5-2021

Semi-Supervised Spatial-Temporal Feature Learning on Anomaly-Based Network Intrusion Detection

Huy Mai

Semi-Supervised Spatial-Temporal Feature Learning on Anomaly-Based Network Intrusion Detection

An Undergraduate Honors College Thesis
in the
Department of Computer Science and Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

By

Huy Mai

May 2021
University of Arkansas

Abstract

Due to a rapid increase in network traffic, it is growing more imperative to have systems that detect attacks that are both known and unknown to networks. Anomaly-based detection methods utilize deep learning techniques, including semi-supervised learning, in order to effectively detect these attacks. Semi-supervision is advantageous as it doesn't fully depend on the labelling of network traffic data points, which may be a daunting task especially considering the amount of traffic data collected. Even though deep learning models such as the convolutional neural network have been integrated into a number of proposed network intrusion detection systems in recent years, little work has been done on spatial-temporal feature extraction for network intrusion anomaly detection using semi-supervised learning. This paper introduces *Anomaly-CNVAE*, a variational autoencoder where the encoding and decoding layers perform convolution and transpose convolution, respectively, in order to account for spatial feature extraction. In addition, in order to account for time-based features in the dataset, the proposed model utilizes 1D-CNN for the convolution operations. The performance of the model in network intrusion detection is evaluated against an autoencoder and a vanilla variational autoencoder. Results show that *Anomaly-CNVAE* significantly outperforms the other semi-supervised learning models with a 5-10 percent increase in evaluation metrics.

Keywords: *Network Intrusion Detection, Semi-Supervised Learning, Anomaly Detection*

TABLE OF CONTENTS

1 Introduction

Cyber-attacks are becoming more prevalent in our society today. In particular, the number of attacks pertaining to the entrance of unauthorized traffic into networks is increasing. Even though some technologies such as firewalls are commonly placed to stop these attacks from intruding into the network, recent advancements, which have highlighted the pitfalls of these common technologies, have been presented to improve the mitigation of this issue, including the development of network intrusion detection systems.

The primary function of a network intrusion detection system (NIDS) is to detect malicious network traffic and raise an alert in the event of such an attack. There are two types of identification: signature-based detection and anomaly detection [1]. Using a database of predefined attacks, signature-based approaches are able to effectively identify known network traffic attacks. Despite their low false alarm rate, these approaches are futile against unknown attacks and zero-day attacks. Furthermore, updating the database seems to be more of a cumbersome task as attacks are growing more frequent and diverse. In contrast, anomaly detection systems utilize normal traffic activity to identify traffic that deviate from normal behavior. Even though they are able to identify both known and unknown attacks, anomaly detection technologies are subject to high false alarm rates.

Regarding network intrusion detection, numerous machine learning approaches have been proposed over the past few years. A majority of these approaches fit under the category of supervised learning, where the models are trained on data that include labels for all network traffic records. The convolutional neural network (CNN), which is a deep learning model commonly utilized for classification tasks in fields such as voice recognition and image processing [2], has been included in several recent proposed models for multiclass classification of network

traffic. Nevertheless, supervised learning models, including the CNN, are dependent on the strenuous labelling of traffic data.

By contrast, semi-supervised learning requires that a certain portion of the data is labelled. Some models that are designed to follow this learning process include the autoencoder and variational autoencoder. In the case of anomaly detection, both models take advantage of dimensionality reduction and input reconstruction to first train on normal data in order to minimize reconstruction loss and then identify data points within testing data that deviate from normal behavior. When considering that benchmark datasets related to network intrusion detection are imbalanced in favor of normal traffic behavior [3], semi-supervised anomaly detection is advantageous. However, little work has been done to explore semi-supervised anomaly detection on network traffic. Moreover, even as [4] proposed a variational autoencoder model for anomaly detection, major improvements, including those related to feature extraction, need to be made to the model.

In this work, I propose a novel semi-supervised learning model, *Anomaly-CNVAE*, where convolution and transposed convolution are added to the variational autoencoder in order to extract and reconstruct spatial features in network traffic data. Moreover, instead of the more common two-dimensional approach used for images, the convolution and transpose convolution are both one-dimensional in order to account for temporal features in the data. *Anomaly-CNVAE* outperforms a few semi-supervised learning models on a number of benchmark datasets.

The remainder of the thesis is organized as follows. Chapter 2 gives a thorough overview of literature related to network intrusion detection systems. Chapter 3 presents background information necessary for the proposed methodology, which is introduced in Chapter 4. After the datasets are introduced in Chapter 5, Chapter 6 breaks down the experiment conducted on the methodology against other semi-supervised models for anomaly-based network intrusion detection, where the results are discussed in Chapter 7. Chapter 8 concludes the paper and offers ideas for future work.

2 Related Work

Anomaly-based network intrusion detection is the other primary means of keeping track of normal and abnormal activity in a network. More research has been conducted within this domain due to the ability of this type of system to identify not only known intrusions, but also unknown and zero-day attacks. For these new attacks, databases do not need to be updated [5]. Anomaly-based network intrusion detection makes use of machine learning methods, where most work has been done on supervised and unsupervised learning for network intrusion detection.

