The School of Mathematics

Exploring the use of Autoencoders and Variational Autoencoders for Anomaly Detection in Cyber Security

by

Kartik

Dissertation Presented for the Degree of
MSc in Statistics with Data Science

June 2023

Supervised by
Dr Miguel de Carvalho, and Dr Amanda Lenzi

Executive Summary

In cybersecurity, anomaly detection involves recognising patterns and behaviours that significantly deviates from the normal, in order to identify potential threats or malicious activity. This report provides a comprehensive analysis of the utilisation of Autoencoders(AEs) and Variational Autoencoders(VAEs) for anomaly detection in the field of cybersecurity. This study focuses on the application of these models to two different datasets- UNSW-NB15 (developed by University of South Wales) and NSL-KDD(developed by University of New Brunswick). Only the first part is utilised in the case of UNSW-NB15 dataset which have 14,278 anomalies after removing redundant observations. And the NSL-KDD dataset contains 70,940 anomalous observations with various forms of cyberattacks.

AEs and VAEs uses an unsupervised approach which enables them to learn patterns and representations directly from the unlabeled data. Specifically, the patterns of the normal observations are learned. And any deviations from these learned patterns will be identified as anomaly.

The findings of the analysis demonstrate the exceptional performance of AE and VAE models compared to other traditional anomaly detection approaches. Specifically, AE exhibited better performance in correctly identifying true anomalies in the UNSW-NB15 dataset, while VAE showcased superior capabilities in the NSL-KDD dataset. But both of the models have had some challenges in accurately identifying the patterns of certain type of attacks.

Word Count: 4370 words(Only includes the main text and executive summary)

Acknowledgments

I am grateful for the valuable advice and guidance given by supervisors of the project, Dr Miguel de Carvalho, Dr Amanda Lenzi. I especially want to thank PhD student helper Jame Chok for providing valuable insights. Moreover, I want to thank the researcher and organisations that make their data publicly available for the research purposes.

I'd also like to thank George Deskas, Stephen O'Sullivan, and Dimitrios Ntakoulas for their crucial advice and support throughout this project. Their knowledge, and suggestions were critical to the successful preparation of this study.

Word Count: 4370 words(Only includes the main text and executive summary)

University of Edinburgh – Own Work Declaration

This sheet must be filled in, signed and dated - your work will not be marked unless this is done.

Name: Kartik
Matriculation Number: s2407270
Title of work: Exploring the use of Autoencoders and Variational Autoencoders for Anomaly Detection in Cyber Security

I confirm that all this work is my own except where indicated, and that I have:

- Clearly referenced/listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Not sought or used the help of any external professional academic agencies for the work
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Complied with any other plagiarism criteria specified in the Course handbook

I understand that any false claim for this work will be penalised in accordance with the University regulations (`https://teaching.maths.ed.ac.uk/main/msc-students/msc-programmes/statistics/data-science/assessment/academic-misconduct`).

Signature: Kartik

Date 30/06/2023

Word Count: 4370 words(Only includes the main text and executive summary)

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

As the world moves towards digitalization unlocking countless opportunities and developments, it is crucial to recognise and handle the inherent risks along with its advantages. The ever-changing world of cyberattacks is one of the most serious risks to our progression towards a digital future. Digital criminals use various methods to create an anomaly which deviates from the normal data, creating a possibility that it was generated using an unusual method. These cyberattacks try to exploit vulnerabilities of the system which then compromises data integrity and violate our privacy. Cyberattacks can originate from both state-sponsored entities and individuals. In an interconnected world where information moves freely across networks and systems, cyberattacks have become more complex and frequent. Detecting abnormal behaviour is critical for maintaining security and confidence in the digital systems.

1.2 Objective

This research aims to compare the performance of autoencoders(AE) and variational autoencoders(VAE) in the context of detecting anomalies in the field of cybersecurity. Additionally, the performance of these techniques is compared with the other traditional anomaly detection approaches such as the isolation forest, one-class SVM(Support vector machine) and K-nearest neighbours(KNN) clustering algorithms.

2 Literature review

An anomaly can be considered as an outlier that is significantly different from the rest of the data. Three groups of anomaly detection techniques can be categorised as- statistical, proximity-based, and deviation-based [2].

Statistical approaches presuppose that data has a particular probability distribution, such as gaussian. These techniques recognises a data point as an anomaly if its likelihood of being generated from a probabilistic distribution is less than a specific threshold. These statistically based methods have several advantages, including that they produce results that are easy to comprehend with certain assumptions. But the real-world data may not always match these assumptions. Furthermore, if the relationships between the variables are not linear, these methods may have difficulty deducing the complex patterns. Z-score based anomaly detection is an example of statistical-based methods.

Proximity-based methods typically attempt to create clusters similar data based on the distance measures. It is predicated that anomalous data in a feature space will be spread out from the majority of clusters. That data is categorised as an anomaly if its separation from the cluster exceeds a specific threshold. The limitation is the requirement of high computation resources in handling high-dimensional data. The K nearest neighbours-clustering algorithm is one example of such a technique.

