St. Cloud State University
The Repository at St. Cloud State

Culminating Projects in Information Assurance | Department of Information Systems

5-2020

Autoencoder-Based Representation Learning to Predict Anomalies in Computer Networks

Shehram Khan

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation

Khan, Shehram, "Autoencoder-Based Representation Learning to Predict Anomalies in Computer Networks" (2020). *Culminating Projects in Information Assurance*. 102.
https://repository.stcloudstate.edu/msia_etds/102

This Thesis is brought to you for free and open access by the Department of Information Systems at The Repository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of The Repository at St. Cloud State. For more information, please contact tdsteman@stcloudstate.edu.

Autoencoder-Based Representation Learning to Predict Anomalies in Computer Networks

by

Shehram Sikander Khan

A Thesis

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

May, 2020

Thesis Committee:
Akalanka Mailewa Dissanayaka, Chairperson
Mark Schmidt
David Robinson

Abstract

With the recent advances in Internet-of-thing devices (IoT), cloud-based services, and diversity in the network data, there has been a growing need for sophisticated anomaly detection algorithms within the network intrusion detection system (NIDS) that can tackle advanced network threats. Advances in Deep and Machine learning (ML) has been garnering considerable interest among researchers since it has the capacity to provide a solution to advanced threats such as the zero-day attack. An Intrusion Detection System (IDS) is the first line of defense against network-based attacks compared to other traditional technologies, such as firewall systems. This report adds to the existing approaches by proposing a novel strategy to incorporate both supervised and unsupervised learning to Intrusion Detection Systems (IDS). Specifically, the study will utilize deep Autoencoder (DAE) as a dimensionality reduction tool and Support Vector Machine (SVM) as a classifier to perform anomaly-based classification. The study diverts from other similar studies by performing a thorough analysis of using deep autoencoders as a valid non-linear dimensionality tool by comparing it against Principal Component Analysis (PCA) and tuning hyperparameters that optimizes for 'F-1 Micro' score and 'Balanced Accuracy' since we are dealing with a dataset with imbalanced classes. The study employs robust analysis tools such as Precision-Recall Curves, Average-Precision score, Train-Test Times, t-SNE, Grid Search, and L1/L2 regularization. Our model will be trained and tested on a publicly available datasets KDDTrain+ and KDDTest+.

Table of Contents

Chapter Page

Chapter Page

List of Figures

List of Tables

Chapter I: Introduction

Intrusion Detection System

The idea of the Intrusion detection system (IDS) as the first line of defense against network intrusion can be traced back to Dorothy Denning's seminal paper named 'An Intrusion-Detection Model' where she first proposed a model for a real-time detection system capable of detecting various forms of threats [1]. Since then, the IDS has come a long way especially with the recent advancements in machine learning, big data, and an industry-wide shift to the cloud.

Intrusion detection systems can be divided into variants depending on its detection method. The first one follows the signature-based detection technique and the second follows an anomaly-based detection technique. An IDS that follows the intrusion-based technique matches up the signature of a potential threat against its database of known attacks and decides accordingly. Under the anomaly-based scheme, IDS are rigorously trained on learning normal traffic flow patterns using machine learning algorithms which allows the IDS to detect abnormal traffic.

The shortcoming of a signature-based IDS is its inability to catch threats that are not known beforehand since it heavily relies upon its database of known attacks. The signature-based detection method has been widely popular because of its high precision rates and low memory consumption [2]; however, attacks have been becoming more sophisticated over the years. Threats such as zero-day attacks are not publicly known before infecting a host system or organization. Attacks of such nature

can wreak havoc since it takes advantage of the required time it takes to patch up an IDS against that threat.

On the other hand, the anomaly-based detection method performs better against zero-day since it is trained on good traffic flow and can detect an anomalous pattern. One criticism of the detection method in question would be its high false-positive and high memory consumption which is required in the training phase of the detection algorithm. Solving this challenge would be one of the main themes of this study.

Machine Learning Algorithms

There has been a growing interest in the use of machine learning (ML) algorithms to catch anomalies since they are considerably better than traditional classification algorithms [3]. The use of these advanced algorithms has played an important role in increasing precision in detecting anomalies. Various machine learning algorithms such as Random Forest (RF) [4], Support Vector Machine (SVM) [4], K-Nearest Neighbors (KNN) [5], and Naïve Bayes have been utilized to optimize the anomaly-based systems.

At its core, anomaly detection is a classification problem. The machine learning algorithms generally do an excellent job of catching threats, but there is an added computational cost, which is a challenge for cybersecurity experts since anomalies must be dealt with in real-time scenarios. In addition, with the advent of edge computing, building an algorithm that is not heavy on computational power is becoming even more crucial.

To tackle these issues, the recent implementation of deep learning, a subfield of ML, in anomaly-based detection methods have been quite promising. Neural nets allow for more robust and thorough learning on inputs owing to its rigorous utilization of optimization techniques based on neural networks. Neural Nets are further built on principles from calculus, linear algebra, and probability.

The most basic structure of a neural net is composed of three layers: an input layer, hidden layer, outer layer. Neural networks containing 2 or more hidden layers are considered *Deep Neural Networks* [6]. Hidden layers in the neural architecture enable backpropagation which allows the neural nets to iteratively adjust the associated weights and biases of a given neuron by comparing it against the outcome labels.

There are many hyperparameters within a given neural network scheme which can be adjusted so that it can govern the way the model is trained on the input data. One of the distinguishing hyperparameters is *Learning Rate, Epochs, Hidden Layers, Neurons,* and *Activation Function*[1]*.*

KDDCUP99 and NSL-KDD Dataset

The KDDCUP99 dataset was prepared and developed by the MIT Lincoln Lab in the year 1998 under the DARPA Intrusion Detection Evaluation Program [7]. The dataset contains 'bad' and 'good' connections acquired through nine weeks of raw TCP dump of a military network environment. Various studies have performed anomaly-

[1] The definition of each of these terms are provided in Chapter IV- *Definition of Terms.*

based modeling on this dataset to gauge their system's performance [8]. Even though there are over 37 attacks in the dataset, they can be broadly categorized in five as four attack types which can be seen in table 1. However, the KDDCUP99 dataset has several issues such as the redundancy of records owing to the synthetic nature of the data. This issue can cause statistical errors since being trained on redundant data can cause the model to be biased towards the frequent records [8]. Because of this reason, we will be implementing our proposed anomaly detection method on the enhanced NSL-KDD dataset which addresses the previously mentioned issue. Our study will focus on this dataset to gauge the accuracy of our proposed algorithms. Our study will evaluate the dataset as both a binary (Attack/Normal) and multi-class.

Table 1

Attack Types in NSL-KDD Dataset [9]

Attack Type	Description	Training Dataset	Testing Dataset
DoS	Denial-of-Service	45927	7456
Probe	Surveillance and other probing	11656	2421
R2L	Unauthorized access from a remote machine	995	2756
U2R	Unauthorized access to local superuser (root) privileges	52	200

Problem Statement

There are potential threats within the network traffic that can compromise a target application or computer. Some well-known attacks that can hinder a legitimate user from accessing system resources are DOS attacks that can flood your system with connection requests, thus rendering your host machine useless. MITM (Man-In-The-Middle) attacks intercept network communication in order to listen in on the exchange of information. Spoofing attacks attempt to mimic an authorized user so that they can convince the system to provide access to the attacker. It does so by sending IP packets from a known Host. User to Root Attack (U2R) bypasses the system security by gaining access through the network option. Application layer attacks exploit the lapses within the application layer, so there might be a security weakness in the server-side.

However, the most relevant for this study would be unseen attacks. Zero-day malware, which is a sophisticated form of attack because there is no known prior information regarding the threat. Training the IDS to tackle this form of attack would

require a security mechanism that is proficient in distinguishing between normal and anomalous network traffic. There is a need to classify the different types of threats based on different features that yield higher detection with increase precision and recall so that the user can stay protected against the modern network-level attacks.

Nature and Significance of the Problem

Malicious threats have the potential to wreak havoc within a system's infrastructure. Therefore, the importance of detecting anomalies within given network traffic is crucial [10]. With the recent boom in the volume of data owing to Cloud-based services, faster internet speeds, and internet-of-things (IoT) devices, there has been an onslaught of more sophisticated attacks that defense mechanisms like Internet Firewall are unequipped to handle. The internet speeds have gone up to 100Gbps or more, and the data is forecasted to grow to 44 ZB [11]. In addition, because of this surge in network data, we are seeing a change in the diversity of data and protocols transmitting through the network traffic. If a computer host or application does not have an effective intrusion detection system, it could pose a threat to data confidentiality, data integrity, and vulnerabilities to denial-of-service (DOS) attacks.

The network intrusion threats that are prevalent today can pose a significant risk to the operational security of big corporations, governments, and individual users. In the past decade, there have been 14 major website breaches, which include attacks on the National Assembly, Shinhan Bank, the defense ministry, web sites of presidential blue house, New York Stock Exchange, among others. In 2009, a google employee

uploaded a malware site that declared the entire Internet to malware for 55 minutes. This breach caused a lot of reputational damage to Google as well as financial damage from ad revenue lost [2].

Objective of the Study

The primary objective of this study is to evaluate the effectiveness of our proposed neural network scheme by measuring the performance against well-known performance and classification metrics. The performance we will be using for this study is the Precision-Recall curve, F1-micro, Prediction Accuracies, and Matthews Correlation Coefficient. It is expected that combining autoencoder-based representation learning with an SVM would lower the computational requirements during the train and test phase of the model. The lowered computational and storage requirement is essential against time-sensitive network threats that an intrusion detection system must face.

Study Questions/Hypotheses

1. Does employing deep autoencoders as a non-linear dimensionality reduction technique lead to better classification metrics than a linear dimensionality reduction technique?
2. Does deep autoencoder provide a better alternative to dimensionality reduction than its linear counterparts from a reduced train/test time and memory consumption viewpoint?

3. Does incorporating the regularization penalty term in our model's loss function enhance the model performance? If so, what form of regularization (*L_1, L_2, None*) is most effective within the proposed neural network scheme?

