Computer Networks 199 (2021) 108399

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

DoS and DDoS mitigation using Variational Autoencoders

Eirik Molde Bårli [a,b], Anis Yazidi [a], Enrique Herrera Viedma [c], Hårek Haugerud [a,*]

[a] Department of Computer Science, OsloMet – Oslo Metropolitan University, P.O. Box 4 St. Olavs plass, N-0130 Oslo, Norway
[b] Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
[c] Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, Granada 18071, Spain

ARTICLE INFO

Keywords:
Variational Autoencoders
Anomaly detection
Cyber-security
Deep learning
DDoS
DoS

ABSTRACT

DoS and DDoS attacks have been growing in size and number over the last decade and existing solutions to mitigate these attacks are largely inefficient. Compared to other types of malicious cyber attacks, DoS and DDoS attacks are particularly challenging to combat. Because of their ability to mask themselves as legitimate traffic, it has proven difficult to develop methods to detect these types of attacks on a packet or flow level. In this paper, we explore the potential of Variational Autoencoders to serve as a component within an intelligent security solution that differentiates between normal and malicious traffic. The motivation behind resorting to Variational Autoencoders is that unlike normal encoders that would code an input flow as a single point, they encode a flow as a distribution over the latent space which avoids overfitting. Intuitively, this allows a Variational Autoencoder to not only learn latent representations of seen input features, but to generalize in a way that allows for an interpretation of unseen flows and flow features with slight variations.

Two methods based on the ability of Variational Autoencoders to learn latent representations from network traffic flows of both benign and malicious traffic, are proposed. The first method resorts to a classifier based on the latent encodings obtained from Variational Autoencoders learned from traffic traces. The second method is an anomaly detection method, where the Variational Autoencoder is used to learn the abstract feature representations of exclusively legitimate traffic. Anomalies are then filtered out by relying on the reconstruction loss of the Variational Autoencoder. In this sense, the construction loss of the autoencoder is fed as input to a classifier that outputs the class of the traffic including benign and malign, and eventually the attack type. Thus, the second approach operates with two separate training processes on two separate data sources: the first training involving only legitimate traffic, and the second training involving all traffic classes. This is different from the first approach which operates only a single training process on the whole traffic dataset. Thus, the autoencoder of the first approach aspires to learn a general feature representation of the flows while the autoencoder of the second approach aims to exclusively learn a representation of the benign traffic. The second approach is thus more susceptible to finding zero day attacks and discovering new attacks as anomalies.

Both of the proposed methods have been thoroughly tested on two separate datasets with a similar feature space. The results show that both methods are promising, with the classifier-based method being slightly superior to the anomaly-based one.

1. Introduction

With the advent of Internet of Things (IoT), the risks of security attacks have grown in magnitude not only due to the vulnerabilities of IoT devices that makes them usually an easy target, but also due to the potential to misuse them to launch malicious network traffic. In 2017, the number of network devices was estimated to be 18 billion units, according to an ongoing initiative by Cisco to track and forecast networking trends [1]. Given the sheer number of units available for network attacks, it would be nearly impossible to manually create solutions to combat the problem of filtering out malicious from harmless traffic.

Indeed, some IoT devices are less secure than others, and more vulnerable to theft in the sense that they can be used as part of a botnet, or as a source of attack by an external party. Among the most notable, and perhaps hardest attacks to prevent, are denial of service (DoS) and distributed DoS (DDoS) attacks. DoS and DDoS attacks have become an immense threat to any Internet-connected machine over the last decade. In 2015, a global survey of a number of companies conducted by Kaspersky found that 50% of DDoS attacks caused a noticeable disruption in services, and 24% led to a complete denial of service [2,3]. As attacks continue to evolve and as the amount of IoT

* Corresponding author.
E-mail address: harek.haugerud@oslomet.no (H. Haugerud).

https://doi.org/10.1016/j.comnet.2021.108399
Received 12 February 2021; Received in revised form 23 June 2021; Accepted 3 August 2021
Available online 18 August 2021

devices available to launch attacks from grows, these percentages could very well increase, as could demand for working mitigation systems.

DoS and DDoS attacks are similar in intention and often similar in execution. The goal for both types of attacks is to cause a denial of service for their target by exhausting either the victim's bandwidth or system resources, such as CPU or memory. In general terms, causing a denial of service means to overload a victim, for example a web server, with traffic. Once a system receives more traffic than its bandwidth or system resources are able to handle, it will fail to receive or send parts of the intended data traffic. DoS attacks are denial of service attacks that come from a single source, while DDoS attacks are denial of service attacks originating from a distributed range of machines and networks. Both types of attacks have proven effective and devastating, but DDoS attacks are the most difficult to handle. The main reason for this is the potential significant size of a DDoS attack.

Cisco reported that the average attack size of a DDoS attack in terms of traffic load was 990 Mbps in 2017 [1]. Because of its distributed nature, a DDoS attack can generate much larger attacks than a normal DoS attack. In late 2016, the peak size was reported to be 1.1 Tbps [4]. It was the result of a DDoS attack consisting of multiple compromised IoT devices. In 2018, the peak size was reported by Cisco to have reached 1.7 Tbps, originating from a vulnerability in the memcached protocol [1], and resulting in the largest DDoS attack to date.

Creating an effective mitigation system requires consideration of multiple facets of both DoS and DDoS attacks. An effective mitigation system needs to be able to handle large amounts of traffic, and to process malicious and normal traffic simultaneously. The system is required to separate harmful from harmless traffic at such a rate that the targeted victim is able to handle the traffic load. Considering that network packets from a DDoS or DoS attack are not harmful in themselves, a mitigation system can let a certain amount of malicious traffic through, while prioritizing letting as much normal traffic through as possible. A mitigation system that fails to meet any of these requirements risks exposing the victim to the attack. In the case of inability to handle the incoming traffic load to a server, the mitigation system would cause a general slowdown of the server, because it needs to inspect each packet before forwarding or allowing the packets through. In the case of inability to separate normal traffic from malicious traffic, the mitigation system risks blocking normal users from accessing the server, or letting too many malicious packets through. Handling large DDoS attacks can be particularly difficult, because the defending system has limited resources to spare for defense, while an attack has the potential to capture and use much larger amounts of devices and their resources to overpower the victim. There are commercial systems that reroute the enormous amounts of traffic in DDoS attacks to a network of dedicated cloud services, which filter out the malign traffic in what are known as cloud scrubbing centers [5,6]. However, such cloud services also need an efficient method to differentiate between the attack packets and the normal packets that are supposed to be forwarded to the destination service.

Current mitigation systems often focus on a form of pattern recognition or frequency-based detection, or a combination of the two. We will give some examples of these approaches. In [7], in order to detect DDoS attacks, the authors use a real time frequency vector based on the target's resource requests that tracks statistics of the target's resource requests. Other approaches track the frequency of the IP addresses and build IP filters based on this information [8,9]. In [10], a hybrid approach was reported which uses frequency to compute traffic entropy which are fed as features into a pattern recognition engine. The work of Hagos et al. [11] is a typical example of a pattern recognition approach where the authors investigated applying machine learning approaches on 41 traffic features.

It is common to have a detection system in place, but this often just means that it is able to know when an attack is happening, while being unable to stop it. This puts system administrators in the precarious situation of having to combat the attack manually, which is usually only feasible after the victim server has been shut down [12]. Detection systems generally encompass systems that are able to recognize when an attack is happening, or has happened in the past, by analyzing the network traffic logs. A mitigation system is different in that it is able to differentiate malicious from normal traffic on the packet or flow level, i.e., based on IP addresses or other packet information during the attack.

Part of what makes it difficult to differentiate between normal and denial-of-service traffic is their similar behavior. If we compare two network packets in isolation, one from a legitimate user, and another from a compromised computer being used as a part of an attack, the differences would be few or none. DoS and DDoS packets are not harmful on their own. It is the amount of traffic they send that overloads a system, drowning out legitimate attempts at access. This makes it difficult to implement solutions that can keep out malicious connections.

Simple methods and algorithms for synthetic datasets can give very good results due to similarity of the generated traffic. For instance, BoNeSi [13] is a popular framework to simulate malicious attacks, however, it might generate too homogeneous traffic making it easy to detect by Machine Learning based intrusion detection systems. For more real-life datasets, the accuracy drops usually as the detection task becomes more difficult compared to synthetic datasets. However, within the field of security, the number of publicly available datasets for malicious traffic are limited. The most known dataset is the KDD-Cup dataset [11] which has several disadvantages as reported in [11,14] and which led to the advent of an improved version called NSL-KDD [11]. One of the most recent and adopted datasets nowadays are CICIDS 2017 and CICIDS 2018 released by the Canadian Institute for Cybersecurity. Both datasets fill the void when it comes to the lack of publicly available datasets for security. However, these datasets are limited to a few days of traffic and a limited number of flows that is limited in the era of Big data and that do not generalize to real life settings. Thus, some authors have even reported 100% accuracy on those types of datasets. For instance, in [15], the authors reported 100% accuracy using SaE-ELM-Ca which is an optimized extreme learning machine on CICIDS 2017 and this can be due to the limited size of the dataset. Even one of the simplest machine learning algorithms, namely KNN, performed well in [16]. With the advent of Artificial Intelligence and the proliferation of Generative Adversarial Networks [17], attackers are gaining more power in generating illegitimate traffic that is hard to distinguish from Normal traffic. For example, in a paper involving some of the authors of the current work [18], we have shown that a subtle modification of the flow parameters and packet padding can make malicious flows evade detection.

In [19] the authors tested machine learning detection with two datasets, one including IP addresses and one without IP address. The accuracy drops when the IP addresses are not fed into the classifier which is a sign of overfitting. However, in our paper the accuracy increases when IP addresses are not used as a part of the training data which shows more robustness against overfitting and also an ability to learn meaningful features of the traffic.

A mitigation system needs to be able to detect what packets or IP addresses are malicious, and to stop them from entering the network. Using machine learning algorithms could be a potential way of creating such a system, since machine learning can itself potentially find relations between packet information and intent.

In the last few years, we have seen a rise in the use and success of machine learning algorithms. Machine learning, specifically deep learning, can be applied to a myriad of different problem domains, including classification of different types of data and anomaly detection, using a variety of different architectures such as convolutional networks or autoencoders [20]. Deep learning has been proven to be effective at analyzing and extracting useful data patterns that manual and automatic approaches are unable to solve in problem domains other than computer network traffic. Manual and automatic approaches rely

on constant updates as well as human interaction to remain effective against DoS and DDoS attacks. A deep learning solution, on the other hand, is autonomous and requires minimal human interaction. These are among the reasons why a deep learning solution could prove to be an effective means of combating DoS and DDoS attacks.

This article proposes two approaches for DoS and DDoS mitigation utilizing the framework of the Variational Autoencoder (VAE) by Kingma and Welling [21]. The first approach is the Latent Layer Classification algorithm (LLC-VAE), which aims to identify different types of computer network traffic from the latent layer representations of the VAE. LLC-VAE resorts to a classifier based on the latent encodings obtained from Variational Autoencoders learned from traffic traces. The second approach is the Loss Based Detection algorithm (LBD-VAE), which tries to identify normal and malicious traffic patterns based on the VAE reconstruction loss function. The construction loss of the autoencoder is fed as input to a classifier that outputs the class of the traffic including benign and malign, and eventually the attack type. Thus, the second approach operates with two separate training processes on two separate data sources: the first training involving only legitimate traffic, and the second training involving all traffic classes. From a methodological point of view, both approaches look similar as they both use autoencoder however the fundamental principles are different as the first uses autoencoder to learn a representation of all traffic types while the second is only interested in learning a representation of benign traffic forming the basis for anomaly detection. Thus, the second approach is more robust against zero day attacks while the first is better at recognizing seen malicious patterns. The two proposed approaches are not meant to be complete solutions for mitigation systems, nor are they aimed at creating solutions that work against all types of DoS and DDoS attacks. The contribution of this article is the research it presents on how deep learning, specifically a VAE, can detect specific types of DoS and DDoS traffic from network flows. In addition, it examines whether this can be generalized to detect other types of DoS and DDoS attacks, and to what extent.

1.1. Outline

The remainder of this article is organized as follows. Section 2 surveys related work within the field of DoS and DDoS detection. Section 3 discusses the two proposed deep learning approaches. Section 4 describes the experimental setup in which the proposed approaches have been tested, as well as the datasets. Section 5 reports on the experimental results achieved by the two proposed approaches. Finally, in Section 6, we make some final remarks and conclude the article. The Appendix is a short review of autoencoders.

2. Related work

Techniques to combat DoS and DDoS attacks are many and varied. Some focus on stateless packet information, while others rely on meta information from stateful packet flows. In this section, we will present a few relevant research papers and topics in order to create an overview of different strategies and methods for both discovering and mitigating DoS and DDoS attacks. We will also present research that aims to improve or adapt existing techniques relevant to our research or future research. A traditional approach for mitigating DDoS attacks is to construct filters based on the historic benign IP-traffic of a site [9] and even to include the geographical location of the IP-addresses in order to improve the efficiency of the IP-filters [8]. In this section we will however mostly focus on approaches related to machine learning techniques.

2.1. Flow-based stacked autoencoder in SDN

Niyaz et al. [22] present a DDoS detection system for Software-Defined Networks (SDN), that uses deep learning to detect multi-vector DDoS attacks from flow traffic. In an SDN, DDoS attacks happens on the data plane or control plane[1], and their system is thus focused on detecting DDoS traffic on these two planes. The detection system consists of three modules, which they call “Traffic Collector and Flow Installer”, “Feature Extractor”, and “Traffic Classifier”. These modules operate by extracting multiple different headers from TCP, UDP, and ICMP packets, and generating a flow to be fed into the DDoS detection system. Each packet belonging to the same flow has the same source and destination IP, the same source and destination ports, and the same protocol type.

Multiple sparse autoencoders (SAE) are used together to form a deep learning network by placing them after each other [22]. There are two ways to do this. One is to encode and decode as usual on the first autoencoder, and feed the decoded output to the second autoencoder, and so on. Another method is to encode and decode as usual to train the network, but, instead of using the decoded outputs, the latent layer outputs are fed into the next autoencoder in the line. So, if we have the raw input x feeding into the first SAE, it will be encoded into the latent layer values g and decoded to $\hat{x}$. The values g are used as input for the second SAE, which encodes to latent layer values h. After this, the authors apply a softmax classifier to the outputs of h [22]. The final stacked autoencoder consists of two models with a classifier at the end.

For training and testing, their flow generation system extracts and creates a total of 68 features [22], although separated into three protocols, TCP, UDP, and ICMP. Attacks were simulated using Hping3, and Niyaz et al. claim to be able to identify individual DDoS classes with an accuracy of about 95%. The accuracy for differentiating between normal and attack traffic is claimed to be 99.82%.

2.2. Anomaly detection on the CICIDS2017 dataset using LSTM

In networking, traffic is sent back and forth between machines, and some packets can be said to have a relation to each other, such as the packets transferred during a three-way handshake. If multiple machines cooperate to send a DDoS attack, they are considered to be parts of the same attack, even though there are multiple sources. Using machine learning to detect the relationship between packets or flows could potentially allow mitigation methods to prevent malicious traffic based on learned patterns in IP addresses and traffic frequency. Pektas and Acarman [23], and Radford et al. [24] proposed two different detection methods in their respective papers using the CICIDS2017 dataset [25] among others, which are used in this paper as well. This dataset, discussed further in Section 4.2, is a modern dataset containing network traffic data, both normal and malicious. It is meant for research, as a benchmark for developing detection and mitigation methods.