Deviation-based approaches detect the anomalies by measuring the difference between the original and reconstructed data. The more the deviation between the two, higher the likelihood of an anomaly being present. Examples of these techniques include AEs.

3 Data

The performance of several approaches are evaluated using data developed specifically for network intrusion detection research. The University of South Wales in Australia generated the UNSW-NB15 dataset[9], while a group of researchers from the University of New Brunswick in Canada created the NSL-KDD dataset[12]. The UNSW-NB15 dataset consists of network traffic that simulates real-world network attacks in a controlled environment, whereas the KDD dataset was collected from a simulated network traffic.

3.1 UNSW-NB15 dataset

For our analysis, the first part of the UNSW-NB15 dataset is used, as specifically requested by the client. After removing the redundant and unwanted observations, significant disparity between normal and anomalous observations is observed, with only 14,278 anomalous cases out of 640,726. Dataset consists of observations related to these categories- Normal, Exploits, Fuzzers, Generic, Reconnaissance, DoS(Denial of service), traffic analysis attacks, backdoor, shellcode and worms. The first class is normal and latter are anomalous categories. Detailed number of observation for each category is given in table 1.

Table 1: UNSW-NB15 dataset

Category	**No. of observations**
Normal	626448
Exploits	4042
Fuzzers	3991
Generic	2833
Reconnaissance	1740
DoS(Denial of service)	825
Traffic analysis attack	301
Backdoor	299
Shellcode	223
Worms	24
Total	640,726

Out of total 49 features, only eleven features are categorical. And each categorical feature has at least five unique categories along with binary features, and some of them have 65,000 unique values. Due to limited computation resources, it is not practical to convert all of them at once to numeric values using one hot encoding. Instead frequency encoding is used which first encodes different categories with label encoding and then encoding those categories with their respective counts within the dataset. For numeric variables, min-max scaling is used to transform the data in these variables between the range of 0 and 1. For a detailed discussion on pre-processing techniques, please refer to Appendix A.

3.2 NSL-KDD dataset

The NSL-KDD dataset is well-balanced, with a total of 70,940 anomalous cases out of 147,907 observations containing variety of attacks which come under five main categories- Normal, DoS, Probe, R2L(Remote to local) and U2R(User to root)7. The number of observations for each category is shown in table 2.

Table 2: NSL KDD dataset

Category	**No. of observations**
Normal	77120
DoS	52987
Probe	13954
U2R	108
R2L	3738
Total	147,907

Out of total 45 features, only three are categorical features. One hot encoding technique is employed to transform these to numeric values. For numeric features, min-max scaling is used to transform data in these variables between the range of 0 and 1.

According to the labels, these datasets are split into normal and anomalous observation. The training set comprises of 70% normal observations, while validation set also consists of 15% normal. Furthermore, the remaining 15% normal observations are in testing data with anomalous observations. We can refer to the appendix A for an in-depth discussion of the pre-processing and manipulation of both datasets.

4 Methodology

4.1 Autoencoder

Autoencoder(AE) is a type of neural network that intends to reconstruct the input through an unsupervised learning process. There are numerous autoencoder architectures, but most of them have three common components- an encoder, code and a decoder that work simultaneously to reconstruct the data.

The **encoder** transforms the input using a non-linear mapping into a lower-dimensional latent space representation. Because this latent space layer contains fewer neurons than the rest of the network, the network is forced to learn a compressed representation of the data. Its primary goal is to identify important patterns in data and produce a **code**.

A **decoder** will take this compressed form as input and attempt to recover the original data by mapping it back to original dimensions.

4.1.1 Architecture

The model architecture that we are adopting for our research is a vanilla AE with one encoding layer, one decoding layer, and a latent representation layer known as the code or bottleneck layer, as shown in figure 1.

