universität
wien

# MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Subverting Network Intrusion Detection:
Crafting Adversarial Examples
Accounting for Domain-Specific Constraints“


verfasst von / submitted by

Martin Teuffenbach, BSc


angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna, 2020

| | |
|---|---|
| Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet: | A 066 910 |
| Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet: | Master Computational Science |
| Betreut von / Supervisor: | Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant |

# Contents


**1 Introduction** **2**

**2 Related Work** **3**
2.1 Use of Machine learning for intrusion detection systems . . . 3
2.2 Adversarial Examples . . . 4
2.2.1 Definition . . . 4
2.2.2 Taxonomy . . . 5
2.2.3 Transferability of adversarial examples . . . 7
2.2.4 Adversarial Examples in Computer Vision . . . 7
2.2.5 Adversarial Examples in Network Intrusion Detection Systems . . . 8
2.2.6 Robustness against Adversarial Examples . . . 8

**3 Preliminaries** **10**
3.1 Threat Model . . . 10
3.2 Models . . . 11
3.2.1 Objective . . . 11
3.2.2 Deep Neural Network . . . 11
3.2.3 Deep Belief Network . . . 12
3.2.4 Outlier Detection . . . 13
3.2.5 Evaluation Metrics . . . 14
3.3 Datasets . . . 15
3.3.1 NSL-KDD . . . 15
3.3.2 CICIDS2017 . . . 16
3.3.3 CIDDS . . . 17
3.3.4 Preprocessing . . . 18
3.4 Attack Algorithms . . . 19
3.4.1 Fast Gradient Sign Method . . . 19
3.4.2 Carlini and Wagner Attack . . . 20
3.4.3 Optimizer . . . 21

**4 Methodology** **22**
4.1 Grouping . . . 22
4.2 Crafting . . . 23
4.3 Vulnerability Score . . . 24

**5 Results** **26**
5.1 Model Evaluation . . . 26
5.1.1 Objective and setup . . . 26
5.1.2 Deep Belief Network . . . 26
5.1.3 Deep Neural Network . . . 28
5.1.4 Outlier Detection . . . 32
5.1.5 Comparative Evaluation of Network-based Intrusion Detection System (NIDS) models . . . 32
5.2 Attack Evaluation . . . 33
5.2.1 Grouping . . . 34

5.2.2 Comparative evaluation of the attack algorithms . . . . . 34
5.2.3 Transferability . . . . . 37
5.2.4 Attack on Network Intrusion Detection models . . . . . 40
5.2.5 Vulnerability Score . . . . . 43

**6 Conclusion** **45**

**7 Appendix** **52**
7.1 Model architecture . . . . . 52
7.2 Results Transferability . . . . . 53
7.3 Results Attack Evaluation . . . . . 54

**Abstract**

With the growing amount of data in today's internet traffic, the demand for faster and more robust NIDS is increasing. Therefore, application of Deep Learning (DL) algorithms to intrusion detection is gaining more interest. Deep learning algorithms outperform other detection algorithms when it comes to computational efficiency and accuracy. However, machine learning based detectors and in particular deep learning approaches have recently been found vulnerable to so called adversarial examples, inputs crafted with the intent of causing classification algorithms to misclassify with high confidence. Adversarial examples were discovered in the field of image recognition, where the concept of 'imperceptibility' for a human observer is vital. Researchers managed to fool an classifier into labelling a STOP-sign as a 'Speed Limit 45' sign with a confidence over 90% by adding a Sticker [1]. With the growing interest in DL applied in NIDS adversarial examples must be considered as a potential threat.
This thesis will showcase how adversarial example attacks might be used to evade network intrusion detection systems. To transfer the idea from computer vision to the network security domain restrictions on the features were formulated and a metric for evaluating the risk was developed. An extended version of the Carlini and Wagner Attack algorithm [2] alongside with feature-restrictions for adversarial examples for network-flow datasets to ensure validity constraints is proposed. To highlight the threat the possibility of a black-box attack utilizing the phenomenon of transferability [3] is elaborated.

**Zusammenfassung**

Durch die stetig wachsenden Datenmengen die heutzutage im Internet verschicket werden ist der Bedarf an schnellen und zuverlässigen Network-based Intrusion Detection System (NIDS) gestiegen. Um dem nachzukommen sind Deep Learning (DL) Algorithmen in den Fokus der Wissenschaft gekommen, da diese akkurater und effizienter sind als andere Detektions-Techniken. Jedoch zeigten aktuelle Publikationen, dass vor allem Deep Neural Network (DNN) basierte Algorithmen, anfällig gegen sogenannte Adversarial Examples sind. Bei Adversarial Examples handelt es sich um Daten, welche manipuliert wurden um einen klassifizierungs Algorithmus zu täuschen. Dieses Phänomen wurde erstmal im Bereich Computer-Vision untersucht. Forscher haben es geschafft, ein Bild eines Stoppschilds so zu verändern, dass das in einem Auto implementierte DNN es fälschlicherweise als Geschwindigkeitsbegrenzung 45mph Schild erkannt hat [1]. In diesem Bereich spielt die Ununterscheidbarkeit für einen menschlichen Beobachter eine wichtige Rolle, das Ziel ist Daten nur minimal zu verändern, sodass kein visueller Unterschied zu echten Daten besteht, der Algorithmus jedoch nicht korrekt klassifiziert.
Mit dem steigenden Interesse von DL Algorithmen im NIDS Bereich müssen Adversarial Examples als reale Bedrohung untersucht werden. Diese Masterarbeit wird zeigen, wie sich dieses Phänomen gegen Algorithmen zur Detektion von Angreifern verwendet werden können. Im Zuge dessen werden für den Internet Verkehr typische Beschränkungen formuliert und eine Metrik zur Evaluierung des Risikos eines Adversarial Example Angriffs entwickelt. Ein in der Forschung etablierten Algorithmus, der Carlini und Wagner Algorithmus [2], wird erweitert um Validitäts Beschränkungen zu berücksichtigen. Die Möglichkeit eines Angriffs auf ein Black-Box NIDS wird durch Anwendung des Transferabiltiy Theorems [3] untersucht.

# Acknowledgements

This Project has been done in a collaboration between the Austrian Institute of Technology (AIT) and the University of Vienna.

I wish to express my gratitude to my supervisors at AIT, Dr. Ewa Piatkowska and Dr. Paul Smith, for giving me the opportunity to do my research and for their guidance, as well as for their encouragement and their passion. Thanks to them and my colleagues at AIT I had the privilege to work in a supportive and friendly environment in the course of the project.
Furthermore, I would like to thank my University supervisor Professor Claudia Plant for her constructive feedback and for helping me to realize my project, and Professor Kerstin Hummer, without whom this thesis would not have been possible.
Last but not least I want to thank my caring partner Victoria and my family for their support throughout the project.

# 1 Introduction

The goal of an Intrusion Detection System (IDS) in the IT domain is to monitor a network or devices for suspicious activities and to detect intrusions. We distinguish two types of detection systems: Host-based Intrusion Detection Systems (HIDS) and Network-based Intrusion Detection System (NIDS). HIDS run on a particular device on the network and monitor activities on the machine, like log files, to detect malware in the system, whereas NIDS are installed on a point on the network, to monitor network segments of all the devices online and monitors for suspicious patterns. Most NIDS rely on a library of known attacks to detect malicious traffic.

Major challenges of NIDS are the rapidly growing volume of network data and the increasing number of different protocols and the diversity of data in modern networks. Detection systems are required to process data with high speed and to accurately deal with unknown behaviour. To match those requirements researchers investigate the use of Machine Learning (ML) techniques for NIDS. ([4] and [5]). Particularly, Deep Learning (DL) methods such as deep neural networks are receiving substantial interest as they outperform shallow networks in accuracy. Several DL algorithms have been proposed for NIDS, some reaching an average accuracy score up to 99% for specific datasets ([6], [7]).

However, DL algorithms were found to be vulnerable to adversarial examples [8], i.e. inputs crafted to cause a misclassification. A lot of research on this phenomenon has been done in the image recognition domain, where a small modification of pixel values, which is imperceptible to a human observer, causes an algorithm to misclassify the image. Most algorithms make use of gradient information of the model to craft adversarial examples ([9], [8], [10]) or query the model in order to approximate the gradient ([11]).

There are several techniques to defend against adversarial examples. One of the main countermeasures is called Adversarial Training ([10]), which simply retrains a model using adversarial examples. Others tried to increase robustness by training an additional binary classifier to detect adversarial examples, stacked on to the Deep Neural Network (DNN) [12] or network distillation, which transfers knowledge from a network to a smaller to improve the training [13]. However, these defence strategies are shown to be effective only for part of attacks and most fail to defend unseen attacks. Stronger defence strategies are urgently required.

This thesis is a continuation/extension of the work done by Teuffenbach *et al.*[14], submitted for the 15th International IFIP Cross Domain Conference for Machine Learning & Knowledge Extraction (CD-MAKE 2020). Parts of the Introduction, Related work and Preliminary Section in [14] have been adapted for this thesis. The Methodology Section (4) is content wise similar. The Experiments (Section 5) have been improved and evaluated again entirely and an extensive Transferability evaluation has been added.
This thesis aims to further improve the understanding of adversarial examples against NIDS. The following steps are elaborated to approach the objective:

1. Transfer the concept of Adversarial Examples to NIDS domain.
   As a first step it is elaborated how the concept of imperceptibility in image recognition translates to an NIDS dataset. In addition, a threat model for attacking a NIDS utilizing adversarial examples are proposed. To approach this attack, challenges and constraints are discussed. A framework to categorize features in order to generalize restrictions is presented. Utilizing this framework, a crafting algorithm for adversarial examples that is aligned with these restrictions are proposed.

2. Reproduce state-of-the-art DL models for NIDS.
   The performance of different classifiers trained to perform the intrusion detection task is investigated. A representative sample of proposed IDS algorithms is implemented and evaluated on benchmark network traffic datasets. These datasets are analysed according to the proposed framework.

3. Evaluate the impact of adversarial example attacks on NIDS.
   The robustness of the DL models against adversarial examples is evaluated with various budget constraint. In this process, a metric to measure the vulnerability against these attacks is proposed.

The reminder of this thesis is structured as follows: Section 2 reviews related work in this domain and in Section 3 the theoretical background for evaluation of the objective will be elaborated. Section 4 derives the grouping of features alongside with the crafting algorithms, as well as the robustness metric. Section 5 will discuss obtained results. Finally, the conclusions are given in Section 6.

# 2 Related Work

In this section, related work is presented. First about DL in the NIDS domain, followed by research about adversarial examples in general, in the computer vision, and finally in the NIDS domain.

## 2.1 Use of Machine learning for intrusion detection systems

An IDS is a software or a device that monitors a systems for malicious activity. The goal is to detect and report any intrusion activity. Hamed *et al.*[15] categorizes IDS components into three parts: *(i)* pre-processing/feature extraction, *(ii)* pattern analyser, which involves knowledge representation, and learning processes, and *(iii)* decision making. Distinguishing normal from different types of anomalous behaviour in a network is usually equivalent to a classification problem, i.e. the goal is to assign a label to every network traffic instance. This could be done in the form of binary classification (normal or anomalous) or multi-label classification.
The first challenge arising is to define an input instance, and to find enough training data to build a robust classifier. Most publicly available datasets are highly imbalanced and outdated, as recent years have seen an increase in the number of new protocols used in modern networks. For example, the KDD99 Cup dataset [16], which is used in around 50% of all NIDS research of the last decade [17], even though it was generated over 20 years ago in 1999. In addition to that, most ML techniques require a comparatively high level of human interaction to process data (identifying useful data and patterns).
To deal with those limitations, DL, a subset of ML, is increasingly utilized by the research community. These algorithms have the potential to extract better representations from the data (unsupervised learning) to create robuster models with fewer training samples.
There are two main approaches used in NIDS, the first one is Anomaly Detection (AD), which is a binary classification into two classes (anomalous and normal). AD methods based on DL techniques can deal with high volumes of data and have the potential to detect novel attacks. Basically, these techniques assign labels based on the distance to the normal behaviour. If the distance exceeds a predefined threshold, anomaly is reported.

The definition of the distance measure depends on the algorithm. One example for an AD algorithm would be an Auto Encoder (AE) [6], which is an unsupervised neural network-based feature extraction algorithm, which can also be used for dimensionality reduction [18]. This method uses the reconstruction error as a measure of distance to find anomalies.
The second main methodology is signature-based detection, mostly in the form of multi-class classification. Hereby, patterns of the input data are extracted and compared to known malicious samples. The major limitation of this approach is that it is restricted to detect known attacks. An obstacle of both methods, signature-based and anomaly-based detection systems, in an IDS setting is the lack of labeled data. To overcome this issue unsupervised (or semi-supervised) learning techniques such as Deep Belief Network (DBN) ([19], [20]) are used. To increase the performance of a single DNN for intrusion detection ([21], [7]) researchers are combining various DL algorithms,[22] for example stacked an AE onto the DNN. Hodo at al. [23] and Xin at al. [24] made a comprehensive analysis of DL techniques in NIDS.

## 2.2 Adversarial Examples

### 2.2.1 Definition

Adversarial Examples where first discussed by Szegedy *et al.*[8]. In this work they are referred to as *blind-spots of neural network*. Later Ian Goodfellow *et al.*[10] named and defined the phenomenon. They formulated the following definition.

**Definition 1.** *Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake.*

The idea is to get the classifier to misclassify an instance by adding a small perturbation to the input. Figure 1 shows an example presented by Goodfellow *et al.*.

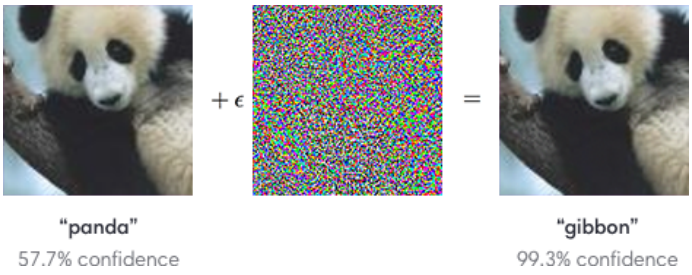



Figure 1: Adversarial Example *Source:* [10]

A small distortion (size $\epsilon$ in figure 1) causes the classifier to change the originally correct prediction. An important aspect of this is that a human observer is not able to distinguish the adversarial example from the original image. This idea is often referred to as imperceptibility for a human observer and is almost always implied when adversarial examples in the computer vision domain are investigated. Formally, for a classifier F:

$$F : x \mapsto y$$

(where $x \in \chi$ sample-space and $y \in \Upsilon$ set of labels) an Adversarial Example $x'$ is defined as

$$x' = x + \delta \quad \textbf{s.t} \quad F(x) \neq F(x') \tag{2.1}$$

In practice, equation (2.1) is often formulated as an optimization problem, where one solves for the minimal distance $\delta$ that fullfills $F(x) \neq F(x+\delta)$.

### 2.2.2 Taxonomy

A classification model is a multidimensional function $F : x \rightarrow y$ where $X$ is the input vector and $Y$ the output vector. In the case of AD, Y would be a scalar indicating whether or not the input is anomalous. The objective of an adversary is to find the optimal perturbation $\delta$ in equation (2.1). Researchers came up with various optimization approaches to this objective, most prominent algorithms are the Fast Gradient Sign Method (FGSM) [10], Jacobian based Saliancy Map Attack (JSMA) [9], Deepfool [25] and the Carlini and Wagner $L_2$ norm Attack (C&W) [26]). A review of proposed adversarial examples crafting algorithms can be found in [27]. The following taxonomy is taken from Papernot *et al.*[9] and Carlini *et al.*[28]. Papernot *et al.*elaborated the Threat model of an adversarial attack by defining the *Adversarial Goal* and the *Adversarial Capabilities*. The *Adversarial Goal* is to impact the integrity of the classifier output. The four primary goals of an adversary, enumerated by increasing difficulty, are:

A. Confidence reduction:
Reduce the output confidence of classification. This goal does not require the model to misclassify, but only to increase the uncertainty.

B. Misclassification/Non targeted attacks:
Alter the output classification to a class different from the original class (equation (2.1))

C. Targeted misclassification:
Produce inputs (generate new input) that are classified as a specific target class $y^*$. The idea is not to add a perturbation to an existing instance, but to generate an instance of class $y$ ($y \neq y^*$) that is misclassified.

D. Source/Target misclassification:
Add perturbation to an existing input to force the classifier to classify it as an specific target class $y^*$. Formally the objective is to find $\delta$ to solve the following equation:

$$\text{argmin}_\delta ||\delta|| \quad \textbf{s.t} \quad F(x+\delta) = y^* \tag{2.2}$$

Similar to the *Adversarial Goal*, the *Adversarial Capabilities* can be categorized in five levels:

A. Training data and network architecture:
The adversary has perfect knowledge of the classification model (weights, biases, number of layers) and access to the training data. The attacker can analyse the training data and has full gradient information.

B. Network architecture:
The adversary has perfect knowledge of the classification model (weights, biases, number of layers) without the training data. With this the adversary has access to the gradient information.

C. Training data:
The adversary can collect pairs of input and output data in the training phase. The adversary has no knowledge of the models architecture, but can reproduce the original model using the training data.

D. Oracle:
The adversary has no information about the original model but can query it. Using various inputs the adversary can approximate the gradient using a finite difference approach.

E. Samples:
The adversary can collect pairs of input and output data in the test phase. He cannot modify these inputs.

The first two bullet points are called white-box attacks and the latter three black-box attacks. Figure 2 depicts this taxonomy.

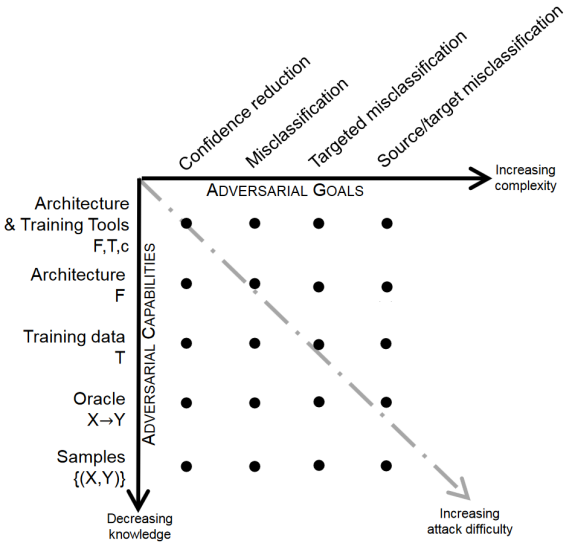



Figure 2: Threat Model Taxonomy *Source:* [9]

In order to tailor this taxonomy to other domains Carlini *et al.*[28] extended the *Adversarial Capabilities* by splitting it into 2 subcategories: *Adversarial Knowledge*, which is defined as the *Adversarial Capabilities* by Papernot *et al.*[9] (see above) and *Adversarial Capabilities*. This includes reasonable constraints for the adversarial instances, for example restrict the norm of the perturbation $\delta$ in equation (2.1) to be small (*perturbation budget*) or restrictions in the feature space (*feature budget*). This convention is used in this thesis.
There is some research done about adversarial examples in the NIDS domain (for example [29]), most papers, however, often treat network-traffic datasets arbitrarily. Adversarial capabilities have not been discussed properly, as they hardly made any restrictions in feature space, which makes sense for attacking an image input, but not for an network-traffic input. To the authors knowledge the feasibility of an adversarial example-attack on a NIDS has not yet been discussed properly.