A method for grouping together network flows into two-dimensional arrays was proposed by Pektas and Acarman [23]. The proposed system aims to detect malicious network traffic using a combination of a convolutional neural network (CNN) and a long short-term memory (LSTM)[2] network, to learn spatial–temporal features. Each network flow is grouped based on its source and destination IP, destination port, protocol, flow direction, and label. The group is then fed through the model as a 2D array, where each row represents one flow, sorted by their timestamps. If a group consists of a number of flows that is lower than a given threshold value, the group is omitted. This could be regarded as a form of frequency-based detection added on top of the detection algorithm using a hyper parameter, since malicious traffic,

[1] The data plane in an SDN forwards traffic, while the control plane manages what route the traffic will take.

[2] LSTM is based on a recurrent neural network architecture.

especially DoS and DDoS attacks, rely on sending large amounts of packets or flows in a given time frame. To configure the model network, a Tree-structured Parzen Estimator (TPE) is used to tune the model automatically based on hyper parameter searching. The authors claim that the model is able to detect attack traffic with an accuracy of 99.09%.

Radford et al. [24] proposed a method for anomaly detection using sequence modeling, utilizing a recurrent neural network (RNN) architecture with an LSTM model[3]. Five different sequence aggregation rules based on the flows provided in the CICIDS2017 dataset are evaluated using an LSTM model. The research in their paper is partly based on their previous work [26]. Once a sequence of computer network traffic flows of length 10 has been generated, it is fed through the LSTM model. There, the sequence is given a prediction of whether it is normal or malicious traffic, based on an outliers score [24] and measured using the mean area under the curve (AUC). For baseline comparison, a simple frequency-based method for outlier detection was used. The sequence aggregation using an LSTM model proved to be slightly better in a few cases, but it was mostly on a par with the frequency-based model, or worse. Whether this was because of the LSTM model or the sequence aggregation methods was deemed uncertain.

2.3. Malicious traffic detection using entropy-based techniques

Entropy, from information theory, is a measurement of uncertainty or disorder in a system, often called Shannon entropy [27]. It is a value representing the average rate of information drawn from a stochastic source of data. A source of data producing an entropy value closer to 1 when normalized is considered hard to predict, and a value closer to 0 is considered easier since there is less uncertainty. Behal et al. proposed using generalized information entropy (GE) and generalized information distance (GID) to separate so-called low rate DDoS (LW-DDoS) and high rate DDoS (HR-DDoS) from normal traffic and flash events (FE) [28]. The idea is to group network traffic into sets, where packets are grouped together in 10-second time frames. The entropy and information distance is then measured for each set. It was discovered that DDoS traffic flows had more similar traffic, as their IP addresses are more closely grouped in relation to time. This leads to lower entropy values within a set containing more DDoS traffic, and higher information distances between normal and DDoS traffic.

To better understand the relationships between different features in network traffic and how they can be used for anomaly detection, Nychis et al. published an empirical evaluation of the subject, using entropy [29]. Features were collected from bi-directional flow data.

The relationships between the features' source IP, destination IP, source port, destination port, in-degree, out-degree, and flow size distribution (FSD) were measured using entropy, and given correlation scores. Note that FSD is the packet per flow measurement. The study found a high correlation between certain features, perhaps most notably between ports and IP addresses. However, the correlation between ports and addresses was found to have limited usability for anomaly detection purposes, and it was argued that they are ineffective for both scanning and flood type attacks. Interestingly, the FSD and degree distribution scores had some success in detecting anomalies, and there was a notable difference between the entropy scores for normal traffic and malicious traffic [29].

[3] An RNN is a neural network with the ability to remember previous data inputs. LSTM is an improvement on this concept, increasing its memory capabilities.

2.4. Flow-based DoS attack detection with techniques based on computer vision

Autoencoders from machine learning can be used for anomaly detection, by separating malicious from normal traffic using pattern recognition. If a data input fed into the model is not recognized, it will be considered an anomaly. Tan et al. proposed the use of computer vision techniques for anomaly detection in network traffic, specifically for DoS attack traffic [30].

Features from inbound network traffic are fed through the detection system, and stored as one-dimensional feature vectors called records. In computer vision, earth mover's distance (EMD) can be used to detect dissimilarities between two images. To apply this idea to their system, Tan et al. transform inbound records into two-dimensional matrices, similar to images. Profiles for normal network traffic are generated based on multivariate correlation analysis (MCA) from previous work [31]. MCA to find correlations between features and by generating normal records from the inbound records originating from the datasets. A reformulation of EMD [32] is then applied to the generated record matrices, measuring dissimilarities between each record. Any unmatched records will be defined as attacks. In an evaluation of the application of the detection system to the KDD'99 [33] dataset, the system was reported to have achieved 99.95% accuracy, and 90.12% on the ISCXIDS2012 [34] dataset.

2.5. Anomaly detection with hidden semi-Markov model

Hidden Markov models (HMM) have a variety of different applications, such as research on time series data or for sequence recognition. Xie and Yu proposed a solution for detecting application layer DDoS attacks as anomalies by learning from user behavior on web pages using a hidden semi-Markov model (HsMM) [35]. HsMM is an extension of HMM that adds an explicit state duration, and is designed for live training. The HsMM is a behavior model that learns from looking at normal user behavior with regard to how they behave when browsing a given web page, using the address bar, hyper links, and reading web page content. From this, a normal user is defined with a mean entropy value that will be used for comparison with the filter. Requests from an outside source reach the victim web page and are stored over time as a request series, or, as it is called in the paper, an HTTP request sequence. This is similar to how network packets would be handled by an RNN. The average entropy of this sequence is calculated in the detection system filter, and used for comparison with the entropy from the generated user behavior characteristics, made by the HsMM. The research and experiments in the paper were only tested on application layer DDoS attacks, but showed promising results, with a detection rate as high as 98% .

Not many solutions for detecting or mitigating DoS and DDoS attacks focus on learning from user behavior. This could be a potential avenue for further research.

2.6. Complete autoencoders and recurrent autonomous autoencoders

Ili et al. proposed the usage of Complete Autoencoders (CA) and Recurrent Autonomous Autoencoders (RAA) for detecting DDoS attacks in [36] and [37] respectively. The main difference between a CA and a regular autoencoder is the fact that a CA exploits the imbalance in the data. The heart of the architecture in the two works rely on an ensemble of a number N individual Autoencoders each acting as classifier and a majority voting mechanism to aggregate the individual results. Whenever the predicted number of benign IPs is a majority under an attack scenario, a class switch is operated in the binary classifier. Each Autoencoder is composed of a feature extractor, a target detector and a netflow identifier. The target detector has an adjustable reference threshold which is the true number of positives among the IP addresses. The reference threshold is determined using

adaptive search. The netflow identifier operates the classification based both on the features and the reference threshold. The model in [37] improves the previous work in [36] in the sense that the RT is not computed from frames before the attack but rather using the principles of recurrent neural networks. The works of Ili et al. [36,37] use a Deep Autoencoder that contains two symmetrical Deep-Belief Networks (DBN). It is worth mentioning that the idea of using recurrent neural networks can enhance our current paper in order to adjust the threshold of the contraction loss. The authors use a rather non-realistic setup by resorting to BoNeSi [13] to simulate malicious attacks consisting of TCP flood, UDP flood, and ICMP SYN flood attacks [37]. In our paper, we use more realistic traffic. The disadvantage of using simulated traffic is that the malicious network packets might be too similar to each other, making the detection easier.

The work of Chen et al. [38] uses a multi-channel CNN(MC-CNN) to detect DDoS attacks and reports experiments using both KDD-Cup 99 and CICIDS2017. The results are encouraging, however only accuracy is reported. MC-CNN permits automatic extraction of features. The authors distinguish different groups of features: packet level features and traffic features which are fed into two channels.

3. The proposed deep learning algorithms for attack mitigation using variational autoencoders

DoS and DDoS mitigation has been researched for many years, and several different approaches have been developed. Popular methods for mitigation include pattern recognition, similar to how viruses are detected, or the detection of malicious sources based on network traffic frequency. Pattern recognition mitigation systems have proven effective in certain scenarios, such as when a victim is the target of a known attack, but they are also known to have several drawbacks. These kinds of systems are prone to human error and require constant maintenance to operate. Code updates are needed every time a new attack surfaces, or whenever a known one is altered. Traffic frequency-based mitigation systems work by blocking network traffic based on high traffic frequency, or allotting a certain amount of bandwidth to each connected IP. Monitoring traffic frequency can be effective if there is a lot of traffic from one source, allowing a mitigation system to block attackers that take up too much of a system's resources. A problem with this approach is that normal users might sometimes be regarded as attackers, for example if they try to refresh a website many times because of slow loading. Checking for frequency alone could also let certain types of attacks through, such as DDoS attacks from a large network of machines with different addresses.

With deep learning, it is possible to let the mitigation system filter out normal from malicious traffic autonomously. Network traffic can be fed through a deep learning algorithm, which filters individual packets or flows based on learned features. For this article, incoming network packets will be transformed into traffic flows, before being fed into the deep learning algorithms. We propose two separate deep learning algorithms to filter network traffic flows: Latent Layer Classification on a Variational Autoencoder (LLC-VAE), and Loss Based Detection on a Variational Autoencoder (LBD-VAE). These two deep learning algorithms learn patterns by themselves, instead of relying on older techniques where the attack patterns must be inserted manually.

The contribution of this article is to present research on how, and how well the proposed deep learning approaches, the LLC-VAE and the LBD-VAE, can filter out malicious from normal traffic. The LLC-VAE and LBD-VAE will be used to learn from a few types of DoS and DDoS attacks. While many more types of malicious computer network traffic exist, the samples in these datasets will be used to analyze whether the two proposed approaches can reliably be used as DoS and DDoS mitigation systems. In the remainder of this section, we will present how the two proposed approaches are designed, and discuss different options for designing them.

3.1. Motivation

As mentioned earlier, network packets moving between a client and a server can vary greatly in shape and form even though they follow the same protocols. Likewise, many network packets can be very similar, with only small differences separating them. The same applies to packets belonging to DDoS and DoS attacks, which can be very similar to other attack packets, and normal packets. One of the goals we aim to achieve in this article is to be able to efficiently separate DDoS and DoS attacks from normal traffic. An autoencoder, particularly a VAE, could be a well-suited tool for this. It is a challenging problem to separate malicious and normal traffic, sometimes with no more than a minute difference in time separating them; hence we need a tool that is able to detect small details and differences, as well as finding features that are important for separating them. Standard autoencoders and other implementations of autoencoders learn features from input data in a discrete fashion when encoded to the latent layer. A VAE, on the other hand, encodes features as probability distributions using variational inference [21,39], which, in our case, causes similar packets to be encoded and decoded in a similar manner. We sample from this distribution to get the latent attributes.

A common problem with machine learning in general is collecting reliable data to train and test on. What is more, deep learning algorithms require large amounts of data to generalize and train robust and deep features. While a VAE needs large amounts of data just like many other deep learning methods, one of its strengths is its ability to generalize over similar features, and ignore noise. With a VAE, it is possible to create a model that is capable of learning smooth latent state representations of the input. There are two parts to the learning process of a VAE: the reconstruction loss and the KL-divergence loss. Using purely reconstruction loss causes it to behave like a normal autoencoder, simply reconstructing the input to the output, with potentially large gaps between different classes. Using purely KL-divergence loss, we end up with outputs that all use the same unit Gaussian, causing different classes of data to be grouped and mixed together. Other autoencoders that do not rely on variational inference have been used for both DDoS and DoS detection, and were discussed in Section 2, Related work. Because the VAE combines these two loss terms into one, a potential use for it is to group different classes of data with similar features close to each other. This is one of the reasons why a VAE is a generative model, since it can extract and generate new data based on data points with likenesses to each other. For detecting and classifying DDoS and DoS, the generative part can be ignored, instead using the VAE to remove noise, and generalizing over similar features in a manner that enables understanding of data not included during training of the model.

Generating new samples using a VAE is straightforward. One simply removes the encoder part after training, leaving z to sample from $\mathcal{N}(0,1)$[40]. We do not need to generate new data from the learned features of DDoS, DoS, and normal data since we are using it for detection and mitigation. However, it should be noted that a general problem with VAE architecture is that the normal and generated outputs come out blurry, or noisy [21,40]. The model will sometimes ignore less prevalent features of the input, potentially causing this blurriness. This weakness is most obvious when looking at images as outputs. How much this affects datasets consisting of network packets, and how important it is for training and testing is difficult to tell.

3.2. First proposed approach: Latent layer classification on a variational autoencoder (LLC-VAE)

The first proposed approach, Latent Layer Classification on a Variational Autoencoder (LLC-VAE), utilizes the strength of the variational autoencoder [21] as the underlying architecture for a latent layer classification network. Based on the ability of the variational autoencoder to learn latent representations of various classes of a dataset, the LLC-VAE

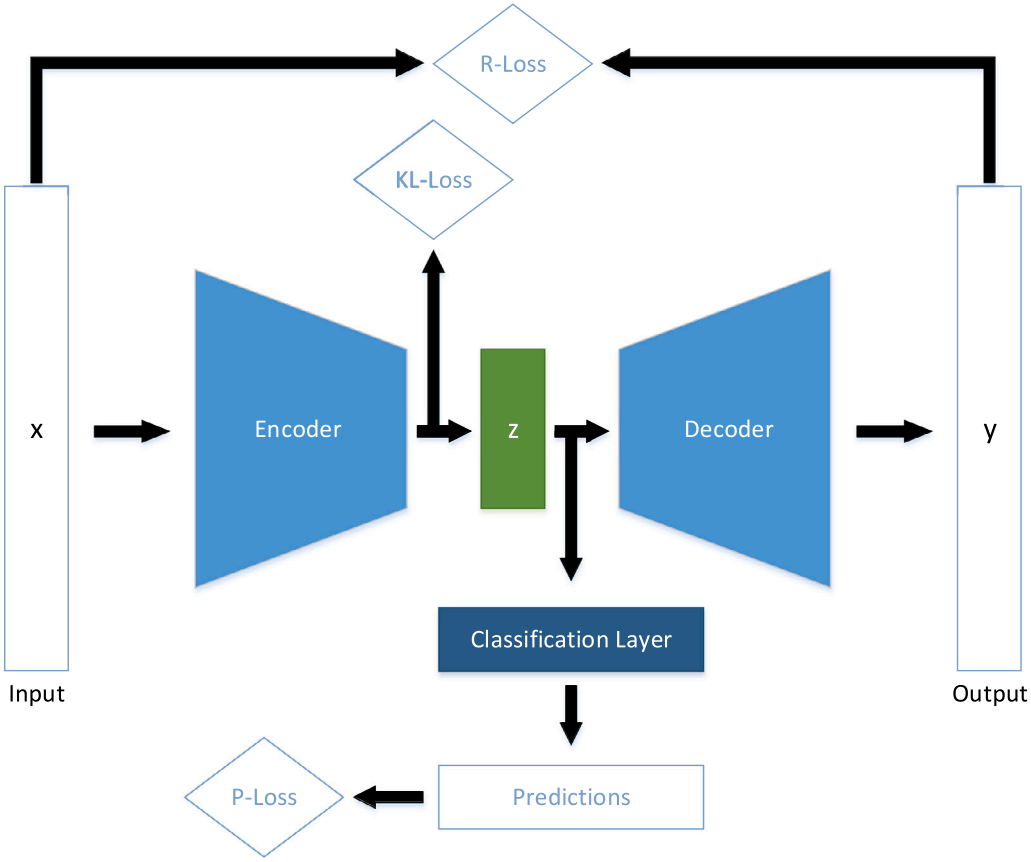


Fig. 1. Latent layer classification on a VAE. Rectangles represent node layers. Diamond shapes represent the loss values. The trapezoids represents the shrinking nature of the encoder, and the expanding nature of the decoder. The classification layer is a fully connected layer with a softmax activation function.

approach aims to classify different types of computer network traffic, and separate normal from malicious traffic. The performance will be documented in 5, Analysis, together with how the model performs in different settings.