Supervised learning methods allow NIDS models to be trained on existing network traffic data to identify unknown activity as normal or abnormal. This opens up for the use of artificial neural networks (ANNs), which are function approximators that model decision making inspired by the interconnections of neurons in the human brain [6]. Researchers in this area have implemented certain types of ANNs such as multi-layer perceptrons (MLPs), which are simple networks that each consists of one input layer, one hidden layer, and one output layer, support vector machines (SVM), which utilize nonlinear mapping to transform the original training data into a new dimension in order to separate the data into two classes with a hyperplane, and k-nearest neighbor (KNN), which takes the proximity of data points into consideration in order to classify a certain data point. One early example of use of these classifiers includes [7], whose MLP was trained on the Defense Advanced Research Projects Agency set to identify and predict network attacks. Nevertheless, the MLP was still weak in classifying unknown attacks as indicated in the significant difference between classification rates for known and unknown attacks. Another example is [8], which proposed a recursive SVM to build upon regular SVM classification in order to account for the extraction of main features of data. [9] proposed an algorithm that combined the KNN and

MARS algorithms in order to classify normal and abnormal behavior based on collections of neighbors.

As stated before, a significant proportion of recent supervised learning models tasked for NID incorporates the CNN architecture. One example model, highlighted in [10], converts a normalized feature vector generated from a benchmark NID dataset to a two-dimensional format frequently used for images, which required the removal of a feature using the coefficient of variance. The two-dimensional input then goes through two convolutional layers and two pooling layers before going through a fully-connected layer in order to classify the input as normal or as an attack class. Additional works have expanded upon this model, a number of which also integrated schemes to solve the problem of class imbalance in NID datasets. For instance, [3] introduces *AS-CNN*, where the ADASYN algorithm augments the data in order to account for an imbalanced data distribution and a split convolution component is incorporated to the CNN architecture to reduce interchannel information redundancy. Another example is [11], which proposed a one-dimensional CNN tasked for supervised learning on time-series NID data. Moreover, more works have applied CNNs to a hierarchical framework that considers other types of features not related to spatial locality in the dataset. This is seen in [12], for example, where a CNN architecture is combined with a bi-directional long short-term memory (LSTM) model that extracts time-based features. Even though most supervised learning models include methods to account for class imbalance in the datasets, this paper neither focuses on supervised learning nor the problem of imbalanced distribution.

On the other hand, more intrusion detection algorithms include unsupervised learning techniques, where the input data does not need a label. These techniques generally incorporate clustering, which allows analysis of patterns within data by the partition of the dataset. The ultimate goal of clustering algorithms is to maximize intraclass similarity and minimize interclass similarity. K-means is a well-known clustering algorithm that partitions data based on the selection of k cluster centroids. [13] was the one of the earliest papers to propose an intrusion

detection algorithm based on K-means. Even though the algorithm is suitable for large datasets, the algorithm is not as robust as other clustering algorithms due to its requirement of a pre-defined k value before running the algorithm. [14] introduced density-based spatial clustering of applications with noise (DBSCAN), which offered more robustness compared to K-means by requiring one input parameter and supporting the user in determining an appropriate value for that parameter.

Semi-supervised learning requires that a certain proportion of data points contains labels while the rest are unlabeled [15], which can be useful as labelling data from a large dataset becomes a daunting task for human annotators. Some models that utilize this type of learning train on the labelled data in order to assign a label to a previously unlabeled point. For example, [16] applied the above task to images outputted by a GAN generator using feature matching, where images are generated based on the statistics of real data. These newly generated images are added to the training set.

In the case of network intrusion detection, works that utilize semi-supervised learning follow the procedure of performing an unsupervised feature extraction and supervised classification, which is usually done using an autoencoder-based architecture. For example, [17] proposed an approach based on a stacked autoencoder, where encoding layers from each autoencoder in a set of n autoencoders are attached for feature extraction after the layers pretrained to minimize reconstruction error.

However, little work has been done to consider a semi-supervised approach for anomaly-based network intrusion detection, where the model is trained on labeled normal data in order to minimize reconstruction error before including both normal and attack data during the testing process. Network traffic attacks are identified if they produce an anomalous reconstruction error. [4] tested autoencoder and variational autoencoder models against a support vector machine to assess performance on the CIC-IDS2017 dataset, showing that the variational autoencoder performed best out of the three models.

3 Background

It is important to have an understanding of preliminary concepts within machine learning before delving into the proposed model. This chapter explores the ideas of artificial neural network, autoencoder, variational autoencoder, and convolutions when used in machine learning technologies.

3.1 Artificial Neural Network

Let $x = (x_1\ x_2\ \dots x_n)^T$ be a column vector. We define an ***artificial neural network*** as a mathematical model of interconnected nodes that includes a set of input nodes that process x and a set of output nodes such that one output node is emphasized given x. The nodes are arranged in layers, where the output of one layer may serve as input to another [18]. To simulate synaptic connection and the transference of information in the brain, a node c_i in one layer of size m is connected to all or a subset of nodes d_j in the next layer of size n using a weighted value w_{mn}. (Let's assume for the sake of this explanation that each node d_i is connected to all nodes d_j.) The value of each node d_j is computed using

$$d_j = f(\sum_{i}^{m} w_{ji}c_i + b),$$

where f is called the *activation function*, which simulates the level of activity of d_j, and b is the bias term, which allows d_j to have a required output value in case it is not possible. Some activation functions such as the sigmoid and hyperbolic tangent functions are commonly utilized to assign a value between a certain range, where the higher value denotes an active node while the lower value denotes an inactive node.

Our proposed model uses a feedforward neural network, where the connections between nodes do not form a cycle. In other words, input to the neural network is generally processed from one layer to another such that the information does not revert back to a layer in order to obtain an output for the input. This output is compared with the actual output for the data point $x^{(i)}$, where the calculated difference is called the ***loss***. Various loss functions such as the mean-squared error and binary cross-entropy are commonly used.

During the training process, the network learns through a process called ***backpropagation***, where loss is fed back through the network, resulting in updates of weights in the network. Different gradient-based techniques such as Adam [19] and stochastic gradient descent have been used to update these weights such that the loss converges as the training process progresses.