### 2.2.3 Transferability of adversarial examples

Papernot *et al.*[3] observed, that adversarial examples generated for a specific model may also fool a different model with a different architecture or even a different ML algorithm. He referred to this phenomenon as transferability. The following theorem is given.

**Definition 2.** *Adversarial examples that affect one model often affect another model, even if the two models have different architectures or were trained on different training sets, so long as both models were trained to perform the same task.*

Mathematically, an adversarial instance $x'$ that full fills equation(2.1) might also fool a different classifier $\tilde{F} : x \to y$, that is trained to perform the same classification task as $F$, but with $\tilde{F}(x) \neq F(x)$:

$$F(x + \delta) = \tilde{F}(x + \delta) = y^* \tag{2.3}$$

This phenomena can be utilized for black-box attacks against an IDS. A substitute model with known architecture is trained to reproduce the output of an detection system in order to craft adversarial examples. The success of those transferred samples strongly depends on the threat model. An attacker with Oracle knowledge, e.g., is able to train a substitute model using efficient sampling techniques to even achieve source/target misclassification. Furthermore, it is possible [3] to use data augmentation to generate samples around the decision boundary of the original classifier to train a substitute model and increase its quality. Training the substitute model is done by maximisation of the quality measure, e.g., a relative number of matching labels of a test set. Achieving robust adversarial examples by training a model with only input-output samples is significantly more difficult and has not been researched yet in the NIDS domain. A threat model with Oracle knowledge is not particularly realistic in an NIDS setting, as querying a system would be detected immediately.

### 2.2.4 Adversarial Examples in Computer Vision

Early research about adverarial examples ([9], [10], [13]) is done almost exclusively for computer vision, due to extensive use of DL in this domain. Publicly available benchmark datasets are used to showcase the phenomenon of adversarial examples. Potential consequences of intentional degradation of image recognition capabilities can have quiet sever safety impact, e.g. in the domain of self driving cars. To demonstrate the risk of an adversarial example attack in the physical world, Eykholt *et al.*[1] proposed the *stop-sign attack*, which is designed to fool the DNN-based classifier deployed in an self driving car to misclassify a stop sign as a 45mph sign. For evaluation the adversarial images were printed, captured with an standard video-camera and forwarded to an convolutional neural network classifier. The proposed algorithm [1] incorporates environmental constraints, like different lightning conditions, limits of printability and spatial constraint, as only perturbations on the actual sign and not the background are possible. To deal with the spatial constraints, a mask-matrix was introduced, which sets perturbations in undesired areas, i.e. in the background of the image, to 0. Despite all this restrictions, success rates are rather high and vary from 73% to 100% for different scenarios were achieved. Figure 3 depicts an example image of the printed adversarial posters. This attack on an AI installed in an self-driving car was one of the first real-world adversarial example attacks.


Figure 3: Stop-Sign attack *Source:* [1]

### 2.2.5 Adversarial Examples in Network Intrusion Detection Systems

Due to the growing interest in adversarial example for image recognition, researchers tried to transfer this concept to the NIDS domain. Currently, the challenge is to design a realistic scenario to attack an intrusion detection system using adversarial examples. Machine Learning in the NIDS domain is quit different in the computer vision domain with respect to feature properties. An image consists only of numerical features (i.e. pixels) with the same numerical range, in a network-traffic instance binary and categorical features, as well as feature dependencies, may appear. Additionally, the range of numerical features can vary significantly. However, early research, for example [29], often treat network-traffic datasets arbitrarily. Adversarial capabilities have not been discussed properly for most proposed attacks, as they hardly made any restrictions in feature space, which makes sense for attacking an image input, but not for an network-traffic input. Recently, Zhang *et al.*[30] investigate a reinforcement learning approach to match IDS dataset specific restrictions. The action space of their algorithm only contains valid actions, meaning actions that would not degrade the compliance of the instance. Hashemi [31] introduced the idea of treating features based on their properties differently when crafting adversarial examples for flow-based NIDS and also takes dependencies of features into account. Crafting valid adversarial Examples to evade network intrusion detection in black-box setting remains an open research challenge.

### 2.2.6 Robustness against Adversarial Examples

In general classification models are characterised by parameters like *accuracy* and *precision*, which reflect the models performance on a given testest. As recent research has showcased that adversarial examples are a potential threat to ML models, the demand of a metric for robustness against those samples arises. Different ideas on how to measure the vulnerability against adversarial examples are proposed frequently. As the concept of adversarial examples in image recognition involves the imperceptibility from the original image for a human observer, a lot of metrics use the distance ($L_1/L_2$ norm) as a measure for robustness ([25], [13]). This makes sense in the computer vision domain, as the distance between original input and adversarial example is correlated with visible differences. Moosavi *et al.*[25] define the expectation value of the *minimal perturbation* over a testset as a measure of robustness.

This *Adversarial Robustness* is defined (see equation (2.4)) as the average distance to the closest decision boundary for each instance in a given testset. The proposed crafting algorithm aims to find the closest decision boundary and ,therefore, has an untargeted misclassification as adversarial goal.

$$\Delta(x; F) := \min_{\delta} ||\delta||_2 \quad \text{subject to} \quad F(x+\delta) \neq F(x) \tag{2.4}$$

The *Adversarial Robustness* $\rho_{adv}(F)$ of a model F is then defined as the expectation value of the minimum perturbation $\Delta(x, F)$ required to misclassify a sample x.

$$\rho_{adv}(F) = \mathbb{E}_{\mu}(\Delta(x, F)) \tag{2.5}$$

Where $\mu$ is the data distribution of the samples x. This score is dependent on the testset. The visualization of this metric is depicted in figure 4.

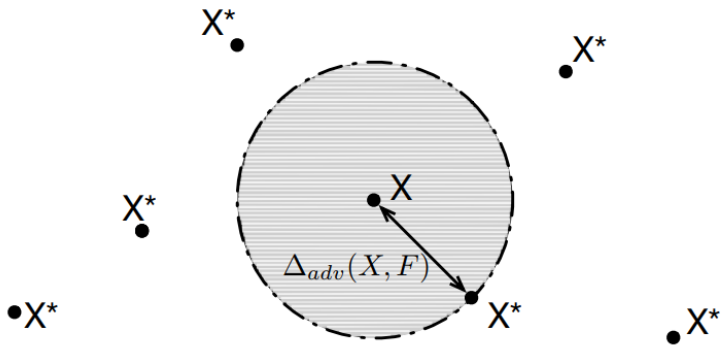



Figure 4: Adversarial Robustness $\Delta(x, F)$ Source: [13]

Using a theoretical approach, Weng *et al.*[32] developed the *CLEVER* score (**C**ross **L**ipschitz **E**xtrem **V**alue **Ne**twork **R**obustness), which is also an estimation of the minimal distance required to fool a neural network classifier. This score uses the Lipschitz constant (approximated using extreme value theory) as a measure, which brings the advantage that it is data-independent. In the NIDS domain Hartl *et al.*[33] developed the *Adversarial Risk Score* (ARS), which is a distance-based robustness score for classifiers trained for network intrusion detection. In the paper [33], they use a recurrent neural network for classification and investigate the feature sensitivity of their classifier. However, without considering properties of features, a distance based approach for measuring effectiveness of adversarial examples against NIDS might not reflect the true security threat. A lot of defence strategies utilise *gradient masking* ([34]), which relies on 'hiding' the gradient information by using hard-labels or by using a non-differentiable classifier instead of the DNN. However, these techniques do not prevent adversarial examples, but hide them.
This thesis aims to show, that in the security domain, the question about the robustness evaluation of NIDS algorithms should check whether or not there are adversarial examples reachable for an adversary, not how hard it is to find them. Therefore, robustness evaluation should consider white-box knowledge with useful gradient information. This work does not focus on how proposed defence techniques would improve the robustness against certain attacks. The goal is to show how vulnerable a given classifier is against a constrained adversary.

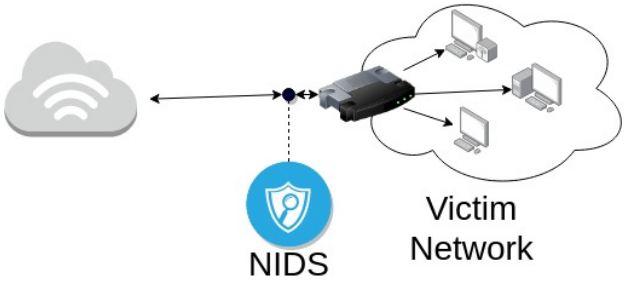
# 3 Preliminaries

## 3.1 Threat Model

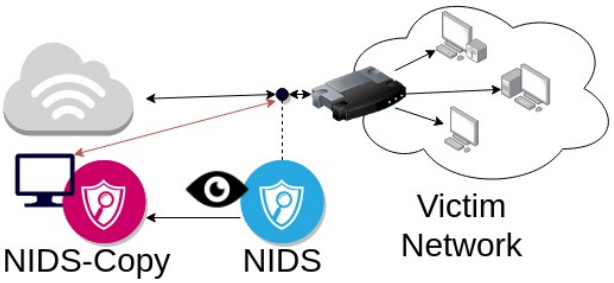
First, the definition of an adversarial examples needs to be adapted to the NIDS domain. As already discussed, the concept of imperceptibility is not applicable anymore. The following equivalent to this concept is proposed.

**Definition 3.** *Adversarial Examples in the NIDS domain are instances with anomalous properties/characteristics that are intentionally designed to cause the detection model to classify them as benign.*

As an adversarial image should appear to be the original image, a network-flow instance should still have the same impact as the original one. With the elaborated taxonomy (Section 2.2.2) and the extended definition above the Treat Model for the adversarial example attack will be derived. One of the main goals for an attack on a NIDS is to stay stealthy. In other words the attacker wants to bypass the detection system, which means attacking while being labelled as benign. In the taxonomy, this would translate to a Source/Target misclassification. Starting with an attack-instance the attacker wants to add perturbation to fool the classifier, while maintaining the characteristics of the attack.
Figure 5 depicts the architecture of the target system. An unknown NIDS is monitoring the network traffic of a small environment (figure 5a), e.g. a business network. Assuming that an attacker is able to monitor the system or to gain access to input-output samples, the attacker is able to reproduce the target detection system (figure 5b). By crafting adversarial examples for this substitute model, an attacker might be able to bypass the original detection system. For the first part of the evaluation presented in this thesis, *samples knowledge* is assumed (knowledge about the training samples). With this approach the transferability between a black-box and a substitute model will be evaluated. For the final robustness evaluation *Network architecture* knowledge is assumed, which is equivalent to assuming the substitute model is a perfect copy of the original system. As mentioned earlier, assuming limited knowledge would be a case of *security via obscurity*, not robustness against adversarial examples.
One key difference in the NIDS to the image recognition domain is reflected in the *Adversarial Capabilities*. Pixels in images can be changed more or less arbitrarily, without the need of taking dependencies and physical limits into account. Network traffic dataset, however, often inherits feature-dependencies value range constraint for certain values. To define the *Adversarial Capabilities* for the attack scenario, an approach for categorizing and a rule-set for manipulating features of an IDS dataset will be elaborated in Section 4.1. This contains a restricted feature-space and a maximum perturbation budget. The idea is to restrict the feature-space to feasible features, i.e. features an attacker can change without losing important properties of the attack, similar to feature-space reduction technique done by Eykholt *et al.*[1].




(a) Target Network




(b) Proposed attack scenario

Figure 5: Attack on a Black-Box NIDS

## 3.2 Models

### 3.2.1 Objective

The objective of this thesis is to fool DL classification models, trained to perform network intrusion detection, utilizing adversarial examples. As a first step towards this goal, different DL models are reproduced, trained and optimized to classify instances from a NIDS dataset. To get a representative sample, the following 3 models were implemented: A Deep Neural Network [22] (DNN) and a Deep Belief Network [19](DBN) for supervised and semi-supervised multiclass classification, and an AE [35] trained to perform outlier detection for unsupervised anomaly detection. The anomaly detection uses a distance based technique to detect outliers, which is fundamentally different from the other two models that compare patterns between known samples. The difference of the models in robustness and accuracy will be evaluated. The goal was to implement models that differ in their training algorithms in order to get a better understanding of feature importance for classification. All three models will be trained with the same datasets and attacked utilizing the same attack-algorithms to get comparable results.

### 3.2.2 Deep Neural Network

A feed-forward neural network, also often referred to as *multilayer perceptron*, is an artificial neural network consisting of an input layer, hidden layers and an output layer. The layers of the network are fully connected with directed connections (input to output). When speaking of a DNN, one refers to an artificial neural network with 2 hidden layers or more.

In this thesis, a DNN is trained using the Back Propagation (BP) algorithm [36] to perform a classification task. To extend the functionality of a regular DNN, which usually works as a supervised classifier, Rezvy *et al.*[22] proposed the idea to stack an AE on top of a neural network. An AE is a neural network trained to reconstruct the input with a low-dimensional hidden layer (smaller than the input-dimension). This technique is often used for dimensionality reduction, in [22] it is used to filter out noise. The network learns how to project an input to a smaller dimension and reconstruct it as accurate as possible and, therefore, learns which features contain most information. Figure 6 depicts the basic architecture of an AE (taken from [37]) with the input layer $l_1$, one hidden layer $l_2$ and an output layer $l_3$. The '+1' in this image represent the bias terms.

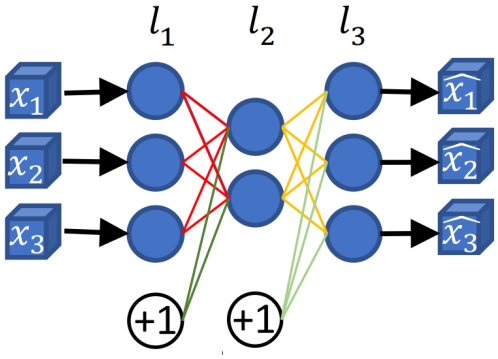
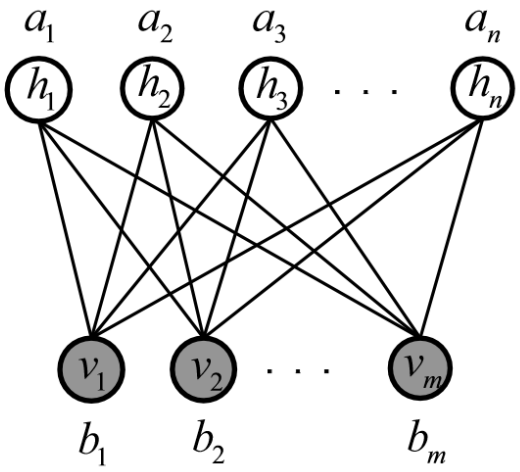



Figure 6: AE architecture *Source:* [37]

The proposed algorithms [22] are trained in three stages: First the unsupervised training of AE is performed, second, the classifier is trained to perform a supervised classification using with the output of the AE as input and in the third step, the network as a whole is trained with a few training samples in supervised manner. By training the DNN on the de-noised samples in the second stage, the classification is restricted the most informative features. This technique is intended to improve the robustness of the decision boundaries. Intuitively, the first two training stages initialize the weights to improve the training done by the last fine-tuning stage. This Auto-encoded Deep Neural Network (AEDNN) should outperform a regular DNN in scenarios, where there is only a limited amount of labelled training-samples available.

### 3.2.3 Deep Belief Network

A DBN is a generative model that consists of several layers of Restricted Boltzmann machines (RBMs), with a classifier stacked on top. A RBM is a generative stochastic artificial neural network that is often used for dimensionality reduction, classification and investigating feature importance.




Analogically to an AE, a RBM is trained to reproduce the input, however, trained with a probability-distribution based algorithm and with only two layers. Given the visible layer $\{v_i\}$ a (lower) dimension set of hidden layers $\{h_i\}$ can be sampled by the posterior distributions p($\{h_i\}|\{v_i\}$). With this hidden layer, a new representation of the visible layer can be sampled by the posterior distributions p($\{v_i\} \mid \{h_i\}$). The parameters of these probability distributions are learned with maximum likelihood learning. The goal is to reproduce the visible layer as close as possible.

Figure 7: RBM architecture *Source:* [19]

In the basic RBM architecture there are no connections between nodes of the same layer and each layer is fully connected to the adjacent layer, with undirected connections. Figure 7 depicts the graphical representation of an RBM.

The second layer (similarly to the low-dimensional hidden layer in the AE) is the input layer of the following RBM when constructing a DBN. A DBN can be represented as a graphical model, where the joint distribution of the visible layer $\nu$ and the hidden layer $h_k$ ($1 \leq k \leq n$, n=number of layers) is defined as follows:

$$p(\nu, h_1, ..., h_n) = p(\nu|h_1) \prod_{k=1}^{n-2} p(h_k|h_{k+1}) p(h_{n-1}|h_n) \tag{3.1}$$

where $\nu = \{\nu_i\}$ are the nodes of the visible layer and $h_i = \{h_j\}$ are the nodes of the $i$-th hidden layer (see Figure 7). The DBN proposed by [19] is trained in 2 phases: First the RBM are trained layer-by-layer in an unsupervised manner with maximum likelihood learning. In the second phase, the parameters of the DBN are fine-tuned using BP. The top layer RBM outputs the prediction of the model using exact gradient descent on a global supervised cost function between the output predictions and the true labels.

### 3.2.4 Outlier Detection

In contrast to the previous models, the implemented outlier-detection model is trained in an unsupervised manner. This classifier utilizes the reconstruction error of an AE as a measure of outlyingness. This concept goes back to Hawkin *et al.*[35]. The idea is to train an AE on only with benign data, such that anomalous (attack) samples are unknown to the model and, have therefore a higher reconstruction error. To detect outliers Hawkins defined the Outlier Factor (OF) of a record $i$ as follows:

$$OF_i = \frac{1}{n} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2 \tag{3.2}$$

Where n is the number of features (input dimension), $x_{ij}$ the j-th feature of record $x_i$ and $\hat{x}_{ij}$ is the j-th feature of the i-th reconstructed record (see Figure 6).