Summary

There has been an extensive number of studies conducted for anomaly detection by using supervised machine learning approaches such as KNN (k-nearest neighbor), support vector machine and artificial neural networks (ANN) [5]. The combined approach of unsupervised deep learning for feature reduction and supervised machine learning for classification has been proven to be better in terms of lowered training and resource consumption viewpoint. In the coming chapters, we will delve deeper into what neural net schemes other researchers have employed to enhance intrusion detection systems.

Chapter II: Background and Review of Literature

Introduction

There has been extensive research concerning the implementation of deep neural networks within the network intrusion detection system domain. In this section, we will explore different scholarly journals that have implemented disparate neural network architecture on the KDDCUP99 and the NSL-KDD. In the first section of the literature review, we will briefly discuss the inner working of machine learning and neural networks in order to understand the calculations being performed in the backend. In the later section of the literature review, we will identify various deep learning structures utilized for Network Intrusion Detection Systems (NIDS) for higher accuracy and prediction. In the last part of this section, there will be a brief discussion regarding how our study deviates from the existing body of research with respect to deep learning and anomaly detection.

Background Related to the Problem

Deep Learning

Deep learning is a subset of machine learning that attempts to model the human brain through mathematical functions that imitate the neuron. It finds patterns from raw data without the need to be explicitly programmed. The deep learning algorithms have been around for decades but it has recently come to the fore owing to the plethora that is available in present-day [12]. The architecture of a neural network is composed of an input layer, a hidden layer, and an output layer. Every layer is

composed of neurons or a perceptron, which his considered the building block of deep learning. Each node connects one neuron with another through successive layers. Each neuron imposes weights and biases on the input provided. Then the product sum of weights and biases is sent through an activation function so that the output accommodates non-linearity, which is crucial when dealing with classification problems. For instance, a typical activation function would be the sigmoid function. When an input is passed onto the sigmoid function, it collapses the product sum of learned weights into a range from 0 to 1. Once the product sum of weights is non-linearized with the activation function, we get the output in the output layer. This entire process is called feedforward propagation. The output is then measured against the actual value and then trained iteratively to minimize the error between the initial prediction and the actual value. This iterative process of adjusting the weights and biases of input features is known as backpropagation. Backpropagation attempts to perform loss optimization. The process of minimizing or optimizing the loss function iteratively is called *gradient descent.* There are many loss functions, but the most used are C*ross-Entropy Loss* and *Mean Squared Error Loss* (RMSE). In figure 1, we can see the neurons in the input layer and its interconnectedness with the subsequent hidden and output layers.

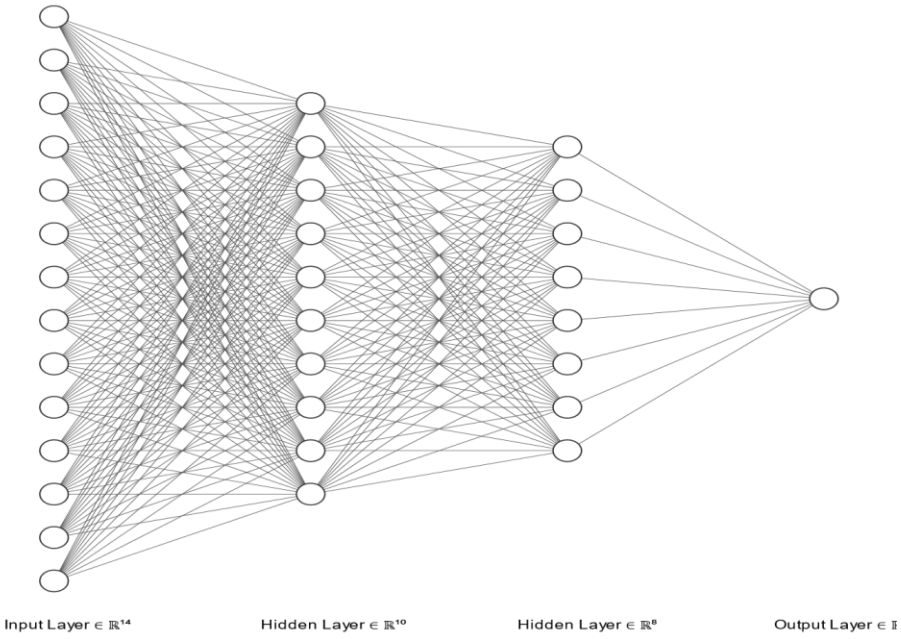

Figure 1. Neural Network Architecture

Autoencoders

The autoencoder is a neural network that is trained to copy its input to its output [13]. The neural network contains two primary mathematical functions that allow the input to be reconstructed into output. The encode function $h = f(x)$ and the decode function $r = g(h)$. ***h*** is the internal representation when the input ***x*** is being converted to ***r*** (called reconstruction). In making the approximate copies of input, the neural network learns the most useful properties of the raw dataset. Typically, the ***h*** is always a lower-dimension subspace of the ***x*** because of this autoencoder neural networks have a bottleneck layer that has a lower number of nodes than the other layers [14].

Activation Functions. The $f(.)$ and $g(.)$ are the activation functions that non-linearize the bias and weight parameters. There are many variations of activation functions that are used within the neural network such as sigmoid, Tanh, and ReLU activation functions [14]. Relu, which stands for rectified linear unit, is a widely used activation function in deep neural networks [15]. The primary reason for the success of this activation function is that it does not require a lot of computational resources to execute it compared to complicated activation functions that lead to increased difficulty in optimization. Mathematically, ReLU is presented as:

$$y = \max(0, x) \tag{1}$$

ReLU activation function yields an output of 0 when $x < 0$, and then draws a linear line with a slope of 1 when $x > 0$.

Our paper incorporates the usage of Scaled Exponential Linear Unit (*SELU)* which is one of the newer activation functions being used in deep learning as of now. In *Self-Normalizing Neural Networks* [16]*,* the author posits SELU activation function which exhibits as a self-normalizing property which is proven using the Banach fixed-point theorem. Essentially, the activations that are closer to zero mean and unit variance allows the network layers to converge to zero mean and unit variance [32].

Having SELU in our scheme is relevant since we will be employing deep autoencoders which employ more than three deep layers. Usually, when employing more than three layers in a neural architecture which can put the forward neural network

at a disadvantage since the lack of normalization within the activation function could lead to gradient issues. Since with SELU, the normalization occurs within the function, we can bypass that issue and take full advantage of this activation function.

SELU allows room for deeper network layers owing to its faster processing speeds, in addition since the activation encourages normalization there is a presence of regularization penalty. In reference to the previously mentioned paper, the authors meticulously derived two fixed parameters used in the feedforward process. For standard scaled inputs (mean 0, standard deviation 1), the parameters are a=1.6732~, and λ= 1.0507 ~. Having fixed parameters our neural network to not backpropagate through these variables. SELU can be mathematically presented as:

$$SELU(x) = \lambda \begin{cases} x & if\ x>0 \\ \alpha e^{x} - \alpha & if\ x \leq 0 \end{cases} \quad (2)$$

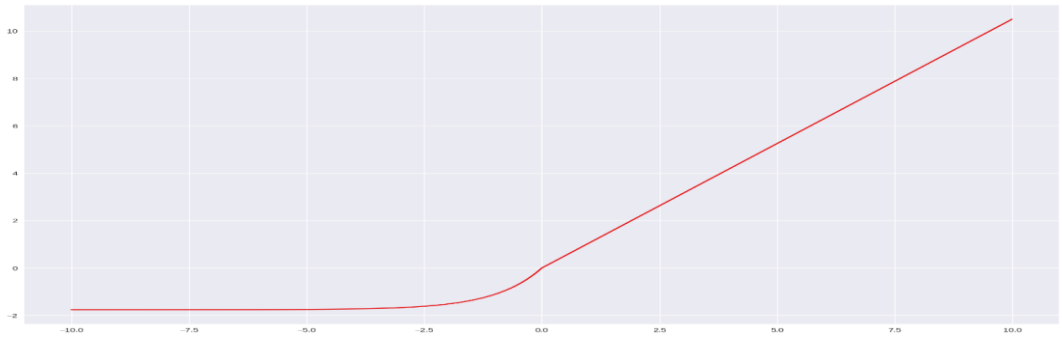

Figure 2: SELU plotted for a=1.6732~, Lambda=1.0507~

Deep Autoencoders as a Dimensionality Reduction Tool. Merely reconstructing inputs into an output variable is not considered useful since it does not achieve much. However, the true power of autoencoder resides in the internal representation of the encoded input *x* [33]. The lower-dimensional representation allows improved performances especially when performing classification tasks. Classifier trains faster on lower dimensions due to a lower need for memory and computational power [34]. The autoencoder learns by minimizing the following loss function:

$$L\left(x, g\left(f(x)\right)\right) \tag{3}$$

In the equation above, the loss function is penalizing $g\left(f(x)\right)$ for $\boldsymbol{x}$**.** In fact, if the decoder function is linear and the $\boldsymbol{L}$ is mean squared error, the resulting subspace is the same as Principal Component Analysis (PCA) [13]. The autoencoder scheme is typically composed of three primary layers. First, the encoder layer, where the inputs are assigned weights and biases. Afterward, the inputs are reduced to the most useful features in the code layer. As seen in figure 2, the number of neurons in the input layer is typically higher than in the 'code layer.' Lastly, the decoder layer consists of a decoder function that reconstructs the input. However, the code layer contains the latent representation of the input vectors which is essential for classification-based tasks. When the code layer has a smaller dimension than the input dimension then it is called **undercomplete** [13]**.**

Autoencoders is a viable non-linear feature reduction technique and tends to outperform other dimensionality reduction techniques. Wang, Yao, and Zhao compare

autoencoder as a dimensionality reduction algorithm against state-of-the-art dimensionality reduction algorithms such as Principal Component Analysis, Linear Discriminant Analysis, Locally linear embedding, and Isomap [17]. The study concludes that auto-encoders not only outperform other techniques in reducing dimensionality, but it is also good at detecting repetitive structures [17]. However, there are other studies that give preference to PCA, linear dimensionality reduction technique, for real-world tasks as opposed to artificial tasks [18].