3.2 Autoencoder

With regards to the proposed model, we first define an ***autoencoder*** as an acyclic feed-forward unsupervised neural network that learns how to reconstruct a given input with minimal loss. An autoencoder generally consists of four components: the encoder layer, the decoder layer, the bottleneck, and the reconstruction loss function.

Let x be any input to an autoencoder. The encoder layer f yield the bottleneck z by computing $z = f(x)$, where z is of lower dimension than x. In other words, the bottleneck z is a compressed representation of the original input x. From there z serves as input for the decoder layer g, producing the reconstruction x' after computing $x' = g(z)$. To train the autoencoder, we examine the minimization of the reconstruction loss $L(x, x') = L(x, (g \circ f)(x))$, which helps us assess the performance of the decoder by measuring how close the reconstructed input is to the original input.

We note that although a traditional autoencoder consists of a single layer for both the encoder and the decoder, multiple layers can be applied to the encoder

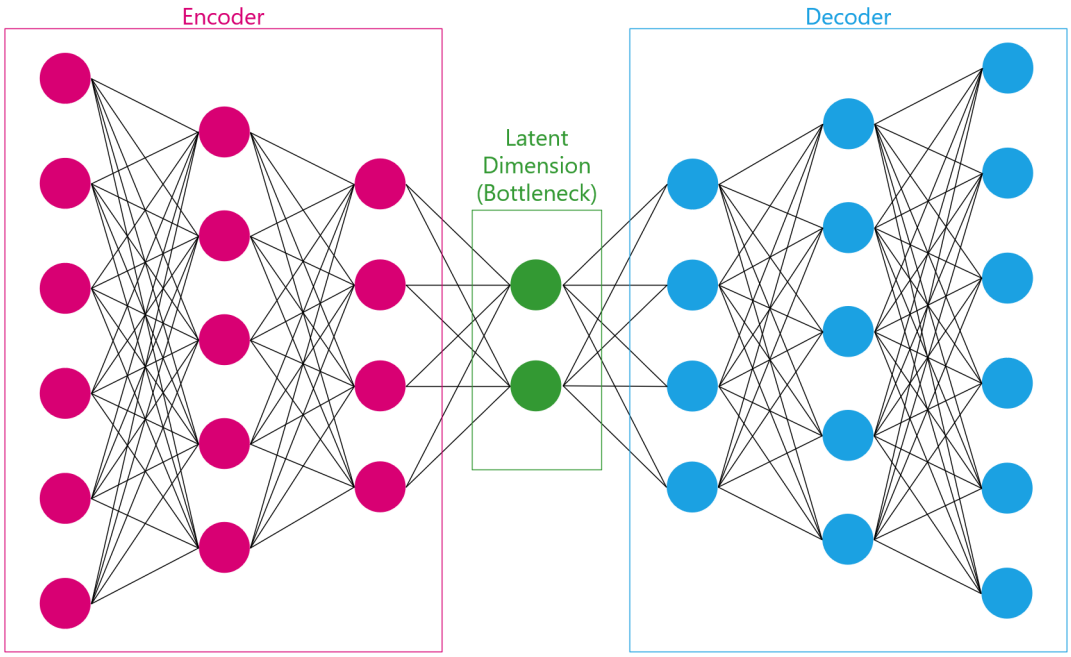

Figure 3.1: Deep Autoencoder

and the decoder, producing an architecture that has autoencoders within autoencoders. We call this particular type of autoencoder a ***stacked*** (or ***deep***) ***autoencoder***, which is shown in Figure 3.1. Stacked autoencoders have been frequently used for dimensionality reduction [20]. As they consist of multiple traditional autoencoders, the training process for a stacked autoencoder can be "layer-wise" for each autoencoder: The process typically starts with the pre-training of each individual autoencoder before attaching encoders and decoders together such that for $i = 1, 2, \ldots, n$, the output of i^{th} encoder layer acts as input for the $(i+1)^{th}$ encoder layer while the output of the $(i+1)^{th}$ decoder layer acts an input for the i^{th} decoder layer.

With a semi-supervised learning paradigm, where the training process involves unlabelled normal data points, the reconstruction loss makes an autoencoder a popular model for detecting anomalies. As a result of $L(x, x')$ being minimized after training, this implies that $L(x, x')$ is relatively low among normal data points.

Hence, when the trained autoencoder is given an anomalous data point, we expect $L(x, x')$ to have a significantly large value. This allows us to enact a threshold θ on a value for $L(x, x')$ of any tested data point x to detect whether x is a normal point or an anomaly using the following output y:

$$y = \begin{cases} normal, & L(x, x') < \theta \\ anomaly, & L(x, x') \geq \theta \end{cases}$$

We do note that there are different ways of determining the threshold θ for an anomaly detection task. One method is to determine θ beforehand and utilize trial and error to optimize θ, which may prove to be tedious. Alternatively, θ can be expressed as a function of the losses of each data point $x^{(i)}$, where the function could be the mean, median, or percentile of the losses [21].

3.3 Variational Autoencoder

Our proposed approach consists of a special type of autoencoder, the ***variational autoencoder***, that involves variational Bayesian inference [22], where the latent variable z is sampled from an approximate posterior inference $q_\phi(z|x)$ given a set of input X. Let ϕ be the parameters associated with the encoding layer and θ be the parameters associated with the decoding layer. Compared to the traditional autoencoder, the encoding layer of a variational autoencoder calculates the approximate posterior inference $q_\phi(z|x)$, where the approximate posterior inference is needed over the true posterior inference $p_\phi(z|x^{(i)})$ as the true posterior is intractable due to the high-dimensional nature of X. Moreover, the decoding layer calculates the approximate marginal inference $p(x|z)$.