In their work, Hawking *et al.*compared the OF of benign and anomalous samples of a network-traffic dataset to illustrate.
As this thesis aims to get the model to output a more general score, which reflects the outlier-probability, it was combined with the method of Azami *et al.*[38] of converting distance based outlier detection methods to probabilities using a sigmoid function:

$$P(\text{i is outlier}) = (1 + exp(\frac{\text{OF}_i - \gamma}{\sigma}))^{-1} \tag{3.3}$$

where $\gamma$ is the anomaly threshold ($\text{OF}_i \geq \gamma \Rightarrow$ P(i is outlier) $\geq 50\%$) and $\sigma$ is a scaling constant. This function maps real numbers to [0, 1]. As this model does not require known attack samples to train, it is also able to detect novel attacks (unknown behaviour). The fact, that the model trains only on benign samples makes it more feasible for application in network systems, as samples with attack-behaviour are rarely available. The algorithm projects the features to a smaller dimension and back to the original dimension. In this process information which is not required to reconstruct original features, is discarded. In order for this anomaly detection to work, the method assumes that anomalous data instances contain information that is lost in the compression step and can therefore not be reconstructed correctly. The algorithm will work only if benign samples in AE representation are significantly different than anomalous samples.

### 3.2.5 Evaluation Metrics

In this thesis, the algorithms are evaluated using the following metrics: accuracy, recall, precision and F1-score are used. They are derived from the values of the confusion matrix True Positive of label i ($TP_i$), which is the number of correctly classified instances with true label i, False Positive of label i ($FP_i$), which is the number instances with true label not i but classified as i, True Negative of label i ($TN_i$)-the number of correctly classified instances with true label not i, and False Negative of label i ($FN_i$), which is the number of misclassified instances with true label i. The metric values are defined in Table 1:

| **Metric** | **Formula** | **Description** |
|---|---|---|
| $\text{Accuracy}_i$ | $\frac{TP_i+TN_i}{TP_i+TN_i+FN_i+FP_i}$ | Relative number of correct predictions |
| $\text{Recall}_i$ | $\frac{TP_i}{TP_i+FN_i}$ | Proportion of actual positives that are correctly classified |
| $\text{Precision}_i$ | $\frac{TP_i}{TP_i+FP_i}$ | Ratio of actual positives that are correctly classified to all correctly classified instances |
| $\text{F1-score}_i$ | $\frac{2*\text{Precision}_i*\text{Recall}_i}{\text{Precision}_i+\text{Recall}_i}$ | Harmonic mean of Precision and Recall. |

Table 1: Metrics used for model evaluation

The overall Accuracy, Recall, Precision and F1-score are then the averages over all labels $i$ weighted by support (the number of true instances for each label). As the testsets of the datasets are primarily benign instances (see Section 3.3), the accuracy alone does not reflect the detection performance of the models. Therefore the precision, which is intuitively measures the amount false alarms, and the recall, which measures the hit rate, are computed as well, alongside with their harmonic mean.

## 3.3 Datasets

For evaluation of the reproduced DL models, three different network-traffic datasets are used. In order to get comparable results, datasets with similar features and attack-labels were chosen. The NSL-KDD [39], the CIDDS-001 [40] and the CICIDS2017 [41] dataset are used for evaluation. All three of them can be considered as benchmark-datasets.

### 3.3.1 NSL-KDD

The NSL-KDD dataset is a refined version of the KDD99 dataset. The KDD99 dataset was developed in 1999 by the Defence Advanced Research Projects Agency (DAPRA) and contains labelled data of several weeks of internet traffic of a test environment. A variety of cyber attacks were launched during the measurement. To cope with the shortcomings of the original KDD dataset, all duplicate records have been removed and the selection of instances was altered to achieve a more realistic dataset, denoted as the NSL-KDD dataset. The NSL-KDD set is already split into a train set (125.973 records) and a test set (22.544 record), both with 41 features (Intrinsic, Content, Time-based and Host-based). In total, the sets contain 49 different labels, grouped into 5 attack categories: *Normal*, *DoS* (Denial of Service), *Probe*, *U2R* and *R2L*.

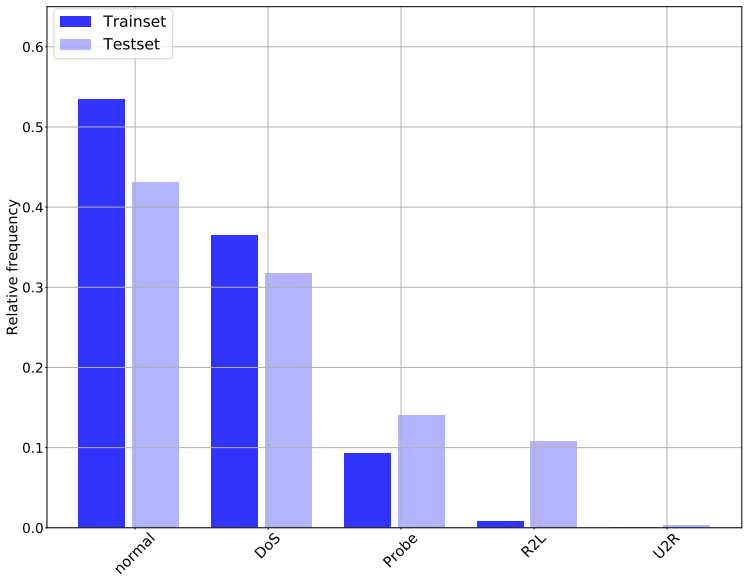



Figure 8: Label distribution of NSL-dataset

Dhanab *et al.*[39] summarized these attacks alongside with a more detailed analysis of the dataset.
*DoS*: Denial of service is an attack category, which depletes the victims resources thereby making it unable to handle legitimate requests.
*Probe*: Surveillance and other probing attacks objective is to gain information about the remote victim, e.g. port scanning

*U2R (User to Root)*: Unauthorized access to local super user (root) privileges is an attack type, by which an attacker uses a normal account to login into a victim system and tries to gain root/administrator privileges by exploiting some vulnerability in the victim, e.g. buffer overflow attacks.
*R2L (Remote to Local)*: Unauthorized access from a remote machine. The attacker intrudes into a remote machine and gains local access.

One challenge about the dataset is the higher diversity in the testset. For examples in the trainset only 53 instances of the group U2R appear and 995 R2L instances, whereas in the testset there are 200 U2R and 2754 R2L instances. There are also several attack-categories in the testset that do not appear in the trainset. Figure 8 depicts this diversity problem.

### 3.3.2 CICIDS2017

The intrusion detection dataset of the Canadian Institute of Cybersecurity (CICIDS2017) was developed by the University of New Brunswick in 2017. They monitored 5 days of internet traffic in a test-environment with 25 simulated users. In this 5 days, they conducted 14 types of attacks (most common attacks based on the 2016 McAfee report). The data was collected using the software CICFlowMeter. A more detailed analysis of the CICIDS2017 dataset is presented in Sharafaldin *et al.*[41]. The most dominant attacks (most instances) present in the dataset are the following:
*DDoS* Attack: It typically occurs when multiple systems,flood the bandwidth or resources of a victim. Such an attack is often the result of multiple compromised systems (for example, a botnet) flooding the targeted system with generating the huge network traffic.
*Infiltration* Attack: The infiltration of the network from inside is normally exploiting a vulnerable software such as Adobe Acrobat Reader. After successful exploitation, a backdoor will be executed on the victim's computer and can conduct different attacks on the victim's network such as IP sweep, full port scan and service enumerations using Nmap.
The *DoS* attack type is not listed here as it was already described in the previous Section. In total, this set contains about 2.8 million records with 80 features per record, grouped in 15 label categories (14 attack + benign).
The CICIDS2017 dataset is highly imbalanced, the attack-label *Heartbleed*, for example, appears only 11 times in the 2.8 million records, the label *Normal* 2.36 million times. This dataset has no predefined train- and testset, to divide the dataset a random 80/20 split was applied, depicted in Figure 9. In this figure, only the top 5 labels are depicted, as the other 10 labels are below 0.3% of the instances.

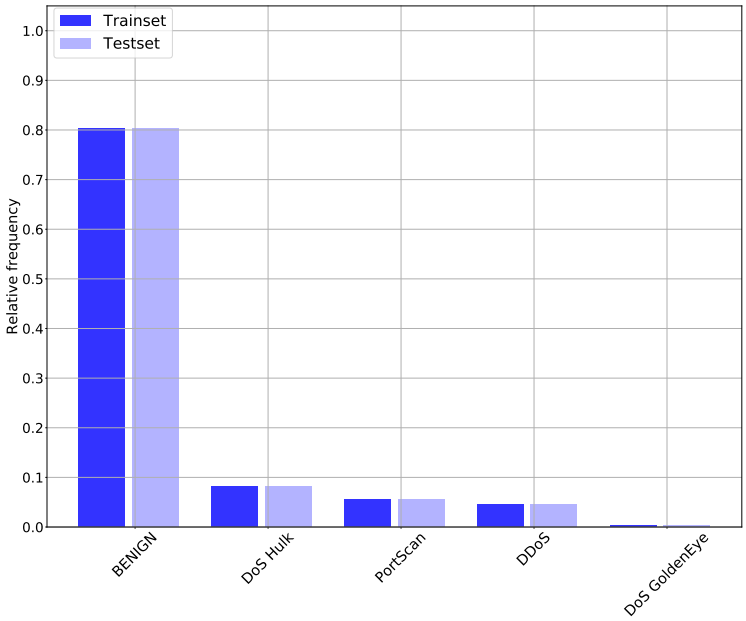



Figure 9: Label distribution of CICIDS2017-dataset

### 3.3.3 CIDDS

CIDDS-001 (Coburg Intrusion Detection Data Sets) [40] is a dataset for anomaly-based NIDS. It is a labelled flow-based set created in a virtual environment using OpenStack. A small business environment was emulated using python scripts. The emulation includes benign traffic and 4 types of malicious traffic, Denial of Service, Brute Force attacks and Port Scans. As these attack types are fairly similar to the attacks of the other datasets they will not be described further here. The dataset contains 2 weeks of records, one with only benign traffic ($\approx$ 6M instances) and one with benign and malicious traffic ($\approx$ 10M instances). As the available data is not split into train- and testset a 80/20 train-test-split was performed.
Figure 10 depicts the label distribution of the $2^{nd}$ week of records. It can be seen that over 90% of the instances are labelled as benign ('—'), which potentially results in a biased classification. In contrast to the other two datasets, this dataset contains, apart from flow-specific features (IP adresses and ports), only 5 numerical and 2 categorical features.

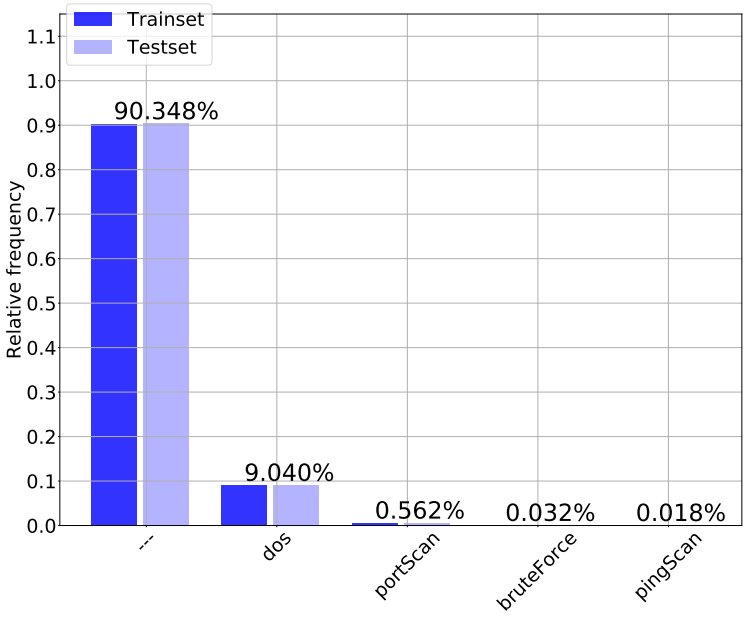



Figure 10: Label distribution of CIDDS-dataset

### 3.3.4 Preprocessing

Similar to Gao *et al.*[19], three preprocessing methods were applied. These are necessary for the datasets to be used as input to machine learning models. First, all categorical features are converted to numerical values and then, all features are normalized to have the range [0, 1]. The following 3 preprocessing steps were done for the datasets:

1. 1 to N Encoding (one-hot-encoding):
   Mapping categorical features to numerical values. For example 'Protocol' feature in the NSL-KDD set (3 distinct categories: 'TCP', 'UDP' and 'ICMP') is replaced by a $3x1$ vector.'TCP' protocol would translate to [1, 0, 0], 'UDP' to [0, 1, 0]. With this preprocessing step, the 41 attributes of the NSL-KDD set are numeralized as 122 features and the 7 features of the CIDDS-001 dataset to 29 features. As there are no categorical features in the available CICIDS2017 Machine learning files (without IP-addresses and ports), this step was skipped.

2. Log-Transformation:
   As the range of the features in the datasets varies significantly (some package-size related features range from 0 to $10^8$, others from 0 to $10^3$), the following log-transformation to all features $x_i$:
   $$x_i = log(1 + x_i)$$
   A value $x_i \in [0, 10^8]$ is transformed to $x_i \in [0, 18]$. This transformation was also applied by Hawkins *et al.*[35]. Without this transformation, outliers would cause most other values to be set to zero in the next step.

3. Min-Max transform:
   To set the range of all features to [0, 1], a min-max transform has been applied to each feature $x_i$ as follows:

$$x_i = \frac{x_i - x_{min}}{x_{max,i} - x_{min,i}}$$

   where $x_{max,i}$ and $x_{min,i}$ are the maximal/minimal appearing value of feature i. After this step the range of all features is [0, 1]

A challenge when using the NSL-KDD dataset is the high diversity of testing samples (the testing set contains labels that are not included in the training set). In addition, for *U2R* and *R2L* attacks, the training set is smaller than the testing set, which can have a negative impact on the accuracy of trained classifiers. To address this issue, Rezvy *et al.*[22] defined their own training and testing subsets by merging and re-splitting the existing sets (further referred to as NSL-KDD*) to train their classifiers with more samples of *U2R* and *R2L* attacks. For the experiments, the goal is to reproduce the results of the Rezvy *et al.*DNN model; therefore, both variants, of the NSL-KDD train- and testset were used for evaluation.
Due to the extensive size of the CICIDS2017 and the CIDDS-001 datasets, both were reduced in size by randomly removing instances. By reducing the set to 10% the size of the two datasets is about the same as in the NSL-KDD-dataset ($\approx$ 220.000 train- and 55.000 testsamples).
To deal with labels with low frequency in the three datasets, an over- and under-sampling technique was utilized in the training phase. The number of instances of each label with low frequency was increased by randomly adding redundant data and the benign label was undersampled by removing random instances. For the training, all labels were oversampled to have at least 10% the size of the benign label. Note that the testsets were not over- nor undersampled.

## 3.4 Attack Algorithms

To find the perturbation $\delta$ that satisfies equation (2.1), three crafting algorithms were implemented. The iterative FGSM [42], the C&W [2], and an extension of the C&W, that will be derived in Section 4.2, are used to evaluate the models. The objective of these algorithms is to find a perturbation for a given instance that satisfies equation (2.1) for a given model by utilizing gradient information. The C&W is still one of the most powerful white-box techniques to craft adversarial examples and therefore used in this paper. The iFGSM is used for comparison.

### 3.4.1 Fast Gradient Sign Method

The FGSM, proposed by Goodfellow *et al.*[10], was the first algorithm designed specifically to fool neural network classifiers to misclassify images. Goodfellow *et al.*implemented a simplified version of gradient descent (GD) in order to proof the theory, that DNN classifiers are particularly vulnerable to the new phenomenon of adversarial examples. They derived Equation (3.4) to showcase this.

$$x^* = x + \epsilon \cdot \text{sign}(\nabla_x F_\theta(x)) \tag{3.4}$$

Where, $x$ is a data sample, $F_\theta(x)$ the output activation function of the targeted classifier with parameters (weights and biases) $\theta$, $\epsilon$ the stepsize and $x^*$ is the adversarial sample with $F(x^*) = t$, t being the targed label. In the original paper, the algorithm is implemented as an one step method. Later, this method was extended by Kurakin *et al.*[42] using an iterative approach.

This simple algorithm was proven to be efficient in white-box settings, where gradient information is available. It does not, however, ensure a minimum perturbation size, which would be desirable for a stealthy attacker. This algorithm was considered in this thesis as it was one of the first and still one of the mostly used adversarial example crafting technique. The iterative version extends equation (3.4) with a while-loop that terminates when the goal is reached, or at a predefined maximum number of iterations. Algorithm 1 depicts this method.

**Algorithm 1:** Iterative FGSM

**Result:** Adversarial Example $x^*$ with $F_\theta(x^*)$=t
1 Classifier $F_\theta$, instance x, desired label t, maximum number of iterations maxiter;
2 **while** $F_\theta(x_i) \neq t$ *and* $i <$*maxiter* **do**
3 $\quad x_{i+1} = x_i + \epsilon \cdot \text{sign}(\nabla_x F_\theta(x_i))$;
4 $\quad$ **if** $F_\theta(x_{i+1})$=$t$ **then**
5 $\quad\quad x^*$=$x_{i+1}$;
6 $\quad\quad$ **return** $x^*$;
7 $\quad$ **else**
8 $\quad\quad$ i = i+1;
9 $\quad$ **end**
10 **end**

To prevent the risk of getting stuck in an local optimum, Dong *et al.*[43] further extended this algorithm with momentum. This is done by adding prior gradients to the current gradient value in line 3 of algorithm 1.

$$g_{i+1} = \mu g_i + \frac{\nabla_x F_\theta(x_i)}{||\nabla_x F_\theta(x_i)||} \tag{3.5}$$

### 3.4.2 Carlini and Wagner Attack

To find Adversarial examples with minimal distance to the original input, Carlini and Wagner [2] formulated the following optimisation problem:

$$\text{find } (\min_\delta ||\delta||_p \ + \ c \cdot g(x+\delta)) \quad \text{s.t. } x + \delta = x^* \in [0,1]^n \tag{3.6}$$

where $\delta = x - x^*$ is the distance between the input and the adversarial sample, c is a coupling constant and $g(x)$ is a target function. This approach ensures minimal perturbation, while maximizing the desired target function. Three distance measurements of perturbation $\delta$ in equation (3.6) are discussed in [2]: The $L_0$, the $L_2$ and the $L_\infty$ norm. The $L_0$ corresponds to the number of features that have been altered in an instance. The $L_2$ distance measures the standard Euclidean distance between x and $x_0$ and the $L_\infty$ the maximum change to any of the features. For each norm an attack algorithm was proposed. For this thesis, the $L_2$-norm attack is utilized. In the original paper [2], seven possible target functions $g(x)$ where listed, where all basically are minimal when the classifier outputs the desired label. The constant c links the minimization of the distance with the minimization of the target function, the smaller c, the more weight goes into the minimization of the distance. An example for such a target function would be the following:

$$g(x) = \max(\max_{i \neq t}(F(x)_i) - F(x)_t, \ -\kappa) \tag{3.7}$$

where t is the target label and kappa a constant. This function is minimal when the output probability of the target class is significantly higher then the output probability of all other classes.