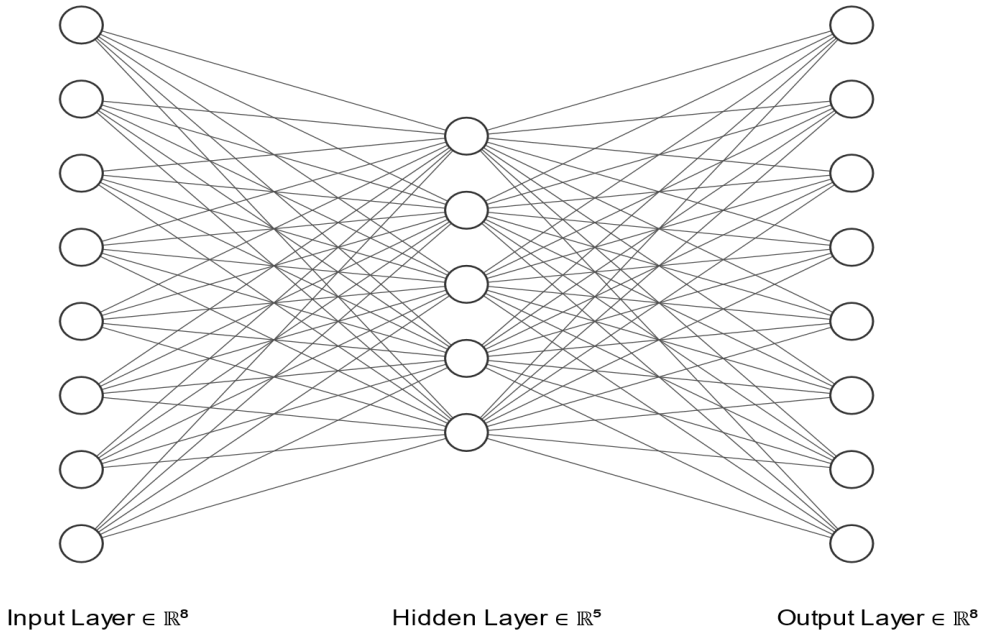

Figure 3: Autoencoder (AE) Neural Architecture

There are many types of autoencoders such as Sparse, Deep, Denoising [19], Convolutional [20], Contractive [21] and variational Autoencoders. The AE that this study would be employing deep autoencoders (also named Stacked Autoencoder) [19].

Deep AE has several hidden layers, and deeper stacked AE is considered to have a better training capability compared to lesser layers [13] [22].

Support Vector Machine (SVM)

Our dataset will be trained using a support vector machine (SVM) for anomaly classification on the NSL-KDD dataset. SVM algorithm attempts to find a hyperplane (subspace with a dimension that is one less than of its ambient space) that distinctly classifies the data points. It does so by making use of support vectors that are points closest to the hyperplane (i.e. decision boundary). The hyperplane is positioned such which allows the support vectors to sit equidistantly to the hyperplane. The algorithm calculates the maximum margin hyperplane which is a margin that yields the highest sum between the two support vectors. Mathematically, the algorithm for a binary SVM classifier would be represented as follows:

$$f(x_i)\begin{cases}\geq 0 & y_i = +1\\ < 0 & y_i = -1\end{cases} \tag{4}$$

Unlike logistic regression which squashes output of its linear function within the range of 0 to 1, SVM squashes its output from the range of -1 to 1. In other words, the SVM algorithm classifies the datapoints as negative and positive 1. SVMs are known to be effective when working with high dimensional data, even when the number of features is higher than the actual number of data points being used. In addition, SVM is a flexible classifier in that we can choose among different kernel functions, such as linear, polynomial, Radial Basis Function (RBF), sigmoid, and even custom kernels using Python. For the purpose of our study, we will choose the RBF kernel since it

allows us to perform non-linear classification on our dataset. The radial basis function could be mathematically written as such:

$$\text{RBF: } \exp(-\gamma \parallel x - x' \parallel^2) \tag{5}$$

Regularization

In order to ensure that our autoencoder representation contains the highest informative weights, we can take advantage of a principle called regularization. Essentially, regularization penalizes complexity and encourages simplicity in the training model. The way it does so is by adding a regularization term in the loss function of our neural network loss function. Having this regularization term in the loss function ensures that there is no risk of overfitting. Overfitting typically happens when our model overlearns our dataset's training set to the point where it also picks up on the specific quirks and outliers. When we incorporate regularization to our neural scheme, we can ensure that our model is built to predict unseen data instead of becoming over-trained to predict data of the existing training dataset instead of the out-of-sample test set. There are various forms of regularization techniques with each having their own advantages and disadvantages depending on the nature of the dataset that we are working with. There are L_1, L_2, and L_0 regularization terms that are typically employed to penalize complexity in each model [23]. For our report, we will use apply all three of these regularization techniques and gauge which one (or lack thereof) helps us minimize the loss on our testing dataset. Mathematically, L_2 regularization could be defined as:

$$\|w\|_2^2 = w_1^2 + w_2^2 + w_3^2 + \cdots + w_n^2 \tag{7}$$

The L_2 regularization term quantifies the model complexity by takin the squared sum of all the calculated weights. The optimization of the neural network is contingent on the minimization of the following loss function and the regularization term:

$$minimize\big(Loss(Data|Model) + \lambda\, complexity(Model)\big) \quad (8)$$

If the lambda in the above equation is zero, then the regularization term is removed completely. Setting the right lambda parameter is important when setting up regularization for our model. The lower the lambda value, the more complex the model becomes and vice versa. On the other hand, L_1 regularization is mathematically defined as:

$$\|w\| = |w_1| + |w_2| + |w_3| + \cdots + |w_n| \quad (9)$$

Essentially, L_2 regularization motivates the model weights to converge around 0 whereas L_1 regularization forces the weights to be exactly 0. Depending on the situation, L_1 may be better than L_2 under certain circumstances. For instance, the model that has sparse vectors is better off with L_1 regularization since they remove many sparse weights thus putting less load on RAM and increasing lowering training and testing time. In our report, we will compare both L_1 and L_2 regularization to gauge which scenario leads to desirable outcomes against our classification metrics.

Literature Related to the Problem

In *Intelligent intrusion detection systems using artificial neural networks* [24]*,* the authors present a model based on artificial neural networks (ANN), which is a supervised deep learning classifier. The study performs the grid search technique and

decides on using a multi-layer perceptron with two hidden layers composed of 30 neurons each. In addition, a 10-fold cross-validation technique was utilized to get more robust results. The model yielded a high area under the ROC curve, which indicates better classification. The average AUROC came out to be 0.98, SD AUROC 0.02, Maximum AUROC 1.00, and Minimum AUROC 0.82.

Apart from the usage of supervised deep neural nets for training IDS on abnormality, there are other deep-learning frameworks that can be adopted. A promising approach and one that is the focus of this study would be the Self-taught learning (STL) framework, which is essentially composed of two stages. The first stage employs unsupervised deep learning for feature and dimension reduction. Whereas in the second stage, there is a use of traditional machine learning models for classification. The combined usage of both approaches yields results that have shown considerably better results than other frameworks based on a study performed by Majjed, Lasheng, Al-Habib, and al-Sabahi [25] . According to Majjed et al., this approach is efficient in terms of computational cost. In addition, the STL approach contributes to an overall increase in detection accuracy compared to other shallow machine learning classifiers.

A study that closely follows the same methodology named *Autoencoder-based Feature Learning for Cyber Security Application* performs an autoencoder based deep learning scheme where the reduced features are then classified based on multiple machine learning algorithms [26]. The study draws its data from two main datasets: KDDCUP99 Dataset and Malware Classification Dataset published by Microsoft at

Kaggle in 2015. The study compares the results with standalone machine learning classifiers as well as Autoencoder-based input features. The study indicated that Gaussian Naïve Bayes coupled with AE-based features proved to have higher intrusion detection accuracy than other deep learning models, such as Xgboost, H20 models, among others.

In a similar vein, Shone, Ngoc, Phai, and Shi combines the use of Non-symmetric Deep Auto-Encoder (NDAE) with Random Forest (RF). Each NDAE has three hidden layers with the same number of neurons in each layer [11]. The primary way NDAE stands out from other autoencoders is that it does not follow the typical encoder-decoder paradigm, but instead, it just employs the encoder formula in the outer layer process, so in that sense, this scheme is considered non-symmetric in nature. The scheme is implemented on both the KDDCUP99 and NSL-KDD dataset. The analysis is performed as a 5-class KDD classification and 13-class KDD classification. The comparisons generated after building the models indicated an improvement of 5% in accuracy and training time reduction to 98.1%.

In the *Comparative Study of Deep Learning Models for Network Intrusion Detection,* the authors posit three main approaches to the anomaly classification problem [27]. The first approach is the STL model, which yields an average accuracy of 98.8% across four classes of anomalies – namely -- DoS, Probe, R2L, and U2R. The second approach is based on Recurrent Neural Network (RNN), which considers previous lags of input feature which allows for an additional memory input. Having that

additional memory input allows adding a temporal dimension to the analysis. However, the RNN based on Long Short-term Memory yielded an average accuracy of 79.2%. Lastly, the deep neural network approach on the KDD dataset yielded an accuracy of 66%.

Summary

One can safely surmise that the implementation of a deep learning approach to network intrusion detection systems is still in its nascent stages. Given how complex the model building process can become given its various configuration (i.e. training, optimization, activation, and classification) and other model-specific configurations (number of hidden layers, learning rate, loss function) we believe that our approach of using deep autoencoders with SVM classifier would prove to be a substantial contribution to the existing body of literature on the topic. In the next chapter, we will precisely discuss the model building process and the methodology of the study.

Chapter III: Methodology

Introduction

In this section, we will set the bounds of the study by defining the metrics of the study. Furthermore, we will address the specifics of the data preprocessing and hyperparameter tuning aspect that has been achieved so far in the study.

Definition of Terms

- **Intrusion Detection System:** An intrusion detection system is a device or software application that monitors a network or system for malicious activity or policy violations.
- **Network-Level Attacks:** Network-delivered threats that typically gain access to the internal operating systems. Common types of network attacks are Denial-of-Service (DOS), spoofing, sniffing, and information gathering.
- **Recall:** quantifies the number of positive class predictions made out of all positive examples in the dataset
- **Precision:** indicates the proportion of correct predictions of intrusions divided by the total of predicted intrusions in the testing process
- **Accuracy:** indicates the proportion of correct classifications of the total records in the testing set
- **F-score:** Provides a single score that balances both the concerns of precision and recall in one number.