Letting $\mathbf{X}$= $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ be input data points from a dataset, the objective function for a variational autoencoder is defined as the variational lower bound of the marginal likelihood of the whole dataset, where the marginal likeli-

hood of the set on parameters θ is computed as

$$\log p_\theta(x^{(1)}, \ldots, x^{(n)}) = \sum_{i=1}^{n} \log p_\theta(x^{(i)}) \tag{3.1}$$

The marginal likelihood of each data point $x^{(i)}$ is expressed as

$$\log p_\theta(x^{(i)}) = D_{KL}(q_\phi(z|x^{(i)})||p_\phi(z|x^{(i)}) + \mathcal{L}(\theta, \phi; x^{(i)}) \tag{3.2}$$

The first term on the right side of Equation 3.2 is the *Kullback-Leibler divergence*, which measures the dissimilarity between the approximate posterior and the actual posterior. Meanwhile, the other term on the right side of equation (3.2) is the *variational lower bound* of the marginal likelihood of $x^{(i)}$, which can be written as the following equation:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)})||p_\phi(z|x^{(i)}) + \mathbf{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)] \tag{3.3}$$

In order to perform backpropagation to train the variational autoencoder, we need to compute the gradient of $\mathcal{L}(\theta, \phi; x^{(i)})$ with respect to the parameters θ and ϕ. While the KL-divergence term is differentiable [22], when considering the second term of $\mathcal{L}(\theta, \phi; x^{(i)})$, however, the computation is a very difficult task for a number of reasons, including the inability to evaluate the second term, which is an expectation function, in closed form [23]. Hence it is necessary to find an estimator of the gradient of $\mathbf{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]$ that is differentiable.

The inclusion of a stochastic gradient estimator may also alter the way the latent variable z is sampled from the approximate posterior $q_\phi(z|x^{(i)})$. This is where $z \sim q_\phi(z|x^{(i)})$ is reparametrized to a function $g_\phi(\epsilon, x)$ that is differentiable, where $\epsilon \sim p(\epsilon)$.

This results in the following estimator $\tilde{\mathcal{L}}(\theta, \phi; x^{(i)})$:

$$\tilde{\mathcal{L}}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)})||p_\phi(z|x^{(i)})) + \log p_\theta(x^{(i)}|z), \tag{3.4}$$

where $z = g_\phi(\epsilon, x^{(i)})$ and $\epsilon \sim p(\epsilon)$. This equation can be interpreted as the overall reconstruction loss function, where the KL-divergence term regularizes the encoding parameters ϕ in order to ensure that $q_\phi(z|x^{(i)})$ is close to the prior $p_\phi(z)$ and the second term is the expected reconstruction error.

3.4 Convolutional Layers in Machine Learning

The CNN architecture is usually implemented in order to capture spatial patterns within data. Unlike regular ANNs, the primary operation that is computed in a layer of a CNN is ***convolution***, denoted as the symbol $*$. In general, the convolution of any two functions $a(t)$ and $b(t)$ gives the following result $h(t)$:

$$h(t) = (a * b)(t) = \int a(s)b(t-s)ds \tag{3.5}$$

In the case of the CNN architecture, the convolution between the input matrix and a kernel matrix produces an output matrix that is often called a feature map. Even though both the input and kernel matrices are of n-dimension for some positive integer n, the size of the kernel is smaller than the size of the input in order to preserve the spatial relationship between input units. This is what we refer to as the *sparse connectivity* of layers as opposed to the full connectivity between layers as seen in artificial neural networks.

As mentioned earlier, the CNN architecture is frequently used on image data, which would require the input matrices to be two-dimensional (or three-dimensional if the image has colors other than those of greyscale). However, the two-dimensional input does not have to involve image data. In the previous chapter, a few key works that utilized methods of converting one-dimensional vectors to two-dimensional image formats were highlighted.

In the case of the input matrix $I \in \mathbf{R}^{a \times b}$ and the kernel matrix $K \in \mathbf{R}^{m \times n}$, where $a > m$ and $b > n$, we have the following for the convolution of I and K on some location $S_{i,j}$ in the resulting convolution S:

$$S_{i,j} = (I * K)_{i,j} = \sum_m \sum_n I_{i-m,j-n} K_{m,n} \tag{3.6}$$

This value can also be activated using an activation function to produce a ***feature map*** that is in $\mathbf{R}^{s \times t}$. We note that multiple kernels, hence multiple feature maps, can be considered for a convolution layer in order to extract different pieces of information from an input. For instance, feature maps for an image can contain

information about edges or various shapes [24]. Having multiple feature maps allows the network to better learn on input.

The resulting set of feature maps may be modified using an operation called ***pooling***, which provides a summary statistic of a small number of regions in the outputted set. There are a number of statistics that can be reported such as the max value, the L^2 norm of the region, or even a weighted average based on the distance from the central pixel in the region. When it comes to network traffic data, even though we do not need to worry about the translation invariance that pooling ensures, we do take advantage of the improved computational efficiency of pooling as it reduces the number of parameters for a training model.

We also note that especially for generative models. reverse operations for convolution, which is called ***transpose convolution*** and pooling, which is called ***unpooling***, exist. The transpose convolution operation on a layer with m feature maps undoes the convolution operation by expanding each of the m feature maps to be the input of the convolution operation, resulting in a matrix $\mathbf{R}^{a \times b}$.

4 Methodology

This chapter discusses the proposed method for anomaly-based network intrusion detection. This method utilizes a semi-supervised learning algorithm through the use of a variational autoencoder, *Anomaly-CNVAE*, that is properly trained on the spatial and temporal features that are represented in benchmark network intrusion datasets. As they contain stochastic latent representations due to their probabilistic nature, variational autoencoders better discern the difference between normal and anomalous data points by accounting for the variability between data points rather than solely the average that autoencoders account for. As a result, anomaly detection using variational autoencoders can produce a lower false alarm rate than detection using autoencoders.