The box-constraints in this algorithm are ensured by introduced a new variable $\omega$:

$$\delta = \frac{1}{2}(tanh(\omega) + 1) - x \tag{3.8}$$

and optimized over this substituted variable. Since $-1 \leq tanh(\omega) \leq 1$, it follows that $0 \leq x+\delta \leq 1$, the box-constraints are matched. Carlini and Wagner used the Adam [44] optimization algorithm [2] to solve the problem. For the implementation in this thesis, the Nadam [45] algorithm, an updated version of the Adam optimizer was used, described in the following Section .

### 3.4.3 Optimizer

The output-function of a DL classifier, as well as the optimisation objective described in equation (3.6) is, is in most cases, assumed to be a non-convex function. To solve this objective function, a variant of stochastic gradient descent (SGD), the Nadam-optimizer [45] (Nesterov-accelerated Adaptive Moment Estimation) was utilized. This algorithm improves the Adam optimizer [44], that is used in Carlini and Wagners paper, by combining it with the Nesterov accelerated gradient (NAG) algorithm. The Adam optimizer extends the regular SGD by adding an exponentially decaying average of past squared gradients and an exponentially decaying average of past gradients. These gradients can be understood as the momentum of the optimization and are used to prevent the algorithm from getting stuck in a local minimum. The NAG algorithm places an update rule for the momentum term in order to perform a more accurate step in the gradient direction. The authors of [45] claim to achieve not only faster convergence, but also improved quality of the models compared to the regular Adam optimizer.

# 4 Methodology

In this section, the *Adversarial Capabilities* described in Section 3.1, are further analysed and an approach for crafting adversarial examples for NIDS is derived. Finally, a metric will be defined to measure NIDS robustness (vulnerability) against adversarial examples. The metric intends to reflect the trade-off between attack budget and success rate.

## 4.1 Grouping

Hashemi *et al.*[31] proposed the idea of grouping flow-based features by their 'feasibility'. He divided them into 4 groups:

1. Features that should not be changed because they are extracted from backward flowing packets
2. Features that can be changed independently of each other by using the legitimate transformations
3. Features whose values depend on the second group and can be calculated directly by a set of them
4. Features that cannot be directly recalculated based on independent features values

Based on this grouping, he developed an iterative crafting algorithm for adversarial examples. With restricted feature space (first only group 2, features, then 2 + parts of 4 and finally 2+4), GD steps are performed until the goal of misclassification is reached. Each step the features of group 3 are recalculated. As the objective function, the output (prediction) function of the classifier is used. The algorithm stops when a certain confidence threshold is reached. In this thesis, the feature grouping is adapted to the considered adversarial example crafting algorithm. Like in Hashemi *et al.*[31], group 1 and 3 are considered as inaccessible by the crafting algorithm. Therefore, those two groups are merged into one group '0'. Next group 2 was split into 'independent and not used to derive group 3 features' (group 1) and 'independent and used to derive group 3 features' (group 2). The last group was not changed, but a more precise description was added. This group considers features that depend on batches of other instances, for example mean-values and frequency-based features. Features with underlying physical constraints are put in this group. Changing those features (e.g., inter arrival time (IAT)) might violate physical limits.
Based the proposed feature grouping, weights (bias-variables) $v$ were assigned to each group of features, that are used for the crafting algorithm. These weights should indicate how difficult it is for an adversary to perturb each feature. The weights are assigned to each group according to the enumeration above. Weight 0 indicates that features of this group are not accessible by the attacker. Weights 1 to 3 should indicate increasing difficulty. The crafting algorithm should favour features with weight 1 over 2 and 3. The comparison of the proposed grouping method with the one form Hashemi *et al.*[31] is summarized in Table 2

| Group | [31] | Description | weight $\upsilon$ |
|---|---|---|---|
| G0 | (1) | Features extracted from backward flows | 0 |
| | (3) | Features whose values depend on the other features and can be calculated directly by a set of them | 0 |
| G1 | (2) | Independent and not used to derive other features | 1 |
| G2 | (2) | Independent and used to derive other features | 2 |
| G3 | (4) | Features dependent on batches of packets (e.g., mean and frequency based features) | 3 |
| | | Features with underlying physical constraints (e.g., IAT) | 3 |

Table 2: The proposed feature grouping, including those from Hashemi *et al.*[31]

## 4.2 Crafting

For crafting adversarial examples the C&W algorithm by Carlini and Wagner [2] is used, as this method is among the most powerful crafting algorithms up to date and a benchmark technique to evaluate robustness of deep learning algorithms. Carlini and Wagner formulated an optimisation based approach to craft adversarial examples. They derived an objective function that maximizes the desired target-prediction while minimizing the size of the perturbation (see equation (3.6)). As this crafting algorithm performs a straight forward distance minimization, it is simple to extend it with the weighted feature grouping idea.

Based on the groups, weights $\upsilon$ (0 to 3, see 2) were assigned to each feature, according to the enumeration of the groups above. Using these weights $\upsilon$ along with a mask (set of features considered by the algorithm), the C&W (equation (3.6)) was extended with the new bias-variable $\upsilon$:

$$\min_{\delta}(||\delta \odot \upsilon||_2 \; + \; c \cdot g(x + \delta \odot mask)) \quad \text{s.t.} \quad x + \delta = x^* \in [0,1]^n \tag{4.1}$$

where $\odot$ indicates an element-wise vector-vector multiplication. The weights added to the distance in equation 4.1 forces the algorithm to favour perturbations on low-weighted features and avoid adding large perturbation on high-weighted features. The mask represents the proposed restriction on the featurespace. This mask sets the perturbation of undesired features to 0, here the mask acts on group 0 features. As the target function $g(x)$ in equation 4.1, the '$f_5$' function was used, presented in [2]:

$$g(x) = log(2 - 2 \cdot F(x)_t) \tag{4.2}$$

where $F(x)_t$ is the prediction of the classification model of target label t. This function becomes minimal when the prediction of the desired class $F(x)_t \approx 1$.

The idea using the weighted crafting algorithms is to get a bias of the perturbation towards easily accessible features. Ideally, an attacker would want to only alter features of group 1 and avoid changes of group 3 features. Here, the weights are set arbitrarily from 1 to 3, if necessary these weights could be adapted to achieve the desired outcome. Further restrictions, like for example an increase only restriction on IAT features, might be added in the update step of the optimizer.

For the evaluation in this thesis, the extended C&W presented in equation (4.1) was utilized. The pipeline of crafting the adversarial examples attack is depicted in Figure 11. As a first step, the features are analysed and grouped accordingly. In this figure, a few example features are listed. After the grouping, a few features are considered as not accessible (i.e. group 0 features, crossed in Figure 11) and weights are assigned to the other groups (different shades of grey in Figure 11). With these weights in place, the adversarial examples are crafted for each targeted instance. For the evaluation, several feature budgets are investigated.

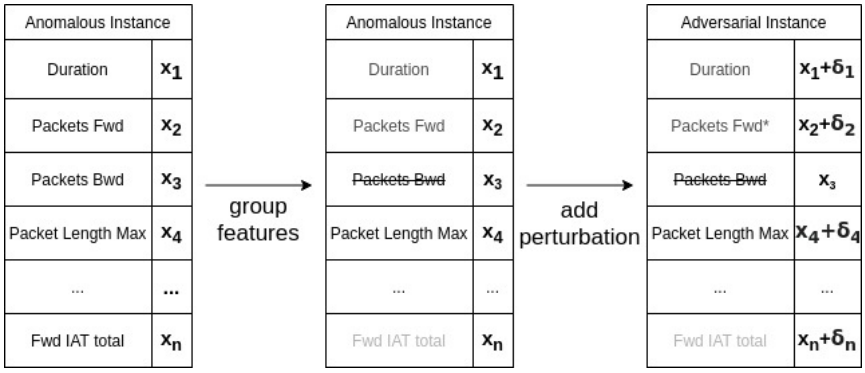



Figure 11: Example pipeline of Adversarial Example Crafting

## 4.3 Vulnerability Score

Finally, utilizing the weights derived in the previous section, a vulnerability score will be derived to measure the robustness of a NIDS against adversarial examples. First, to quantify the feature-budget and to link it to the crafting algorithm, the weighted featurespace-size f is introduced. This parameter gives an indication of the cost of an attack. The f-budget is calculated with the following formula:

$$f = \frac{mask \cdot v}{|v|_1} \tag{4.3}$$

where $mask \cdot v$ indicates a vector-vector product. If all features of group 1-3 are used, the f-budget is equal to 1, with fewer features goes to 0. Therefore, an attack is considered as a strong attack if the f-budget is close to 0 (only few features with low weights). The size of the mask is set with a predefined budget f. In addition to the restriction on the featurespace, the attacks are restricted with a maximum perturbation $\delta_{max}$. The second parameter for the vulnerability metric is the success rate of a given dataset. The goal is to compute the worst-case success rate for a given perturbation and feature budget. This value is dependent on the classifier F, the desired target label t and the distribution of attack-instances $D_j$:

$$s_{j \to t} = \frac{1}{N} \sum_{x \in D_j} \mathbb{1}_{F(x+\delta_x)=t} \quad \text{s.t. } |\bar{\delta_x}| \leq \delta_{max} \tag{4.4}$$

Where F is a given classifier, with $F : D \to Y$ ($Y$ being a set of labels), t the desired label ($t \in Y$) and $D_j$ is the data-distribution of instances with true and predicted label j ($D_j \subset D$ with $f(x) = j$ for all $x \in D_j$ and $|D_j| = N$).

$\mathbb{1}_{\text{condition}}$ is an indicator function, which is 1 if the condition is true, 0 otherwise. The perturbation $\delta_x$ which aims to turn x into an Adversarial Example is restricted with the mask, alongside with a maximum perturbation restriction on the average perturbation per feature $|\bar{\delta_x}|$. As only attacks that aim to be stealthy (target label t is benign) are considered, $s_{j \to t}$ as is denoted $s_j$ for the reminder of this thesis. The overall success rate $s$ would then be the average over all labels j $s_j$. Note, that this value only makes sense for a sufficiently large N. For one-hot-encoded labels the predicted label i of instance x is defined as $i =:$ argmax$(F(x))$. As a matter of course, this success rate is dependent on the technique used for crafting adversarial examples. The algorithm could fail to find the optimal $\delta_x$, that would be within my constraints. Therefore, the evaluation must be restricted to the *empirical success rate*, which is the overall success rate for a given crafting algorithm. In the evaluation section, s denotes the *empirical success rate*. Goodfellow *et al.*[10] interpreted adversarial examples as *blind spots* due to incomplete training data. Using this metaphor, the success rate can intuitively be seen as the relative number of attack-instances that have *benign-blind-spots* within a sphere in the 'mask'-hyperspace, with radius $\delta_{max}$. This parameter gives an insight of the reliability of the models decision boundaries. With all the parameters in place, the Vulnerability Score (VS) is proposed, as a metric for measuring the robustness of NIDS against adversarial examples:

$$VS = \frac{2 \cdot (1/f) \cdot s}{(1/f) + s} \tag{4.5}$$

which is the harmonic mean between the success rate (given the restrictions) and the inverse-f-budget. Therefore, the VS $\in$[0, 2], a score close to 2 means high vulnerability. The harmonic mean is chosen, as it is commonly used to compute the average of rates. This score reflects the trade-off between s and the size of the hyperspace. It gives an intuition of how difficult it is to bypass NIDS. For the evaluation of the NIDS, the value s was determined for various perturbation and f-budgets.

# 5 Results

This section is structured as follows. First, the model evaluation will be presented. The objective and the setup of this evaluation is outlined, followed by the architectures and classification results of the implemented models. Finally, the most important part of this section is the attack evaluation. In this final part, the process of the grouping, the transferability evaluation and the success-evaluation, alongside with the VS score, is presented and analysed.

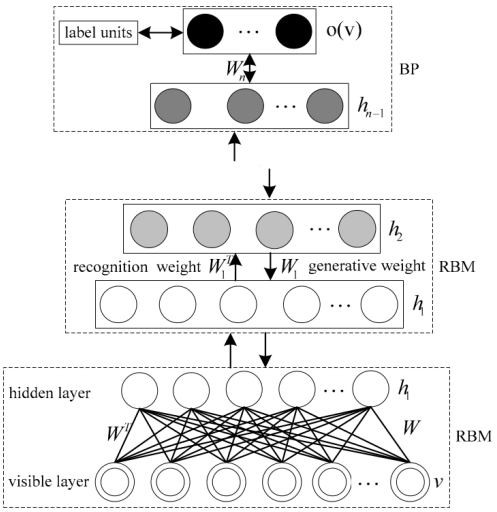
## 5.1 Model Evaluation

### 5.1.1 Objective and setup

The NIDS models described in Section 3.2 were trained and evaluated using the three datasets described in Section 3.3. The two multiclass classifiers, (AEDNN and DBN), were evaluated on the CICIDS2017, the CIDDS-001, the original NSL-KDD and the modified NSL-KDD dataset. The preprocessing steps, elaborated in Section 3.3.4, were performed for all three datasets. The anomaly-based classifier (AE) was evaluated on the CICIDS2017, the CIDDS-001 and the original NSL-KDD dataset. For this model, no over-/undersampling was performed. Furthermore, the evaluation metrics of all models were optimized using a trail-and error approach.
The architecture and parameters, that yielded the best results, are listed in Table 9 in the Appendix.
As a contribution of this thesis, a framework was developed to enable evaluation and testing of NIDS against adversarial examples. The proposed framework was implemented using the programming language *python 3* [46], with the modules *keras* [47] and *tensorflow* [48].
In addition, the sklearn [49] framework was used to derive performance metrics (see Section 3.2.5). More specifically, the functions *confusion_matrix*, *accuracy_score*, *precision_score* and *recall_score* were used. In the following sections, the parametrization of the above-mentioned models are described in more detail.

### 5.1.2 Deep Belief Network

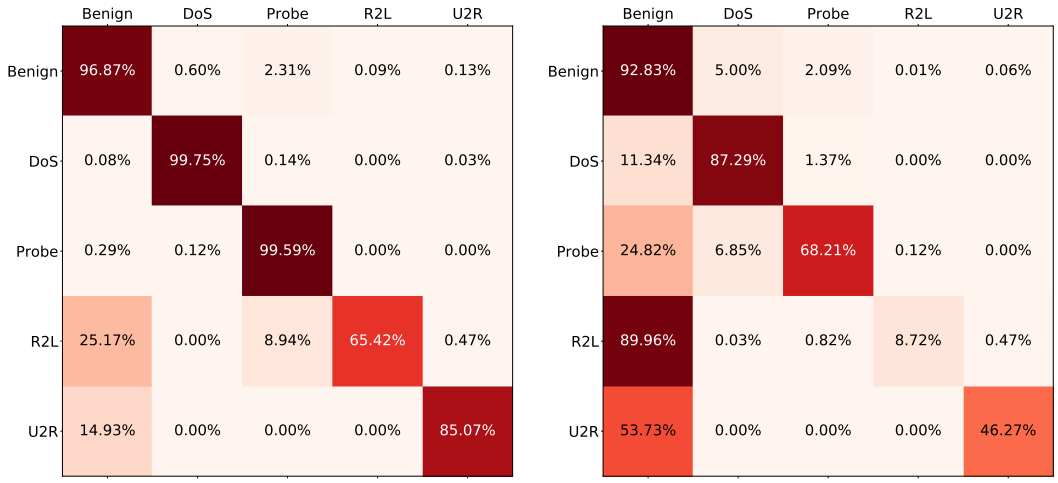
In the implementation of the Deep Belief Network, the model and parameters proposed in Gao *et al.* [19] were used.
Figure 12 depicts the architecture of the model, which was developed for the KDD99 dataset. In this thesis, a publicly available framework [50], which implements the DBN proposed by Hinton *et al.*[18], was utilized.




A two-RBM-layer model was trained for all three datasets. Table 9 (Appendix Section 7.1) depicts the architectures chosen for each dataset. For the modified NSL-KDD dataset, an overall accuracy of 0.985 was achieved, respectively 0.764 for the original dataset. These results are aligned with 0.935 accuracy reported by Gao *et al.*[19], as the testset of the KDD99 dataset is less challenging than the NSL-KDD testset. The model yielded an accuracy of 0.993 for the CICIDS2017, and 0.989 for the CIDDS-001 dataset. The resulting confusion matrix for the two variants of the NSL-KDD dataset can be seen in Figure 13, and the confusion matrix for the other two datasets in Figure 14.

Figure 12: DBN architecture *Source:* [19]

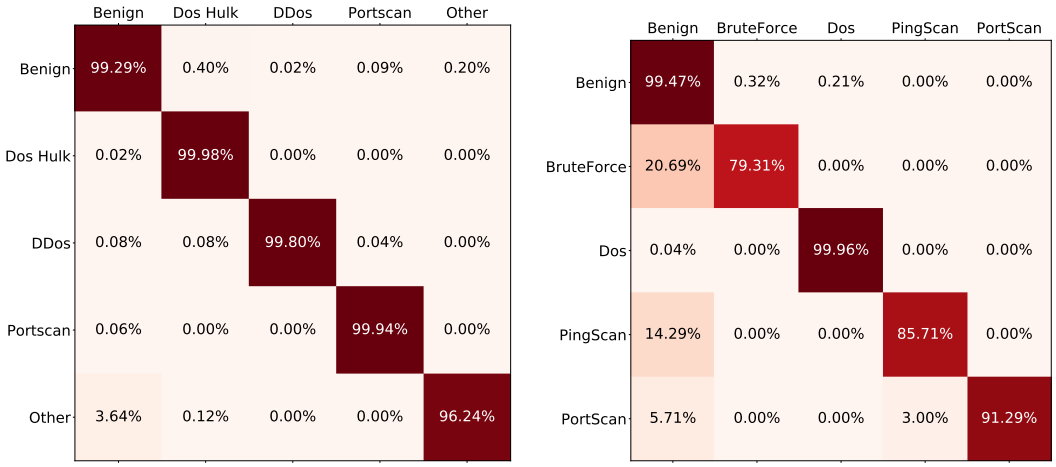
For the visualization of the CICIDS2017 dataset, all labels with less than 200 instances were put together and denoted as 'Other'.




(a) Confusion Matrix NSL-KDD* DBN (b) Confusion Matrix NSL-KDD DBN

Figure 13: Normalized confusion matrices DBN for NSL-KDD dataset

As expected, the models performance of the *R2L* and *U2R* label is rather poor, as depicted in Figure 13b. After merging and re-splitting, however, the results show a significant improvement. Furthermore, the results of the CICIDS2017 and the CIDDS-001 dataset are sufficiently good. In Table 6, a more detailed analysis of the results is listed.




(a) Confusion Matrix CICIDS2017 DBN (b) Confusion Matrix CIDDS-001 DBN

Figure 14: Normalized confusion matrices DBN for CICIDS2017 and CIDDS-001 dataset

#### 5.1.3 Deep Neural Network

Figure 15 depicts the architecture of the model proposed by Rezvy *et al.* [22]. The same basic architecture was implemented for the CICIDS2017, the CIDDS and the NSL-KDD model (classification and detection), but with a different layer dimension of the classifier's hidden-layer (61 instead of 66 neurons depicted in Figure 15). For all datasets, the hidden layer of the AE was set to be $\frac{d_i}{2}$, $d_i$ being the input dimension (for example 122 for the NSL-KDD dataset). Fifteen fine-tune epochs were performed for all datasets. The parameters were optimised with a trail-and-error approach. The python I framework was used for the implementation of the AEDNN model. The *binary-crossentropy* was used as the loss function and the Adam algorithm for the objective optimization. The architecture and parameters are summarized in Table 9 (Appendix, Section7.1).