A representation of the LLC-VAE deep learning model is shown below in Fig. 1. First, one flow from the dataset is loaded into memory and transformed into a readable format by the model. One flow is represented as one feature vector and, together, they are grouped into mini-batches before being fed into the encoder. The encoder performs dimensionality reduction on the mini-batch over multiple layers, further transforming the feature vectors until they have been encoded to the latent layers[4] of means and standard deviations. Based on this, the latent layer z is sampled, and the KL-Loss value is produced. This output, now represented as a vector of nodes with reduced dimensions, is sent to a fully connected layer that outputs the unscaled class predictions. The softmax function, short for softargmax [41], is applied to the class predictions, so that the nodes are normalized to a legal probability density function (PDF), where each node represents a single class. To optimize the predictions, we use cross entropy over the softmax predictions to generate a numerical loss value, which is called `prediction loss`, or P-Loss for short. The latent layer z also feeds its vector of nodes into the decoder. The decoder aims to accomplish the opposite of the encoder, increasing the dimensionality through multiple layers to generate a reconstruction of the original feature vector. The reconstructed feature vector is an approximation of the input. These two vectors are compared to generate a reconstruction loss, or R-Loss for short, using mean squared error:

$$L(x) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$$

Here, x represents the model input, y the output, and n the number of features. The values generated from each of the loss functions, R-Loss, KL-Loss, and P-Loss, are combined and optimized. We use the Adam optimizer developed by Kingma and Ba [42], to perform stochastic gradient-based optimization of the model. The optimizer will backpropagate through the network, updating the weights between each layer based on the total loss.

Deciding what type of layers to use. A variational autoencoder is inherently a deep learning algorithm, with multiple hidden layers. There are a minimum of three hidden layers, excluding the input and output layers, with the potential to add more. The encoder and decoder have a minimum of one hidden layer each, while there is one hidden layer, z, as seen in Fig. 1, also called the latent layer. Various types of layers are available to use for the encoder and decoder in a VAE, some of which will be explored here. How many hidden layers should be present in the encoder and decoder will be decided through tuning in 5, Analysis.

Fully connected layers. A fully connected layer is the layer typically associated with classification problems in multilayer perceptrons, but it is also available to use with other neural networks, such as a VAE. The implementation is fairly simple. All nodes in a layer are connected to all nodes in each adjacent layer, where each node stores the node values and the connections store the weight values. These values are used to predict the outcome of a given problem during the training process. The outputs of this layer type are calculated using the linear function outputs = activation(inputs · kernel + bias), where kernel is a weight matrix created by the layer. Using multiple fully connected layers allows for the classification of nonlinear problems [43].

[4] The hidden layers between the VAE encoder and decoder are called latent layers in this article.

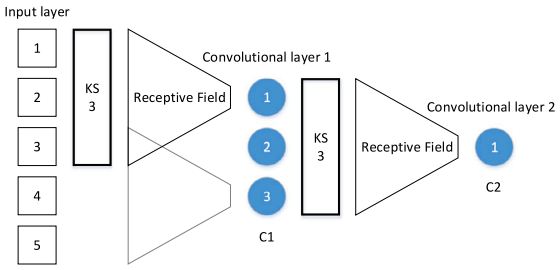


Fig. 2. This figure is a simplification of how the encoder part of the VAE works with convolutional layers. The decoder is similar, but performs the same operations in reverse. The rectangles represent the kernels of size 3 nodes across the height and width dimensions, respectively. The squares from the input layer represents 5 features. The trapezoids represents the concept of `Receptive Fields` in ML. The circles represent the abstract feature nodes of the convolutional layers.

Recurrent layers. Recurrence is primarily used for sequence modeling, that is to say a recurrent layer is able to remember what has been seen previously. Intuitively, implementing memory in a neural network to observe relationships between different traffic flows seems like a good idea, enabling a deep learning model to remember previously seen, as well as ongoing, attacks and their sources. This method has shown some success, as discussed in Section 2, Related work.

Convolutional layers. Convolutional layers can help greatly when classifying network traffic flows, because of their ability to create complex feature abstractions from simpler ones in order to understand complex feature relationships. Convolutional layers calculate the outputs linearly in a sliding window manner, using the convolution operation often denoted as $(x * w)(t)$, where x is the input function at a given time t and w is the weighting function [41,44]. Convolutional layers have the ability to capture spatial and temporal dependencies, and are therefore excellent for use in object detection [45], and image recognition [46]. Convolutional layers have proven effective in relation to a variety of different problems, including multiple classification problems, and they are a strong candidate for use with a VAE. While a fully connected layer learns a representation of an input based on each feature, a convolutional layer selects important features with sliding window detectors, making it better at ignoring redundant information and learning useful representations through multiple abstract feature layers.

A convolutional layer learns abstract features from the input layer using a technique called `sliding window`. A matrix, called a kernel, performs mathematical operations on the node values in a layer, beginning with the feature nodes from the input layer. In the example in Fig. 2, a total of five features from the input layer can be seen. The traffic flows used in this article are one-dimensional; hence, when defining the kernel size, we only need to define the size of a single dimension. In the example, each node in the first convolutional layer learns from nodes 1, 2, and 3 in the input layer. The second node in the first convolutional layer learns from input nodes 2, 3, and 4, while the last node learns from input nodes 3, 4, and 5. Put another way, each node from the first convolutional layer has a receptive field of size 3. Likewise, the second layer also has a local receptive field of 3. When using multiple convolutional layers, each node in each layer has a local receptive field of a given size, but the effective receptive field size increases every time a layer is added. For example, the first layer in Fig. 2 has a receptive field size of 3. The following layer will learn from the feature abstractions of the first layer, causing it to have a local size of 3, but an effective size of 5, in the input layer. This means that the node in the second layer has learned an abstraction from the relations between all the nodes in the input layer.

Regularization. The term regularization has a wide range of uses. Regularizing a deep learning model means preventing overfitting, avoiding exploding and vanishing gradients, and generally keeping the training phase stable and improving various issues. An autoencoder inherently performs a form of regularization. A VAE, specifically, performs dimensionality reduction, and, as a result, forces the model to choose the most important features through node selection.

Weight regularization is the addition of a penalty term to prevent exploding gradients. It can be applied to the hidden layers of the encoder and decoder when using convolutional or fully connected layers. Exploding gradients are a problem for many different neural network architectures, where the layer weights grow out of control, causing various issues with the model loss. This could occur, for example, when the loss does not gain traction and does not improve, or if the loss ends up with a NaN value, due to floating point overflow. We use the weight decay regularization technique, L2, typically called ridge regression, on the kernel values of the convolutional layers. This will encourage the layer weights to grow towards smaller values around 0. Weight regularization might not be necessary, but it does not harm the model performance, and there is therefore no reason not to implement it.

A regularizing layer can be added after each layer in the encoder and decoder. Two different regularizing layers can be used for the approaches in this article: dropout layers [47] or batch normalization layers [48]. Adding either of these to an encoder and a decoder is meant to ensure stable learning, prevent overfitting, and to improve the exploding and vanishing gradients problem. Dropout is a technique that assigns a keep probability to each node in a given model network that is updated during training. If a node has a keep probability that has fallen below a given threshold, the weights will be multiplied by 0 during the forward pass, causing the gradients to become 0 during backpropagation. As a result of this, a number of nodes in the model network will effectively be removed, forcing the model to learn more robust features and teaching each layer to rely on different nodes from the previous layer. Batch normalization can be used instead of dropout to prevent overfitting, as well as to prevent vanishing and exploding gradients.

Latent layer classification. The encoder and decoder of the VAE can be modified to use different types of layers. To represent the vectors μ, σ, and z, we use fully connected layers. The output from z, as seen in Fig. 1, is sent to another fully connected layer, which attempts to predict the classes that the different flows belong to. A softmax cross entropy loss function is used to improve the classification by performing cross-entropy between the predicted classes and the labeled true classes. In the dataset CSECICIDS2018 [49], there are a total of 8 classes, each used for training and validation by the LLC-VAE. Either of the cross-entropy functions softmax or sigmoid could be utilized as loss functions for the LLC. Each flow can only belong to one class, and the higher the probability of a flow belonging to one specific class, the smaller the probability of that flow belonging to another class. That is to say the class predictions are dependent on each other. For the LLC-VAE, we will use the softmax activation function.

The dataset CICIDS2017 [25] has two fewer DDoS classes, a total of 6. This dataset will be used for testing. Having fewer classes for the test network compared to the training network is not a problem, as these six classes are present in both datasets, and comparison can be done on these classes alone. However, the test results will be less comprehensive than the validation results. Classifying specific DoS and DDoS attacks could pose a different problem. Although the LLC-VAE is able to classify specific attacks and normal computer network traffic, doing so could lead to less accuracy when differentiating between normal and malicious traffic. When the model needs to carry out more than just anomaly detection, it has to train more precisely tuned weights for the different flow features. This could mean that the model achieves more precise predictions for specific attacks at the expense of overall accuracy. For this reason, both a multi-class variant and an anomaly detection variant will be examined in 5, Analysis.

Latent layer z. At the core of the VAE is the latent layer z that samples from the vector of means and vector of standard deviations. The two

vectors of means and standard deviations are implemented using fully connected layers that are initialized with random values. Previously, we discussed how, in a VAE, the term $-D_{KL}(q(z|x)) \parallel p(z))$ is used to approximate the true posterior. This is the KL-Loss term, and can be written as:

$$\frac{1}{2}\sum_{j=1}^{J} = (1 + log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2)$$

The KL-Loss term is used as a measurement of the divergence between two probability distributions, the vector z that samples from $\mathcal{N}(\mu_i, \sigma_i^2)$, and the standard normal distribution. The sampled vector z is implemented as $z = \mu + \sigma \cdot \varepsilon$ where ε is an auxiliary noise variable drawn from the standard normal distribution, $\varepsilon \sim \mathcal{N}(0, 1)$. When minimizing the KL-Loss term, the vector of means μ and vector of standard deviations σ will be optimized to resemble the target distribution. This means that we can initialize the two fully connected layers with random values, and let them learn the target distribution during model training.

The latent layer z not only learns to describe the input flows x, but also representations of x that have similar features. The mean and standard deviation vectors that z samples from give a distribution to each flow. Each flow is represented as a data point, and each point has a probability distribution. Other points within the probability distribution have a higher chance of belonging to the same class, as opposed to a normal autoencoder, where each point has a direct encoding that only decodes specific encodings in the latent space. Intuitively, this allows a VAE to not only learn latent representations of seen input features, but to generalize in a way that allows for an interpretation of unseen flows and flow features with slight variations.

3.3. Second proposed approach: Loss-based detection on a variational autoencoder (LBD-VAE)

The second proposed approach, Loss-Based Detection on a Variational Autoencoder (LBD-VAE), is a variant of the first approach. Instead of classifying the different traffic types in the latent layer, the LBD-VAE performs anomaly detection based on the reconstruction loss of the VAE after it has been trained. The LBD is based on the notion that an autoencoder is only able to reconstruct data that have previously been shown to it. Both of the proposed approaches use the underlying framework of the VAE. This means that the decisions about the VAE structure, what layers to use, when to use regularization, how the latent layer operates, etc., will mostly remain the same. The LBD-VAE could end up running on different tuning settings compared to the LLC-VAE, but the parameters available for tuning will mostly remain the same.

An overview of the LBD-VAE can be seen in Fig. 3. The figure is separated into two parts, "Model 1" and "Model 2". They are both part of the LBD-VAE model, but will be separated for easier understanding of how the second approach works. The first part, Model 1, is simply an implementation of the VAE developed by Kingma and Welling [21]. It will be trained separately from the second part, Model 2. To begin training the LBD-VAE, we will first create a dataset exclusively containing benign computer network traffic from the CSECICIDS2018 [49] dataset. The idea is that, by only training Model 1 on benign data, the VAE will only learn the patterns of those types of flows, and, as a side effect, be unable to recognize malicious data. Training only on benign data also means that the LBD-VAE has the potential to be a robust mitigation system against any type of attack.

Loss-based detection. When training the LBD-VAE, we first need to train the first part, Model 1, to completion. After having trained Model 1 on exclusively benign data, the idea is that the reconstruction loss will be low for benign data, and higher for malicious data. After that, the second part, Model 2, is trained based on the reconstruction loss from Model 1. During the training of Model 2, we will use a mixed dataset with both benign and malicious computer network traffic flows. The input data are fed through the whole model, starting from input x as seen in Fig. 3. Each flow will generate a KL-Loss and an R-Loss value, but only the R-Loss will be used to train Model 2. The R-Loss will be the only feature used in training a fully connected layer that outputs the unscaled class predictions. To generate a prediction loss value (P-Loss), we perform cross-entropy on the unscaled class predictions with either a softmax or sigmoid activation function[5]. When performing backpropagation based on the P-Loss to update the weights and node values of model 2, it is important to prevent backpropagation through model 1. In Tensorflow, this can be done by treating the reconstruction loss as a constant, using the function `tf.stop_gradient()`. This ensures that Model 2 will be trained separately from Model 1, and prevent the P-Loss from interfering with the weights and node values of Model 1. Failing to prevent backpropagation through Model 1 from Model 2 would cause the first part to learn from malicious data, which would defeat the purpose of the LBD-VAE approach.

4. Experimental setup

This section will be used to present the proposed algorithms from the previous section. We will explain the specifications of the system used to run the programs, the computer network setup that the datasets came from, and what tools that were used to create the proposed algorithms. Before moving on to the analysis section, we will explain how the models can be tuned, and suggest some presets.

4.1. System specifications

All the experiments were performed on a single system, using the two datasets CSECICIDS2018 [49] and CICIDS2017 [25]. The computer system runs on Windows 10 64-bit version. For deep learning, the most important hardware components are the memory, CPU and GPU. The system has 16GB of RAM, uses an Intel Core i7-5930K CPU with 6 cores at 3.50 GHz (stock frequency), and a Nvidia GeForce GTX 1080 graphics card with 8GB of dedicated GPU-memory.

When running the experiments, it is possible to adjust the batch size that is fed into the deep learning models as a hyperparameter. The number of flows in one batch should usually be as high as possible to ensure optimal learning and speed up the model convergence time [50,51]. Because the models mainly run on the GPU, the only limitation on batch size is the dedicated GPU-memory. After running a few experiments, we found a batch size of about 1000 flows per batch to be the optimal amount to ensure stable training. The deep learning models mainly run using the GPU, so the CPU is at about 15% usage during runtime when the GPU is at max memory load. RAM usage by the model is about 6.5GB during runtime, but it should be noted that this is because we load the whole training dataset into memory to reduce training times. It is possible to rely more on the disk to reduce memory usage, but this would result in slower training times.

4.2. Datasets

For this article, the Intrusion Detection System (IDS) dataset created in collaboration between the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC), CSECICIDS2018 [49], will be used to train the deep learning-based models. Furthermore, we will use the CICIDS2017 dataset [25] as a test set for the deep learning models[6]. For an analysis of the two IDS datasets and their features, see the paper by Sharafaldin et al. [52]. A similar dataset, the ISCXIDS 2012 by Shiravi et al. [34], containing

[5] Both functions can be used here, and can be considered equivalent to a binary classification problem: Sigmoid $= \frac{1}{1+e^{-x}} = \frac{e^x}{e^0+e^x} =$ Softmax

[6] Both datasets, CSECICIDS2018 and CICIDS2017, are from the Canadian Institute for Cybersecurity. This is also the case for the datasets ISCX2012 and NSL-KDD.