In order to learn on the spatial and temporal features of network traffic data, we incorporate the use of one-dimensional convolutional layers as opposed to two-dimensional convolutional layers. Even though it is a popular choice for CNNs, the option of using two-dimensional convolutional layers comes down to optimizing the size of the input, which may result in loss of information especially if the number of features for each data point in the set cannot be evenly converted to an n-by-n format.

The following sections provide a detailed description of each component of the proposed method of identifying anomalies in network traffic data: data preprocessing, architecture, and the anomaly detection method.

4.1 Data Preprocessing

Let $\mathbf{X} = \{x^{(1)}, \ldots, x^{(n)}\}$ be the set of input network traffic data such that each data point $x^{(i)}$ is m-dimensional, with m as the number of features. We first

eliminate any data points in $\mathbf{X}$ that have missing values.

Our semi-supervised learning algorithm requires data to be numerical. This is problematic for benchmark datasets dealing with network activity, some of which having a few features containing categorical data, which is non-numerical. Therefore, in order for these features to be processed through our model, we perform **one-hot encoding** for each categorical feature in $\mathbf{X}$ if $\mathbf{X}$ contains any non-numerical features. The encoding process generates new columns for X with each new column titled a category represented in categorical data. Each new column consists of binary labels 0 and 1 where 0 denotes that a data point $x^{(i)}$ does not take on that category and 1 denotes that $x^{(i)}$ takes on that category.

The last step of data preprocessing is the normalization of $\mathbf{X}$ in order to improve the performance of *Anomaly-CNVAE*. We chose to scale features to have values between 0 and 1 by computing the normalized value $x'^{(ij)}$ for $1 \leq j \leq m$ with the following equation:

$$x'^{(ij)} = \frac{x^{(ij)} - min(x^{(i1)}, \ldots, x^{(im)})}{max(x^{(i1)}, \ldots, x^{(im)}) - min(x^{(i1)}, \ldots, x^{(im)})} \tag{4.1}$$

4.2 Architecture

After the input data is preprocessed, the architecture takes each normalized $x'^{(i)}$ as input. The architecture of **Anomaly-CNVAE**, which is diagrammed in Figure 4.1, consists of four components: the encoding convolutional layers, a latent representation, the decoding transpose-convolutional layers, and a loss function associated with the model.

First, each $x'^{(i)}$ is encoded through 1 one-dimensional convolution layer that is followed by a pooling layer. With each kernel having size 7, the convolution layer is going to consist of 40 feature maps, which are of size p, where

$$p = (\text{number of features}) - 6.$$

Each feature map y_j produced using the following equation, with a as input for the feature map, σ denoting an activation function, K denoting a kernel matrix,

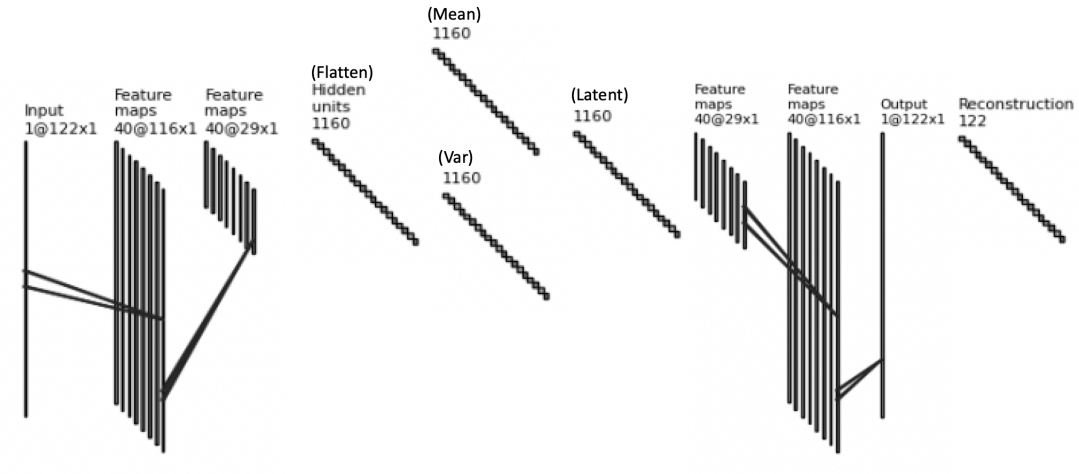

Figure 4.1: The architecture of *Anomaly-CNVAE* using input with 122 features

and b as bias:

$$y_j = \sigma(Ka + b)$$

The feature maps then go through maximum pooling of kernel size 4 and stride 4, reducing the dimensionality of the feature maps by half. After the sequence of one convolutional layer and one max pooling layer, the feature maps are then flattened into a single layer of size q, where

$$q = (p/4) - 3.$$

The flattened layer is then split into the mean and variance vectors. The latent representation z, which has the same dimension as the mean and variance vectors, is obtained using the reparametrization trick discussed earlier in chapter 3. Assuming that the approximate posterior $q(z|x^{(i)})$ is a multivariate Gaussian distribution $\mathcal{N}(\mu^{(i)}, \sigma^{2(i)})$, we have a reparametrization

$$z = g_\phi(\epsilon, x^{(i)}) = \mu^{(i)} + \sigma^{2(i)} \odot \epsilon, \tag{4.2}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The latent representation z, which is initially transformed to have 40 feature maps, is then decompressed through a sequence of max unpooling and one-dimensional transpose convolution. The decompression starts by performing max unpooling on z using a kernel size of 4 and a stride of 4 in order to revert the max pooling operation that was done $x'^{(i)}$ in the encoding layer. After that, the data is taken through a one-dimensional transpose convolutional layer using kernel size 7 to undo the one-dimensional convolution operation performed in the encoding layer. This will result in an output that has one feature map. Finally, in order to account for the loss function, we attach a fully-connected layer of input size, producing a reconstruction of the input $x'^{(i)}$.