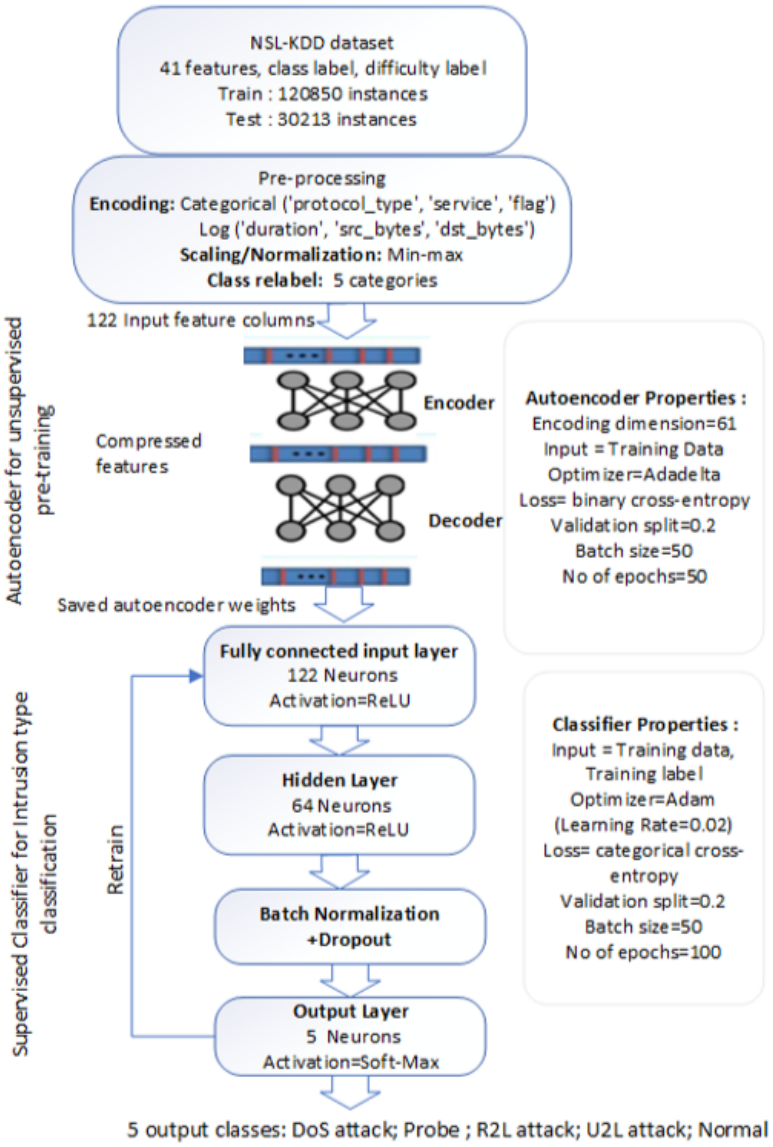



Figure 15: DNN architecture *Source:* [22]

Table 3 compares the accuracy per label achieved by the algorithms implementation in this project, and the results reported in Rezvy *et al.*[22]. As expected, the undersampling decreases the accuracy for the *normal* label, therefore, the overall-accuracy. However, this technique significantly improved the results of the *Probe*, *R2L* and *U2R*.

| | Normal | Dos | Probe | R2L | U2R | Overall |
|---|---|---|---|---|---|---|
| **Rezvys Model [22]** | 0.996 | 0.999 | 0.984 | 0.943 | 0.892 | 0.993 |
| **Reproduced Model** | 0.989 | 0.996 | 0.995 | 0.979 | 0.955 | 0.991 |

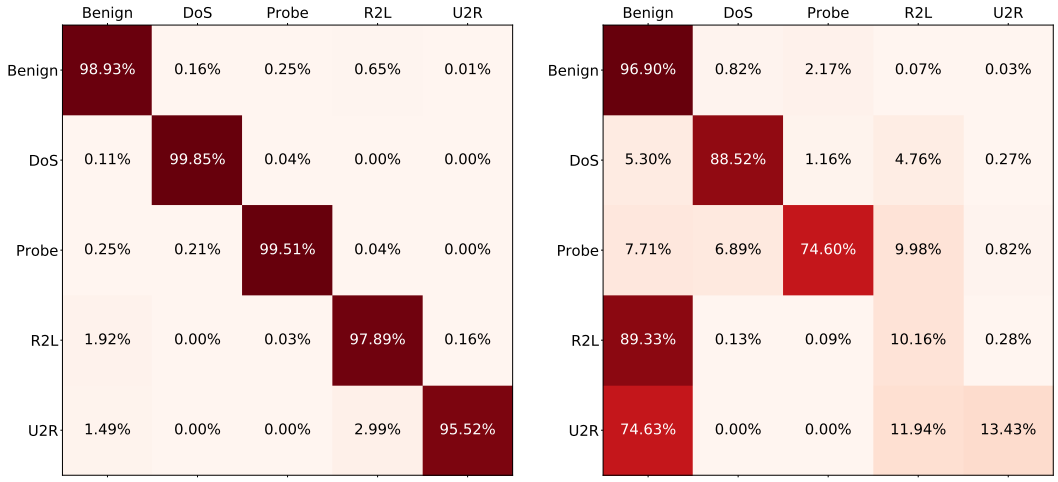
Table 3: Accuracy per label achieved by the AutoEncoded DNN on NSL-KDD* dataset

The key difference between the proposed model in [22] and a regular DNN are the first two training stages. In [22] the model is initialized utilizing an AE. To investigate the benefits and shortcomings of this approach, the algorithm was compared to DNN with the exact same architecture. The latter one is trained in only one stage, in a supervised manner, with given subset of the training samples S1. The extended AEDNN is pre-trained in the first unsupervised training stage, with a subset S2 and then trained in supervised way with the S1 set. Both methods were trained with the same number of epochs for the supervised stage. The impact of different sizes of supervised training set (S1) on the overall detection accuracy were evaluated. The results are listed in Table 4 For the evaluation, S2 was set to be 50% of the NSL-KDD* trainset and the full S1 (100% in table 4) the complementary 50%.

| % of S1 | **100%** | **50%** | **10%** | **1%** |
|---|---|---|---|---|
| **Regular DNN** | 0.99 | 0.99 | 0.97 | 0.78 |
| **Auto encoded DNN** | 0.99 | 0.98 | 0.98 | 0.91 |

Table 4: Overall accuracy of full and reduced NSL-KDD* S1 training set

It seems, that for a large enough supervised training phase, the AEDNN does not provide better performance than a regular DNN. However, by reducing the S1 set, one can see the benefit of the pre-training. For classification tasks with a limited amount of labelled input-output samples, the AEDNN algorithm might improve the performance, however, if a sufficient amount of data is available, a one-stage training might even perform slightly better.
Figure 16 depicts the normalized confusion matrix of the classifier, trained on the NSL-KDD*-set (16a) and the original NSL-KDD-set (16b). As for the DBN, the lack of samples with label *R2L* and *U2R* is reflected by the poor performance of the model trained on the original set. However, the results for the modified dataset are remarkably better than the DBN results for the same dataset.

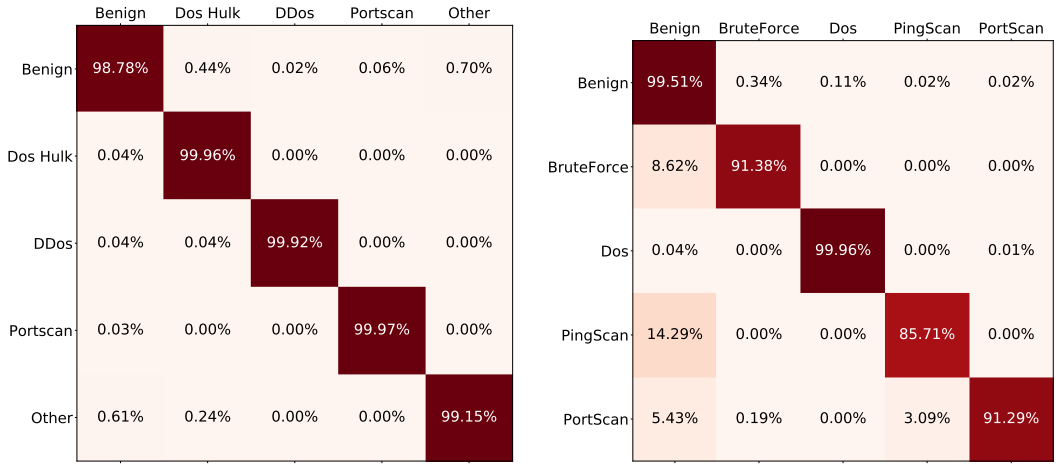



(a) Confusion Matrix NSL-KDD* (modified) AEDNN

(b) Confusion Matrix NSL-KDD (original) AEDNN

Figure 16: Normalized confusion matrices AEDNN for NSL-KDD dataset

The resulting confusion matrix for the CICIDS2017 and the CIDDS-001 dataset can be seen in Figure 17. Again, all labels with less than 200 instances were aggregated as *Other*.




(a) Confusion Matrix CICIDS2017 AEDNN

(b) Confusion Matrix CIDDS-001 AEDNN

Figure 17: Normalized confusion matrices AEDNN for CICIDS2017 and CIDDS-001 dataset

The results are pretty similar to the DBNs results, with a slightly better performance on the NSL-KDD* dataset.

### 5.1.4 Outlier Detection

Although Hawkins *et al.*[35] used the KDD99 dataset for evaluation of his algorithm, their results can not be directly compared with the implementation proposed in this work, due to differences in features pre-processing stage. Instead of using one-hot encoding, the authors [?] grouped the dataset by categories and evaluated the model for each category separately. They concluded, that for most categories, the records with the highest reconstruction error are instances with an attack label. For this thesis, their evaluation was extended to three datasets and the accuracy was measured. In the implementation of this thesis, parameter $\sigma$ in equation (3.3) was set as the standard derivation of the outlier factors of the train-set. The hidden-layer dimension and the percentile were optimized with a trail-and-error approach. As this model only trains on benign samples, no over- nor under-sampling of the datasets was performed. Table 5 depicts the results of the evaluation, alongside with the model architecture (dimension of hidden layers) and the threshold (percentile) used for all three datasets.

| | **Layer-Dimensions** | **Percentile** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|---|---|---|
| **CICIDS2017** | 78-4-78 | 75 | 0.770 | 0.455 | 0.831 | 0.588 |
| **CIDDS-001** | 29-3-29 | 83 | 0.840 | 0.524 | 0.952 | 0.675 |
| **NSL-KDD** | 122-20-122 | 80 | 0.863 | 0.862 | 0.904 | 0.883 |

Table 5: Results and hidden layer dimension Outlier Detection

Surprisingly, the Outlier detection outperformed the AEDNN on the original NSL-KDD testset (as can be seen in Table 5). The fact that the diversity in this testset is higher seems to increase the performance of the distance-based anomaly detection. By evaluating the benign-samples of the test-set, an accuracy of 0.809 is achieved, and 0.904 for the anomaly-samples. The results for the benign sample reflect, as expected, the 80% percentile.
Choosing the threshold is a trade-off between the False-Alarm-Rate (FAR) and the False-Negative-Rate (FNR). A threshold of 95%, for example, would decrease the FAR to 5%, however, a lot more attacks would not be detected (high FNR). As 80% of the instances in the test-set of the CICIDS2017 dataset are benign samples, a high accuracy could be achieved by setting the percentile $\approx$99%, however, hardly any anomalies would be detected. To overcome this problem, the optimal threshold would result in balanced (similar) benign as well as anomaly sample accuracy. With a 75% threshold, an accuracy of 0.754 for benign samples and 0.831 for anomaly instances was achieved. This results in a low FNR, but high number of false positives (FAR), which is reflected by low precision, as depicted in Table 5. The results of the CICIDS2017 dataset are significantly lower then the results of the NSL-KDD dataset, however, a detection rate 80% of the anomalies is adequate for a unsupervised model. For the CIDDS, the accuracy on the attacks reached 95.2%. With a FAR of only 17%, these results can be considered quite impressive.

### 5.1.5 Comparative Evaluation of NIDS models

A summary of results achieved by the detection models on the three datasets are listed in Table 6. The denomination NSL-KDD is the test-set with the original train-test-split, and NSL-KDD* the test-set that has been merged, oversampled and re-splitted. The results are reasonable and aligned with the results of the original papers ([22], [19]).

| Model | Dataset | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|
| **AEDNN** | CICIDS2017 | 0.989 | 0.993 | 0.989 | 0.991 |
| | CIDDS-001 | 0.995 | 0.998 | 0.995 | 0.996 |
| | NSL-KDD | 0.794 | 0.763 | 0.794 | 0.778 |
| | NSL-KDD* | 0.997 | 0.997 | 0.997 | 0.997 |
| **DBN** | CICIDS2017 | 0.993 | 0.994 | 0.993 | 0.993 |
| | CIDDS-001 | 0.989 | 0.995 | 0.989 | 0.992 |
| | NSL-KDD | 0.764 | 0.810 | 0.764 | 0.786 |
| | NSL-KDD* | 0.985 | 0.985 | 0.985 | 0.985 |
| **AE** | CIDDS-001 | 0.840 | 0.524 | 0.952 | 0.675 |
| | CICIDS2017 | 0.770 | 0.455 | 0.831 | 0.588 |
| | NSL-KDD | 0.863 | 0.862 | 0.904 | 0.883 |

Table 6: Results NIDS models

The AEDNN yields slightly better results than the DBN model for all three datasets. Both models have the advantage of unsupervised pre-training steps to improve the performance.The F1-score of the AE model appear to be fairly poor, however, as discussed in Section 5.1.4 the objective was to optimize the FNR. As the testset consists of over 80% benign samples, this objective leads to a poor F1 score, but to a 80% detection rate. To conclude, in this thesis a sample of deep-learning algorithms for NIDS was reproduced. The Accuracy and F1-score (Table 6 prove that these models achieve a sufficient performance. As the research interest in this field is high, there are new techniques, like Recurrent Neural Network (RNN) based [33], proposed which outperform previous models. However, the goal of this work was to craft adversarial examples against representative classifiers from the field, which can achieve decent accuracy. As the classifiers show significantly better results for the merged and re-shuffled NSL-KDD* set, than for the original NSL-KDD, these models (except for the Auto Encoder) were used in the following Sections in order to challenge the attack algorithm. The aim to get comparable results for the datasets only makes sense if the classifiers achieve similar results.

## 5.2 Attack Evaluation

Using the implemented NIDS models, adversarial example attacks will be investigated. This Section will be structured in five parts. Before the attacks against the NIDS models are launched, the grouping of features described in Section 4.1 is done for the three considered datasets (NSL-KDD, CICIDS2017, CIDDS-01). Next, the attack algorithms described in Section 3.4 are compared with the proposed version of the C&W algorithm (Section 5.2.2). This comparison aims to showcase the characteristics of the different algorithms .
Then, inter-model transferability is presented (Section 5.2.3). As mentioned before, the threat model considered in this work is to launch a stealthy attack on a NIDS. The idea is to add perturbations to an instance with an anomaly label (true and predicted), such that it is misclassified as benign. To justify that assumption, transferability between the NIDS models and more importantly, transferability from a substitute model to the original models, is investigated.
Subsequently, the success rate of the proposed adversarial example attack on the original NIDS-models is evaluated (Section 5.2.4). This evaluation is focused on finding the highest success rate possible for a given budget.

Several attacks were simulated, with various feature- and perturbation -budgets, and the best success rate recorded.
Finally, the results from Section 5.2.4 are used to compute the VS in Section 5.2.5.

### 5.2.1 Grouping

In Table 7, the grouping of features are presented for the datasets considered in evaluation. There are 78 (CICIDS), 29 (CIDDS) and 122 (NSL-KDD) features in the sets, only the general categories and a few examples of the grouping are listed.

| Dataset | Group (0) | Group (1) | Group (2) | Group (3) |
|---|---|---|---|---|
| **CICIDS 2017** | Flows in bwrd dir<br>Features considering bwrd flows<br>Total: 46 features | Flows in fwd not used for calculation of mean values<br>'Fwd Packet Length Max',<br>...<br>Total: 7 features | Flows in fwd direction used for calculation of mean values<br>Total: 4 features | IAT-features<br>Mean values of flow in fwd direction<br>Total: 21 features |
| **CIDDS-001** | Categorical Features<br>Total: 26 features | Duration, Packets and Bytes<br>Total: 3 features | - | - |
| **NSL-KDD** | Categorical features<br>Binary features<br>'src_bytes' (bytes send from source to dst)<br>Total: 92 features | 'Duration<br>'dst_bytes' (bytes from dst to source),<br>...<br>Total: 5 features | Counters used to compute frequency based features<br>'count',<br>...<br>Total: 5 features | Frequency based features<br>'Serror_rate' (% of connections with the 'flag' feature aggregated in 'count')<br>Total: 20 features |

Table 7: Features of datasets grouped into categories by their accessibility

Overall, 32 out of 78 (41%) for the CICIDS2017 set, 3 out of 29 for the CIDDS-001 and 30 out of 122 (25%) features were considered for the NSL-KDD set as accessible features. The number of considered features has a direct impact on the previously mentioned feature-budget parameter. The upper bound on the f-budget is defined by the number of features which are feasible to change. In other words, the f-budget equal to 1 would mean that all feasible features are used. Note that the set of accessible features of CIDDS-001 only consists of 3 features. The evaluation of this dataset will demonstrate, whether or not an attacker can fool a NIDS by only increasing/decreasing the duration of the flow, the number and size of packets sent.