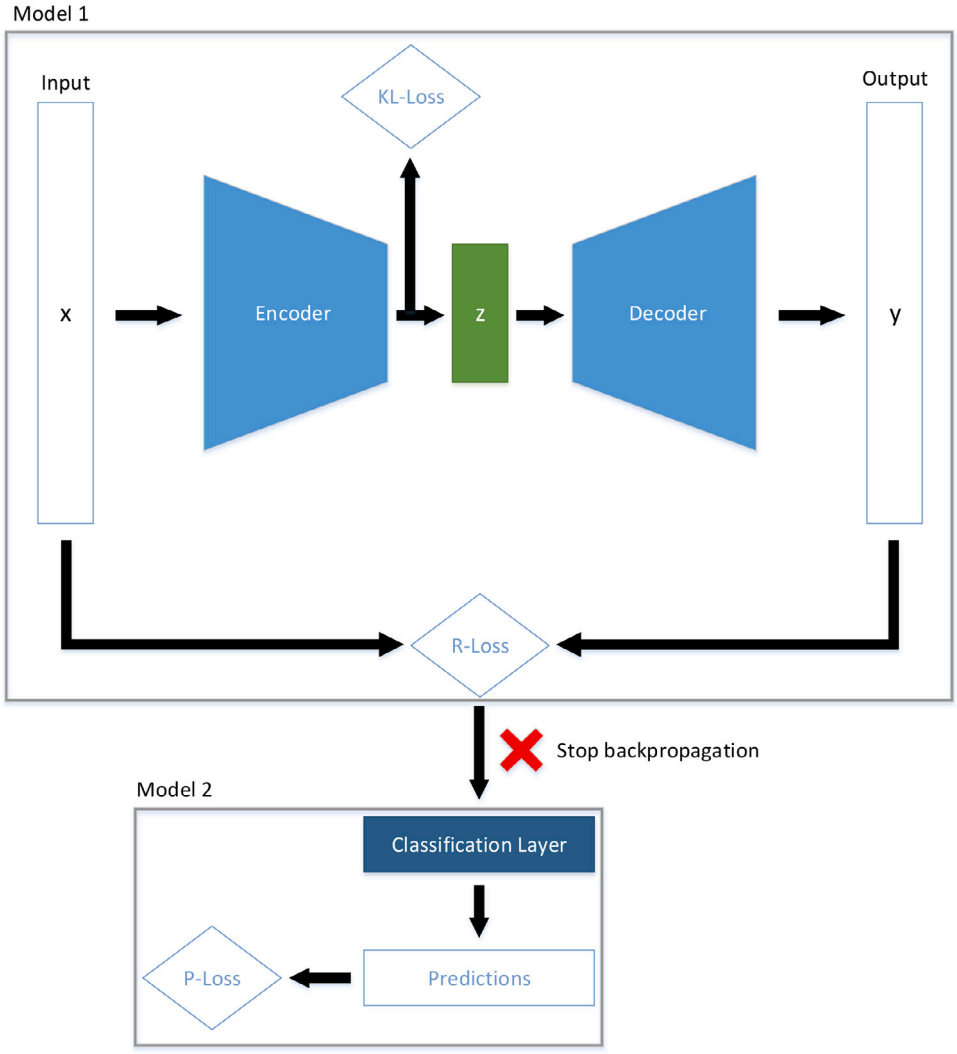


Fig. 3. Loss-based detection on a VAE. Rectangles represent node layers. Diamond shapes represent the loss values. The trapezoids represent the shrinking nature of the encoder, and the expanding nature of the decoder. The classification layer is a fully connected layer with a sigmoid activation function. The surrounding frames depict two stages of model training. First, Model 1 is trained to completion, then Model 2 is trained based on the reconstruction loss of Model 1.

many of the same types of attacks, has also been considered for use in this article. Due to the similarities, only one or two of these datasets were needed, and the newer 2017 and 2018 versions were chosen for their more recent content. For a list of the features used in both datasets, see Table 1. The features were extracted using CICFlowmeter[7] [53], turning the corresponding raw PCAP data into network flows in a CSV file format. The flows are bidirectional, where one flow represents both the forward and backward direction of a group of packets. TCP flows were terminated by a FIN packet, while UDP flows were terminated by a flow timeout. The CSV files are available for public download at [25,49].

CICIDS2017. This dataset, CICIDS2017 [25], contains a variety of both malicious and benign computer network traffic records, the majority of which are flows of normal traffic, labeled `Benign`. The DoS and DDoS attacks are labeled `DoS Slowloris`, `DoS Slowhttptest`, `DoS Hulk`, `DoS Goldeneye`, and `DDoS`. The DDoS attack was simulated using a tool called Low Orbit Ion Cannon (LOIC). LOIC is a tool that can be used for HTTP/TCP/UDP flooding of a server. A single LOIC does not generate enough traffic to cause a denial of service, and it is therefore typically used with many computers simultaneously, comprising a DDoS attack. In total, there are over 2.2 million flows of benign traffic, and over 380 thousand flows of DoS and DDoS traffic. For a complete enumeration of the data, their classes, and the attack vectors, see Table 2. The amount of normal traffic heavily outweighs the amount of malicious traffic. This unevenness of the distributions will be taken into account during training and testing. Deep learning algorithms need large amounts of data to be able to learn robust, general, and accurate deep features. Whether this dataset and the CSECICIDS2018 dataset provide a sufficient amount of flows for the two proposed approaches will be discussed in 5, Analysis. Although the distribution between normal and malicious traffic is uneven, there is still a considerable amount of malicious traffic that can be used, and previous research using CICIDS2017 has shown that this dataset can be used to achieve adequate results, see 2.2 from the Related work section. Using this dataset for training purposes can pose a challenge when it comes to the Slowloris, Slowhttptest, and Goldeneye attacks, as there are very few examples of those classes compared to the other types of traffic.

CSECICIDS2018. Like the 2017 dataset [25], this version consists of a variety of malicious traffic types, in addition to benign traffic [49]. The DoS and DDoS attacks are labeled `DoS Slowloris`, `DoS Slowhttptest`, `DoS Hulk`, `DoS Goldeneye`, `DDoS LOIC-HTTP`, `DDoS LOIC-UDP`, and `DDoS HOIC-HTTP`. The DDoS attacks

[7] CICFlowmeter is a tool created in the Java programming language, used to generate traffic flow data from network packets.

Table 1
All features used from the CICIDS2017 and CSECICIDS2018 datasets. The features are in order, starting from top to bottom, and from left to right. For an explanation of the features and what they represent, see the dataset information page [49].

Column 1	Column 2	Column 3	Column 4
Src IP	Flow IAT Min	Pkt Len Std	Subflow Fwd Pkts
Dst IP	Fwd IAT Tot	Pkt Len Var	Subflow Fwd Byts
Dst Port	Fwd IAT Mean	FIN Flag Cnt	Subflow Bwd Pkts
Protocol	Fwd IAT Std	SYN Flag Cnt	Subflow Bwd Byts
Flow Duration	Fwd IAT Max	RST Flag Cnt	Init Fwd Win Byts
Tot Fwd Pkts	Fwd IAT Min	PSH Flag Cnt	Init Bwd Win Byts
Tot Bwd Pkts	Bwd IAT Tot	ACK Flag Cnt	Fwd Act Data Pkts
TotLen Fwd Pkts	Bwd IAT Mean	URG Flag Cnt	Fwd Seg Size Min
TotLen Bwd Pkts	Bwd IAT Std	CWE Flag Count	Active Mean
Fwd Pkt Len Max	Bwd IAT Max	ECE Flag Cnt	Active Std
Fwd Pkt Len Min	Bwd IAT Min	Down/Up Ratio	Active Max
Fwd Pkt Len Mean	Fwd PSH Flags	Pkt Size Avg	Active Min
Fwd Pkt Len Std	Bwd PSH Flags	Fwd Seg Size Avg	Idle Mean
Bwd Pkt Len Max	Fwd URG Flags	Bwd Seg Size Avg	Idle Std
Bwd Pkt Len Min	Bwd URG Flags	Fwd Byts/b Avg	Idle Max
Bwd Pkt Len Mean	Fwd Header Len	Fwd Pkts/b Avg	Idle Min
Bwd Pkt Len Std	Bwd Header Len	Fwd Blk Rate Avg	Label
Flow IAT Mean	Pkt Len Min	Bwd Byts/b Avg	
Flow IAT Std	Pkt Len Max	Bwd Pkts/b Avg	
Flow IAT Max	Pkt Len Mean	Bwd Blk Rate Avg	

Table 2
Overview of traffic flow data in the CICIDS2017 [25] dataset.

Traffic type	Number of flows	Attack vector
Benign	2,273,098	None
DoS Slowloris	5,796	HTTP/TCP-SYN
DoS Slowhttptest	5,499	HTTP
DoS Hulk	231,073	HTTP
DoS Goldeneye	10,293	HTTP/TCP
DDoS LOIC-HTTP	128,027	HTTP

Table 3
Overview of traffic flow data in the CSECICIDS2018 [49] dataset.

Traffic type	Number of flows	Attack vector
Benign	7,372,557	None
DoS Slowloris	10,990	HTTP/TCP-SYN
DoS Slowhttptest	139,890	HTTP
DoS Hulk	461,912	HTTP
DoS Goldeneye	41,508	HTTP/TCP
DDoS LOIC-HTTP	576,191	HTTP
DDoS LOIC-UDP	1,730	UDP
DDoS HOIC-HTTP	686,012	HTTP

in the 2018 dataset are similar to the attacks in the 2017 version, with the exception of an LOIC-UDP, and a High Orbit Ion Cannon (HIOC) attack class. HIOC is an attack tool used to generate a flood attack by overflowing a victim with HTTP GET and POST requests. It was created as an improvement on the previously discussed LOIC, and to fix some of its shortcomings. From this dataset, we have more than 7.3 million flows of benign traffic available, and over 1.9 million flows of malicious traffic, comprising various types of DoS and DDoS attacks. Although similar to the 2017 version, most of the attack types have a considerable amount of flows each, the `Slowloris`, `Goldeneye`, and `LOIC-UDP` attacks have notably fewer flows than the other types.

The low amount of training data might negatively impact the accuracy of the proposed mitigation methods in relation to these types of attacks, as the two proposed approaches might have too little data to generalize properly. In 5, Analysis, we will discuss how much, if at all, this impacts the learning process in the two approaches.

Dataset files. For both datasets, we use the pre-generated CSV files publicly available for download [25,49]. The PCAP files are available as well, but, due to time constraints and the overall good quality of the CSV files, we will not generate new datasets. The current datasets will need transformation, however, to streamline the feature names and the number of features, to ensure consistency between the two datasets used, as well as to fix weaknesses in the datasets. One of the most notable weaknesses in CSECICIDS2018 is the lack of source IP, destination IP, and source port for a subset of the flow traffic. To solve this problem, we looked up the attack source and destination IP for each type of attack reported in the computer network, as seen in Fig. 4, and added them to each flow lacking these data. It was not feasible to recover the source ports from the original PCAP files in a reasonable time, and they will therefore be dropped. Another potential issue is the fact that both datasets use simulated benign data, and not recorded real-world traffic. There is no real way to mitigate this shortcoming without using other datasets. Some labels in the pre-generated CSV files might be mislabeled, which could lead to unwanted or erroneous results [23]. After inspecting the datasets, this problem seems to be the exception rather than the rule. As long as it is an anomaly in the data, and numerically clearly in the minority, the already noisy and random techniques of machine learning make their impact negligible.

For the two proposed approaches presented in this article, we use the CSECICIDS2018 dataset [49] for training and validation, and the CICIDS2017 dataset [25] for testing. The terms validation set and test set, are often used interchangeably to describe the same thing, a dataset for improving a machine learning model. In the remainder of this article, when we discuss validation sets, this means flows from the same computer network as the training set. The CSECICIDS2018 dataset will be split into two subsets: one is the training set, and the other the validation set. When discussing test sets, this means flows from a different computer network than the ones featured in the training and validation sets, but that follow the same probability distribution. The CICIDS2017 dataset will be used for this. Although the test dataset, CICIDS2017, has fewer malicious flow types than the training and validation set from CSECICIDS2018, all of the flow types in CICIDS2017 are present in CSECICIDS2018, and will thus not pose a problem as regards using CICIDS2017 as a test set.

A simplified version of the computer networks used to simulate the flows can be seen in Fig. 4. The CICIDS2017 network is on the left, while the CSECICIDS2018 network is on the right. In both computer networks, the benign data had multiple, different sources and destinations. The attacks always targeted the victim network, and came from a separate attack network. The IP addresses of the attack flows can be seen in Fig. 4, below their respective networks.

4.3. Tuning presets

An important part of any deep learning model is to tune its settings so that it is able to produce a useful result. To properly capture the nuances of the two proposed approaches, and in an attempt to explore which settings work well, we experiment with multiple, promising presets. These will be used for comparison purposes, to show which settings the models perform well on, and on which settings they come up short. Six presets will be presented initially, with more later, each with a reference name for easy lookup. The LBD-VAE will have its own presets, detailed in 5.2.

The hyperparameters that will be used for tuning are:

- Optimizer learning rate (LR)
- KL-Loss multiplier (KLM)
- Number of training batches iterated (Steps)

The dataset transformations used for tuning are:

- Scaling technique (ST)
  - No scaling (None)

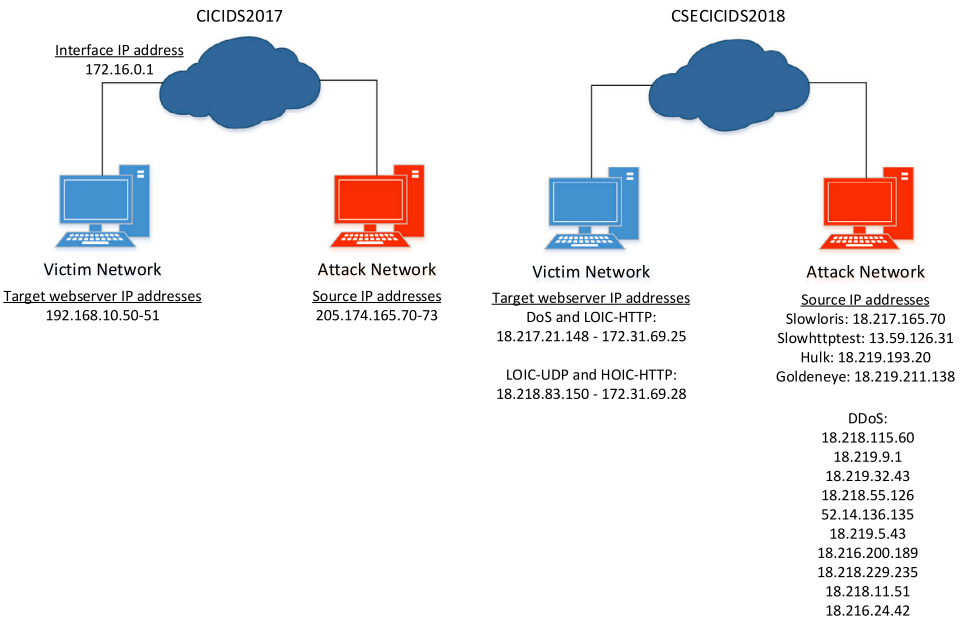


Fig. 4. Simple example of the computer network setups used to generate the CICIDS2017 dataset (left), and the CSECICIDS2018 dataset (right). For the full network setup of the CICIDS2017 dataset, see [52]. For the full network setup of the CSECICIDS2018 dataset, see [49].