The following equation is the loss function used to train and perform anomaly detection using the *Anomaly-CNVAE* model:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = \frac{1}{2} \sum_{j=1}^{q} (1 + \log((\sigma^{(i)})^2) - (\mu^{(i)})^2 - (\sigma^{(i)})^2) + \log\, p_\theta(x^{(i)}|z^{(i)}). \quad (4.3)$$

4.3 Anomaly Detection

Our proposed model *Anomaly-CNVAE* is designed to perform anomaly detection on network traffic data. Rather than using a pre-specified threshold, we demonstrate the robustness of the detection mechanism by considering a threshold function t as the 95th percentile of the distribution of losses L_{train} obtained from the last epoch of the training process. The last epoch is the theoretical point where the average loss of data converges, i.e. the point where the loss remains almost stagnant from epoch to epoch. In addition, the threshold function t produces a threshold relative to the dataset rather than merely choosing a value for the threshold, which could be a bit cumbersome when working with multiple benchmark datasets.

As shown in Equation 4.4, we determine whether a data point $x^{(i)}$ is normal or anomalous using the trained *Anomaly-CNVAE* model, which has a learned loss function $\mathcal{L}(\theta, \phi; x^{(i)})$.

$$result = \begin{cases} normal, & \mathcal{L}(\theta, \phi; x^{(i)}) < t(L_{train}) \\ anomaly, & \mathcal{L}(\theta, \phi; x^{(i)}) \geq t(L_{train}) \end{cases} \tag{4.4}$$

5 Experimentation

5.1 Datasets

Dataset	No. of Features	Normal	Abnormal (Attack)
NSL-KDD	122	77,054	10,719
UNSW-NB15	198	93,000	13,950
TRAbID	43	43,676	6,551

Table 5.1: Data Used for Experiments

Table 5.1 shows statistics that are considered for the experimentation of *Anomaly-CNVAE* for each dataset, with the number of features being computed after one-hot encoding during the data preprocessing stage. In this section, we discuss three benchmark datasets chosen for preprocessing and the testing of each model for network intrusion detection.

5.1.1 NSL-KDD

The NSL-KDD dataset, proposed by [25], was collected at the Canadian Institute for Cybersecurity as an improvement on another benchmark dataset: the KDD-CUP99 set. Even though it's been widely utilized, KDDCUP99 has its fair share of issues, namely redundancy in records in the set, which would cause biases toward the records, and a low difficulty of prediction for the set. NSL-KDD not only removes redundant records from KDDCUP99 but also increases the difficulty for classifiers when they learn on the set.

Features of the NSL-KDD dataset, which remained unchanged from the KDDCUP99 dataset, are divided into three groups: basic features, where values

are extracted from the TCP/IP connection, traffic features, which contain time-based values that are calculated in relation to a window of 2 seconds (or 100 connections to account for attacks that do not produce intrusion patterns within the two-second window), and content features, which examine the data portion of packets to account for attacks such as R2L and U2R that do not have time-based intrusion patterns.

Attacks represented in the dataset fall into one of four categories: Denial of Service (DoS), where attackers may force computing resources to be unavailable to legitimate users, User to Root Attack (U2R), where attackers initially have access to a system as normal users en route to root access to the system, Remote to Local Attack (R2L), where the attacker has the ability to send packets to a machine over a network without having an account, and probing, where the attacker seeks to gather information about a network in order to gain access through a weak point in the network.

5.1.2 UNSW-NB15

The UNSW-NB15 dataset was created by researchers at the Cyber Range Lab of the Australian Centre for Cyber Security to include more modern examples than other records in sets such as KDDCUP99 and NSL-KDD. The set contains a mixture of real normal activities and synthetic attack abnormal network traffic, where nine families of attacks are represented [26]. However, in order to address the lack of low footprint attacks, where the attacks don't install new software to a computer, most of these attacks differ from attacks represented in the NSL-KDD dataset. Besides DoS, the UNSW-NB15 dataset includes groups of low footprint attacks such as generic, where the attacker targets block ciphers, fuzzers, where the attacker attempts to suspend a computer program or network by giving it randomly generated data, and exploits, where the attacker has enough knowledge of a pitfall in the network's security to exploit its vulnerability.

Using two tools that process raw packet files, Argus and Bro-IDS, the first

35 features of the UNSW-NB15 dataset are obtained by matching the generated features that come from the output files in an SQL Server database. These features include packet-based features, where values come from examining the payload of a network packet, and flow-based features, where values come from examining packets that are under network connections. The last 12 features of the UNSW-NB15 dataset are considered general purpose and connection features.

5.1.3 TRAbID

The TRAbID dataset [27] is named after the paper "Towards a Reliable Anomaly-Based Intrusion Detection in real-world environments", from which the set was proposed. In order to address the lack of datasets that represent real-world network traffic, sixteen intrusion databases were created with network data points generated from activity of two automated users to a honeypot server: normal client and attacker. For the background traffic, each client, which executes random behavior not reliant on a statistical distribution, requests services such as HTTP, SMTP, and SSH that are common in network behavior. The traffic is determined to be valid if the client sends a real and valid request to the server and receives a real and valid reply from the server. On the other hand, the automated attacker can generate one of two types of attacks, probing and DoS, resulting in different groups of attack similarity such as network-level vulnerabilities and service-level vulnerabilities that are represented in the databases.

