlating, Hypothesizing, and Predicting Intrusion Alerts. *Computer Communications*, 29:2917–2933, 2006.

[42] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy. Training Deep Neural Networks on Imbalanced Datasets. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4368–4374. IEEE, 2016.

[43] A. Yulianto, P. Sukarno, and N. A. Suwastika. Improving Adaboost-based Intrusion Detection System (IDS) Performance on CICIDS2017 Dataset. In *Journal of Physics: Conference Series*, volume 1192, page 012018. IOP Publishing, 2019.

Feature reduction method	Features chosen
Random forest	Source IP, Init_Win_bytes_backward, Destination Port, Source Port, Bwd Packet Length Max, Destination IP, Bwd Packet Length Mean, Avg Bwd Segment Size, min_seg_size_forward
Correlation (9 features)	Source IP, Destination IP, Bwd Packet Length Std, Flow IAT Std, Flow IAT Max, Packet Length Mean, Packet Length Variance, Average Packet Size, Init_Win_bytes_backward
Correlation (17 features)	Featureset of size 9 and Flow Duration, Bwd Packet Length Mean, Flow Packets/s, Flow IAT Mean, Fwd Packets/s, Bwd Packets/s, Packet Length Std, Avg Bwd Segment Size
Correlation (36 features)	Featureset of size 9, 17 and Destination Port, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Bwd IAT Mean, Bwd IAT Max, Max Packet Length, Subflow Fwd Bytes, Subflow Bwd Bytes, Idle Mean, Idle Max, Idle Min
Correlation (54 features)	Featureset of size 9, 17, 36 and Source Port, Total Fwd Packets, Total Backward Packets, Fwd Packet Length Mean, Flow Bytes/s, Bwd IAT Total, Bwd IAT Std, Bwd IAT Min, Fwd Header Length, Bwd Header Length, FIN Flag Count, PSH Flag Count, Avg Fwd Segment Size, Subflow Fwd Packets, Subflow Bwd Packets, Init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward
K-means (9 features)	Avg Bwd Segment Size, Subflow Bwd Bytes, Total Length of Bwd Packets, Fwd Packet Length Max, Bwd Packet Length Max, Average Packet Size, Packet Length Mean, Packet Length Std, Packet Length Variance
K-means (17 features)	Featuresubset of size 9 and Destination Port, Fwd IAT Mean, Init_Win_bytes_backward, Source IP, Destination IP, Max Packet Length, Bwd Packet Length Std, Bwd Packet Length Mean
K-means (36 features)	Featuresubsets of size 9, 17 and Total Fwd Packets, Subflow Fwd Packets, Subflow Bwd Packets, Total Backward Packets, min_seg_size_forward, Flow Bytes/s, Init_Win_bytes_forward, Avg Fwd Segment Size, Fwd Packet Length Mean, Fwd IAT Total, Fwd IAT Std, Flow IAT Max, Fwd Packet Length Std, Flow IAT Std, Flow IAT Mean, Subflow Fwd Bytes, Total Length of Fwd Packets, Fwd IAT Max, Bwd Header Length
K-means (54 features)	Featuresubsets of size 9, 17, 36 and Fwd IAT Min, Active Min, Fwd Packets/s, Bwd IAT Total, Idle Min, Idle Max, Bwd IAT Min, Bwd Packets/s, Active Max, Idle Mean, Flow Packets/s, Active Mean, Bwd IAT Std, Flow Duration, Bwd IAT Mean, Bwd IAT Max, act_data_pkt_fwd, Fwd Header Length
K-means (67 features)	Featuresubsets of size 9, 17, 36, 54 and Fwd PSH Flags, Active Std, ACK Flag Count, FIN Flag Count, Idle Std, URG Flag Count, Down/Up Ratio, PSH Flag Count, Fwd Packet Length Min, Min Packet Length, Flow IAT Min, Source Port, Bwd Packet Length Min
Search	Packet Length Std, Total Fwd Packets, Source IP

Table 19: Features chosen by different reduction methods.