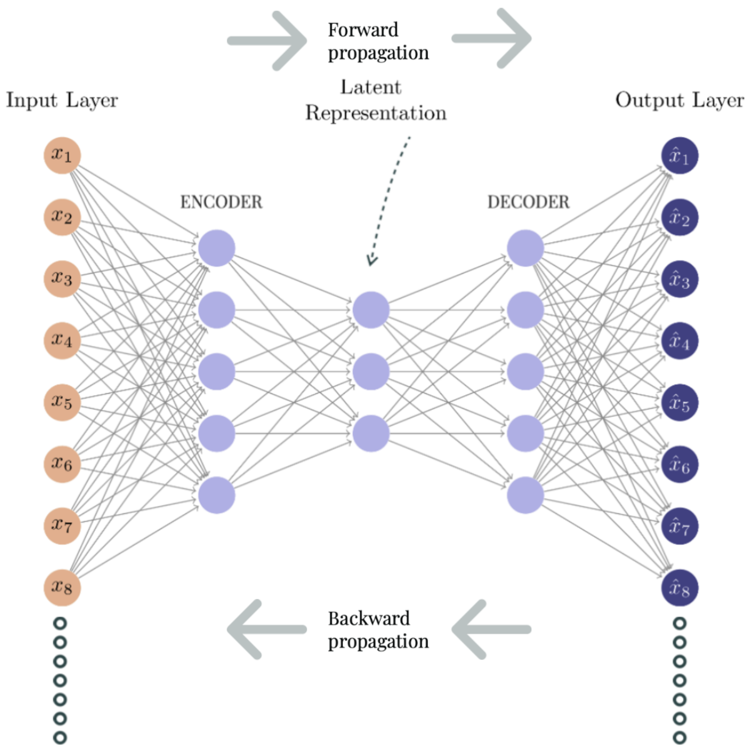


Figure 1: Vanilla Autoencoder

Vanilla autoencoders are designed to be symmetric, which implies that the architecture of the decoder mirrors that of the encoder. Because of this symmetry, AE can learn efficiently.

Let's say that the input data x has dimensions $n \times m$, where n is the number of input datapoints and m is the size of the feature space. The **encoder** will attempt to encode the input:

$$z = \sigma(W_{(i \times m)} \cdot x^{\top} + b_{(i,)}) \tag{4.1}$$

where $W_{i \times m}$ represents the weight matrix which connects the encoder layer with m neurons to the next layer, with i neurons and b represents the bias vector of shape $(i,)$ of the neural network respectively and σ is the non-linear activation function.

Leaky relu[8] activation function is used to introduce non-linearity σ. Leaky relu(eq. 4.2) is chosen to avoid dead neurons in the network. As a result, better information flow and gradient propagation through the network.

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \beta x, & \text{otherwise} \end{cases} \tag{4.2}$$

where β in equation 4.2 is a hyper parameter which represents the slope for negative inputs typically set to small positive value such as 0.03.

Decoder will then try to reconstruct the input data using the following equation:

$$\hat{x} = \sigma(W_{(m \times j)} \cdot z^T + b_{(m,)}) \tag{4.3}$$

where $W_{m \times j}$ in eq. 4.3 represents the weight matrix connecting the previous layer(with j neurons) representation to the reconstructed output, and b corresponds to the bias vector. This results in the reconstructed output having the same dimensions as the input. This reconstruct output is mapped to a particular range, i.e., between 0 and 1, in the output layer using a sigmoid activation function.

$$\hat{\sigma}(\hat{x}) = \frac{1}{1 + e^{-\hat{x}}} \tag{4.4}$$

This is **forward propagation**, where the weights and biases of both the encoder and decoder are utilised to reconstruct the data. During the **backward propagation**, the model optimises its weights and biases based on suitable gradient optimiser to minimize the reconstruction loss. This procedure of passing through forward and backward propagation, adjusting the weights and biases of the network based on gradient until the convergence is referred to as training phase. The training algorithm for autoencoder is shown in algorithm 1.

Algorithm 1 Vanilla Autoencoder Training

```
Require: Input data: $X$, Encoder network: $E(\cdot)$, Decoder network: $D(\cdot)$, Loss function: $L(\cdot)$, Learning rate: $\alpha$
Ensure: Trained autoencoder: $AE(\cdot)$
 1: procedure TrainAutoencoder($X, E(\cdot), D(\cdot), L(\cdot), \alpha$)
 2:     Initialize the parameters of $E(\cdot)$ and $D(\cdot)$
 3:     while not converged do
 4:         Randomly sample a batch of training examples: $X_{\text{batch}}$
 5:         Compute the encoded representation: $Z_{\text{batch}} = \text{Encode}(X_{\text{batch}}, W_E, b_E)$
 6:         Reconstruct the input: $\hat{X}_{\text{batch}} = \text{Decode}(Z_{\text{batch}}, W_D, b_D)$
 7:         Compute the reconstruction loss: $L_{\text{batch}} = L(X_{\text{batch}}, \hat{X}_{\text{batch}})$
 8:         Compute the gradients: $\nabla_{\theta_E} L_{\text{batch}}$, $\nabla_{\theta_D} L_{\text{batch}}$
 9:         Update the parameters of $E(\cdot)$ and $D(\cdot)$ using gradient descent:
10:             $W \leftarrow W - \alpha \nabla_W L_{\text{batch}}$
11:             $b \leftarrow b - \alpha \nabla_b L_{\text{batch}}$
12:     return Trained autoencoder: $AE(\cdot) = D(E(\cdot))$
```

4.1.2 Anomaly detection using autoencoder

The reconstructed observations are not an exact replica of the data, but rather the goal is to reconstruct the input data as accurately as possible; based on which the reconstruction error is determined using a suitable loss function. It serves as the anomaly detection criterion. The mean absolute error(MAE) loss function is used to determine the reconstruction error.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |x_i - \hat{x}_i| \tag{4.5}$$

Since the autoencoder is trained solely on the normal observations and not abnormal ones, as a result the reconstruction error for the anomalous observations is large. This is because the autoencoder did not meet any anomalous data during the training phase, this prevents the autoencoder from successfully reconstructing those observations. Algorithm 2 shows the steps taken to detect anomalies using an AE.