- **Training and Testing Time:** the number of seconds it takes for neural network or classifier to train and test respectively on the dataset.
- **Machine Learning:** Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.
- **Deep Learning:** Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled.
- **Activation Function**: the activation function of a node or neuron defines a numerical output by taking input or a set of inputs.

Data Preprocessing

During the preprocessing phase of the study, the categorical features within the dataset were encoded using one-hot encoder which binarizes the categorical values into 0 and 1. The numeric variables were normalized using L2 normalization. *Sklearn.preprocessing.Normalizer* library was used to perform the normalization. Normalized numeric inputs are a requirement for many neural network schemes [28]. Furthermore, the normalization of numeric inputs helps to avoid outliers that might be present in the dataset.

Hardware and Software Environment

Operating System: Windows 10 Home 64-bit (10.0, Build 17763)

BIOS: X510UAR.309 (type: UEFI)

Processor: Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz (8 CPUs), ~1.8GHz

Memory: 8192MB RAM

DxDiag Version: 10.00.17763.0001 64bit Unicode

Software Environment: Python 3.7.5 64-bit | Qt 5.9.6 | PyQt5 5.9.2 | Windows 10

Python Packages: TensorFlow 2.0.0, NumPy 1.17.2, Pandas 0.25.2

Design and Implementation of the Study

The study is going to be quantitative, as anomaly detection is essentially a classification problem. We have selected deep autoencoders as the first stage for dimension reduction and Support Vector Machine (SVM) classifier as the second part for performing classification on the encoded vector.

The number of hidden layers and activation neurons will be contingent on the grid search algorithm, which performs hyperparameter tuning and selects parameters that optimize the loss function. Instead of utilizing PCA, LDA, or other forms of dimension reduction, we will use deep autoencoders as a form of non-linear dimensionality reduction [30]. Once we receive the subset vector of reduced features, we will employ a support vector machine for the classification process.

The python libraries that are relevant to the study are *Numpy, Pandas, Scikit-learn* and *Keras*. Python was chosen over other statistical programming languages since Python has a broader range of libraries, which makes it an ideal choice for performing deep learning analysis. The classification will be performed on binary as well

as multi-class labels. The first scenario binarizes the label as a simple normal and attack label. Whereas, in the multi-class scenario, we have classified the labels in five total attack type labels.

Tools and Techniques

Grid search is a technique used for performing hyperparameter tuning. The technique provides the optimal hyperparameters by performing an exhaustive search through the provided parameter grid. The provided grid for the paper contained the following parameters: epochs, loss function, kernel function, activation function, and the number of k-folds for cross-validation. K-fold cross-validation allows the user to segment the training dataset into multiple folds so that the results are less biased and not overfitted. The loss function for the model would be RMSE, which is the square root of the average of squared differences between prediction and actual observation. RMSE loss function is mathematically presented as

$$RMSE = \sqrt{\frac{1}{n}(Y_j + \check{Y}_j)^2} \quad (9)$$

Performance Evaluation

In a two-by-two confusion matrix, the four possible outcomes are as follows:

1) *True Positive (TP): Attack data that is correctly grouped as an attack.*
2) *False Positive (FP): Normal data that is incorrectly grouped as an attack.*
3) *True Negative (TN): Normal data that is correctly grouped as normal.*
4) *False Negative (FN): Attack data that is incorrectly grouped as an attack.*

Most of the performance metrics that will be discussed are based on these four possible outcomes [29]. The concept of *classification threshold* is tightly linked to confusion matrix outcomes since we must define a threshold value that helps us make the decision of when to indicate an outcome 'abnormal', and 'normal'.

Accuracy

Based on the four measures computed from the confusion matrix, we can compute *accuracy* which is the fraction of the number of correct predictions to the total number of predictions. We can formulate accuracy as such:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{10}$$

The accuracy rates would be the key performance indicator for the several machines and deep learning classifiers we will be working on throughout the study.

Precision-Recall Curve

The *precision* and *recall* metrics are very important measures when dealing with imbalanced classes. *Precision* calculates the proportion of positive identifications that were correct. It could be mathematically defined as:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

Recall is defined as the proportion of actual positives correctly identified. It can be written as so:

$$Recall = \frac{TP}{TP+FN} \tag{12}$$

Typically, there is a trade-off between precision and recall as the decision threshold is adjusted. An increase in the classification threshold always causes the recall to decrease or stay the same. Whereas, the increase in classification threshold increases the precision [30]. The precision-recall curve shows the inverse relationship graphically.

F-measure

The F-measure is a single value metric that is based on *Precision* and *Recall.* The range of the F-measure is between 0 and 1. F-measure allows to take the precision and recall into account simultaneously through just one measure as opposed to looking at them form a trade-off point of view. The F-measure can be mathematically defined as:

$$FM = \frac{(1 + \beta^1) * Recall * Precision}{Recall + Precision} \tag{13}$$

Test and Train Timings

The test and train timings are crucial for this study owing to the time-sensitive nature of network-level attacks. Therefore, having a model that can train much faster than other algorithms would prove to be quite important for our study.

Chapter IV: Results

Visualizing Data using t-SNE

To visualize our dataset, we employ t-distributed Stochastic Neighbor Embedding (t-SNE) which is a tool for visualizing high-dimensional data. T-SNE employs Kullback-Leibler (KL) divergence which is a measure of the difference between two probability distributions [31]. KL divergence could be interpreted as a dimensionality reduction technique since it converts observations into joint probabilities which lower the overall information being processed. However, when dealing with higher dimensional data (generally over 50 dimensions), it is recommended to apply prior dimensionality reduction technique which would bring manageable subspace that T-SNE can handle effectively since it demands a considerable amount of computing power. In addition, performing T-SNE on reduced dimensions also manages the noise without distorting the interpoint distances. In our situation, we would apply PCA (linear dimensionality reduction) and deep autoencoder (non-linear dimensionality reduction) and see how t-SNE visualizes the reduced dimensions from both the techniques. We were able to bring down the initial feature space of 122 inputs to a subspace of 10 dimensions using deep autoencoders. We will be using t-SNE to visualize a 2D manifold for our dataset with adjustments to different perplexity values. Perplexity can be interpreted as a smooth measure of the effective number of neighbors. Generally, the perplexity value ranges from 5 to 50. However, when dealing with larger datasets, it is fine to go with higher perplexity value. In the following figures, we can see the t-SNE representation of

PCA and AE-encoded dimensions with adjusted perplexity and iterations. We can observe that both PCA and AE is able to cluster different attack groups in higher perplexity and iterations as opposed to lower perplexity and iterations.

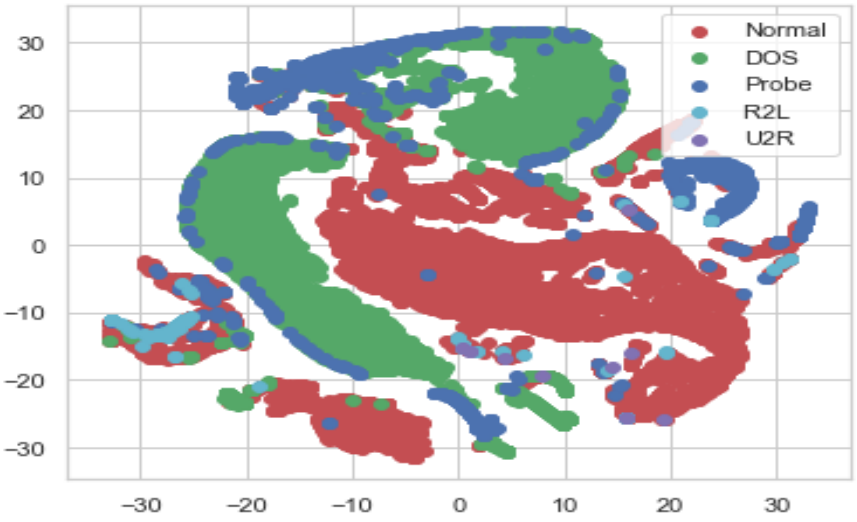

Figure 4 t-SNE Representation of Encoded Representation (Perplexity= 50, Iterations=500)

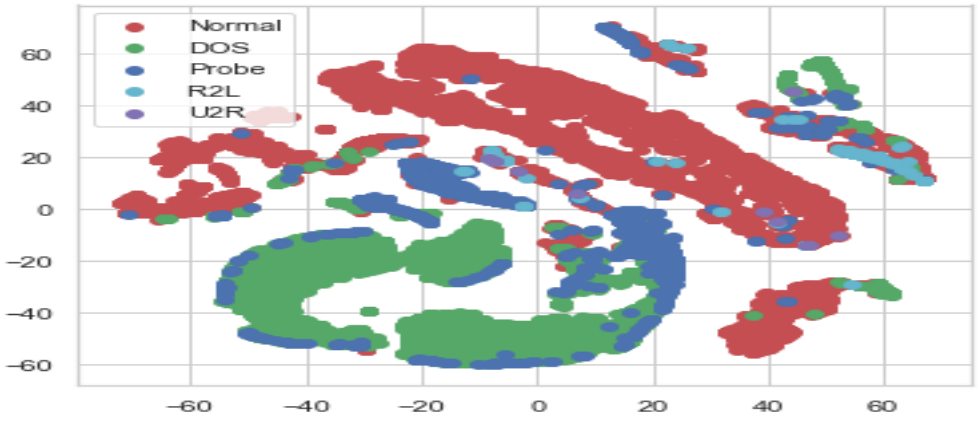

Figure 5 t-SNE Representation of Encoded Representation (Perplexity= 100, Iterations=500)

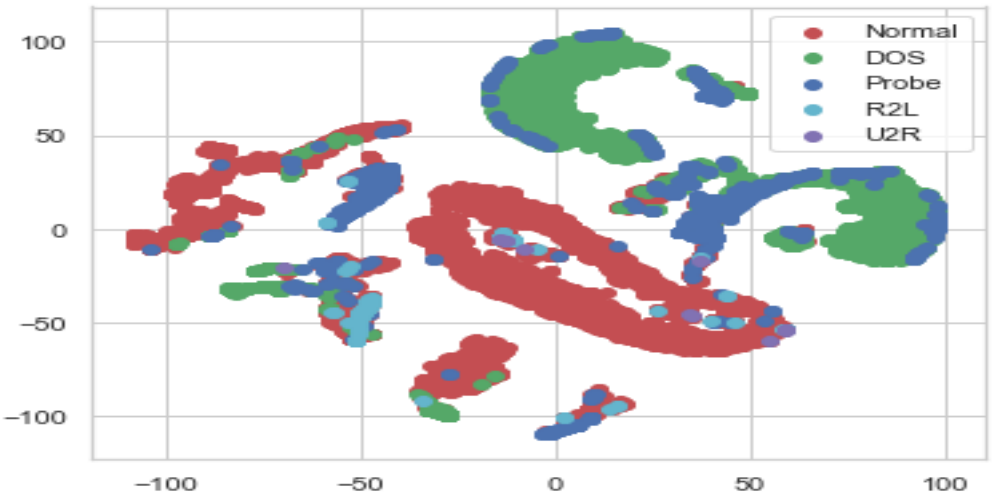

Figure 6 t-SNE Representation of Encoded Representation (Perplexity= 50, Iterations=1000)

The data points in the figures are color-coded according to five attack classes within the NSL-KDD dataset. The vertical and horizontal axis in the graph above is generated using the KL-divergence algorithm and is used to embed high-dimensional data space into a lower subspace.