- Min–Max normalization (N-18) or (N-18/17)[8]
- Logarithmic scaling (Log)

Other settings that will be used for tuning:

- Type of layers used for encoding and decoding (LT)
  - Convolutional layers (Conv)
  - Fully connected layers (Dense)
- Regularizing layer type (RLT)
  - Batch normalization (Batch)

Unless noted otherwise, all training done with convolutional layers will use a kernel size of 5 for each layer, a stride size of 2 for the first layer, and a stride size of 1 for the other two layers. This means that the 76 input features will be reduced to 38 abstract features in the first convolutional layer of the encoder, 34 features in the second layer, and 30 features in the third layer and the latent layer.

4.3.1. Presets

All the above presets were tested in [54]. Preset 4 and Preset 6 were retained as the most promising ones.

5. Analysis

The performance of the two proposed approaches is measured by logging key features of the models, using Tensorboard plots and confusion matrices. From Tensorboard, the graphs are plotted every 50 steps, where one step represents one batch (1 024) of flows being fed through a model. Graphs are plotted for the training dataset, the validation set, and the test set. The plotted graphs show the prediction accuracy, P-Loss, KL-Loss, R-Loss,[9] and total loss mean values, as they develop over time during training, validation, and testing. The prediction accuracy mean values represent the mean accuracy of each batch. In our dataset, there is an overweight of benign traffic in the datasets used, which will influence the overall accuracy score by giving more weight to benign accuracy. With this in mind, we will only use the mean accuracy score for development and performance improvement, and not as a metric to show model results. As an alternative, we can also use fewer benign flows, or only draw a certain percentage of each traffic type, from the datasets. To determine how well each model performs on different tuning settings, we will use confusion matrices to show the overall benign versus malicious flow traffic accuracy, as well as the accuracy within individual attack classes, separated and unaffected by the other flow accuracy scores.

5.1. LLC-VAE results

The first approach that will be analyzed is the LLC-VAE. We have trained the model using the settings defined by the presets from 4, Experimental setup, defined in Table 4 as reported in [54]. For all training runs, we will use the CSECICIDS2018 dataset, with a 60–40 split between the training set and validation set, respectively, unless noted otherwise. To prevent learning bias, the training set has been modified to contain an equal amount of benign and malicious flows. For the test set, we will use the whole CICIDS2017 dataset.

5.1.1. Further adjustments

In the following subsections, we will present further adjustments to the two most promising presets, Preset 4 4 and Preset 6 4f. Compared with the other presets, Preset 4 and Preset 6 had the best test results overall, and achieved much higher test accuracy, as reported in [54]. We use test accuracy as the deciding factor for choosing to build on these presets, because it is the best metric to show how well the LLC-VAE generalizes. Further adjustments will be made to examine how the LLC-VAE can be further improved, and to determine whether it can be considered as a method for creating a mitigation system.

LIME. In addition to the feature selection carried out by the convolutional layers, and the selection through dimensionality reduction by the encoder of the LLC-VAE model, we can manually select which input features the model should learn from. Using LIME [55], we can analyze how the presets weighted each feature, to find out which ones had

[8] Normalization uses sampled min and max values from the used datasets. A notation of (N-18) means the values are sampled from CSECICIDS2018, while a notation of (N-18/17) means the values are sampled from both CSECICIDS2018 and CICIDS2017.

[9] As discussed in Section 3.2

Table 4
All presets parameters. The parameters that change between presets have been highlighted in gray. Preset 6 (in bold) only uses P-Loss to improve, which means that it is not a VAE, just a simple convolutional neural network that performs dimensionality reduction.

(a) Preset 1

Parameter	Value
LR	$1 \cdot 10^{-2}$
KLM	1
Steps	15,000
ST	None
LT	Conv
RLT	Batch

(b) Preset 2

Parameter	Value
LR	$1 \cdot 10^{-2}$
KLM	$1 \cdot 10^{-2}$
Steps	15,000
ST	N-18
LT	Conv
RLT	Batch

(c) Preset 3

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	$1 \cdot 10^{-2}$
Steps	30,000
ST	N-18/17
LT	Conv
RLT	Batch

(d) Preset 4

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	$1 \cdot 10^{-4}$
Steps	30,000
ST	Log
LT	Conv
RLT	Batch

(e) Preset 5

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	$1 \cdot 10^{-4}$
Steps	30,000
ST	Log
LT	Dense
RLT	Batch

(f) **Preset 6**

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	None
Steps	30,000
ST	Log
LT	Conv
RLT	Batch

Table 5
Top 40 features remaining after using LIME on the LLC-VAE with Preset 4 4d. The features are ordered from most to least impactful, from top to bottom and left to right.

Column 1	Column 2	Column 3	Column 4
Bwd IAT Std	Bwd Pkt Len Mean	Fwd Header Len	Fwd IAT Tot
Pkt Len Mean	Bwd Pkt Len Max	Pkt Len Std	Bwd Pkt Len Min
Fwd IAT Max	Active Min	Fwd Pkt Len Max	Bwd IAT Tot
Bwd IAT Min	Idle Max	TotLen Fwd Pkts	Flow IAT Max
Dst Port	Flow Duration	Subflow Fwd Byts	Bwd Header Len
Init Bwd Win Byts	Bwd Pkt Len Std	Flow IAT Min	Subflow Fwd Pkts
Pkt Size Avg	Fwd IAT Std	Fwd IAT Mean	Active Std
Pkt Len Max	Bwd IAT Mean	Flow IAT Mean	TotLen Bwd Pkts
Idle Mean	Pkt Len Var	Fwd Seg Size Avg	Idle Min
Init Fwd Win Byts	Bwd IAT Max	Fwd IAT Min	Pkt Len Min

the largest, and which ones had the smallest impact on the learning process. LIME shows how much impact each input feature from a flow had on a prediction. An example of how LIME visualizes the prediction probabilities for a single flow is shown in Fig. 5. The flow from the example figure has been predicted to be a DoS Hulk attack, with a probability of 62%. On the right-hand side, we see the six most impactful input features ordered from most to least impact. Features on the side named "Not DoS Hulk" weigh against this particular flow being predicted to be a DoS Hulk attack, and vice versa for the opposite side named "DoS Hulk". The number next to each feature name represents how much they are weighted by the model, rounded off to the closest, second decimal point. We can use LIME to understand how the LLC-VAE classifies flows, as well as for manual feature removal, to ensure better generalization.

To find out which input features were the most influential on the LLC-VAE training process, we ran LIME on the model after it had been trained using Preset 4, with the same training and validation datasets that were used previously. Running LIME is a time-consuming process, so we have shortened the process by using a sample size of 50 000 flows. This produced a list of the most impactful features, sorted in order.

Results after feature selection with LIME. To test whether manual feature selection is constructive for the LLC-VAE model performance, we tried to remove the least impactful features, only keeping the top 40, as seen in Table 5. By removing the least impactful features, we allow the model to give more emphasis to the remaining features. This could help prevent overfitting, since the model no longer tries to learn from less important data. Another method would be to select individual features if they are deemed to be potential causes of overfitting. This would be much more time-consuming, and will not be done for this article.

The confusion matrices from Fig. 6, and Fig. 7, shows the test results of Preset 4 and Preset 6, respectively, before performing manual feature selection using LIME.[10] Preset 6 showed slightly better benign accuracy than Preset 4, but both models had trouble classifying the malicious traffic flows. Preset 4 was particularly poor at classifying the attack `DoS SlowHTTPTest`, while Preset 6 was mostly unable to classify the attacks `DoS Slowloris` and `DoS SlowHTTPTest`. Recall Table 3, showing the number of flows for each flow type from the training dataset, CSECICIDS2018. `DoS Slowloris` had relatively few samples compared to the other attack types, which could explain why the convolutional network, using Preset 6, was unable to generalize well enough to correctly classify this attack. The attack `DoS SlowHTTPTest` had relatively many flow samples in the training set, but the LLC-VAE and the convolutional network were still unable to generalize well, using Presets 4 and 6, respectively. This conceptualizes the idea that the attack `DoS SlowHTTPTest` is too similar to benign traffic flows for the LLC-VAE and the convolutional network, using Presets 4 and 6, to classify it correctly.

After having used LIME to find the least impactful features, we kept the top 40 most impactful features, as seen in Table 5, and ran more tests on variations of Preset 4 and Preset 6, as seen in Table 6. Since differentiating between individual attack classes seemed to be difficult for the LLC-VAE to handle, we decided to compress them into one class named `malicious`, and instead perform binary classification. In addition, we also changed the latent layer size, by changing the convolutional layers. For Preset 4 and Preset 6, the kernel size was 5 for each layer, and the first layer had a stride size of 2, as described in 4.3. The kernel size remains the same for these tests, at a size of 5, but now each of the layers has a stride size of 1. Hence, the top 40 input features, 5, will be reduced to 28 abstract features in the latent layer, through dimensionality reduction. The confusion matrix for the test results for Preset **4a** 6a can be seen in Fig. 8, while the test results for Preset **6a** 6b can be seen in Fig. 9. From both figures, we can see that the benign accuracy has decreased by about 2%, but that the malicious accuracy has increased dramatically. As discussed previously, Preset 6, and inherently 6a, is a simple, reducing convolutional network that is used for baseline comparison with the LLC-VAE performance. The LLC-VAE performed less than 1% worse in terms of benign accuracy, but almost 5% better in terms of malicious accuracy. Overall, the LLC-VAE performed better in this test, but could still be improved with some fine tuning.

When we trained the six presets [54], reducing the KL-Loss multiplier showed some improvement in overall model performance. We

[10] Note that, at the bottom of the confusion matrices, there are no LOIC-UDP or HOIC-HTTP attacks. This is because the test dataset contains no samples of those attack types.

Table 6
Modified Preset 4 4d and Preset 6 4e. The parameters are as follows; `Preset` is the base preset, CT is short for classification type, `Features` specifies the number of input features, KS is short for kernel size. `Stride` specifies the stride size in the format $i_1 - i_2 - i_3$, where i is the size of the first, second, and third layer, respectively.

(a) Preset 4a

Parameter	Value
Preset	4
CT	Binary
Steps	30,000
Features	40
KS	5
Stride	1-1-1

(b) Preset 6a

Parameter	Value
Preset	6
CT	Binary
Steps	30,000
Features	40
KS	5
Stride	1-1-1

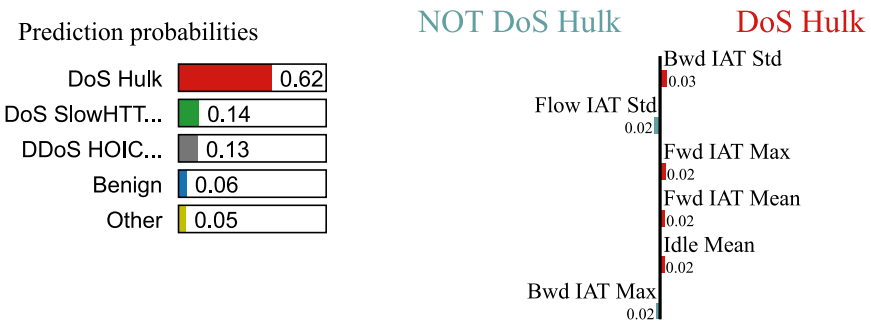


Fig. 5. LIME example figure for a single flow. Result achieved by running LIME on the LLC-VAE using Preset 4 4d.

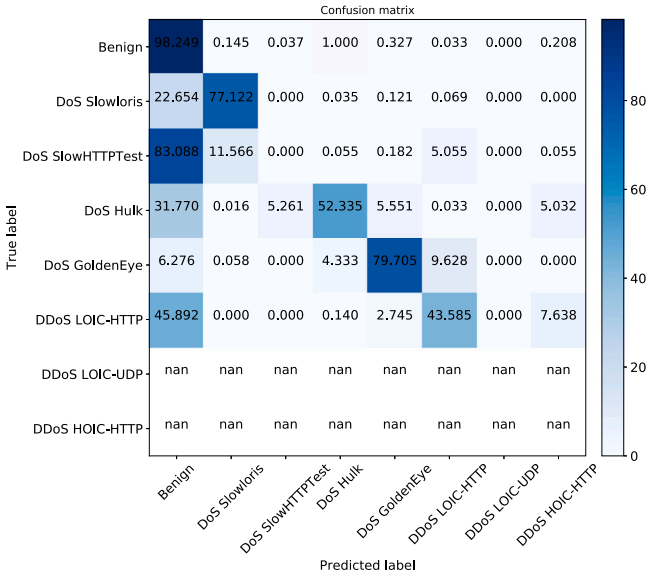


Fig. 6. Confusion matrix of test dataset for preset 4.

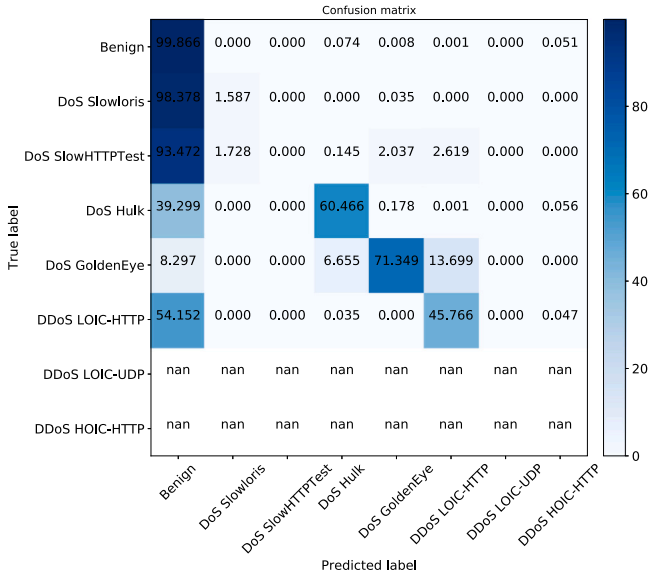
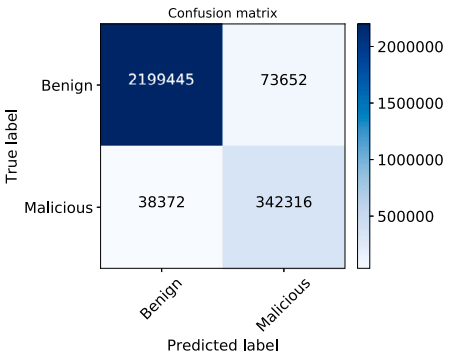


Fig. 7. Confusion matrix of test dataset for preset 6.

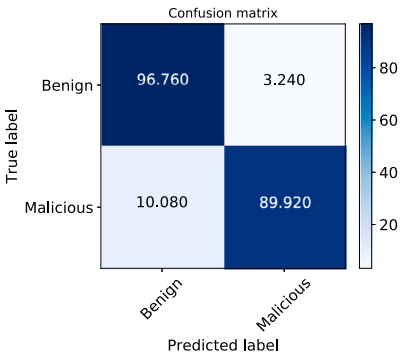
tried to reduce it even further using Preset 4b 7a, changing the multiplier to $1 \cdot 10^{-6}$ and the number of steps to 50 000. The result of this can be seen in Fig. 10. The benign accuracy was mostly the same as when using a KL-Loss multiplier of $1 \cdot 10^{-4}$, but the malicious accuracy increased by more than 3%, compared to the results of Preset 4a 8. Further reducing the KL-Loss multiplier caused the LLC-VAE model performance to degrade, indicating that a multiplier of $1 \cdot 10^{-6}$ will provide the best results.