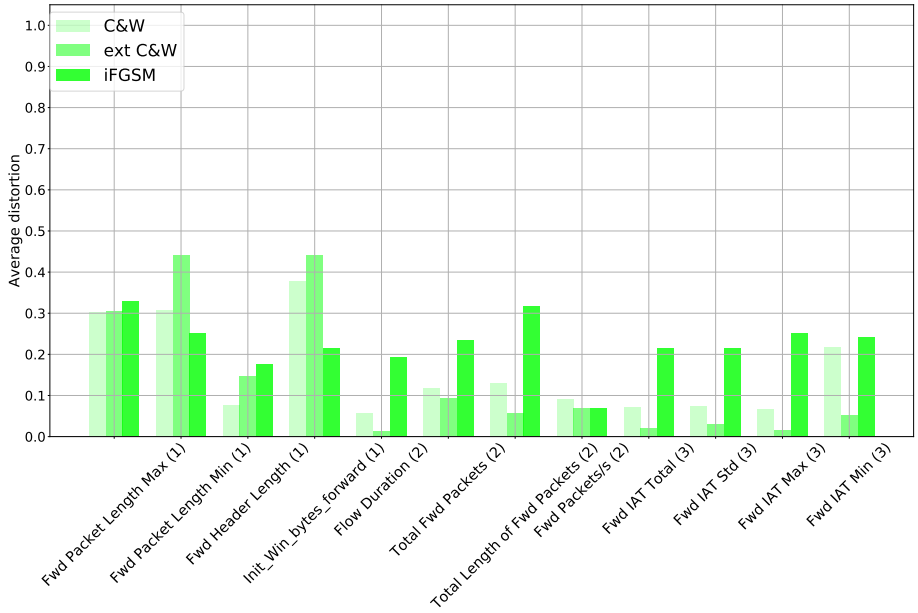
### 5.2.2 Comparative evaluation of the attack algorithms

The first part of the evaluation aims to showcase the impact of the weights in equation (4.1). The AEDNN classifier of the evaluation trained with the CICIDS2017 dataset was attacked. The label *DDoS* was used to attack. No maximum distance constraint was applied, as this restriction does not influence crafting algorithm itself. The following three different algorithms were investigated: Regular C&W [2], proposed extended C&W attack and iterative FGSM [42]. Two different c-values (see equation (4.1)) were used, and the average distortion for the successful adversarial examples for each feature and algorithm, was stored.

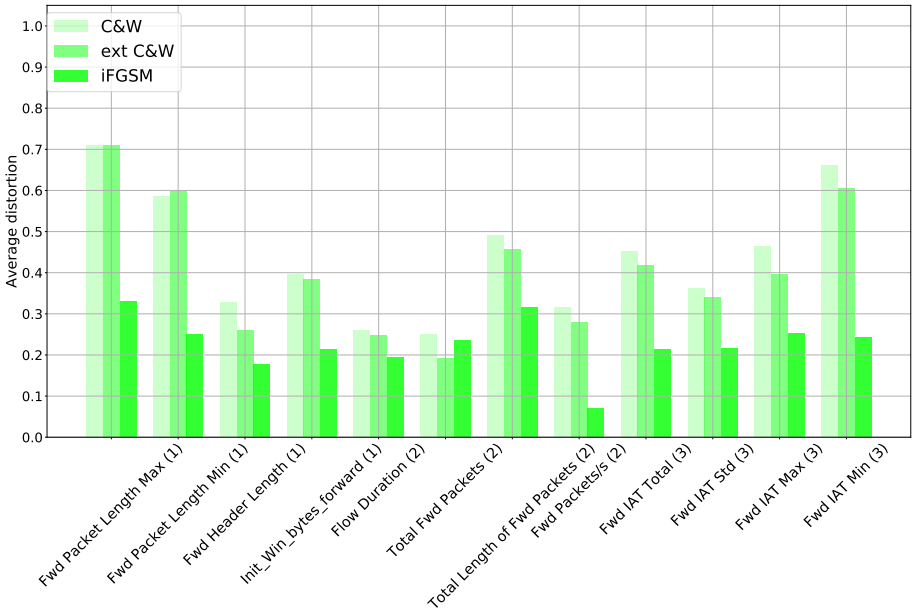
As considered features, four arbitrarily chosen features of each group were picked. The f-budget is computed using equation (4.3), with the grouping depicted in table 7 as follows:

$$f = \frac{\text{number of considered features times their weight}}{\text{number of accessible features times their weight}} = \frac{4 \cdot (1+2+3)}{7 \cdot 1 + 4 \cdot 2 + 21 \cdot 3} = \underline{0.31} \tag{5.1}$$

This adds up to an f-budget of 0.31. In Figures 18a and 18b, the results of the *DDos* attack of the CICIDS2017 dataset are depicted. The number in parenthesis indicate to which group the feature belongs. The results for the iFGSM were only computed once and used in both plots. As mentioned before, the hyper-parameter c is a trade-off between distance and target-function minimization. A low c-value (here c=1 in figure 18a) pushes the algorithm to enforce a small distance over the desired misclassification. With this configuration, the weights of the extended C&W algorithm strongly influence the results. Comparing the results of the original and extended C&W algorithms, it can be observed that the algorithm favours features of group 1 over group 2 and 3. In Figure 18b this influence can still be seen, however, not as apparent as in Figure 18a. Those two figures also visualize the advantage of being able to control the perturbation size of the C&W over the iFGSM. For an adversary who intents to attack a NIDS utilizing adversarial examples, it is vital not just to restrict the featurespace to accessible features, but also to avoid larger perturbations to certain features, while keeping the overall perturbation as small as possible. The results suggest that this goal can be achieved with the proposed C&W approach. The weighted algorithm managed to keep the changes in IAT features (group 3, important for *DoS* attacks) low, which supports maintaining the impact of the attack. In the experiments, weight values from 1 to 3 (1:2:3) were assigned, however, the impact depicted in figure 18a could be amplified by using different weight-ratios. Carlini and Wagner developed their algorithm for computer vision tasks. In [2] they compared the success rate of a targeted adversarial example attack, utilizing the C&W and other state-of-the-art crafting algorithms, including the iFGSM, against a DNN model. The authors observe a success rate of 100% for the C&W and iFGSM algorithms, evaluated on 3 different computer vision datasets. However, their C&W attack achieved a 2 to 10 times lower distortion than the iFGSM. With the weight-extension, the C&W algorithm can also be used in the NIDS domain, as with this modification the distortion is restricted to easy to access features.




(a) Average distortion c=1

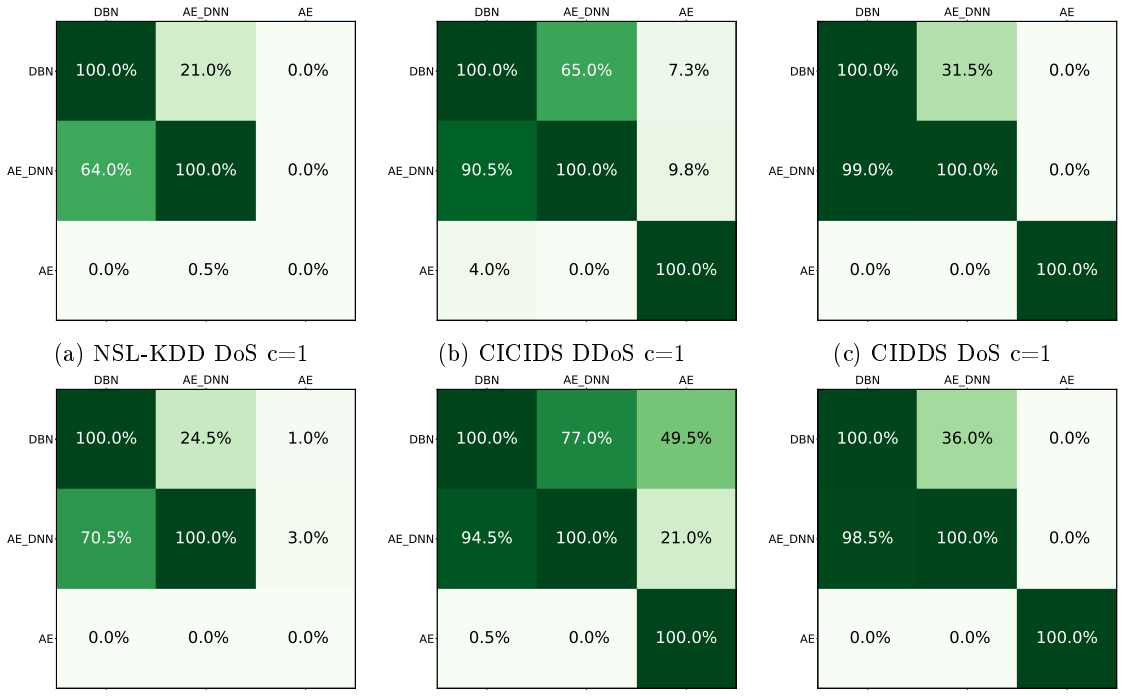



(b) Average distortion c=10

Figure 18: Average distortion per feature AEDNN model with (a) c=1 and (b) c=10

### 5.2.3 Transferability

This section aims to answer two questions about the transferability:
(1) To what extent adversarial examples can be transferred between two classifiers trained to perform the same task and (2) how effective adversarial examples crafted for a substitute model are on the original model.
To investigate the first question, 200 adversarial samples were crafted for each classifier (AE, DBN and AEDNN) for two attack-labels of each dataset, utilizing the extended C&W algorithm ($f_5$ function), solved with the Nadam-Optimizer ($\alpha = 0.02$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). This evaluation was done twice, with c=1 (equal weight on distance and confidence optimization) and c=10 (bias towards confidence optimization). For the second c value, one can expect larger distances, and higher confidences for the misclassification. To get an intuition of the impact of this hyperparamer, Figure 20 depicts the average perturbation per feature for the two c values for the DBN model.
Figure 19 depicts the results for the *DoS* attack with a colormap, where columns represent the classifiers that the samples were crafted for and rows the evaluated classifiers. For example, the samples of the NSL dataset with c=1 crafted for the DBN achieved a 64.0% success rate for the AEDNN. It is worth to note, that the compared models are trained with the same training-set to perform the same task. The colormaps for the other attacks (*PortScan* and *Probe*) are depicted in Figure 25 (Appendix Section 7.2).




(a) NSL-KDD DoS c=1 (b) CICIDS DDoS c=1 (c) CIDDS DoS c=1

(d) NSL-KDD DoS c=10 (e) CICIDS DDoS c=10 (f) CIDDS DoS c=10

Figure 19: Transferability success DoS attack

Analysing the results, one can observe the signature-based classifiers and the distance (anomaly) based AE use different patterns to classify, as samples hardly transfer at all between those two types. As for transferability between the two signature-based models, a rather high success rates can be observed, especially for the CICIDS2017 dataset, where a success rate up to 99% for DBN to AEDNN has been achieved. Surprisingly, the other direction (AEDNN to DBN) has a significantly lower success rate. This indicates, that the AEDNN, despite having achieved a higher accuracy in Section 5.1.5 than the DBN, is less robust with respect to adversarial examples.
Assuming the knowledge of the underlying model of the target NIDS, it is possible to craft adversarial samples indirectly and attack the system. The key factor is to know the approach used for detection. Signature-based models appear to be vulnerable against transferred samples of other signature-based models, however, not as vulnerable against samples crafted for distance-based models.
Figure 20 depicts the average perturbation per features for the two c-values of the three models for the *DoS* attack type. Except for the AE models result on the CIDDS dataset, the higher c value provides a larger average distance per feature.

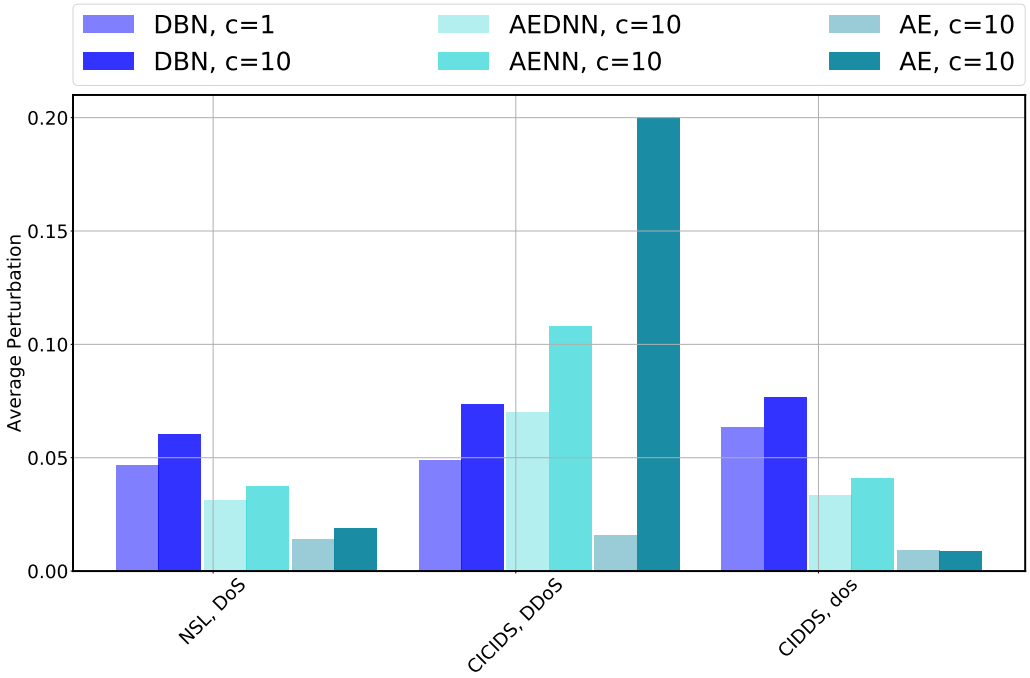



Figure 20: Average perturbation per features DBN model

The second part was approached as follow (for each NIDS model as target model):

1. Train a substitute Model
   With the test-set of each dataset, labels were generated utilizing the predictions of the target model. Then, the substitute model was trained with the test-set as input and the predictions as labels.

2. Craft Adversarial Samples
   Perform a white-box attack on the substitute model. Utilizing the extended C&W algorithm (equation (4.1)), 200 adversarial samples were crafted per label. For this evaluation, only successful adversarial examples were considered, meaning instances which are classified as benign by the classifier after adding the perturbation. Additionally, only instances which were classified correctly at the beginning are considered. To see the impact of the trade-off parameter c (see equation (4.1)), different c-values were evaluated.

3. Transfer the Samples back to the target model
   Evaluating the relative amount of samples classified as benign by the target model.

The threat model assumes white-box knowledge of the NIDS, which is not very realistic in a real world scenario. However, with only access to input-output samples of the system, a substitute model can be trained and used to attack the system. To train this model, samples from the test-set were used, together with the prediction of the original classifier (target model) as labels.
A DNN with the same architecture and parameters for each classifier was trained. All values and parameters can be found in the Appendix. Evaluated against the predicted labels of the original classifier on the test-sets, all transfer models achieved over 97% accuracy.
As in the first part, 200 adversarial samples were crafted for two different c-values, for each transfer model. Then, the relative amount of samples which are classified as benign were evaluated by the target model. Table 8 depicts the results of this evaluation.

| Model | c | NSL DoS | NSL Probe | CICIDS DDoS | CICIDS PortScan | CIDDS | CIDDS |
|---|---|---|---|---|---|---|---|
| DBN | 1 | 0.34 | 0.44 | 0.90 | 0.83 | 0.75 | 0.95 |
| | 10 | 0.46 | 0.42 | 0.90 | 0.86 | 0.87 | 0.91 |
| AEDNN | 1 | 0.39 | 0.42 | 0.70 | 0.88 | 0.39 | 0.34 |
| | 10 | 0.46 | 0.5 | 0.80 | 0.92 | 0.76 | 0.75 |
| AE | 1 | 0.08 | 0.1 | 0.0 | 0.0 | 0.68 | 0.15 |
| | 10 | 0.05 | 0.08 | 0.0 | 0.0 | 0.38 | 0.15 |

Table 8: Success rate Transferabilty between Transfer DNN and target model

Again, it can be observed, that transferability between the anomaly-based and the signature-based models is not very successful. So far the AE model, even though it is trained with an unsupervised algorithm using only benign samples, appears to have the highest robustness against adversarial examples. As before, rather high success rates for the DBN and AEDNN model were achieved. This observation is fairly surprising, given the fact, that the substitute DNN model is more similar to the AEDNN model. The highest success rate for transferred samples was achieved for the CICIDS2017 dataset.
Given access to the monitoring system (NIDS), it is possible to perform undetected attacks by bypassing the NIDS with adversarial examples. A transferability success rate of at least 30% for signature-based detection is for this scenario a considerable threat.

### 5.2.4 Attack on Network Intrusion Detection models

In this Section, the empirical success rate for each classifier is evaluated under various conditions. To get an overview, two labels of each dataset are used for an attack. This choice was made based on high label-accuracy, sufficient number of records and comparability. For the NSL-KDD dataset, the *DoS* and *Probe* label, for the CICIDS2017 set, *DDoS*, as it was the *DoS* type attack with the highest detection accuracy, and *PortScan*, as it is similar to the NSL-KDD *Probe* attack. For the CIDDS-001 set, the same two attack-types, *DoS* and *PortScan*, are used.

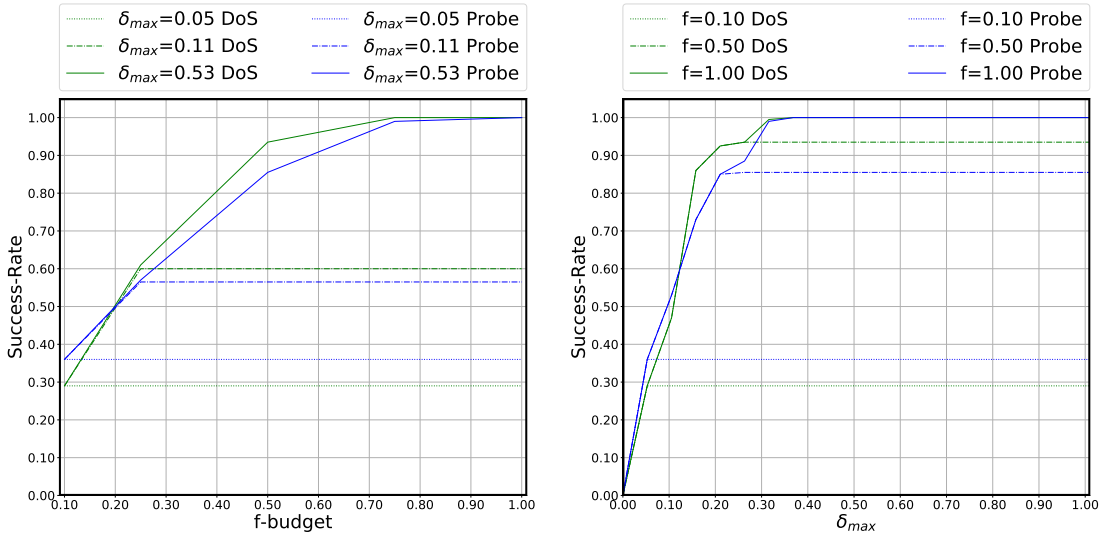
First, parameter c in equation (4.1) was set empirically with a trail-and error approach. Next, restrictions on the success rate were placed. The following list sums up the restrictions under which the attacks were evaluated:

- An attack is considered successful, if an attack-instance, that was predicted correctly with at least 80% confidence, was transformed by adding a perturbation $\delta$ to an instance that is classified as benign
- The average weighted perturbation per feature must be smaller than a predefined $\delta_{max}$
- To incorporate the weights, the perturbation of each feature was multiplied with the corresponding group-weight of the features ($\delta = \text{avg}(\nu \cdot (\text{adversarial - original sample}))$). By doing this, perturbations of group 3 features have higher influence on the average perturbation than group 1 features.
- Only features in the *considered features* list can be changed (group 1 to 3)

Attacks were launched with five different f-budgets $\in [0, 1]$ (three for the CIDDS-001 dataset as there are only three features that can be accessed). The considered features (mask) were defined by randomly adding features to a list, starting by group 1, then group 2 and finally group 3, until the desired f-budget is reached. If $f = 1$, all features of group 1 to 3 were added, if $f \approx 0$, only a few features from group 1 were considered. For all classifiers, the same combinations of *considered features* were used for a given budget, to achieve comparable results. The average of the weighted absolute distances between original and adversarial sample is then calculated and compared to $\delta_{max}$. As before two different labels were used for attacking.
Figures 21, 22 and 23 depict the success rate plotted against the f-budget (21a, 22a and 23a) and against the maximum distance $\delta_{max}$ (21b, 22b and 23b) for the models trained on the NSL-KDD* dataset. The results of the other two datasets can be found in the Appendix (Section 7.3 figures 26 to 31). Each plot depicts the resulting success rates for three different $\delta_{max}$ and three different f-budgets, respectively.
Analysing the righthandside of those plots, one can see that for almost every attack, there exists a threshold-distance, that includes all adversarial samples. Further increase of the distance does not improve the success rate anymore. For most attacks, this threshold is at $\delta \approx 0.2$, which denotes an average weighted change per feature of 0.2. Surprisingly, this is also true for the CIDDS dataset, even though only 3 out of the 29 are considered as feasible features.
Furthermore, the success rates are also dependent on the f-budget utilized for the attack, as one can see in plots. The left side of the plots depict the success rate plotted against the f-budget. Note that for the CIDDS dataset only three different budgets are possible. As expected, the success rate grows with the budget. A threshold for the f-budget, similar to one for the distance restriction ($\delta$-budget), does not seem to exist.