As the perplexity and iterations become higher, we can observe 'normal' class data points (red dots) coalescing. A normal class is distinctly separate from the other attacks labels which indicate that PCA and DAE are quite effective at separating 'normal' and 'attack'. However, when focusing on the other four attack class, we see a different scenario; except for 'DOS' attack class, we see the other three attack types being jumbled together and not forming a clear cluster. The t-SNE representation of principal component analysis and deep autoencoder suggest that classifying the other three attack types might be challenging to classify.

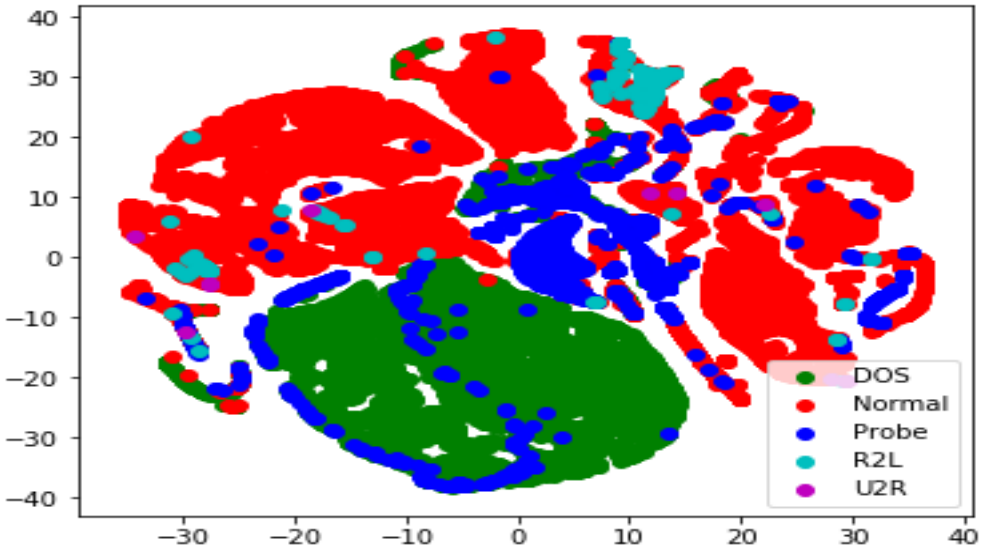

Figure 7 t-SNE Representation of PCA (Perplexity= 50, Iterations= 500)

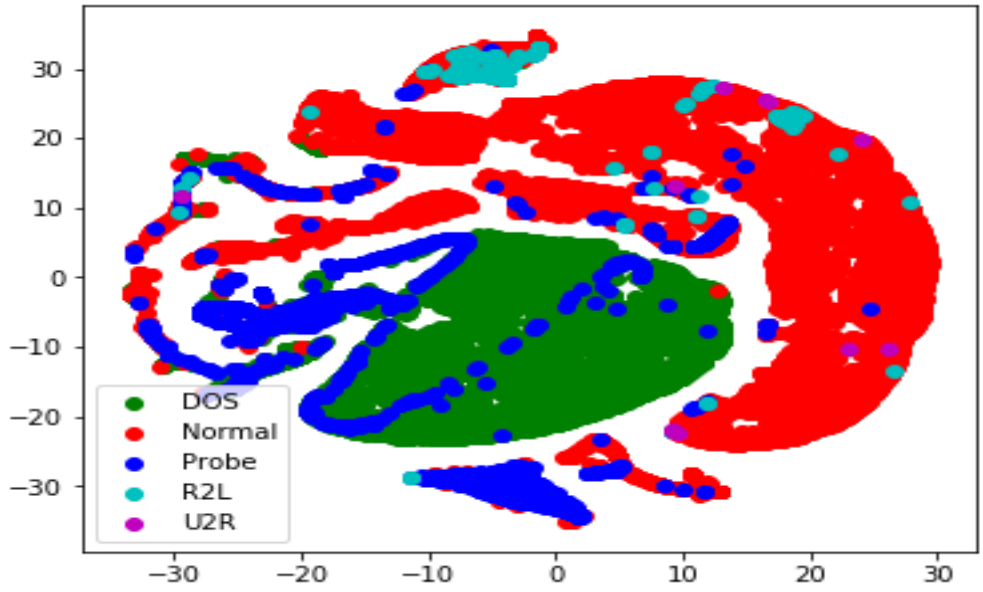

Figure 8 t-SNE Representation of PCA (Perplexity= 100, Iterations= 500)

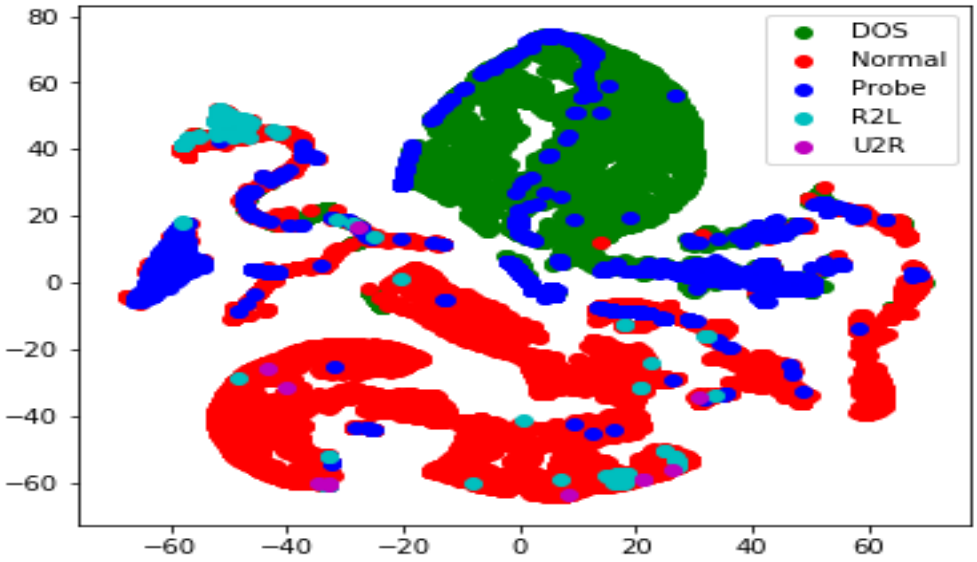

Figure 9 t-SNE Representation of PCA (Perplexity= 100, Iterations= 1000)

Grid Search

In order to choose parameters for our classifier optimally, we employed the GridSearchCV available to via scikit-learn which is an open-source machine learning library available on Python The GridSearchCV attempts to perform an exhaustive search over specified parameter values for our SVM classifier based off against important score. For our purpose, we will base our grid search against Micro-recall and balanced accuracy.

Important parameters to consider when working with SVM with RBF kernel function is the selection of *Gamma* and *C* parameters. *C* variable is seen as a regularization term that keeps sparse variables in check. The optimal parameter selection is contingent on the level of noise and balance that the dataset exhibit.

Since our dataset is non-linear, we employed the kernel functions available in the scikit-learn python library. We can choose out of four kernel functions –namely—linear, polynomial, Radial Basis Function (RBF), and sigmoid. For our analysis, we employed 'RBF' since our grid search predicted RBF to be the best kernel function in order to optimize F1-Micro (Table 2).

Table 2

Grid Search with 'F1-Micro' Scoring

F1-Micro Score	***Std***	***C***	***kernel***
0.704	(+/-0.258)	1	linear
0.815	(+/-0.011)	1	rbf
0.781	(+/-0.019)	1	poly
0.497	(+/-0.017)	10	linear
0.883	(+/-0.010)	10	rbf
0.851	(+/-0.013)	10	poly
0.567	(+/-0.023)	100	linear
0.928	(+/-0.011)	100	rbf
0.903	(+/-0.014)	100	poly
0.665	(+/-0.028)	1000	linear
0.949	**(+/-0.013)**	**1000**	**rbf**
0.937	(+/-0.012)	1000	poly

Table 3

Grid Search with 'Balanced Accuracy' Scoring

Balanced Accuracy	***Std***	***C***	***kernel***
0.458	(+/-0.035)	1	linear
0.823	(+/-0.094)	1	rbf
0.755	(+/-0.081)	1	poly
0.611	(+/-0.056)	10	linear
0.841	(+/-0.154)	10	rbf
0.851	(+/-0.100)	10	poly
0.66	(+/-0.079)	100	linear
0.849	(+/-0.126)	100	rbf
0.875	**(+/-0.106)**	**100**	**poly**
0.688	(+/-0.109)	1000	linear
0.828	(+/-0.091)	1000	rbf
0.845	(+/-0.135)	1000	poly

Classification Metrics

Accuracy, Precision-Recall, F-Score

In this section, we will closely examine three main models to interpret their effectiveness at predicting anomalies as well as the quality of their prediction by looking at subtler yet important measures such as precision, recall, and F-score. Simply looking at accuracy is not enough to gauge the effectiveness of our model, especially owing to the imbalanced label distribution of the NSL-KDD dataset.