Adjusting receptive field. The tuning presets that were used to train the LLC-VAE worked as a way of trying different methods to improve the model, as well as to see which settings worked, and which did not. Both Preset 4a 6a and Preset 6a 6b used convolutional layers for the encoder and decoder. Within convolutional layers, it is possible to adjust the kernel size and the stride size of the sliding window, as discussed in Section 3.2. We can thereby adjust the effective size of the receptive field of the latent layer to make it learn an abstraction of the relations between a larger or smaller number of features in the input layer.

All of the six original presets 4 use a kernel size of 5, where the first layer has a stride size of 2, which leads the latent layer to have an effective receptive field size of 21. This means that each latent layer node is able to learn relations between 21 adjacent input features. In order for each node in the latent layer to learn from the relations between all the original 76 input features, as seen in Table 1, the kernel size of each layer needs to be set to 13, and the first two layers need to have a stride size of 2. A figure showing how the theoretical receptive field increases in this case can be seen in 11.

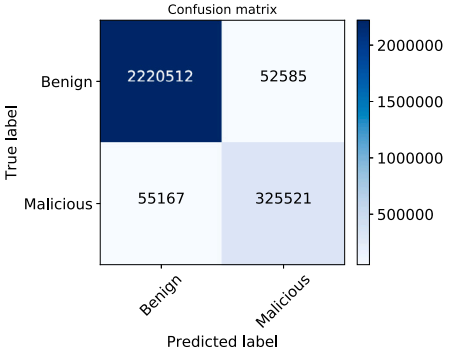


(a) Amount of flows for each classification.

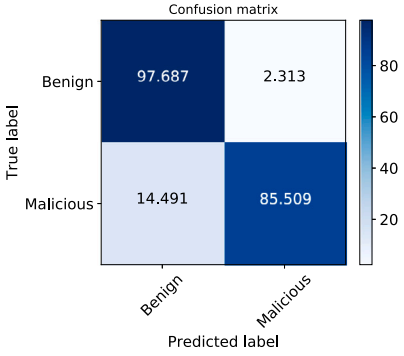


(b) Accuracy values for each classification.

Fig. 8. Confusion matrix of the test results on preset 4a.



(a) Amount of flows for each classification.



(b) Accuracy values for each classification.

Fig. 9. Confusion matrix of the test results on preset 6a.

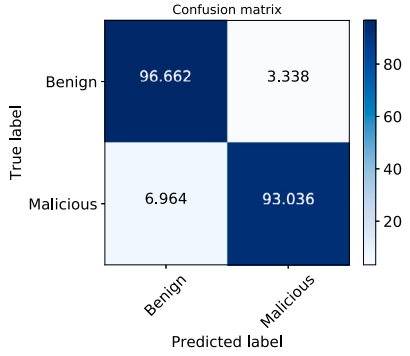


Fig. 10. Confusion matrix of the test results on preset 4b, after adjusting the KL-Loss multiplier from $1 \cdot 10^{-4}$ to $1 \cdot 10^{-6}$.

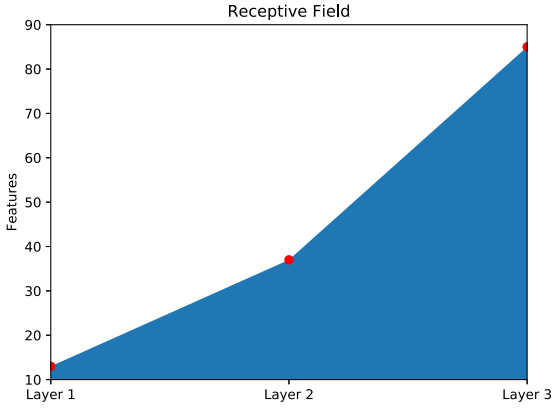
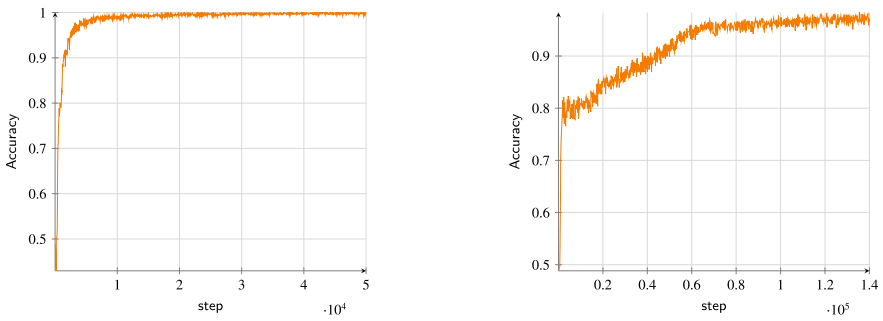


Fig. 11. Example figure of the receptive field size for a model using three layers, a kernel size of 13, and the first two layers having a stride size of 2.

Thus far, the best results have been achieved after training the model on Preset **4b** 7a with a KL-Loss multiplier of $1 \cdot 10^{-6}$, as seen in Fig. 10, with the reduced input feature set from Table 5. To achieve an effective receptive field size that covers all the 40 features from the reduced input feature set, we adjusted the kernel size of the convolutional layers to 7, and the first and second layer to have a stride size of 2, as seen in Preset **4c** 7b. This gives the latent layer nodes a theoretical receptive field size that covers 43 features. In Fig. 12(a), we see the accuracy plot of the LLC-VAE performance using Preset 4b on the validation set. In Fig. 12(b), we see the accuracy plot of the LLC-VAE performance using Preset 4c on the same set. We can see from Fig. 12 that increasing the receptive field size did not improve the validation accuracy, but rather degraded it. The model converged to a solution at about step 130 000.

Other changes. To better understand what works well and what does not for the LLC-VAE, we have tried further fine tuning of the model based on Preset 4, and with the improvements done with LIME and the modifications on the convolutional layers. We have tried to adjust the KL-Loss multiplier, and ended up with a multiplier of $1 \cdot 10^{-6}$, as seen in Preset 4b 7a, to achieve the best performance. Adjusting the P-Loss or R-Loss multipliers showed no improvement. Further adjustments of the convolutional layers showed that the optimal settings for the kernel sizes were 5 with a stride size of 1 for all layers, leading to a latent layer size of 28. Both larger and smaller latent layer sizes produced poorer end results, even though larger latent layer sizes caused the model to converge faster.



(a) Preset 4b (b) Preset 4c

Fig. 12. Comparison of the validation accuracy when using two different receptive field sizes with the LLC-VAE on Presets 4b and 4c.

Earlier, when discussing the datasets, we saw that some of the malicious traffic flows from the training set CSECICIDS2018 had few samples, see Section 4.2. The attacks `DoS Slowloris`, `DoS Goldeneye`, and `DDoS LOIC-UDP` had relatively few samples compared to the other attack classes. To test whether this had a negative impact on the training process, we removed them for a training run. Contrary to our initial belief, removing these attacks from the training set decreased model accuracy. The training and validation accuracy remained the same as before, but the test accuracy decreased. This could point to the model generalizing better when subjected to a variety of different attack types with few samples.

The features that were removed after using LIME included the source IP and destination IP addresses. It is common for mitigation systems, when determining whether a network packet or flow is an attack or not, to analyze the source and destination IP addresses. By doing so, a mitigation system can determine the intent of the source by connecting their IP address to their behavior and traffic frequency. The LLC-VAE performed better, however, after removing the IP addresses, improving overall model performance for the test dataset. We performed a training run on the settings used with Preset 4 4d, where we removed the IP address input features, and got the test results seen in Fig. 13(b). Fig. 13(a) shows the confusion matrix for Preset 4 when performing binary classification, which used the IP addresses during training. The benign detection accuracy decreased by about 1% when the IP addresses were removed. The malicious detection accuracy increased by about 16%, which is a significant leap in overall model performance. This points to the LLC-VAE overfitting on the IP address input features, learning specific IP addresses instead of the relationships between the IP addresses and the other input features.

5.2. LBD-VAE results

The second approach, LBD-VAE, will use similar tuning settings to the first approach. The presets will not be used directly, but variations will be tested, based on the findings concerning what worked well for the LLC-VAE. These new settings can be seen in Table 8.

The LBD-VAE consists of two different models that need to be trained in order, where the first model is trained in isolation from the second one, as seen in Fig. 3. As previously, all training runs will use the CSECICIDS2018 dataset with a 60–40 split between the training set and the validation set, respectively. To prevent learning bias, the training set has been modified to contain an equal amount of benign and malicious flows.

5.3. Comparison of results

Convolutional neural networks (CNNs) have proven extremely successful in various contexts and, in this paper, we have therefore chosen to compare our approach to CNNs. In our proposed methods, we use convolutional encoding and decoding layers in the variational autoencoders, and by comparing them to a pure CNN we try to detect whether the variational autoencoders actually make a difference. A CNN performs dimensionality reduction and can act as a simple baseline that can be compared to the data reduction capabilities of variational autoencoders.

At the currently best settings, using Preset 4b 7a, the LLC-VAE achieved a benign accuracy of 99.59% and a malicious accuracy of 99.98% on the validation set,[11] as seen in Fig. 14(a). This means that, based on the validation set, 4 out of every 1000 legitimate flows, and the corresponding source IP addresses, would be blocked. At the same time, only 2 out of every 10 000 malicious flows would be let through the LLC-VAE. The LBD-VAE prioritized benign accuracy, which is better than prioritizing malicious accuracy. However, the overall accuracy, as seen in 14(b), was much lower than the accuracy of the LLC-VAE.

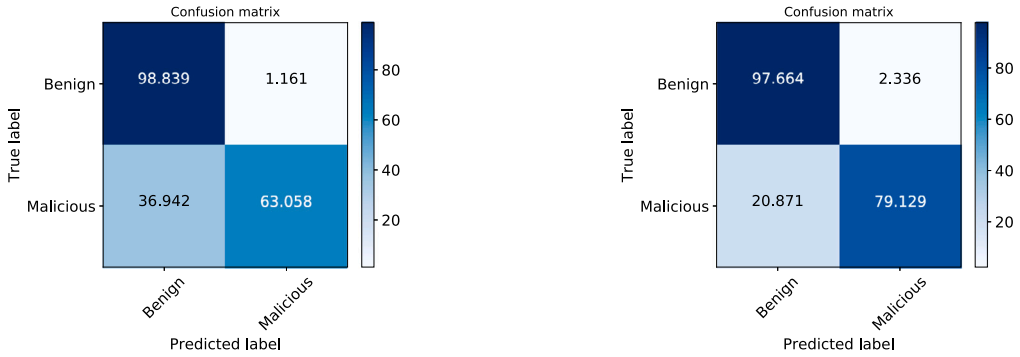
Compared with the simple convolutional neural network, the LLC-VAE performed better overall on the test set. The best test results for the LLC-VAE were achieved using Preset 4b 7a, as can be seen in Fig. 15(a). The best test results for the simple convolutional network were achieved using Preset 6a 6b, as can be seen in Fig. 15(b). The LLC-VAE had less than 1% poorer benign accuracy, but substantially increased malicious accuracy, at about 7.5% higher. For both models, the validation accuracy was significantly higher, but, ultimately, the test accuracy gives a better understanding of how the models would perform in general, since the test set comes from a different computer network. Hence, there are fewer similarities that could cause overfitting, as the datasets are internally correlated to a large degree.

Using the LLC-VAE as a part of a mitigation system is reasonable, even though 3 of every 100 benign flows would be blocked. However, relying solely on the detection capabilities of the LLC-VAE would not be a good solution. Instead, it should be incorporated as a part of a larger mitigation system.

5.3.1. Flow processing time

After fully training the LLC-VAE on the best preset, Preset 4b 7a, we measured the speed of the model. The LLC-VAE was estimated to be able to process one batch in about 10.8 ms ± 0.3 ms, where one batch contains 1024 flows. That is between 90 and 95 batches per second, which translates to about 92 000 to 97 000 traffic flows per second. These measurements are only from the LLC-VAE, and do not include the time taken for data transformations. Furthermore, the model has not been optimized for speed either. If this approach were to be added as a part of a full mitigation system, the overall flow processing time would be expected to increase, so these measurements only give an indication of what to expect from the LLC-VAE.

[11] We use validation set here for comparison between the LLC-VAE and the LBD-VAE, since we did not do any test runs using the LBD-VAE because of poor training performance.



(a) With IP addresses. (b) Without IP addresses.

Fig. 13. Comparison of test results of Preset 4, with and without IP addresses.

Table 7
Two modifications on Preset 4a 6a. As before, KLM is short for KL-Loss multiplier.

(a) Preset 4b

Parameter	Value
Preset	4a
KLM	$1 \cdot 10^{-6}$
Steps	50,000
Features	40
KS	5
Stride	1-1-1

(b) Preset 4c

Parameter	Value
Preset	4a
KLM	$1 \cdot 10^{-6}$
Steps	140,000
Features	40
KS	7
Stride	2-2-1

Table 8
Settings for the LBD-VAE Model. Steps 1 is the number of iterations for "model 1", while Steps 2 is the number of iterations for "model 2".

(a) LBD-preset 1

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	1
Steps 1	20,000
Steps 2	30,000
ST	Log
LT	Conv
RLT	Batch
Features	76
KS	5
Stride	2-1-1

(b) LBD-preset 2

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	4
Steps 1	20,000
Steps 2	30,000
ST	Log
LT	Conv
RLT	Batch
Features	40
KS	5
Stride	1-1-1

(c) LBD-preset 3

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	$1 \cdot 10^{-6}$
Steps 1	1,500
Steps 2	30,000
ST	Log
LT	Conv
RLT	Batch
Features	40
KS	5
Stride	1-1-1

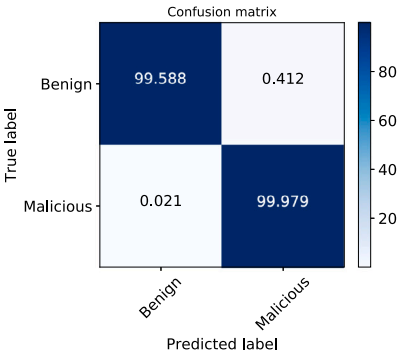
(d) LBD-preset 4

Parameter	Value
LR	$1 \cdot 10^{-4}$
KLM	$1 \cdot 10^{-6}$
Steps 1	100,000
Steps 2	30,000
ST	Log
LT	Conv
RLT	Batch
Features	40
KS	7
Stride	2-2-1

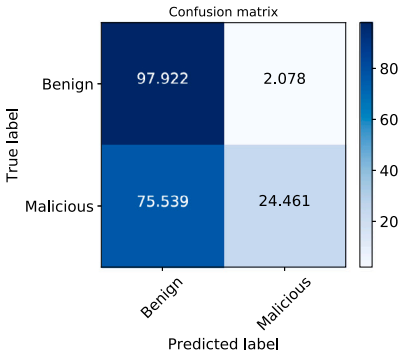
5.4. Comparison with related works

As mentioned in the Related Works section, quite a few studies have used the CICIDS datasets. However, there are a few obstacles when trying to compare these results directly to the results in this paper. Most of the studies report aggregated results for all the attacks in the datasets, not only DoS and DDoS attacks, as in this work. In addition, in order to make the testing phase as realistic as possible, we have trained and validated the models using the CICIDS2017 [25] dataset and tested them using the CSECICIDS2018 [49] dataset. This approach is different from the approach taken in other studies. Nevertheless, in Table 9 we show some results from related works that make the interpretation of our results more transparent. The result shown in the first row of the table is for the Deep Belief Network (DBN) model proposed by Manimurugan et al. [56], which is reported to perform better than the results of the more conventional models seen in the next four rows. The Partial model in the last row is a Partial Decision Tree approach that reports a very high accuracy [57].