To extract features, a set of 50 predetermined features that was determined and experimented by the same group in an earlier study was considered for each network packet. (In the dataset, 43 features are represented.) Each feature fits into one of three categories: header-based, where values are based on the network packet header, host-based, which examines the communication history between hosts, and service-based, which examines the communication history between services.

5.2 Experiments

Algorithm 1: Training Process for *Anomaly-CNVAE*

Input : $\mathbf{X}_{train}$, M epochs, ϕ, θ
Output: L_{train}

Initialize the ϕ and θ parameters using Xavier Normal Initialization;
for k *in* $[1, M]$ **do**
  $L_{train} = [\,]$;
  for $x^{(i)} \in \mathbf{X}_{train}$ **do**
    $\mu^{(i)}$, $\sigma^{2(i)}$ = Flatten(Maxpool(Convolution($x^{(i)}$));
    $z^{(i)} = \mu^{(i)} + \sigma^{2(i)} \odot \epsilon$;
    Transform $z^{(i)}$ for unpooling;
    $x'^{(i)}$ = Fully-Connected(Transpose-Conv(Unpool($x^{(i)}$)));
    $reconstruction\_loss = \mathcal{L}(\theta, \phi; x^{(i)})$;
    Add $reconstruction\_loss$ to L_{train};
  end
end
return L_{train};

The training algorithm is shown in Algorithm 1. The performance of Anomaly-CNVAE on network intrusion detection was evaluated against two other semi-supervised models: a vanilla deep autoencoder, where all layers are fully connected, and a variational autoencoder, which were both implemented by [4]. The experiments were implemented using Python 3.7 along with built-in libraries such as Sci-kit Learn and Pytorch, both of which are commonly used to program machine learning models. In particular, the Pytorch library contains built-in implementation for various layers such as convolution, transpose convolution, max pooling, and linear that were used to construct the model. The experiments were conducted on a Dell Precision 7920 with an Intel Xeon Gold 5220 CPU and a NVIDIA Quatro RTX 8000.

In general, the training process was conducted on a train/test split for each dataset, where 75% of data points, all normal, were used for training while the other 25%, which contains both normal and anomalous data points, were used to evaluate anomaly detection.

The following sections highlight specific hyperparameters used to test each

model, and two evaluation metrics to analyze the results of the experiment.

5.2.1 Hyperparameters For Each Model

The three models shared a number of hyperparameters, including ones for the Stochastic Gradient Descent optimizer algorithm such as learning rate, which was set to 0.001, the momentum, which was set to 0.3, and the L2 penalty, which was set to 0.001. Moreover, a batch size of 256 and an epoch size of 30 were used to train each model on each of the three datasets. These values were chosen after tuning on the hyperparameters.

It is also important to note the following activation functions and layers used for each model. For the vanilla deep autoencoder, three hidden layers of size 512, 256, and 64 were used with the latent representation having size 64. For all layers except for the very last decoding layer, the *sigmoid function* $S(x)$, defined as

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{5.1}$$

was chosen as the activation function. The *softmax function* $\sigma(x)_i$, defined for all $1 \leq i \leq m$ as

$$\sigma(x) = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}, \tag{5.2}$$

was chosen as the activation function for the final decoding layer, which produces a reconstruction of the input.

The sizes of the encoding and decoding layers for the vanilla variational autoencoder were the same as the vanilla autoencoder. However, instead of the sigmoid function, the *Leaky ReLU function* $f(x)$, defined as

$$f(x) = \begin{cases} x, & x > 0 \\ 0.01x & x \leq 0 \end{cases} \tag{5.3}$$

was replaced as the activation for almost all the layers except the final decoding layer, which still used the softmax function as its activation function, in order to resolve the vanishing gradient problem that the sigmoid function runs into [28].

For *Anomaly CN-VAE*, the sizes of the one-dimensional convolution, pooling, max unpooling, and one-dimensional transpose convolution layers were stated earlier in chapter 4. Both the convolution and transpose convolution layers used the Leaky ReLU function as their activation functions.

5.2.2 Evaluation Metrics

The performance of the three models were evaluated on each test set of the benchmark datasets using two metrics. One metric is **accuracy**, which is simply the percentage of data points that the model correctly detects as either normal or anomalous. However, it is not enough to have accuracy as the sole metric for this experiment as there is a larger proportion of normal data points compared to anomalous data points across all datasets.

Hence, in addition to accuracy, our experiments use the **F1 score** to determine how well the models detect the network traffic data points. The metric, defined in equation (6.4), is the harmonic mean of the following two metrics: **precision** P, which is the proportion of true normal results to the number of data points classified as normal, and **recall** R, which is the proportion of true normal results to the number of data points labelled normal.

$$\text{F1 score} = 2 \cdot \frac{P \cdot R}{P + R} \tag{5.4}$$

5.3 Results

Model	*NSL-KDD*	*UNSW-NB15*	*TRAbID*
Vanilla AE [4]	69.28	58.61	62.44
Vanilla VAE [4]	72.63	70.60	64.19
Anomaly-CNVAE	**83.60**	**79.49**	**70.35**

Table 5.2: Accuracy for each model by dataset

Model	*NSL-KDD*	*UNSW-NB15*	*TRAbID*
Vanilla AE [4]	66.94	53.52	41.11
Vanilla VAE [4]	70.71	70.67	47.19
Anomaly-CNVAE	**81.02**	**78.49**	**57.22**

Table 5.3: F1 score for each model by dataset

Tables 7.1 and 7.2 indicate the accuracy and F1 score, respectively, of each model on the four benchmark datasets. For the first three datasets, the *Anomaly-CNVAE* model was significantly more accurate in its detection of network traffic anomalies. Moreover, there was an apparent increase in F1 score for the *Anomaly-CNVAE* model, indicating that model's accuracy does not favor the set of normal data points, which is substantially larger than the set of anomalous data points.