We need to give extra importance in analyzing recall values for the models in question since false negatives (Type 1 error) can cause intrusions that could lead to a breach in the information system of another individual or organization. To avoid that scenario, the recall metric holds considerable importance to us as cybersecurity experts.

It is also important to look at precision, recall, and f1-score values of specific attack types, as that will help us understand a finer picture of the model's performance. For instance, not having a high recall in 'U2R' attack type would mean potentially giving root access information of the system to a hacker. Thus, measuring our models' effectiveness against specific attack type would be as important as focusing on the overall model, if we intend to build a robust anomaly-based IDS.

The first model that we will examine would be the standalone SVM model with input data that is scaled using L2 normalization. The predictions of the model are

generated against an unseen validation set provided in the NSL-KDD. In the binary label scenario (Table 4), we find the overall prediction accuracy to be 77%, whereas the model yields a weighted average f1-score of 76%.

Table 4

Binary Label Classification Report for Standalone SVM

Label	***precision***	***recall***	***f1-score***	***support***
Normal	65%	97%	78%	9711
Attack	97%	61%	75%	12833
Other Measures				
Accuracy			77%	22544
Macro Average	81%	79%	77%	22544
Weighted Average	83%	77%	76%	22544

In table 5, we use the same model on a multi-class scenario where attack label is further divided into four classes. The overall accuracy as well as weighted f1, recall, and precision drop quite significantly when performing multi-label predictions.

The second model incorporates dimensionality reduction using PCA with similarly adjusted SVM. Bearing on the binary data presented in table 6, we observe a reduced accuracy and recall of 74 percent. In so far as multiclass labels are concerned, we also

observed a decrease in weighted averages (Table 7) for recall, f1-score, precision, and overall accuracy compared to the multi-class standalone SVM model.

Table 5

Multi-Label Classification Report for Standalone SVM

Label	***precision***	***recall***	***f1-score***	***support***
Normal	70%	84%	77%	9711
DOS	90%	70%	79%	7460
Probe	59%	62%	60%	2421
R2L	42%	23%	30%	2885
U2R	4%	60%	8%	67
Other Measures				
Accuracy			69%	22544
Macro Average	53%	60%	51%	22544
Weighted Average	72%	69%	69%	22544

Our proposed model shows considerable increases in accuracy as well as average recall rates when handling binary labels (Table 8). Our proposed model does far better in terms of weighted averages of precision, recall, and f1-score, as well as accuracy, compared to the metrics of the previous two models when working with

multiple classes. However, when focusing on U2R, we find that the recall rates are considerably higher in the standalone SVM case than it is with our model (Table 9).

Table 6

Binary Label Classification Report for PCA+SVM Classifier

Label	***precision***	***recall***	***f1-score***	***support***
Normal	63%	96%	76%	9711
Attack	95%	56%	71%	12833
Other Measures				
Accuracy			74%	22544
Macro Average	79%	76%	73%	22544
Weighted Average	81%	74%	73%	22544

Table 7

Multi-Label Classification Report for PCA+SVM Classifier

Label	***precision***	***recall***	***f1-score***	***support***
Normal	61%	55%	58%	9711
DOS	88%	68%	77%	7460
Probe	48%	63%	54%	2421

R2L	19%	10%	13%	2885
U2R	1%	52%	2%	67
Other Measures				
Accuracy			55%	22544
Macro Average	44%	50%	41%	22544
Weighted Average	63%	55%	58%	22544

Table 8

Binary Label Classification Report for AE-Encoded + SVM Classifier (L2 Regularization)

Label	***precision***	***recall***	***f1-score***	***support***
Normal	67%	97%	80%	9711
Attack	96%	65%	77%	12833
Other Measures				
Accuracy			78%	22544
Macro Average	82%	81%	78%	22544
Weighted Average	84%	78%	78%	22544

Table 9

Multi-Label Classification Report for AE-Encoded + SVM Classifier

	precision	***recall***	***f1-score***	***support***
Label				
Normal	71%	89%	79%	9711
DOS	90%	74%	81%	7460
Probe	70%	62%	65%	2421
R2L	50%	23%	32%	2885
U2R	5%	48%	9%	67
Other Measures				
Accuracy			73%	22544
Macro Average	57%	59%	53%	22544
Weighted Average	74%	73%	72%	22544

Precision-Recall Curves

Another useful way to look at precision and recall metrics would be visualizing it through the precision-recall curve. The curve helps us in gauging our classifier's output quality. The relationship between precision and recall metric is typically based on a tradeoff between one another; this quality is exhibited in the precision-recall curve since it plots the two metrics based on various decision boundaries. When we move along the curve the decision threshold (AKA classification threshold) decreases, the number of false positives increases but false negatives decrease. As a result, precision decreases, while recall increases (refer to formulas in the previous chapter).

Another quality metric to closely examine the precision-recall curve is computing the average precision score (AP) which is essentially the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. Mathematically it is represented as:

$$\mathrm{AP} = \sum_{n} (R_n - R_{n-1}) P_n \tag{14}$$

Where R represents recall, P representing Precision, and N representing the nth threshold. One of the important things to look out for when examining precision-recall curves is the area beneath it. AP and trapezoidal rules are ways to approximate the area beneath the curve. A higher precision-recall curve suggests a high value for both the metrics, which is considered important for a robust, valid model. Within this section, we will only compare the binary precision-recall curve for scaled, PCA, and DAE

encoded datasets which classified using SVM. The precision-recall graphs for other algorithms can be found in the appendix section for further examination.

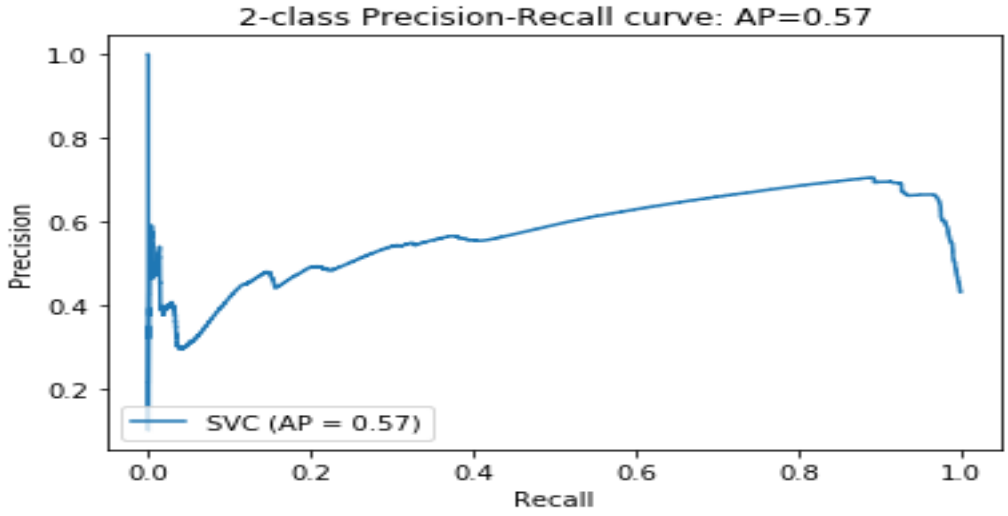

Figure 10 Standalone SVM for Binary Class Precision-Recall Curve

The graph represents the precision-recall curve for a standalone SVM operating on an RBF kernel function. The average precision score is a mere 0.57 which is not a desirable outcome for an IDS.

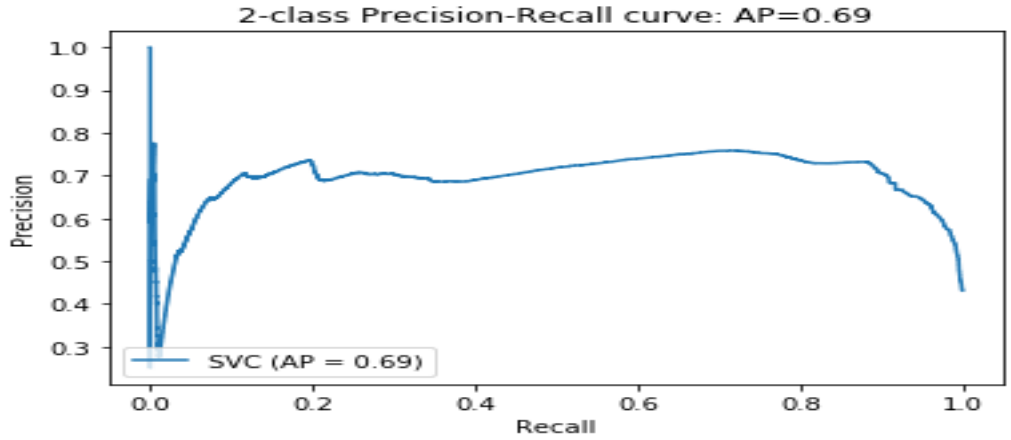

Figure 11 *PCA+SVM Binary Class Precision-Recall Curve*

The graph presents data relating to PCA encoded inputs coupled with the Support Vector Machine classifier. The shape of the curve is relatively outwards facing, suggesting an increase in the area beneath the curve compared to the data presented in the previous figure. The algorithm receives an AP score of 0.69.