Over all three datasets, the CICIDS2017 yields the highest success rates for all models. As 41% of the features in the CICIDS2017 dataset, and only 25% of the NSL-KDD dataset (see Section 5.2.1) are considered accessible, this result is reasonable. The fact that the CICIDS2017 datasets uses several features related to 'packages in forward direction' for classification increases the risk of an adversary managing a stealthy attack. As the CIDDS-001 dataset uses only 3 features for detection, that can be realistically accessed by an attacker, the attacks on this dataset appear to be weaker.

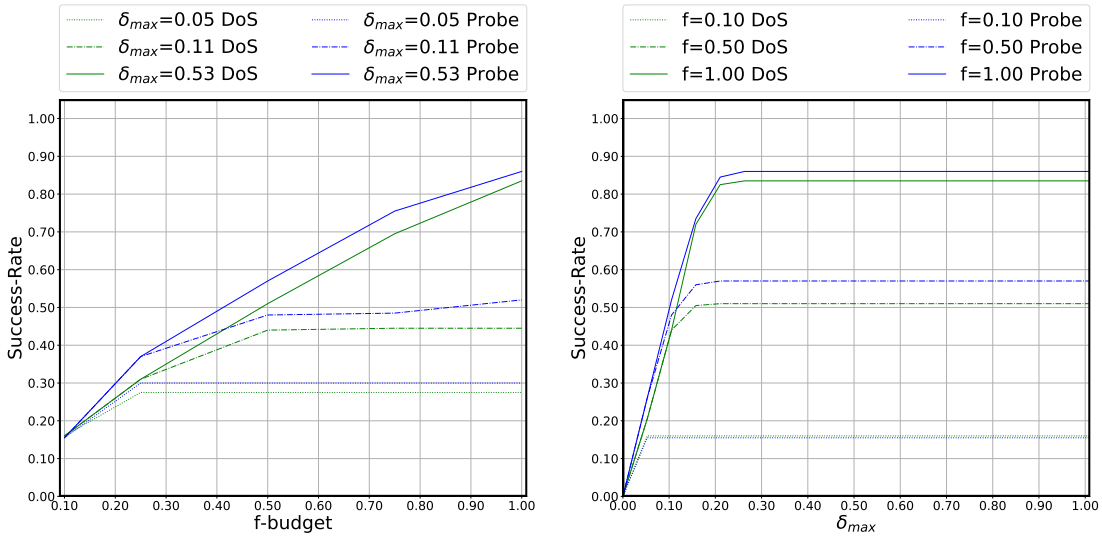
Considering the different models, it appears that the outlier detection (AE) is by far the most robust model against adversarial example attacks. It can be observed that it is not possible to find a lot of adversarial instances for low budgets. The results for the DBN and the AEDNN are comparable, the AEDNN being slightly more robust for most attacks.




(a) DBN success rate vs. f-budget NSL

(b) DBN success rate vs. $\delta_{max}$ NSL

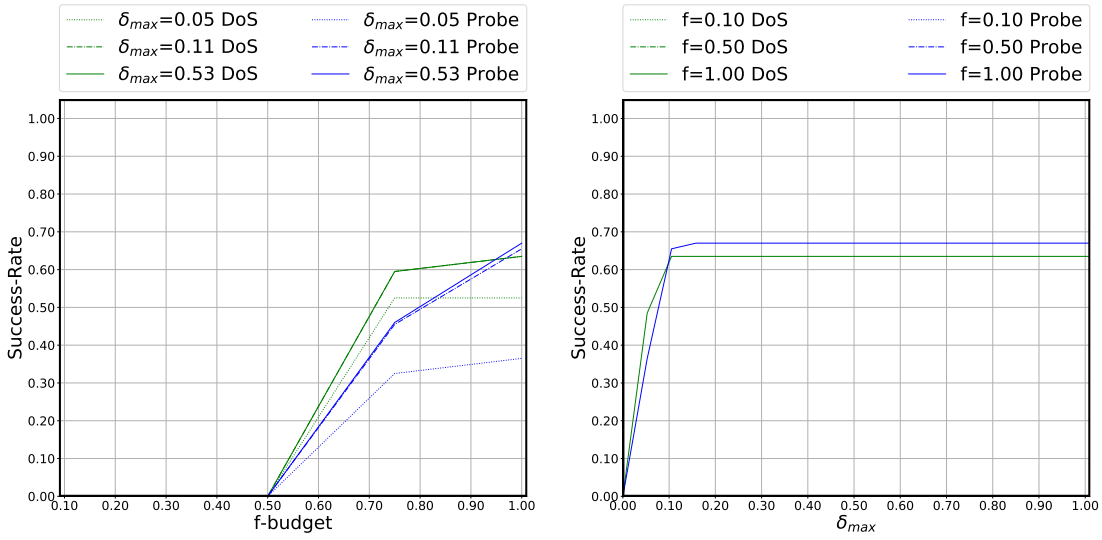
Figure 21: Success rate DBN NSL-KDD dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AEDNN success rate vs. f-budget NSL

(b) AEDNN success rate vs. $\delta_{max}$ NSL

Figure 22: Success rate AEDNN NSL-KDD dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AE success rate vs. f-budget NSL

(b) AE success rate vs. $\delta_{max}$ NSL

Figure 23: Success rate AE NSL-KDD dataset plotted (a) against f-budget (b) against $\delta_{max}$

### 5.2.5 Vulnerability Score

In the previous Section, the results of the success rate of the attacks against the models were presented and discussed. This Section aims to show, how the proposed metric (Vulnerability Score described in Section 4.3) reflects the robustness of a classifier. The Vulnerability Score (VS) should give an intuition, of how vulnerable the considered NIDS classifiers are to an adversarial example threat. For evaluation, a reasonable value for $\delta_{max}$ should be chosen. Standard deviation of each feature or similar measures could be used here. It is worth to mention that this score does not require normalized data, because the size of the distortion is not included in the calculation. Figure 24 depicts the highest achieved Vulnerability Score for the $\delta_{max}$ values 0.05, 0.11 and 0.53. As before, a weighted average distance was used.

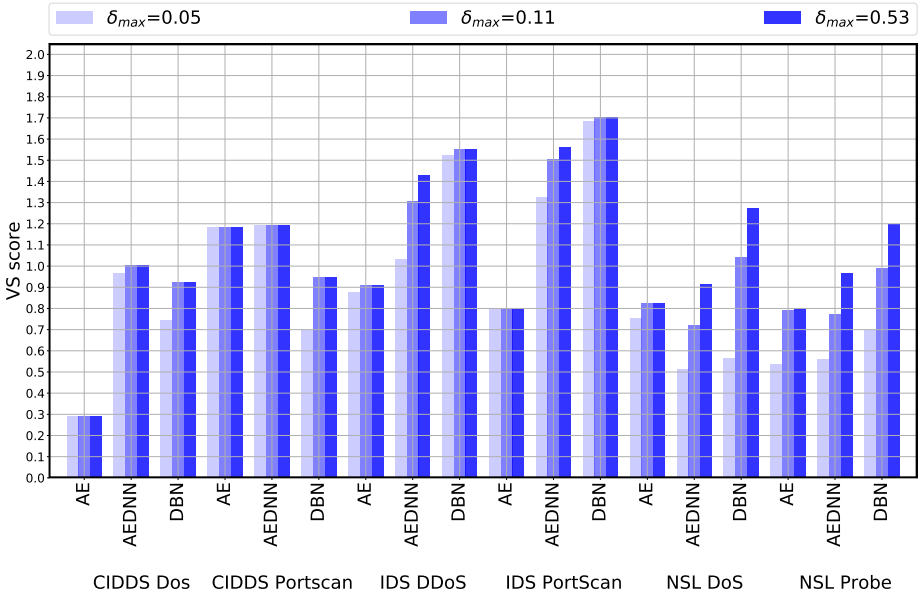



Figure 24: Vulnerability Score for all classifiers using the attacks from CIDDS, CICIDS, and NSL-KDD dataset

The results of this score reflect the comparatively high success rates for the CICIDS2017 dataset, as well as the advantage of the outlier detection. One can observe, that there is hardly any difference for the 0.11 and 0.53 $\delta_{max}$ restriction. Quite a few model-attack combinations reached a VS above 1, indicating a rather high success rate for low budgets, e.g. DBN on CICIDS2017 *PortScan*. Under the assumption of an possible average perturbation per feature of 0.05, the classifications models are more robust against the *DoS*, than against the *Probe*/*PortScan* attacks. In the proposed feature grouping, IAT-features and frequency based features are considered as group 3 features (i.e. features that are hard to access by an attacker). Considering using a lot of features of this group would increase the f-budget and therefore decrease the VS. These types of features, however, are characteristic for a Denial of Service attack, which results in a low VS for these attacks. The VS for the CIDDS dataset (except for the AE model) are considerably high, as only 3 out of 29 features are considered feasible.

Given that the lowest possible budget is $\frac{1}{3}$, the highest achievable score is 1.5. By far the highest score was achieved by the DBN model for the *PortScan* attack of the CICIDS2017 dataset.
The purpose of the Vulnerability Score is to get a theoretical estimation of the risk of an adversarial threat. With knowledge of the features properties used for classification, one can group them according to their accessibility and estimate a possible perturbation per feature to estimate the threat of an adversarial example attack. Achieving the highest possible VS is related to a knapsack-problem, as the optimal combination of features that leads to a high success rate with a low f-budget is not known beforehand and can not be determined exactly without a brute-force approach. However, simulating all possible combinations is computationally not feasible, one might perform a feature-sensitivity evaluation ([33]) to estimate a promising combination.

# 6 Conclusion

In this thesis, a framework was developed to enable evaluation and testing of NIDS against adversarial examples, including a novel approach to crafting adversarial examples, which accounts for domain-specific constraints. To create an adversarial example, the algorithm only considers features that are accessible, i.e. possible to modify without violating the validity of input data. For evaluation, weights were assigned to features to reflect the difficulty of their modification.
Furthermore, a metric has been proposed to assess the vulnerability of NIDS models with respect to relation between success rate of an attack and difficulty of its implementation (constraints). The success rates achieved by adversarial examples presented in this thesis, demonstrate credibility of this type of threat in the NIDS domain. It has been shown that a classifiers can be fooled with high success rate ($\approx 98\%$) even at a low budget.
To showcase the possibility of a black-box attack against a NIDS, the phenomenon of transferability has been utilized. Two approaches were used to investigate transferability of the adversarial examples. In the first one, samples are transferred between different models trained on the same dataset. It has been demonstrated that from 21%, up to 99% of adversarial examples can be transferred from one NIDS model to another. The second approach evaluates transferability between a NIDS model and a substitute model, trained with input-output samples. Around 90% of adversarial instances of an *PortScan* attack (CICIDS2017) were transferable from the substitute model to the original AEDNN classifier. This observation verifies that attacks with a low budget against a black-box system are feasible and should be considered when deploying the NIDS.
To improve robustness against those attacks an ensemble-detection approach with different detection techniques could be used to decrease the risk. Since the transferability between signature- and anomaly-based was proven to be ineffective, a combination of both solutions could be very beneficial. With the increasing DL models deployed in safety-critical environments, the research of potential threats becomes vital. For computer-vision models real-world attacks have been proven possible. In a matter of time, attacks like this will be possible on various intrusion detection systems.
For future work, the realisation of adversarial examples in network traffic will be investigated. In this process, the goal is to identify further restrictions on the crafting algorithm to enable the implementation of those attacks. The goal is to derive a comprehensive framework to evaluate robustness of DL models deployed in NIDS.

# Acronyms

**AD** Anomaly Detection.

**AE** Auto Encoder.

**AEDNN** Auto-encoded Deep Neural Network.

**BP** Back Propagation.

**C&W** Carlini and Wagner $L_2$ norm Attack.

**DBN** Deep Belief Network.

**DL** Deep Learning.

**DNN** Deep Neural Network.

**FAR** False-Alarm-Rate.

**FGSM** Fast Gradient Sign Method.

**FNR** False-Negative-Rate.

**GD** gradient descent.

**IAT** inter arrival time.

**IDS** Intrusion Detection System.

**JSMA** Jacobian based Saliancy Map Attack.

**ML** Machine Learning.

**NIDS** Network-based Intrusion Detection System.

**OF** Outlier Factor.

**RBM** Restricted Boltzmann machine.

**RNN** Recurrent Neural Network.

**SGD** stochastic gradient descent.

**VS** Vulnerability Score.

# List of Figures


1 Adversarial Example: [10] . . . 4
2 Threat Model Source: [9] . . . 6
3 Stop-Sign attack Source: [1] . . . 8
4 Adversarial Robustness Source: [13] . . . 9
5 Attack on a Black-Box NIDS Source: author . . . 11
6 AE architecture Source: [37] . . . 12
7 RBM architecture Source: [19] . . . 13
8 Label distribution of NSL-dataset Source: author . . . 15
9 Label distribution of CICIDS2017-dataset Source: author . . . 17
10 Label distribution of CIDDS-dataset Source: author . . . 18
11 Example pipeline of Adversarial Example Crafting Source: author . . . 24
12 DBN architecture Source: [19] . . . 27
13 Normalized confusion matrices DBN for NSL-KDD dataset Source: author . . . 27
14 Normalized confusion matrices DBN for CICIDS2017 and CIDDS-001 dataset Source: author . . . 28
15 DNN architecture Source: [22] . . . 29
16 Normalized confusion matrices AEDNN for NSL-KDD dataset Source: author . . . 31
17 Normalized confusion matrices AEDNN for CICIDS2017 and CIDDS-001 dataset Source: author . . . 31
18 Average distortion per feature Source: author . . . 36
19 Transferability success DoS attack Source: author . . . 37
20 Average perturbation per features DBN model Source: author . . . 38
21 Success rate DBN NSL-KDD dataset Source: author . . . 41
22 Success rate AEDNN NSL-KDD dataset Source: author . . . 42
23 Success rate AE NSL-KDD dataset Source: author . . . 42
24 Vulnerability Score Source: author . . . 43
25 Transferability success Portscan/Probe attack Source: author . . . 53
26 Success rate DBN CICIDS2017 Source: author . . . 54
27 Success rate AEDNN CICIDS2017 dataset Source: author . . . 54
28 Success rate AE CICIDS2017 dataset Source: author . . . 55
29 Success rate DBN CIDDS-001 dataset Source: author . . . 55
30 Success rate AEDNN CIDDS-001 dataset Source: author . . . 56
31 Success rate AE CIDDS-001 dataset Source: author . . . 56

# References


[1] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust Physical-World Attacks on Deep Learning Models. *arXiv:1707.08945 [cs]*, 2017.

[2] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks. *arXiv:1608.04644 [cs]*, 2016.

[3] N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv:1605.07277 [cs]*, 2016.

[4] B. Dong and X. Wang. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pages 581–585. IEEE, 2016.

[5] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli. A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection. *IEEE Communications Surveys Tutorials*, pages 686–728, 2019.

[6] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pages 41–50, 2018.

[7] Jin K., Nara S., S. Y. Jo, and Sang H. K. Method of intrusion detection using deep neural network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 313–316, 2017.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, 2013.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. In *arXiv:1511.07528 [cs, stat]*, pages 372–387, March 2016.

[10] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, 2014.

[11] P. Chen, H. Zhang, Y. Sharma, J. Yi, and C. Hsieh. ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec '17*, pages 15–26, 2017.

[12] J. Lu, T. Issaranon, and D. Forsyth. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly. *arXiv:1704.00103 [cs]*, 2017. arXiv: 1704.00103.

[13] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. *arXiv:1511.04508 [cs, stat]*, 2015.

[14] M. Teuffenbach, E. Piatkowska, and P. Smith. Subverting network intrusion detection: crafting adversarial examples
accounting for domain-specific constraints. In *CD-Make Conference 2020*, 2020.

[15] Kremer S.C. Hamed T., Ernst J.B. A survey and taxonomy of classifiers of intrusion detection systems. *Daimi K. (eds) Computer and Network Security Essentials*, 2018.

[16] A. Shiravi, H. Shiravi, M. Tavallaee, and A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 2012.

[17] H. Hindy, E. Bayne, A. Seeam, C. Tachtatzi, R. Atkinson, and X. Bellekens. A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems. *IEEE Access*.

[18] G. E. Hinton. Reducing the Dimensionality of Data with Neural Networks. *Science*, pages 504–507, 2006.

[19] N. Gao, L. Gao, Q. Gao, and H. Wang. An Intrusion Detection Model Based on Deep Belief Networks. In *2014 Second International Conference on Advanced Cloud and Big Data*, pages 247–252, 2014.

[20] K. Alrawashdeh and C. Purdy. Toward an Online Anomaly Intrusion Detection System Based on Deep Learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 195–200, 2016.

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

[22] S. Rezvy, M. Petridis, and A. Lasebae. Intrusion detection and classification with autoencoded deep neural network. *Springer International Publishing*, 2018.

[23] E. Hodo, X. Bellekens, A. Hamilton, and C. Tachtatzis. Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey. *arXiv:1701.02145*, 2017.

[24] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang. Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access*, pages 35365–35381, 2018.

[25] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[26] N. Carlini and D. Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *arXiv:1705.07263 [cs]*, 2017.

[27] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–20, 2019.