Algorithm 2 Anomaly Detection using Autoencoder

```
Require: Dataset $X$, Trained AE $AE(\cdot) = D(E(\cdot))$, threshold $\gamma$
Ensure: Anomaly labels for instances in $X$
1: for all instance $x$ in $X$ do
2:     Reconstruction $x$ using AE: $\hat{x} = AE(x)$
3:     Calculate reconstruction error: $Loss = \frac{1}{N}\sum_{i=1}^{N} |x_i - \hat{x}_i|$
4:     if $Loss > \gamma$ then
5:         Label $x$ as an anomaly
6:     else
7:         Label $x$ as normal
```

4.2 Variational Autoencoders

Variational autoencoder(VAE)[6] is a type of autoencoder that incorporates probabilistic modelling approach to reconstruct the data. Unlike traditional autoencoders, variational autoencoders learn the parameters of a probability distribution modeling our data rather than some arbitrary function through a neural network[1]. This probabilistic distribution enables the generation of new data input data by sampling from it, hence also considered as a generative model.

4.2.1 Architecture

VAEs consists of three main components- encoder, code(latent-space), and a decoder.

The **encoder** network attempts to approximate the posterior distribution of the latent variables given the input data.

$$q_\phi(z|x) = q(z; f(x, \phi)) \tag{4.6}$$

where $q_\phi(z|x)$ in equation 4.6 denotes the conditional distribution of the latent variables z given the input x, parameterised by the encoder network with parameters ϕ.

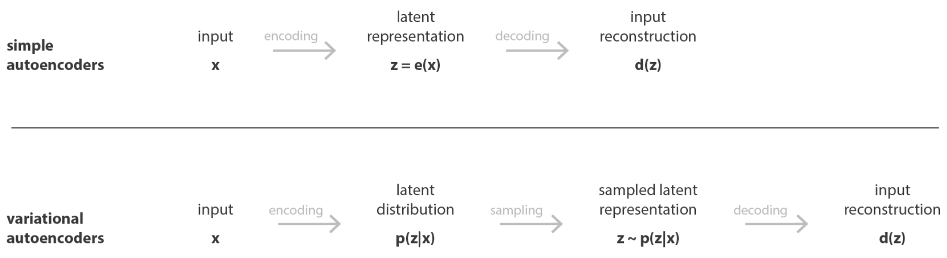


Figure 2: Difference between AE and VAE architecture([11])

Unlike autoencoder, instead of directly encoding the input data into fixed-dimensional space, VAE models the latent representation as a probabilistic distribution. The type of distribution depends on the nature of the data. Because our data is continuous, a multivariate gaussian distribution is used as prior for the latent-space.

Hence, the variational distribution $q(z; f(x, \phi))$ can be specified as:

$$q(z; f(x, \phi)) = \mathcal{N}(z; \mu, \Sigma) \tag{4.7}$$

where μ and Σ are the mean vector and covariance matrix of the gaussian respectively, which are determined by the encoder network. Figure 3 shows the detailed workflow of the VAE model.

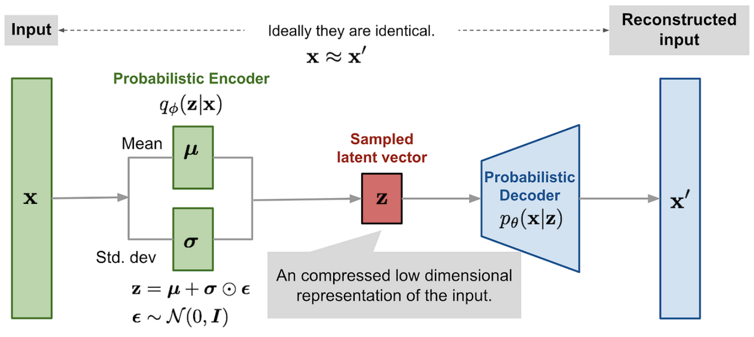

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{x}$
Sampled latent vector
$\mathbf{z}$
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
An compressed low dimensional representation of the input.

Figure 3: VAE model([4])

The value of the latent variable z can be obtained by sampling from the variational distribution.

$$z \sim q(z; f(x, \phi)) \tag{4.8}$$

Sampling techniques such as Monte Carlo methods can be used but are susceptible to high variance[3]. So, instead a reparameterisation trick is used, in which a random variable ϵ from standard normal is used to transform.

$$z \sim q_\phi(z|x) \Leftrightarrow z = t_\phi(\epsilon, x), \quad \epsilon \sim N(0, 1) \tag{4.9}$$

where $t_\phi(\epsilon, x)$ is the reparameterisation function. Due to the gaussian assumption, sampling from the latent-space can be done as follows:

$$z = \mu + \sigma * \epsilon, \quad \epsilon \sim N(0, 1) \tag{4.10}$$

where μ and σ are the mean and variance of the gaussian distribution(prior).