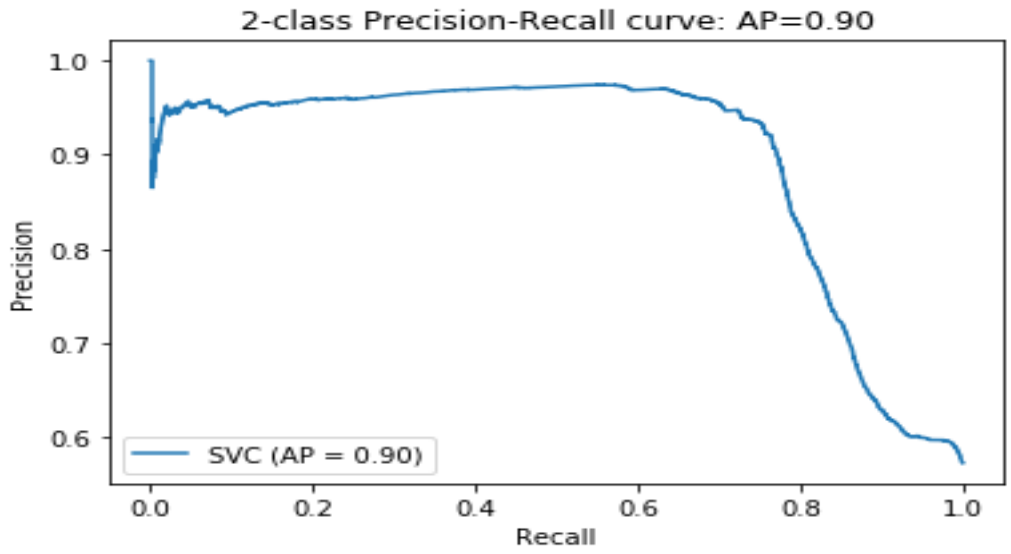

Figure 12 *AE+SVM Precision-Recall Curve (Polynomial Kernel)*

This graph clearly shows the precision-recall trade-off that was mentioned at the start of this section. We can observe as the decision boundary decreases; the recall tends to increase. We can clearly see the area beneath the curve is much greater than the two previous graphs of SVM and PCA-SVM respectively. Our proposed model receives an AP score of 0.90 which is considerably high.

We use the models in multiclass but with an extra adjustment of weight balancing. It should be noted that because the precision-recall curve is used primarily for binary labels, we will be binarizing individual curves for the five classes. Since there

is a class imbalance in our dataset, we chose to micro-average than the macro-average of all five classes which may lead to a different interpretation.

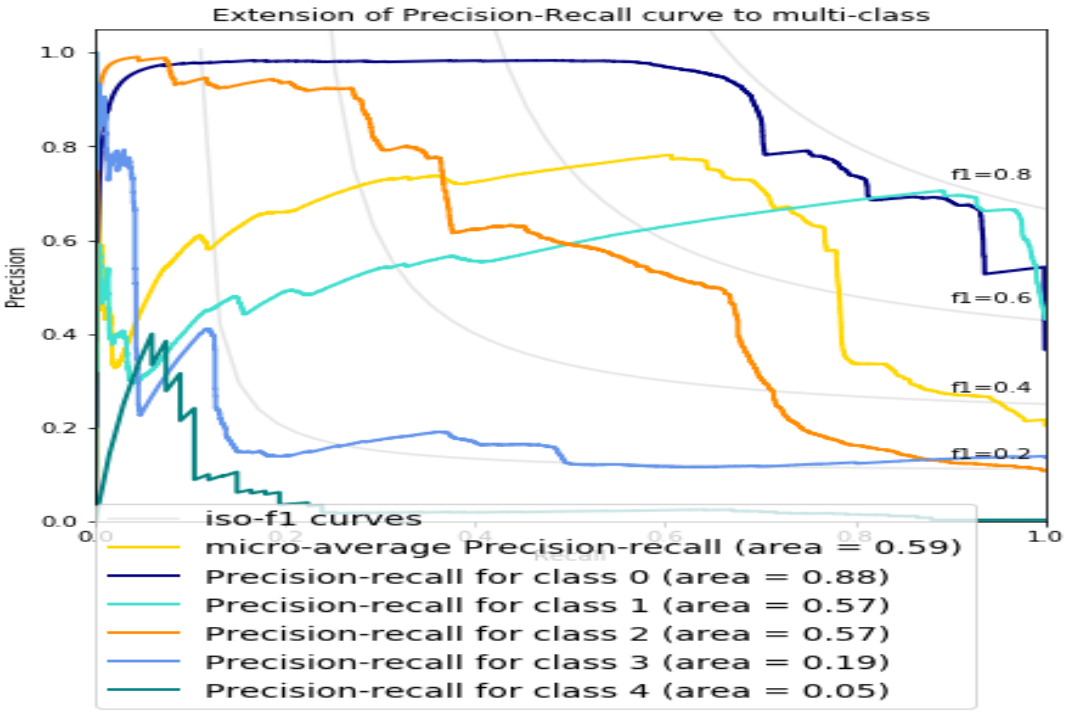

Figure 13 *Standalone SVM for MultiClass Precision-Recall Curves*

Figure 13 presents precision-recall curves based on a standalone SVM classifier for normalized inputs. An important measure in the above graph is the micro-average of precision-recall curves for all five classes by aggregating the contributions of the five classes to compute the average metric.

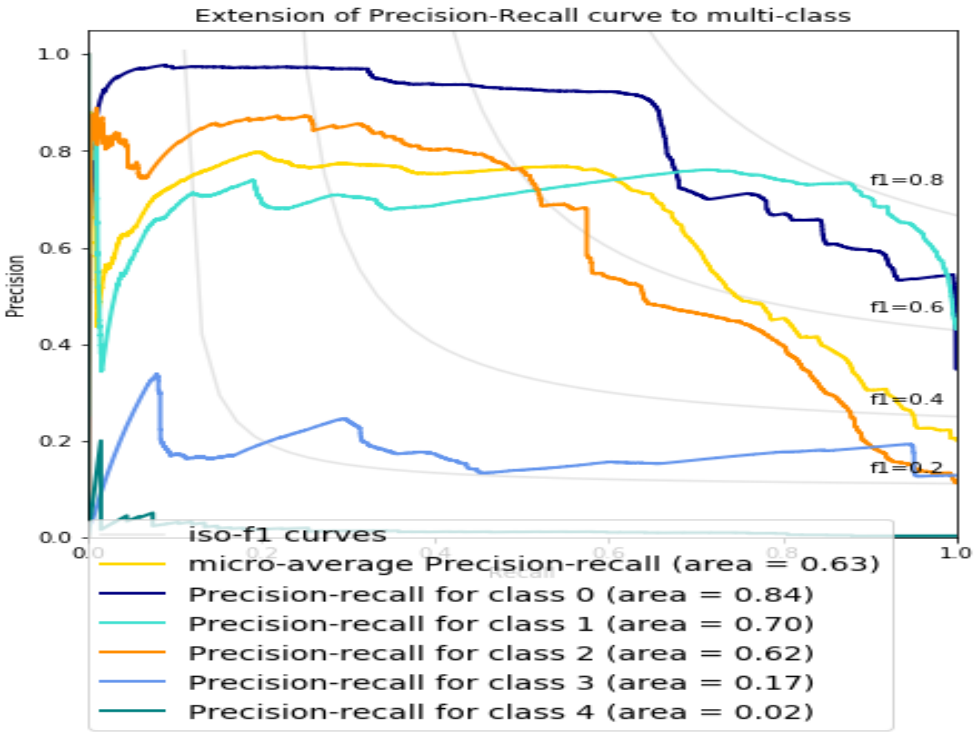

Figure 14 *PCA+SVM MultiClass Precision-Recall Curves*

PCA-encoded inputs lead to a considerable increase in the micro-average of precision-recall, specifically from 0.59 to 0.63. It should be noted that our graphs also display iso curves at different f1 values. Iso-curves are convex-shaped curves that in this case follows a combination of precision and recall values for a given f1-score. The iso-curves gives us a reliable reference point to better understand the precision-recall curve.

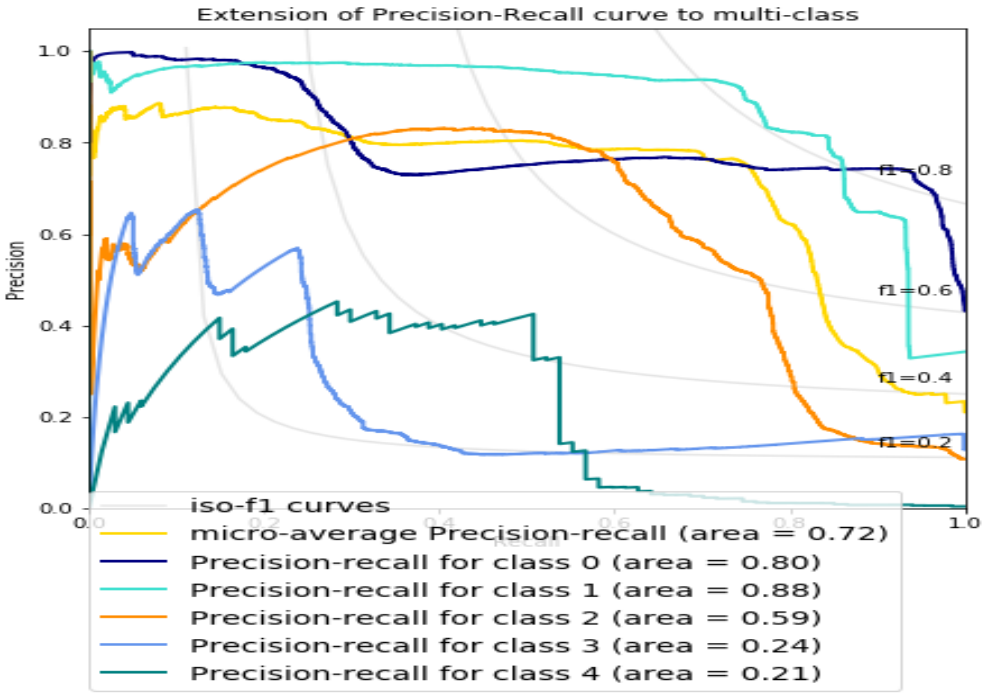

Figure 15 *AE+SVM MultiClass Precision-Recall Curves*

Figure 17 reflects data given from DAE inputs which are classified on SVM. The data indicates a much higher micro-average than the previous two models. Specifically, the area under the micro-average precision-recall is 0.72 which is 10 basis points higher than the previous model.

Performance Metrics

Train and Test Time

One of the most important features of a good model is the time and computational power that it requires to generate its predictions. We used the *time* module in python to measure the time it takes for our models to perform training and testing on the NSL-KDD dataset. The table below shows the training and testing time in seconds for the algorithms used in this paper.