[28] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On Evaluating Adversarial Robustness. *arXiv:1902.06705 [cs, stat]*, 2019.

[29] K. Yang, J. Liu, C. Zhang, and Y. Fang. Adversarial Examples Against the Deep Learning Based Network Intrusion Detection Systems. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 559–564, 2018.

[30] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, and J. Chen. Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks. *IEEE Access*, 2020.

[31] M. Hashemi, G. Cusack, and E. Keller. Towards Evaluation of NIDSs in Adversarial Setting. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks - Big-DAMA '19*, pages 14–21, 2019.

[32] T. Weng, H. Zhang, P. Chen, J. Yi, D. Su, Y. Gao, C. Hsieh, and L. Daniel. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. *arXiv:1801.10578 [cs, stat]*, 2018.

[33] A. Hartl, M. Bachl, J. Fabini, and T. Zseby. Explainability and Adversarial Robustness for RNNs. *arXiv:1912.09855 [cs, stat]*, 2020.

[34] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical Black-Box Attacks against Machine Learning. *arXiv:1602.02697 [cs]*, 2016.

[35] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier Detection Using Replicator Neural Networks. In *Data Warehousing and Knowledge Discovery*, pages 170–180, 2002.

[36] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[37] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *arXiv:1802.09089 [cs]*, 2018.

[38] M. Azami, C. Lartizien, and S. Canu. Converting SVDD Scores into Probability Estimates. *Computational Intelligence*, 2016.

[39] L. Dhanabal and S. P. Shantharajah. A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms. *Computer Science*, 2015.

[40] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho. Flow-based benchmark data sets for intrusion detection. *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*, 2017.

[41] I. Sharafaldin, A. Habibi, and A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pages 108–116, 2018.

[42] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial Machine Learning at Scale. *arXiv:1611.01236 [cs, stat]*, 2017.

[43] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li. Boosting Adversarial Attacks with Momentum. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, 2018.

[44] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, 2017.

[45] T. Dozat. Incorporating Nesterov Momentum into Adam. 2016.

[46] G. Van Rossum and F. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[47] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[50] albertbup. A python implementation of deep belief networks built upon numpy and tensorflow with scikit-learn compatibility, 2017.

# 7 Appendix

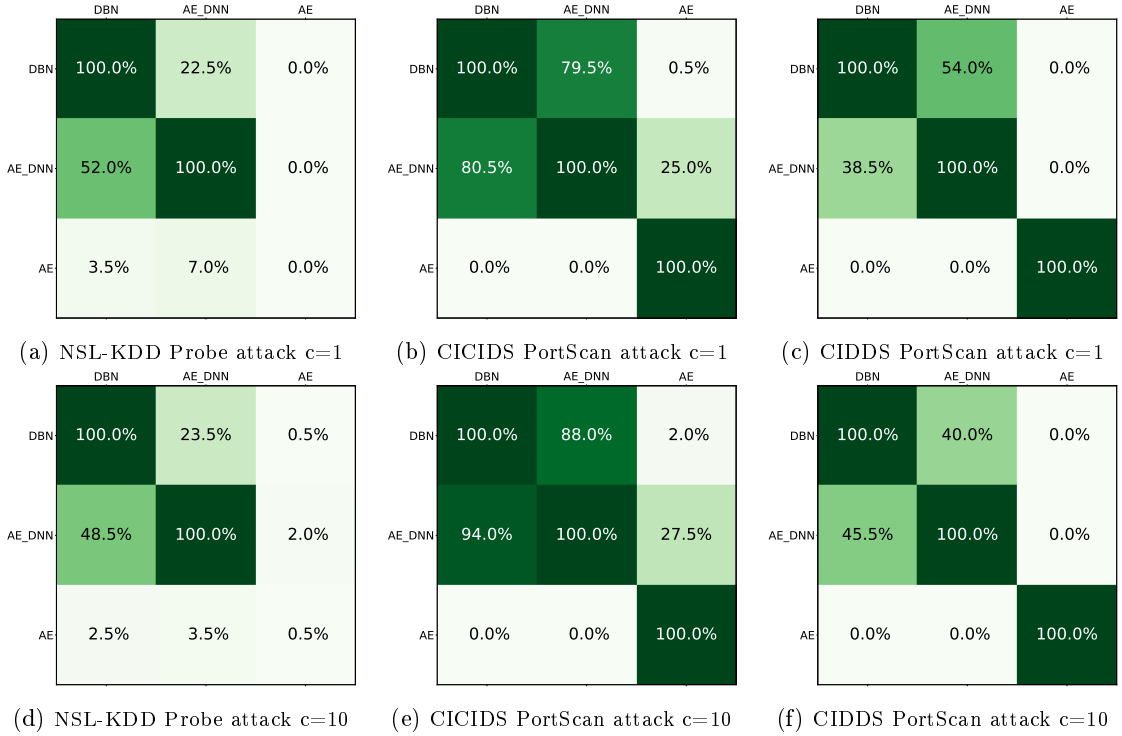
## 7.1 Model architecture

| Model | Dataset | Layer dim | Epochs | Percentile | Optimizer | Loss |
|---|---|---|---|---|---|---|
| DBN | NSL | 1-6/5-4/5 | 5-150 | - | SGD | reduce mean |
| | CICIDS | 1-6/5-4/5 | 5-150 | - | SGD | reduce mean |
| | CIDDS | 1-6/5-4/5 | 5-150 | - | SGD | reduce mean |
| AEDNN | NSL | 1-1/2-1-3/2 | 50-150-40 | - | Adadelta-Adam | binary crossentropy |
| | CICIDS | 1-1/2-1-3/2 | 50-150-25 | - | Adadelta-Adam | binary crossentropy |
| | CIDDS | 1-1/2-1-3/2 | 50-150-20 | - | Adadelta-Adam | binary crossentropy |
| AE | NSL | 1-1/6-1 | 15 | 80 | Adam | mean sqare error |
| | CICIDS | 1-1/20-1 | 6 | 75 | Adam | mean square error |
| | CIDDS | 1-1/15-1 | 2 | 83 | Adam | mean square error |
| Subst. DNN | all 3 | 1-5/3-2/3-1/3 | 200 | - | Adam | poisson |

| Model | Dataset | Batch size | activation | output activation | dropout | batchnoorm/regularizer |
|---|---|---|---|---|---|---|
| DBN | NSL | 50 | relu | sigmoid | 0.2 | Flase/False |
| | CICIDS | 40 | relu | sigmoid | 0 | Flase/False |
| | CIDDS | 80 | relu | sigmoid | 0 | False/False |
| AEDNN | NSL | 50 | relu | sigmoid | 0.3 last layer | True/True |
| | CICIDS | 30 | relu | sigmoid | 0.3 last layer | True/True |
| | CIDDS | 30 | relu | sigmoid | 0.3 last layer | True/True |
| AE | NSL | 200 | relu | sigmoid | 0 | False/False |
| | CICIDS | 100 | relu | sigmoid | 0 | False/False |
| | CIDDS | 200 | relu | sigmoid | 0 | False/False |
| Subst. DNN | all 3 | 40 | relu | sigmoid | 0.2 last layer | True/True |

Table 9: Model architecture. Layer dim as multiples of the input dimensionS

## 7.2 Results Transferability

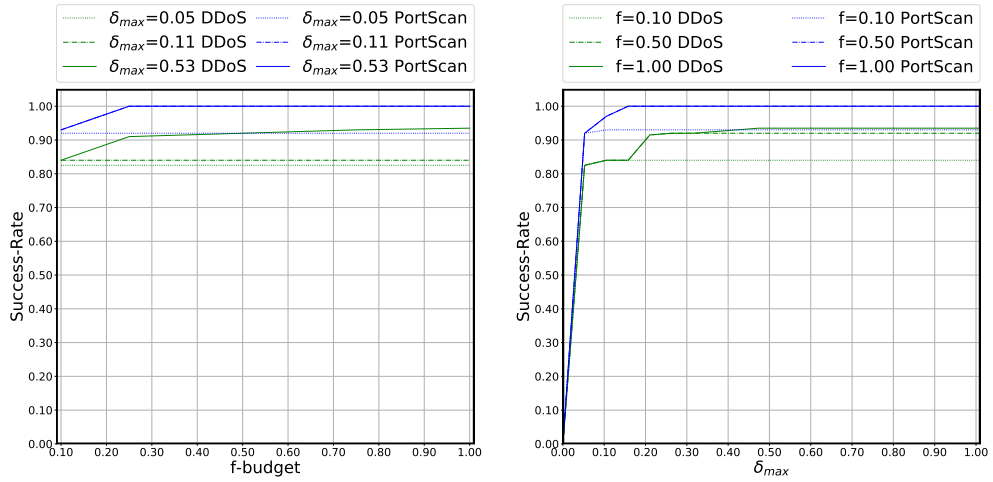



(a) NSL-KDD Probe attack c=1 (b) CICIDS PortScan attack c=1 (c) CIDDS PortScan attack c=1

(d) NSL-KDD Probe attack c=10 (e) CICIDS PortScan attack c=10 (f) CIDDS PortScan attack c=10

Figure 25: Transferability success Portscan/Probe attack

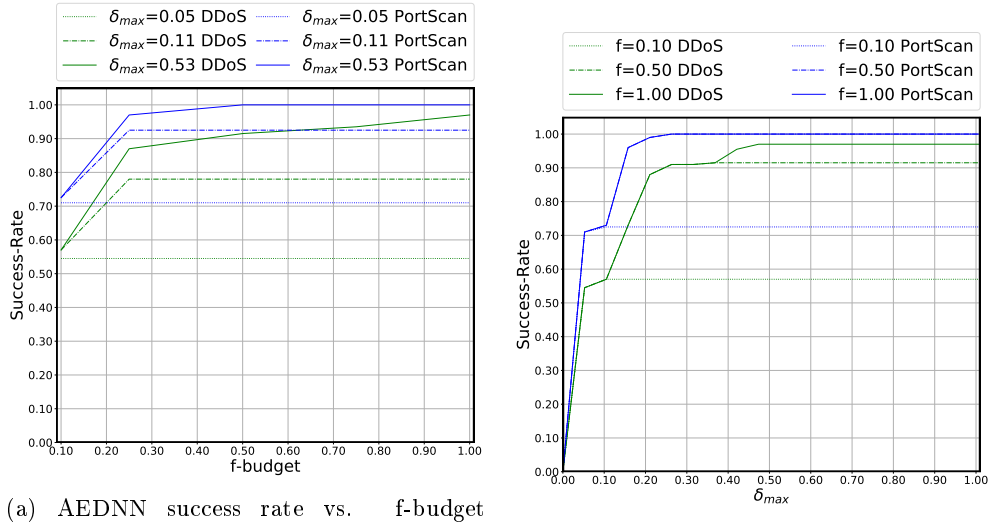
## 7.3 Results Attack Evaluation




(a) DBN success rate vs. f-budget CICIDS

(b) DBN success rate vs. $\delta_{max}$ CICIDS

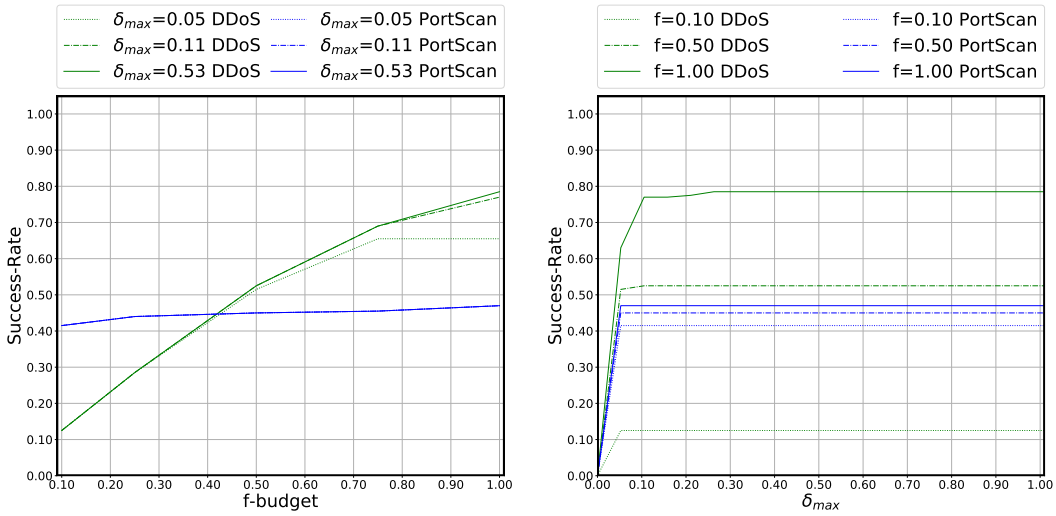
Figure 26: Success rate DBN CICIDS2017 dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AEDNN success rate vs. f-budget CICIDS

(b) AEDNN success rate vs. $\delta_{max}$ CICIDS

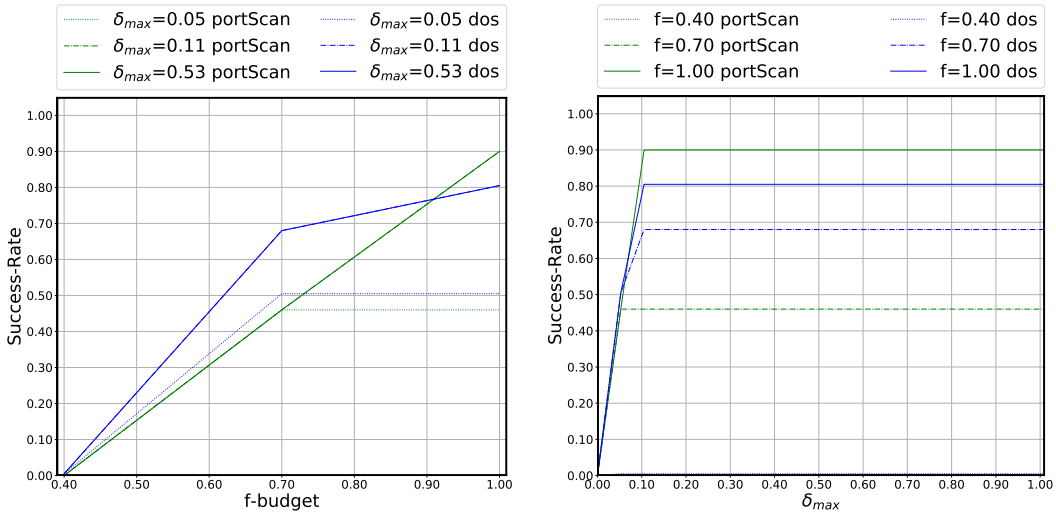
Figure 27: Success rate AEDNN CICIDS2017 dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AE success rate vs. f-budget CICIDS

(b) AE success rate vs. $\delta_{max}$ CICIDS

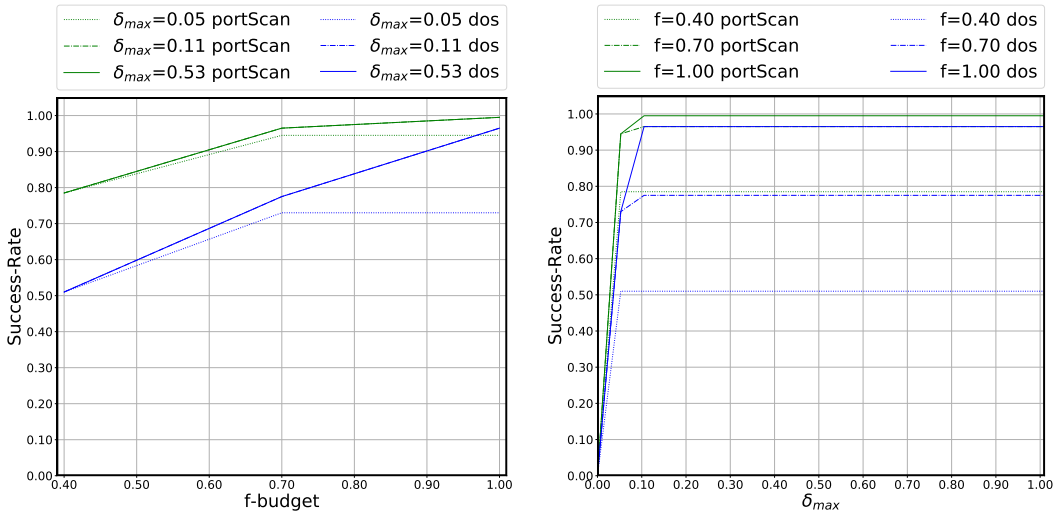
Figure 28: Success rate AE CICIDS2017 dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) DBN success rate vs. f-budget CIDDS

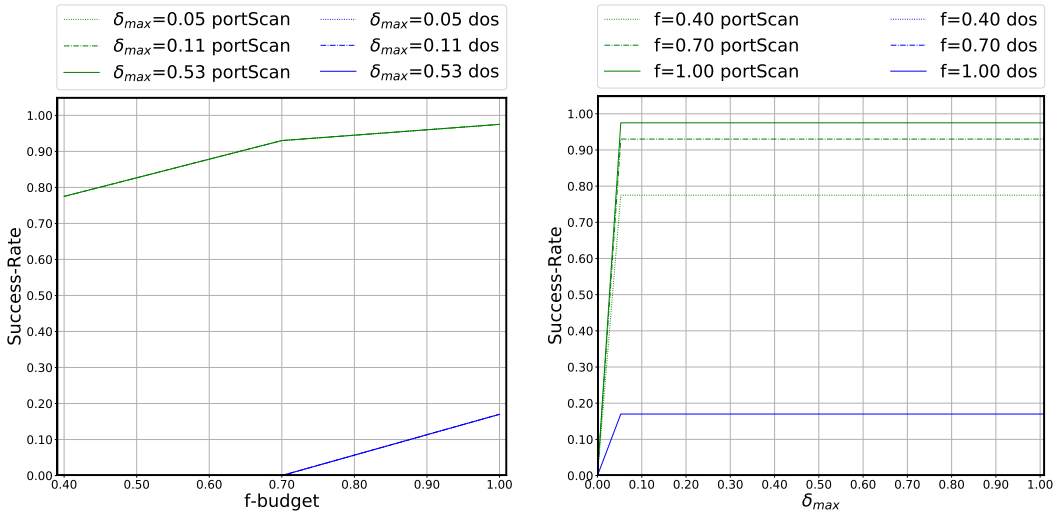
(b) DBN success rate vs. $\delta_{max}$ CIDDS

Figure 29: Success rate DBN CIDDS-001 dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AEDNN success rate vs. f-budget CIDDS (b) AEDNN success rate vs. $\delta_{max}$ CIDDS

Figure 30: Success rate AEDNN CIDDS-001 dataset plotted (a) against f-budget (b) against $\delta_{max}$




(a) AE success rate vs. f-budget CIDDS (b) AE success rate vs. $\delta_{max}$ CIDDS

Figure 31: Success rate AE CIDDS-001 dataset plotted (a) against f-budget (b) against $\delta_{max}$