Decoder attempts to reconstruct the input data based on the samples from the latent-space by learning the $p_\theta(x|z)$(θ in the parametric-form represents the parameters of the decoder network). Decoder maps these samples back to original data space using a series of non-linear transformation(such as leaky relu).

The main objective of the VAE is to optimise the network parameters ϕ and θ to minimise the Kullback-Leibler(KL) divergence. This can be achieved by maximising the lower bound of the marginal likelihood(also known as evidence lower bound(ELBO)), because the marginal likelihood of the distribution is intractable. The marginal log-likelihood of the input data is

$$\log p(x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z; \theta)) \tag{4.11}$$

where the $\log p(x)$ is the marginal log-likelihood of the input data.

The ELBO $L_x(q)$ is the lower bound on the marginal log-likelihood which can be written as:

$$\log p(x) \geq L_x(q) \tag{4.12}$$

Since the KL divergence is always ≥ 0 (If the distributions are the same, the value is 0; otherwise, the value is positive). Hence, from equation 4.11 and 4.12 ELBO can be defined as:

$$L_x(q) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z; \theta)) \tag{4.13}$$

where the first term $\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)]$ represents the expected log-likelihood of reconstructing the input data x given z(**reconstruction error**), and the second term $\mathrm{KL}(q_\phi(z|x)||p(z; \theta))$ acts as the **regularisation** which measures the difference between approximate posterior and prior distribution. And encourages the latent space to align the posterior with the desired prior distribution.

From equation 4.12 and 4.13, it can be inferred that by maximizing the ELBO, VAE indirectly maximises the log-likelihood of the data. Hence, the optimisation problem for the VAE network can be defined as:

$$\arg\max_{\phi,\theta} L_x(q) = \arg\max_{\phi,\theta} \left(\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z; \theta)) \right) \tag{4.14}$$

During the training the decoder is simultaneously optimised alongside the encoder using gradient-based methods. Hence, the VAE learn to encode the input data into meaningful latent-representation and generate reconstructions using the decoder. The training algorithm for VAE is shown in algorithm 3.

Algorithm 3 Training Algorithm for VAE

```
Require: Dataset X, Initialize parameters $\phi$ and $\theta$, Initialize learning rate $\alpha$, Set number of training iterations $T$
Ensure: Trained VAE
 1: for $t = 1$ to $T$ do
 2:     Sample a batch of input data $\{x^{(i)}\}$ from the training dataset X.
 3:     //Encoder
 4:     Compute the mean and log-variance of the latent variables: $\mu, \log \sigma^2 = \text{Encoder}(x^{(i)}, \phi)$
 5:     Sample a latent variable $z^{(i)}$ from the Gaussian with parameters $\mu, \sigma^2$
 6:     //Decoder
 7:     Reconstruct input data: $\hat{x}^{(i)} = \text{Decoder}(z^{(i)}, \theta)$
 8:     //Loss Computation
 9:     Compute the reconstruction loss: $L_{\text{recon}} = -\frac{1}{N} \sum_{i=1}^{N} \log p(x_i|\hat{x}_i)$
10:     Compute Kullback-Leibler(KL) divergence loss: $L_{KL} = \frac{1}{2N} \sum_{i=1}^{N} (\sigma_i^2 + \mu_i^2 - \log(\sigma_i^2) - 1)$
11:     Compute the total loss: $Loss = L_{\text{recon}} + L_{\text{KL}}$
12:     // Back-propagation and Parameter Updates
13:     Update encoder and decoder network parameters: $\phi \leftarrow \phi - \alpha \cdot \nabla_\phi L$ and $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta L$
```

4.2.2 Anomaly detection using VAEs

VAE based anomaly detection follows a similar approach as an autoencoder. During the training phase, the input data is encoded into a lower-dimensional latent space, which follows a probabilistic distribution. In order to reconstruct the input data, decoder decodes the samples drawn from this probabilistic distribution. Hence the loss function for training is different than AEs as described in equation 4.13.

But during the testing phase, only the difference between the original and reconstructed data is considered, hence using only MSE(mean square error) as our loss function.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{4.15}$$

The reason was based on that the training procedure already allows the latent space to capture a meaningful representation of the underlying data distribution. As a result, inclusion of KL loss in the testing phase was unnecessary.

Based on this reconstruction error, a suitable threshold(γ) is defined to label the anomalies. The Anomaly detection algorithm using VAEs is defined in algorithm 4.