Moreover, the difference between accuracy and F1 score either stayed consistent or even better for *Anomaly-CNVAE* than the other two models. This is especially apparent for experiments on the three models for the TRAbID dataset, where the dropoff was greatest. Despite the significant dropoff, we also see that the reduction of the dropoff was greatest for *Anomaly-CNVAE* on top of its improved accuracy and F1 score.

Even though we see increases in accuracy and F1 score for *Anomaly-CNVAE*, the results also indicate that more work needs to be done to further improve the model's performance. We do note that the performance of the models need to be evaluated on a few more benchmark network traffic datasets in order to confirm not only that the *Anomaly-CNVAE* outperforms the autoencoder and the variational autoencoder models but also that its accuracy and F1 score allow the model to be integrated into future network intrusion detection systems. As seen in the results for the TRAbID dataset, a 57.22 F1-score for the *Anomaly-CNVAE* is an indicator that the model is still not as precise in detecting anomalous network behavior as it should be.

6 Conclusion

In this paper we introduced a new variational autoencoder model that performed anomaly-based detection on network intrusion using semi-supervised learning. In order to learn on features in network traffic datasets, which are both content-based (spatial) and time-based (temporal), we infused one-dimensional convolutional and transpose convolutional layers into the model. After training the model on normal traffic data points to reduce reconstruction loss, we incorporated the model into an overall detection scheme that determined whether a traffic record was normal or anomalous based on a threshold computed using a percentile of losses obtained from the training of the model. With an increase in accuracy and F1 score, experiments on four benchmark datasets show that model significantly outperformed other semi-supervised learning models in detecting network intrusions.

Even though the model showed improvement compared to previous semi-supervised learning models that perform anomaly-based network intrusion detection, it is also clear from the results that extensive work still needs to be carried out to create a model best fit for the task. For future work, we may explore the performance of learning for preprocessing techniques other than one-hot encoding in order to generate and test different sets of features.

There may be some possible applications of a model similar to *Anomaly-CNVAE* to network intrusion. For example, the model could be part of a pipeline that not only identifies anomaly-based intrusions but also classifies attacks based on the model's detection. To go even further, since unknown attacks commonly occur in network traffic analysis, especially zero-day attacks where new vulnerabilities pertaining to a computer network are exploited [29], the results from the model can help identify new attacks in real time.

Bibliography

[1] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," in *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, 2018, pp. 3369–3388.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[3] Z. Hu, L. Wang, L. Qi, Y. Li, and W. Yang, "A novel wireless network intrusion detection method based on adaptive synthetic sampling and an improved convolutional neural network," *IEEE Access*, vol. 8, pp. 195 741–195 751, 2020.

[4] S. Zavrak and M. İskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, pp. 108 346–108 358, 2020.

[5] R. Samrin and D. Vasumathi, "Review on anomaly based network intrusion detection system," 2017.

[6] S. C. Wang, *Artificial Neural Network*, 2003, pp. 81–100. [Online]. Available: https://doi.org/10.1007/978-1-4615-0377-4_5

[7] J. Shun and H. A. Malki, "Network intrusion detection system using neural networks," 2008.

[8] G. Shang-fu and Z. Chun-lan, "Intrusion detection system based on classification," 2012.

[9] X. Cheng, B.-X. Liu, K. Li, and J. Yan, "Intrusion detection system based on knn-mars," 2009.

[10] K. Wu, Z. Chen, and W. Li, "A novel intrusion detection model for a massive network using convolutional neural networks," *IEEE Access*, vol. 6, pp. 50 850–50 859, 2018.

[11] M. Azizjon, A. Jumabek, and W. Kim, "1d cnn based network intrusion detection with normalization on imbalanced data," in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2020, pp. 218–224.

[12] K. Jiang, W. Wang, A. Wang, and H. Wu, "Network intrusion detection combined hybrid sampling with deep hierarchical network," *IEEE Access*, vol. 8, pp. 32 464–32 476, 2020.

[13] M. Jianliang, S. Haikun, and B. Ling, "The application on intrusion detection based on k-means cluster algorithm," 2009.

[14] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density- based algorithm for discovering clusters in large spatial databases with noise," 1996, pp. 226–231.

[15] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–130, 2009. [Online]. Available: https://doi.org/10.2200/S00196ED1V01Y200906AIM006

[16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

[17] O. Aouedi, K. Piamrat, and D. Bagadthey, "A semi-supervised stacked autoencoder approach for network traffic classification," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–6.

[18] S. Walczak and N. Cerpa, "Artificial neural networks," in *Encyclopedia of Physical Science and Technology (Third Edition)*, third edition ed., R. A. Meyers, Ed. New York: Academic Press, 2003, pp. 631–645. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0122274105008371

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[20] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://science.sciencemag.org/content/313/5786/504

[21] M. U. Ndubuaku, A. Anjum, and A. Liotta, "Unsupervised anomaly thresholding from reconstruction errors," in *Internet and Distributed Computing Systems*, R. Montella, A. Ciaramella, G. Fortino, A. Guerrieri, and A. Liotta, Eds. Cham: Springer International Publishing, 2019, pp. 123–129.

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[23] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," 2020.

[24] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[25] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[26] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[27] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, pp. 200–216, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128617303225

[28] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München," 1991.

[29] L. Bilge and T. Dumitras, "Before we knew it: An empirical study of zero-day attacks in the real world," 10 2012, pp. 833–844.