Based on our data, we can conclude that L1 regularized DAE coupled with SVM is the most time effective when dealing with binary class. Whereas, in the multi-class case, we find that DAE-SVM without any regularization term takes the least time to train and test the NSL-KDD dataset which gives us a clear answer to our initial research question that asks the effectiveness of an autoencoder based dimensionality reduction tool. A standalone support vector machine appears to take the longest time in both multi and binary class scenarios.

Table 10

Train and Test Time based on Seconds

Algorithm	Training Time (sec)	Testing Time (sec)	Total Time (sec)	Class
SVM	392.70	39.68	432.38	Binary
DAE-SVM	59.13 + 86.60	2.28	148.01	Binary
DAE-SVM L1	52.55 + 90.49	2.80	145.84*[2]	Binary
DAE-SVM L2	62.22 + 89.64	2.69	154.53	Binary
SVM	1545.64	133.23	1,678.87	Multi-Class
PCA-SVM	270.16	29.83	299.99	Multi-Class
DAE-SVM	48.09 + 89.75	7.78	145.62*	Multi-Class
DAE-SVM L1	49.20 + 121.04	10.72	180.96	Multi-Class
DAE-SVM L2	48.60 + 114.15	9.51	172.26*	Multi-Class

Conclusion

The results yielded during this study have addressed all the topics posed earlier in the research question section. To recapitulate the findings of our research, we found DAE+SVM based neural network scheme being effective based on various classification

[2] * Algorithm that takes the least time to train and test on the NSL-KDD dataset.

and performance metrics. Autoencoders were much more effective at capturing useful properties of inputs which were demonstrated through t-SNE to embed higher dimension inputs on a two-dimensional plane and were compared to its linear PCA counterpart.

Specifically, in terms of training and testing time, autoencoder encoded inputs proved to be much more time-efficient in the training and testing phase of the model. In addition, our proposed neural scheme proves to be better at classification metrics like the weighted average of Recall, F-score, and Accuracy in multi-class scenario compared to standalone SVM and PCA-encoded SVM.

By focusing on metrics such as precision and recall, we were able to get a more refined perspective of our proposed neural scheme's performance by focusing on Type 1 (False Positives) and Type II (False Negatives) errors. Since we are dealing with anomaly detection, more importance should be given to Type II error since allowing an anomaly to infiltrate through our intrusion detection system has the potential to wreak havoc on our system's resources.

After rigorously examining the classification metrics, we can safely conclude the reliability and robustness of Autoencoders as a viable dimensionality reduction tool compared to PCA for anomaly detection based on NSL-KDD Dataset.

References

[1] D. E. Denning, "An Intrusion-Detection Model," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* vol. 13, no. 2, p. 222, 1987.

[2] N. Harale and B. B. Meshram, "Network Based Intrusion Detection and Prevention Systems: Attack Classification , Methodologies and Tools," *International Journal of Engineering And Science,* vol. 6, no. 5, pp. 1-12, 2016.

[3] S. NASEER, Y. SALEEM, S. KHALID, M. K. BASHIR, J. HAN, M. M. IQBAL and K. HAN, "Enhanced Network Anomaly Detection Based on Deep Neural Networks," *IEEE Access,* vol. 6, pp. 48231-48246, 2018.

[4] S. Mukkamala, G. Janoski and A. Sung, "Intrusion Detection Using Neural Networks and Support Vector Machines," *Computers in Biology and Medicine,* vol. 96, pp. 116-127, 2018.

[5] Y. Liao and V. R. Vemuri, "Use of K-nearest neighbor classifier for intrusion detection," *Computers & Security,* vol. 21, no. 5, pp. 439-448, 2002.

[6] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[7] "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," *Proceedings - DARPA Information Survivability Conference and Exposition, DISCEX ,* vol. 2, no. February, pp. 12-26, 2000.

[8] M. Tavallaee, E. Bagheri, W. Lu and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA ,* pp. 1-6, 2009.

[9] "KDD-CUP-99 Task Description," [Online]. Available: https://kdd.ics.uci.edu/databases/kddcup99/task.html.

[10] M. K. Asif, T. A. Khan, T. A. Taj, U. Naeem and S. Yakoob, "Network Intrusion Detection and its Strategic Importance," *IEEE Business Engineering and Industrial Applications Colloquium (,* pp. 140-144, 2013.

[11] N. Shone, T. N. Ngoc and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE,* vol. 2, no. 1, p. 45, 2018.

[12] M. Minsky and S. A. Papert, Perceptrons, MA, USA: MIT Press, 1969.

[13] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA: MIT Press, 2016.

[14] D. Liu, "A Practical Guide to ReLU," 30 November 2017. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. [Accessed 12 December 2019].

[15] P. Ramachandran, B. Zoph and Q. V. Le, "SEARCHING FOR ACTIVATION FUNCTIONS," *6th International Conference on Learning Representations, ICLR - Workshop Track Proceedings,* pp. 1-13, 2018.

[16] G. Klambauer, T. Unterthiner and A. Mayr, Self-Normalizing Neural Networks, Linz: Advances in Neural Information Processing Systems (NIPS) , 2017.

[17] Y. Wang, H. Yao and S. Zhao , "Auto-encoder based dimensionality reduction," *Neurocomputing,* vol. 184, no. November, pp. 232-242, 2015.

[18] L. v. d. Maaten, E. Postma and J. . v. d. Herik, "Dimensionality Reduction: A Comparative," *Journal of Machine Learning Research,* vol. 10, pp. 1-41, 2009.

[19] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P.-A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal ofMachine Learning Research,* vol. 11, pp. 3371-3408, 2010.

[20] J. Masci, U. Meier, D. Ciresan and J. Schmidhuber, Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction, Berlin, Heidelberg: Springer, 2011.

[21] S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," *Proceedings of the 28th International Conference on Machine Learning,,* no. 1, pp. 833-840, 2011.

[22] Q. Xu, C. Zhang, L. Zhang and Y. Song, "The Learning Effect of Different Hidden Layers Stacked Autoencoder," *International Conference on Intelligent Human-Machine Systems and Cybernetics,* 2016.

[23] Y. A. Ng, "Feature selection, L1 vs. L2 regularization,," in *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[24] A. Shenfield, D. Day and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express,* vol. 4, no. 2, pp. 95-99, 2018.

[25] M. Al-Qatf, Y. Lasheng, M. Al-Habib and . K. Al-Sabahi, "Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection," *IEEE Access,* vol. 6, 2018.

[26] M. Yousefi-Aza, V. Varadharajan, L. Hamey and U. Tupakula, "Autoencoder-based Feature Learning for Cyber Security Applications," *nternational Joint Conference on Neural Networks,* 2017.

[27] B. Lee, S. Amaresh, C. Green and D. Engels, "Comparative Study of Deep Learning Models for Network Intrusion Detection," *SMU Data Science Review,* vol. 1, no. 8, 2018.

[28] S. K. Patro and K. K. Sahu, "Normalization: A Preprocessing Stage," *CoRR,* pp. 20-22, 2015.

[29] N. Seliya, T. M. Khoshgoftaar and J. V. Hulse, "A Study on the Relationships of Classifier Performance Metrics Naeem," *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI,* pp. 59-66, 2009.

[30] "Classification: Precision and Recall," Google, [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. [Accessed 9 12 2019].

[31] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research ,* vol. 9, pp. 2579-2605 , 2008.

[32] A. Dertat, "Applied Deep Learning - Part 3: Autoencoders," 3 October 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798. [Accessed 3 12 2019].

[33] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science,* vol. 313, no. 5786, p. 504–507, 2006.

[34] H. Chauhan, V. Kumar, S. Pundir and E. S. Pilli, "A Comparative Study of Classification Techniques for Intrusion Detection," *Proceedings - 2013 International Symposium on Computational and Business Intelligence,* pp. 40-43, 2013.

Appendix

The appendix includes supplementary information regarding the study. The table in the appendix section contains a list of features that is present in the KDD 99 dataset. In the second exhibit, we see the attack types of KDD 99 further segmented to get a better picture of the dataset in question.

List of Features in KDD 99 Dataset

Feature Type	No.	Feature Name	Data Type
Basic Features	1	Duration	Continuous
	2	Protocol_type	Symbolic
	3	Service	Symbolic
	4	Flag	Symbolic
	5	Src_bytes	Continuous
	6	Dst_bytes	Continuous
	7	Land	Symbolic
	8	Wrong_fragment	Continuous
	9	Urgent	Continuous
Content Features	10	Hot	Continuous
	11	Num_failed_logins	Continuous
	12	Logged in	Symbolic
	13	Num_compromised	Continuous

	14	Root_shell	Continuous
	15	Su_attempted	Continuous
	16	Num_root	Continuous
	17	Num_file_creations	Continuous
	18	Num_shells	Continuous
	19	Num_access_files	Continuous
	20	Num_outbound_cmds	Continuous
	21	Is_host_login	Symbolic
	22	Is_guest_login	Symbolic
Traffic Features	23	Count	Continuous
	24	Srv_count	Continuous
	25	Serror_rate	Continuous
	26	Srv_serror_rate	Continuous
	27	Rerror_rate	Continuous
	28	Srv_rerror_rate	Continuous
	29	Same_srv_rate	Continuous
	30	Diff_srv_rate	Continuous
	31	Srv_diff_host_rate	Continuous
	32	Dst_host_count	Continuous
	33	Dst_host_srv_count	Continuous

	34	Dst_host_same_srv_rate	Continuous
	35	Dst_host_diff_srv_rate	Continuous
	36	Dst_host_same_src_port_rate	Continuous
	37	Dst_host_same_src_host_rate	Continuous
	38	Dst_host_serror_rate	Continuous
	39	Dst_host_srv_serror_rate	Continuous
	40	Dst_host_rerror_rate	Continuous
	41	Dst_host_srv_rerror_rate	Continuous

Attack Types in KDDCUP99 dataset

Denial of Service (DoS)	User to Root (U2R)	Remote to Local (R2L)	Probing (Probe)
Back	Buffer Overflow	FTP write	IPSweep
Land	Load module	Guess Password	NMAP
Neptune	Perl	IMAP	Port Sweep
Ping of Death	Rootkit	MultiHop	Satan
Smurf		Phf	
Teardrop		SPY	
		Warezclient	
		WarezMaster	