The overall accuracy of our LLC-VAE model, corresponding to the confusion matrix in Fig. 15(a), is 94.85%. It performs similarly to the other neural networks, but, again, it should be noted that the testing phase was conducted using another dataset. When comparing the results of the LLC-VAE model to the CNN model that was exposed to exactly the same datasets, the accuracy of the CNN model was 91.60%, corresponding to the confusion matrix in Fig. 15(b).

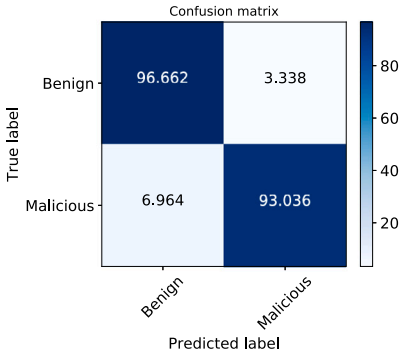


(a) LLC-VAE validation accuracy.

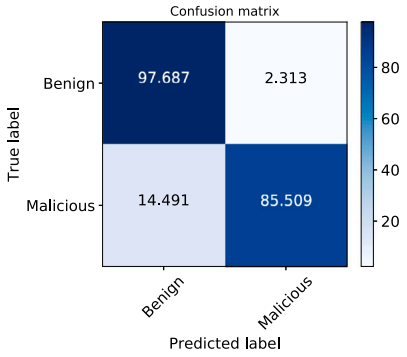


(b) LBD-VAE validation accuracy.

Fig. 14. Validation accuracy comparison.



(a) LLC-VAE test accuracy.



(b) CNN test accuracy.

Fig. 15. Test accuracy comparison.

Table 9
Model accuracy reported by studies focusing on the DoS and DDos parts of the CICIDS 2017 dataset.

Model	Accuracy (%)
DBN [56]	96.67
SVM [56]	95.55
RNN [56]	94.40
SNN [56]	93.30
FNN [56]	92.25
Partial [57]	99.97

6. Conclusion

This article has presented two different approaches to DoS and DDoS mitigation using deep learning. Both approaches build on the framework of a Variational Autoencoder, using pre-generated datasets to classify different types of computer network traffic. These datasets provide the two approaches with input features from network traffic flows, allowing them to learn to filter normal and malicious traffic.

The first approach, LLC-VAE, is a latent layer classification network that utilizes the latent layer encodings of a Variational Autoencoder. The LLC-VAE showed clear signs of overfitting to the training dataset in the beginning, but the generalization capabilities have been greatly improved through various tunings. Improved generalization has in large part been achieved by adjusting the KL-Loss weight, manual feature selection, and tuning of the convolutional layers.

The performance of the LLC-VAE has been tested on two different datasets, split into a training set, a validation set, and a test set. The training and validation sets contain data that are internally correlated; hence we use the test set to record model results. When we compared the LLC-VAE performance to a simple convolutional network of a similar structure, the LLC-VAE was better overall at generalizing, achieving better results on the test set. At the core of the VAE is the KL-Loss value, which regulates the latent layers. Lowering the KL-Loss weight improved overall model performance. When the weight was reduced past a certain point, however, the overall performance of the LLC-VAE declined. This means that using a Variational Autoencoder over a standard autoencoder had a positive impact on the ability of the model to classify normal and malicious traffic flows.

The second approach, LBD-VAE, relies on the VAE to separate normal and malicious traffic flows into two different probability distributions. A Loss Based Detector is applied to the reconstruction loss, classifying traffic flows using a linear classification layer. The VAE is trained exclusively on normal traffic, while the LBD is trained on a combination of normal and malicious traffic. Since the VAE is only trained using normal traffic, the LBD-VAE is theoretically capable of classifying DoS and DDoS attack types not seen during training. This is because malicious flows would not fit into the probability distribution of the normal flows. The results from the training runs using the LBD-VAE showed that it is currently unsuitable for use as a part of a mitigation system. The VAE had difficulty separating the two probability distributions, and the classifier therefore achieved unsatisfactory results. Still, if the LBD-VAE were to be further tuned, it has the potential to become a viable mitigation method.

This article has proven that deep learning-based techniques can be effective at countering DoS and DDoS attacks. The second approach, LBD-VAE, does not currently perform well enough to be used as a mitigation system, but it is theoretically promising. More research should be conducted to explore other possibilities. The best test results overall were achieved by the LLC-VAE, which was able to classify benign and malicious traffic at upwards of 97% and 93% accuracy, respectively,

on simulated data in the generalized case. The LLC-VAE has proven to be capable of competing with traditional mitigation methods, but will need further tuning to ensure even better performance.

CRediT authorship contribution statement

Eirik Molde Bårli: Conceptualization, Methodology, Software, Writing, Visualization, Validation, Resources. **Anis Yazidi:** Methodology, Writing, Investigation. **Enrique Herrera Viedma:** Writing – review & editing, Validation. **Hårek Haugerud:** Conceptualization, Methodology, Writing, Project administration, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Autoencoders

An autoencoder is a deep learning framework that utilizes an NN framework to perform a variety of different tasks, primarily for unsupervised learning, where the backpropagation target values are set to be equal to the input. Variations of an autoencoder allow for classification, anomaly detection, and generative tasks, among other uses. Autoencoders are feedforward networks, meaning that the representative ANN is a directed acyclic graph, and that they use backpropagation for training. The composition of an autoencoder always features at least two parts, an encoder and a decoder. The encoder encodes the input x to a hidden layer h, selecting which dimensions to learn from with the function $h = f(x)$. The decoder tries to make a reconstruction, r, of the input from the hidden layer, with the function $r = g(h)$. The concept of autoencoders have been around for over a decade, with one example, from Bourland and Kamp, dating back to 1988[58]. Historically the hidden layer mapping has been deterministic, but more modern solutions use stochastic mapping, with the functions $p_{encoder}(h|x)$ and $p_{decoder}(x|h)$[41, p. 499].

The reconstruction of the hidden layer is not a perfect replication, and nor should it be, and it is recipient to noise. For an autoencoder model to be useful, it needs to generalize over training data to avoid ending up with a model that performs poorly on foreign data, as we will see later in Section 5, Analysis. One of the major advantages of using this kind of model is its capability to determine which parts of the input are important, by forcing it to learn the useful properties of the data it is given during training. In a way, when using an autoencoder, we are often more interested in the encodings of the data and the latent layer representations than we are in the actual reconstruction of the decoder. A good autoencoder is one that is able to properly select which dimensions of the input to use in the hidden layer, and to what degree. This enables the decoder to produce a good approximation on as few dimensions as possible. Two main methods for dimension selection are used in autoencoders: dimensionality reduction and regularization. Dimensionality reduction is when each hidden layer in the model contains fewer nodes than the preceding layer. This forces the model to select the most important features from the previous layer, and discard the least important ones. Regularization selects the nodes with the greatest positive impact on the model's result, and lessens the impact of the other nodes. There are multiple, different regularization techniques, some of which will be discussed in the following sections.

The imperfect reconstruction of the input data can be both an advantage and a disadvantage in machine learning. Autoencoders, like other machine learning algorithms, must be applied to problems they are fit to solve. Reconstructing the input is only possible if the model has seen similar data during training. If an autoencoder is only given images of cats during training, it will not be able recognize images of, e.g., birds. For this article, this means that the model will not be able to classify a DDoS or DoS attack if we only train it on normal data. However, because of this precise property, an autoencoder could be used as an anomaly detector, essentially differentiating the two classes by only recognizing normal data, and being unable to recognize anomalies.

Algorithm 1: A simple autoencoder

for *each input* x **do**
  Feedforward x and compute activations for each layer;
  Sample the hidden layer z;
  Obtain output y;
  Measure deviation of y from x;
  Backpropagate to update weights and node values;
end

A.1. Sparse autoencoder

A sparse autoencoder (SAE) is a regularized variation of an autoencoder with potentially more nodes in the hidden layer than in the input layer. This means that, to extract useful features to learn from in the input layer, the SAE appends a regularizing function to the normal loss function of an autoencoder [59][41, p. 502]. Because of this, the hidden layer only has a few select nodes active at a time, forcing the SAE to learn the most useful properties of the input. Furthermore, each type of input activates different nodes in the hidden layer. There is normally an overlap between properties of, for example, network packets. Hence they will often activate some of the same nodes, but the point of an SAE is to only activate the relevant nodes in the hidden layer, customized to those types of input. This is called the sparsity constraint. In theory, this means that the total amount of active nodes in the hidden layer could be as large as the different properties of the input, leading to some SAEs having larger hidden layers than input.

The loss function of an autoencoder can be described as $L(x, g(f(x)))$, where f is an encoder and g is a decoder. The goal is to minimize the difference between the input and the output. This is done by penalizing $g(f(x))$ for being dissimilar from input x. A sparse autoencoder adds a sparsity penalty to the loss function, which is commonly done in one of two ways. One way is to use L1 regularization, also called Lasso Regression,[12] which ends up looking like this: $L(x, g(f(x))) + \Omega(f(x))$, where $\Omega(f(x))$ is the regularization term. Note that regularized networks typically regularize the weights that connect the nodes. However, SAEs apply regularization to the activations of the nodes. The term $\Omega(f(x))$, where $f(x)$ is the hidden layer h, can be simplified as

$$\Omega(h) = \lambda \sum_i |h_i|$$

Here λ is a hyperparameter, and the following formula is the absolute sum of all activations of the nodes i in the hidden layer.

Another way to apply a sparsity penalty is by using Kullback–Leibler divergence, or KL-Divergence for short. KL-divergence is a measure of the divergence between two probability distributions. It is used to measure their similarities or dissimilarities. Given the probability distributions p and q, KL-divergence is a measure of how well q approximates p, by calculating the cross-entropy $H(p, q)$ minus the entropy $H(p)$, to get the KL-term $D_{KL}(p \parallel q) = H(p, q) - H(p)$. The KL-divergence is a central part of the Variational Autoencoder, which will be explained in detail in Appendix A.3. Niyaz et al. proposed using a stacked sparse autoencoder in order to detect DDoS attacks in software-defined networking (SDN) [22]. In their paper, they use KL-divergence for their sparse autoencoders to put a constraint on the hidden layer in order to maintain low average activation values. They also present a

[12] From statistics. Lasso Regression shrinks the coefficient of less important features, reducing their impact.

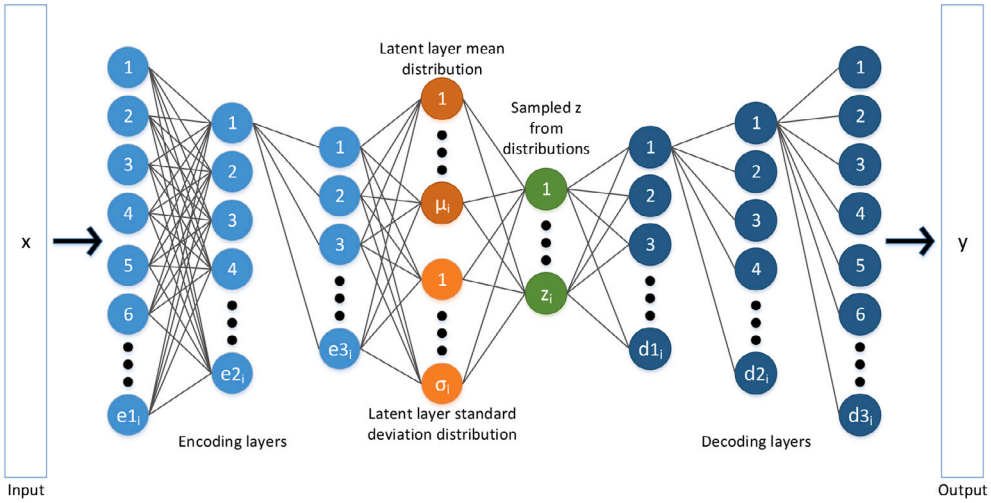


Fig. 16. A VAE with fully connected layers. Each vertical line of nodes represents one layer. The latent mean and latent variance layers are two separate layers, which the layer z samples from. All nodes in a layer have weights connected to the nodes in the adjacent layer, like seen in the first two encoding layers. (For simplification, only some of the weights are drawn in this figure).

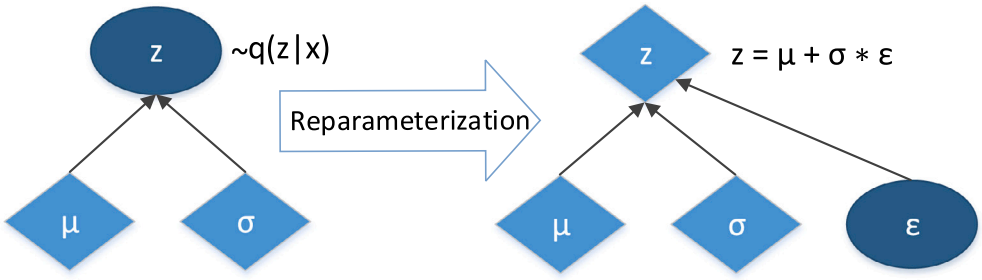


Fig. 17. Diamond shapes represent deterministic dependencies, and oval shapes represent random variables.

method for layering the sparse autoencoders to use as a classifier. The sparsity penalty term can be written as

$$\beta \sum_{j=1}^{N} KL(p \parallel \hat{p}_j)$$

where β is a hyperparameter to adjust the sparsity penalty term, $\hat{p}_j$ is the average activation value of a hidden node j over all the training inputs, and p is a Bernoulli random variable[13] that represents the ideal value distribution. The KL-loss is at a minimum when $p = \hat{p}_j$.

A.2. Denoising autoencoders

The principle of a denoising autoencoder (DAE) is simple. As explained earlier, an autoencoder aims to optimize the loss function by minimizing the difference between the input and the reconstructed output. A DAE is regularized and can be overcomplete, meaning that it uses regularization to extract useful features. Unlike an SAE, a DAE does not apply a penalty to the loss function, but instead changes the reconstruction error term. The loss function is changed from the vanilla version $L(x, g(f(x)))$ to $L(x, g(f(\tilde{x})))$, trying to optimize on $f(\tilde{x})$ instead of $f(x)$, where $\tilde{x}$ is a corruption of the input. The output is then compared to the uncorrupted input. DAEs only differ in a minor way from vanilla autoencoders. By adding noise to the input data, a DAE is forced to learn the most prominent features to be able to reconstruct the original input, essentially learning to remove the noise.

A DAE is generally used to create outputs free of noise. If applied to images, it is possible to reconstruct missing parts; for example, if there is lens-flare covering part of the image, the DAE could provide a copy of the image without the flare. A DAE could also be used to restore missing or hard to read letters and words in a text, or unclear sound could be repaired to make it sound cleaner. Vincent et al. [60] presented a simple stacked DAE and tested it on a variety of different datasets, including the MNIST image dataset [61]. What they showed us is that, by using a denoising criterion, we can learn useful higher level representations of the input data. In the paper by Vincent et al. [60], the input is corrupted with simple generic corruption processes, and they mainly perform tests on image and audio samples. A denoising criterion could be useful to help the learning process of an autoencoder to perform DDoS and DoS classification based on the output loss function.

A.3. Variational autoencoder

The Variational Autoencoder (VAE) introduced by Kingma and Welling [21] is a generative model that uses the same encoding as a normal autoencoder, the difference being in how the latent variables are handled. It is based on variational Bayes,[14] which is an analytical approximation of the intractable posterior distribution of the latent variables. It is used to derive a lower bound for the marginal likelihood of the observed data. The VAE presents a change to variational Bayes, by reparameterization of the variational lower bound, which is called the Stochastic Gradient Variational Bayes (SGVB) estimator. Since the

[13] From statistics, a Bernoulli distribution is the discrete probability distribution of a random variable with Boolean values.