Algorithm 4 Anomaly Detection using Variational Autoencoder

```
Require: Test Dataset $X$, Trained VAE $VAE(\cdot) = D(E(\cdot))$, threshold $\gamma$
Ensure: Anomaly labels for instances in $X$
 1: for all instance $x$ in $X$ do
 2:     Encode $x$ using VAE: $z = E(x)$
 3:     Reconstruct $x$ using VAE: $\hat{x} = D(z)$
 4:     Calculate reconstruction loss: $Loss = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2$
 5:     if $Loss > \gamma$ then
 6:         Label $x$ as an anomaly
 7:     else
 8:         Label $x$ as normal
```

5 Experimental Results and Discussions

5.1 Model Fitting

Autoencoder and VAE were created in *python* using *keras* package from TensorFlow library, an open-source machine learning framework. For the autoencoder and VAE, the encoder and decoder were both single layer with 36 dimensions and the latent-space was 8 dimensional. The parameters of the autoencoder and VAEs neural network were tuned via back-propagation using the Adam optimiser[5], which utilise adaptive learning in order to efficiently update the model parameters during training.

K-fold cross-validation was used to evaluate model performance and generalisation by partitioning the data(training and validation set only) into k subsets, training the model on k-1 subsets, and assessing it on the remaining subset, and repeating this procedure K times. Also, Early stopping technique was employed in order to regularise the autoencoder and prevent overfitting based on held-out validation set. VAEs are already regularised, hence there was no need to regularise. The maximum number of training iterations was set to 22 for both the autoencoder and VAE models, and the batch size was 64.

5.2 Performance evaluation

To evaluate the effectiveness of anomaly detection algorithms, we used precision(specificity) and recall(sensitivity), with a particular emphasis on recall metric. Recall also knows as true positive rate, measures the proportion of actual anomalies that are correctly identified. F1-score metric is also used which provides a balanced evaluation that considers both precision and recall simultaneously.

Area under the receiver operating characteristic(AUC-ROC) is also calculated under various threshold values, which represents the trade-off between true positive rate against the false positive rate. This AUC-ROC metric is also used as a metric; higher values indicate better performance.

5.3 Interpreting Predictions

In order to explain the predictions of our models, two explainable techniques are employed- LIME(Local Interpretable Model-agnostic Explanations)[10] and SHAP(Shapley Additive Explanations)[7].

LIME is used to generate local explanation for the best model on the test set. LIME focuses on individual predictions generated by a black-box model. It creates a explainer(such as linear regression) around a local instance, and observe the change in predictions by perturbing its features. It also allows us to understand the impact of different features on that particular instance.

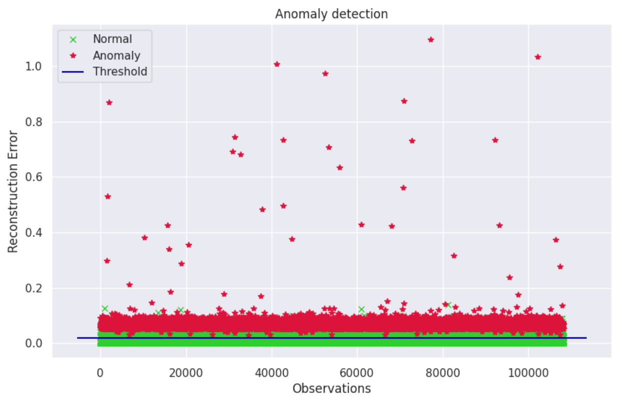
SHAP is used to generate a global explanation around that prediction. It is based on the concept of Shapely values which are derived from the game theory. Shapley values quantifies the contribution of each feature by considering all the feature combinations and it also calculates the average impact of those features on the predictions. SHAP provides a more comprehensive approach for understanding the impact of each feature on the predictions.

LIME is used to test prediction for one instance of anomalous observation, while simultaneously incorporating the global impact using SHAP.

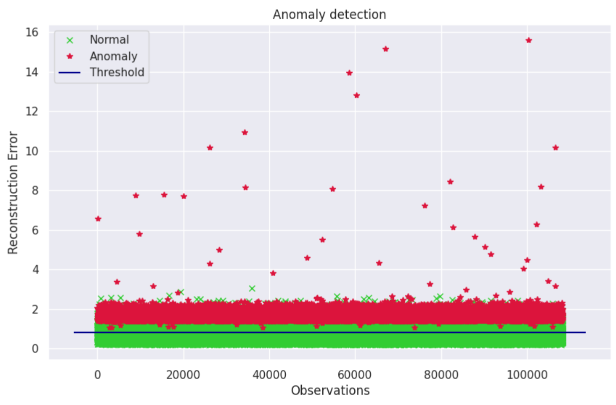
5.4 Results

5.4.1 Reconstruction of UNSW-NB15 dataset

Figure 4, shows the reconstruction error of the observations present in the test set; where the observations greater than threshold are considered as anomaly. From figure 4a and 4b, both models are successful in detecting most of the anomalies. It can also be seen that the majority of normal cases have reconstruction errors near to zero, and the error for the anomalous cases is greater than zero. It implies that both models have efficiently learned the underlying patterns present in the normal observations, enabling them to identify the anomalies using the reconstruction error.



(a) Anomaly detection using AE(γ=0.0383)



(b) Anomaly detection using VAE(γ=1.163)

Figure 4: Anomaly detection comparison on UNSW-NB15 dataset

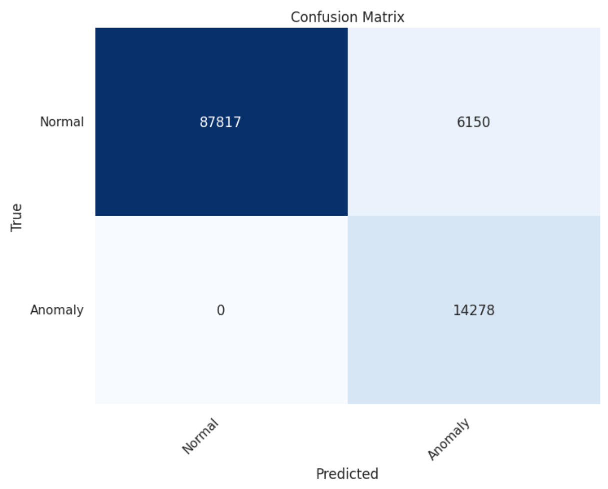
Table 3 shows the comparison of the performance evaluation metrics of AE and VAEs. In the table, highlighted rows represent the optimal threshold values for the respective models. The threshold tuning is done on the training reconstruction error as well as held-out validation set error.

Table 3: AE and VAE Model Comparison(UNSW-NB15 dataset)

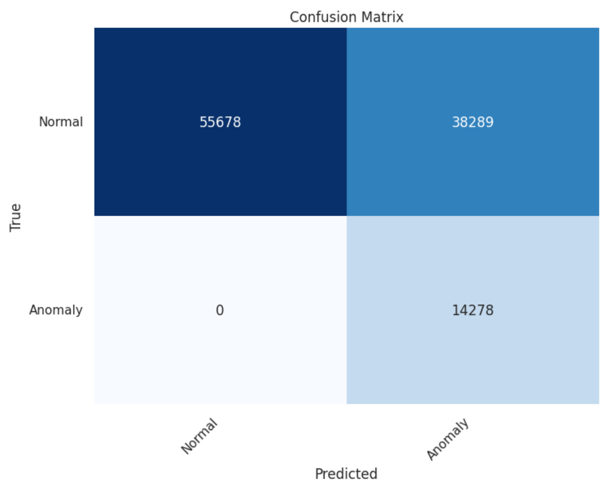
Model	Threshold (γ)	Performance Metrics			
		F1-Score	Precision	Recall	AUC-ROC
AE	0.0389	0.92	0.86	0.99	0.98
AE	0.0383	0.92	0.85	0.99	0.98
AE	0.0180	0.69	0.53	1.0	0.93
VAE	1.162	0.64	0.47	0.99	0.91
VAE	1.163	0.65	0.48	0.99	0.92
VAE	0.802	0.42	0.27	1.0	0.79

Based on the performance metrics in the table 3, the AE model appears to perform better than the VAE model. AE model consistently achieves higher score in all categories across different threshold values. VAEs are also successful in detecting most anomalies accurately based on recall scores, but fall behind in other performance metrics. In order to improve their performance, one may potentially try to increase the number of layers inside the encoder and decoder to capture more complex patterns. Also from table 3, it can be seen that both models are very sensitive to the choice of the threshold.

The confusion matrix for both of the models(relative to optimal thresholds) can be visualised in figure 5. This is to evaluate their performance in correctly detecting the true anomalies accurately.



(a) Confusion matrix for AE



(b) Confusion matrix for VAE

Figure 5: Confusion matrix(UNSW-NB15 dataset)

From the figure 5; despite the imbalanced dataset, both models successfully identifies all of the 14,278 anomalies. However, VAE also identified 38,289 normal cases as anomalies, which explains the low precision.

Comparison with other models

Table 4 shows the comparison of various anomaly detection algorithms including as KNN(K-nearest neighbours), One-class SVM, isolation forest with AE and VAEs. Detailed working of other models can be found in Appendix C. To ensure a fair comparison, the KNN model was trained using the entire dataset, while the SVM and isolation forest were trained using normal observations in the training set.

Table 4: Model Performance Comparison(UNSW-NB15 dataset)

Model	Performance Metrics			
	F1-Score	Precision	Recall	AUC-ROC
KNN	0.088	0.04	0.61	0.66
One-class SVM	0.85	0.86	0.83	0.90
Isolation Forest	0.88	0.83	0.95	0.91
AE ($\gamma = 0.0383$)	0.92	0.85	0.99	0.98
VAE ($\gamma = 1.163$)	0.65	0.48	0.99	0.92

Based on the performance metrics in the table 4, KNN performed very poorly on all of the metrics. On the other hand, isolation-forest and one-class SVM performed better than KNN. SVM and isolation forest are able to detect majority of the anomalies but had some difficulty identifying all of the cases correctly. This indicates that the these models are struggling to find useful patterns in detecting the anomalies. In contrast, the high recall scores of AEs and VAEs imply that they are able to capture the most important patterns while reconstructing the data.