[14] Bayes here refers to Bayesian inference. Variational Bayes methods are used to approximate intractable integrals arising from Bayesian inference.

VAE is a generative model, its primary strength lies in how well it can create new outputs based on features learned from training. In addition, it is possible to extend a VAE to use it for data classification. An example of this can be seen in the VAE of Y. Pu et al. [62], which used convolutional layers of the encoder and decoder to perform semi-supervised learning on image datasets.

The main difference between a traditional autoencoder and a VAE is how they use the layer between the encoder and the decoder, commonly referred to as the latent layer. An autoencoder uses the latent variables directly and decodes them to enable comparison between the input and output. A VAE will instead encode into two vectors of size n, the vector of means μ, and the vector standard deviations σ. A sampled vector is created from a collection of elements z_i that is assumed to follow a Gaussian distribution,[15] where each element i comes from the ith element in μ and σ. Thus, we can write each element in the sampled vector as $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$.[16] (See Fig. 16.)

A.3.1. VAE in detail

Let z be a latent representation of the unobserved variables, and $g(z)$ a differentiable generator network. x is sampled from a distribution $p(x; g(z))$, which can be written as $p(x|z)$. Here $p(x|z)$ represents a probabilistic decoder presenting a distribution over the possible values of x given z. When using the probabilistic decoder, we get an observation x from the hidden variable z. However, what we want is to infer the characteristics of z; thus, we need $p(z|x)$ and the integral marginal likelihood $p(x)$. The problem is that $p(x)$ is intractable, which means that we cannot evaluate or differentiate the marginal likelihood[17]. The solution to this is to create an approximation of the true posterior with another distribution $q(z|x)$, which will be the recognition model, a probabilistic encoder. We can use KL-divergence to measure the difference between these two probability distributions, as discussed earlier in Appendix A.1. The goal is to minimize the difference in order for the two distributions to be as similar as possible. We then get $\min KL(q(z|x) \parallel p(z|x))$. This can be used to maximize the lower bound $\mathcal{L}(q)$ of the marginal likelihood of the observed data, so that we get $\mathcal{L}(q) = E_{z\sim q(z|x)}\ log\ p(x|z) - D_{KL}(q(z|x)) \parallel p(z))$. The first term $E_{z\sim q(z|x)}\ log\ p(x|z)$ represents the reconstruction term, while $D_{KL}(q(z|x)) \parallel p(z))$ represents the Kullback–Leibler (KL) term. It ensures that the approximate posterior q is similar to the true posterior p.

A.3.2. Reparameterization trick

We have now seen the basic explanation of how a VAE works, and the math behind it. To fix the problem of the integral marginal likelihood $p(x)$, the "reparameterization trick" is introduced.

Since an autoencoder relies on an NN to forward data and backpropagate for training, we should not have a latent variable z as a random variable sampled from $q(z|x)$. An NN generally displays poor performance when performing backpropagation on random variables. This would lead to the decoded output being too different from the input. We know that the probabilistic encoder $q(z|x)$ is Gaussian, because it produces a distribution over the possible values of z from a data point x. In other words, $q(z|x) = \mathcal{N}(\mu, \sigma^2)$. Now, let ϵ be an auxiliary noise variable $\epsilon \sim \mathcal{N}(0, 1)$. We can reparameterize the encoder $q(z|x)$, so that we get z= $\mu + \sigma \cdot \epsilon$, as seen on the right-hand side of Fig. 17.

[15] A Gaussian distribution, also called a normal distribution, is a function that represents the distribution of a group of random variables as a symmetrical bell-shaped graph with the mean value at the center.

[16] From statistics. It reads: "z drawn from a normal distribution with mean μ and standard deviation σ"

[17] For further details about this problem, see the original paper by Kingma and Welling [21].

References

[1] Inc. Cisco Systems, Cisco visual networking index: Forecast and trends, 2017–2022, 2019, https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[2] Kaspersky Lab, Denial of service: How businesses evaluate the threat of ddos attacks, 2015, https://media.kasperskycontenthub.com/wp-content/uploads/sites/45/2018/03/08234158/IT_Risks_Survey_Report_Threat_of_DDoS_Attacks.pdf.

[3] Kaspersky Lab, Global it security risks survey, 2015, https://media.kaspersky.com/en/business-security/it-security-risks-survey-2015.pdf.

[4] James Scott, Drew Spaniel, Rise of the machines: The dyn attack was just a practice run, 2016.

[5] Giovane C.M. Moura, Cristian Hesselman, Gerald Schaapman, Nick Boerman, Octavia de Weerdt, 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS & PW), 2020.

[6] Wencong You, Lei Jiao, Jun Li, Ruiting Zhou, Scheduling ddos cloud scrubbing in isp networks via randomized online auctions, in: IEEE International Conference on Computer Communications, INFOCOM, 2020.

[7] Wei Zhou, Weijia Jia, Sheng Wen, Yang Xiang, Wanlei Zhou, Detection and defense of application-layer ddos attacks in backbone web traffic, Future Gener. Comput. Syst. 38 (2014) 36–46.

[8] Madeleine Kongshavn, Hårek Haugerud, Anis Yazidi, Torleiv Maseng, Hugo Hammer, Mitigating ddos using weight-based geographical clustering, Concurr. Comput.: Pract. Exper. 32 (11) (2020) e5679.

[9] Markus Goldstein, Christoph Lampert, Matthias Reif, Armin Stahl, Thomas M. Breuel, Bayes Optimal ddos mitigation by adaptive history-based ip filtering, in: Seventh International Conference on Networking, ICN 2008, vol. 4, IEEE Computer Society Press, 2008, pp. 174–179.

[10] Irom Lalit Meitei, Khundrakpam Johnson Singh, Tanmay De, Detection of ddos dns amplification attack using classification algorithm, in: Proceedings of the International Conference on Informatics and Analytics, 2016, pp. 1–6.

[11] Desta Haileselassie Hagos, Anis Yazidi, Øivind Kure, Paal E. Engelstad, Enhancing security attacks analysis using regularized machine learning techniques, in: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2017, pp. 909–918.

[12] S.T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks, IEEE Commun. Surv. Tutor. 15 (4) (2013) 2046–2069, Fourth.

[13] M. Goldstein, Bonesi ddos simulator, 2008, https://github.com/Markus-Go/bonesi.

[14] Maheshkumar Sabhnani, Gursel Serpen, Why machine learning algorithms fail in misuse detection on kdd intrusion detection data set, Intell. Data Anal. 8 (4) (2004) 403–415.

[15] Gopal Singh Kushwah, Virender Ranga, Optimized extreme learning machine for detecting ddos attacks in cloud computing, Comput. Secur. 105 (2021) 102260.

[16] Ilhan Firat Kilincer, Fatih Ertam, Abdulkadir Sengur, Machine learning methods for cyber security intrusion detection: Datasets and comparative study, Comput. Netw. 188 (2021) 107840.

[17] Maria. Rigaki, Sebastian Garcia, Bringing a gan to a knife-fight: Adapting malware communication to avoid detection, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 70–75.

[18] Torgeir Fladby, Hårek Haugerud, Stefano Nichele, Kyrre Begnum, Anis Yazidi, Evading a machine learning-based intrusion detection system through adversarial perturbations, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, 2020, pp. 161–166.

[19] Gabriel C. Fernández, Shouhuai Xu, A case study on using deep learning for network intrusion detection, in: MILCOM 2019-2019 IEEE Military Communications Conference, MILCOM, IEEE, 2019, pp. 1–6.

[20] Pramuditha Perera, Vishal M. Patel, Learning deep features for one-class classification, 2018, arXiv preprint arXiv:1801.05365.

[21] Diederik P. Kingma, Max Welling, Auto-encoding variational Bayes, 2013, arXiv e-prints, arXiv:1312.6114.

[22] Quamar Niyaz, Weiqing Sun, Ahmad Y. Javaid, A deep learning based ddos detection system in software-defined networking (sdn), 2016, arXiv preprint arXiv:1611.07400.

[23] Abdurrahman Pektaş, Tankut Acarman, A deep learning method to detect network intrusion through flow-based features, Int. J. Network Manage. (2018) e2050.

[24] Benjamin J. Radford, Bartley D. Richardson, Shawn E. Davis, Sequence aggregation rules for anomaly detection in computer network traffic, 2018, arXiv e-prints, arXiv:1805.03735.

[25] Canadian Institute for Cybersecurity, Cicids2017 dataset download and information, 2017, https://www.unb.ca/cic/datasets/ids-2017.html.

[26] Benjamin J. Radford, Leonardo M. Apolonio, Antonio J. Trias, Jim A. Simpson, Network traffic anomaly detection using recurrent neural networks, 2018, arXiv e-prints, arXiv:1803.10769.

[27] C.E. Shannon, A mathematical theory of communication, SIGMOBILE Mob. Comput. Commun. Rev. 5 (1) (2001) 3–55.

[28] Sunny Behal, Krishan Kumar, Detection of ddos attacks and flash events using information theory metrics–an empirical investigation, Comput. Commun. 103 (2017) 18–28.

[29] George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, Hui Zhang, An empirical evaluation of entropy-based traffic anomaly detection, in: Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement, IMC '08, ACM, New York, NY, USA, 2008, pp. 151–156.
[30] Z. Tan, A. Jamdagni, X. He, P. Nanda, R.P. Liu, J. Hu, Detection of denial-of-service attacks based on computer vision techniques, IEEE Trans. Comput. 64 (9) (2015) 2519–2533.
[31] Z. Tan, A. Jamdagni, X. He, P. Nanda, R.P. Liu, A system for denial-of-service attack detection based on multivariate correlation analysis, IEEE Trans. Parallel Distrib. Syst. 25 (2) (2014) 447–456.
[32] H. Ling, K. Okada, An efficient earth mover's distance algorithm for robust histogram comparison, IEEE Trans. Pattern Anal. Mach. Intell. 29 (5) (2007) 840–853.
[33] Irvine University of California, Kdd99 dataset download and information, 1999, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[34] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, Ali A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (3) (2012) 357–374, Dataset download link: https://www.unb.ca/cic/datasets/ids.html.
[35] Y. Xie, S. Yu, A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors, IEEE/ACM Trans. Netw. 17 (1) (2009) 54–65.
[36] Ili Ko, Desmond Chambers, Enda Barrett, Adaptable feature-selecting and threshold-moving complete autoencoder for ddos flood attack mitigation, J. Inform. Secur. Appl. 55 (2020) 102647.
[37] Ili Ko, Desmond Chambers, Enda Barrett, Recurrent autonomous autoencoder for intelligent ddos attack mitigation within the isp domain, Int. J. Mach. Learn. Cybern. (2021) 1–23.
[38] Jinyin Chen, Yi-tao Yang, Ke-ke Hu, Hai-bin Zheng, Zhen Wang, Dad-mcnn: Ddos attack detection via multi-channel cnn, in: Proceedings of the 2019 11th International Conference on Machine Learning and Computing, 2019, pp. 484–488.
[39] David M. Blei, Alp Kucukelbir, Jon D. McAuliffe, Variational inference: A review for statisticians, J. Amer. Statist. Assoc. 112 (518) (2017) 859–877.
[40] Carl Doersch, Tutorial on variational autoencoders, 2016, arXiv e-prints, arXiv:1606.05908.
[41] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[42] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv e-prints, arXiv:1412.6980.
[43] Irie, Miyake, Capabilities of three-layered perceptrons, in: IEEE 1988 International Conference on Neural Networks, 1 (1988) 641–648.
[44] Steven W. Smith, The Scientist and Engineer's Guide To Digital Signal Processing, California Technical Publishing, San Diego, CA, USA, 1997, http://www.dspguide.com/ch13/2.html.
[45] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun, OverFeat: Integrated recognition, localization and detection using convolutional networks, 2013, arXiv e-prints, arXiv:1312.6229.
[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: The IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016.
[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[48] Sergey Ioffe, Christian Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, arXiv e-prints, arXiv:1502.03167.
[49] The Communications Security Establishment and the Canadian Institute for Cybersecurity, Csecicids2018 dataset download and information, 2018, Dataset information: https://www.unb.ca/cic/datasets/ids-2018.html, and download: https://registry.opendata.aws/cse-cic-ids2018/.
[50] Samuel L. Smith, Pieter-Jan Kindermans, Quoc V. Le, Don't decay the learning rate, increase the batch size, 2017, CoRR, arXiv:abs/1711.00489.
[51] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, On large-batch training for deep learning: Generalization gap and sharp minima, 2016, CoRR, arXiv:abs/1609.04836.
[52] Iman Sharafaldin, Arash Habibi Lashkari, Ali Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, 01 (2018) 108–116.
[53] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, Ali A. Ghorbani, Characterization of tor traffic using time based features, in: ICISSP, 2017, pp. 253–262.
[54] Eirik Molde Bårli, Ddos and dos mitigation using a variational autoencoder (Master's thesis), 2019.
[55] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, Why should I trust you?: Explaining the predictions of any classifier, 2016, Github Project Page: https://github.com/marcotcr/lime arXiv e-prints, arXiv:1602.04938.
[56] S. Manimurugan, Saad Al-Mutairi, Majed Mohammed Aborokbah, Naveen Chilamkurti, Subramaniam Ganesan, Rizwan Patan, Effective attack detection in internet of medical things smart environment using a deep belief neural network, IEEE Access 8 (2020) 77396–77404.
[57] John Sheppard, A partial approach to intrusion detection, in: International Conference on Digital Forensics and Cyber Crime, Springer, 2020, pp. 78–97.
[58] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biol. Cybernet. 59 (4) (1988) 291–294.
[59] Andrew. Ng, et al., Sparse autoencoder, CS294A Lecture Notes 72 (2011) 1–19.
[60] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (Dec) (2010) 3371–3408.
[61] Yann LeCun, Corianna Cortes, Christopher J.C. Burges, Mnist dataset download and information, 1998, http://yann.lecun.com/exdb/mnist/.
[62] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, Lawrence Carin, Variational autoencoder for deep learning of images, labels and captions, in: D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016, pp. 2352–2360.

Eirik Molde Bårli received his M.Sc. degree in Computer Science from the University of Oslo in 2019 and has since then worked as a Full Stack Web Developer for Kongsberg Target Systems, Kongsberg, Norway

Professor **Anis Yazidi** received the M.Sc.and Ph.D. degrees from the University of Agder, Grimstad, Norway, in 2008 and 2012, respectively. He was a Researcher with Teknova AS, Grimstad, Norway. He is currently a Full Professor with the Department of Computer Science, Oslo Metropolitan University, Oslo, Norway, where he is leading the research group in Applied Artificial Intelligence. His current research interests include machine learning, learning automata, stochastic optimization, and autonomous computing.

Enrique Herrera-Viedma (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain, in 1993 and 1996, respectively. He is currently a Professor of Computer Science and the Vice-President of Research and Knowledge Transfer with the University of Granada. His H-index is 85 with more than 25000 citations received in Web of Science and 97 in Google Scholar with more than 38500 citations received. His current research interests include group decision making, consensus models, linguistic modeling, aggregation of information, information retrieval, bibliometric, digital libraries, Web quality evaluation, recommender systems, and social media. He has been identified as one of the World's Most Influential Researchers by Shanghai Center and Thomson Reuters/Clarivate Analytics in both the computer science and engineering scientific categories in 2014–2020.

Dr. **Hårek Haugerud** received his M.Sc. and Ph.D. degrees from the University of Oslo and is currently an associate professor at Oslo Metropolitan University (OsloMet). He joined OsloMet in 1998. He is member of the research group Autonomous Systems and Networks and the Applied Artificial Intelligence research group.