Interpreting AE model

In order to understand the performance of AE, LIME is used to explain the reconstructed output of the anomalous observation as shown in figure 6.

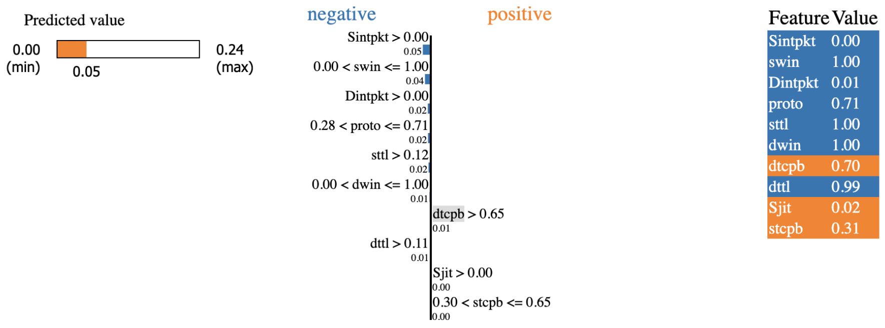


Figure 6: LIME explanation for anomalous instance of UNSW-NB15 dataset

In the Figure 6, the left part represents the average contribution of each feature towards the reconstruction. Furthermore, the middle part shows the feature importance with positive part showing features contributed most towards more accurate reconstruction, and the negative part shows which contribute less to the accurate reconstruction. These negative features can lead to high reconstruction error. The right part shows the top 10 features influencing the reconstruction of that particular observation locally. We can observe that the features- dtcpb(Destination TCP base sequence number), Sjit(Source jitter), Stcpb(Source TCP base sequence number) have a positive influence, while the other features are having a negative influence.

The impact of reconstruction using the AE model on global scale can be visualised in figure 7.

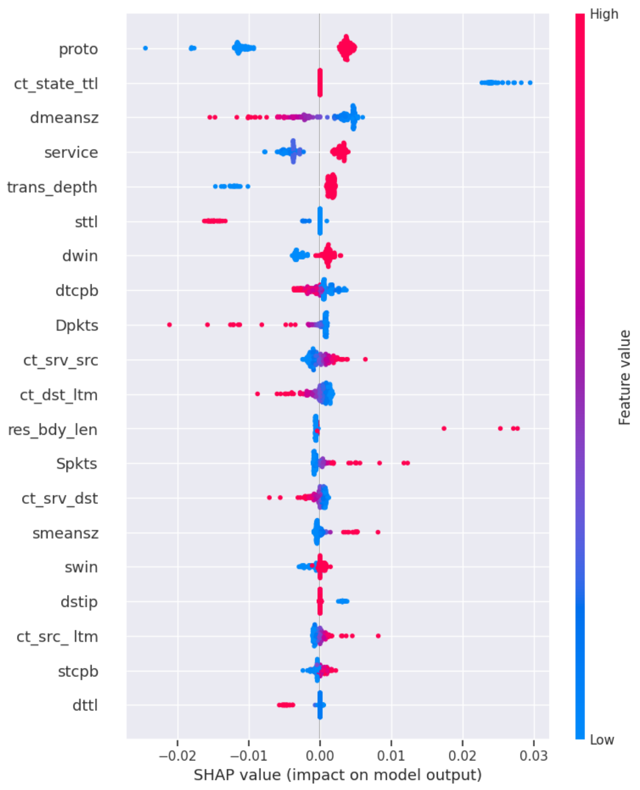


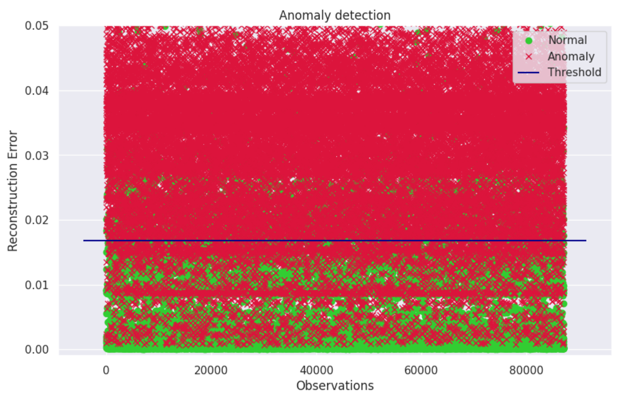
Figure 7: SHAP global explanation for UNSW-NB15 dataset

Figure 7 shows the features that have the most significant impact on the reconstruction process. The position of a feature in the SHAP plot indicates its relative importance or impact on the reconstruction(higher-placed features in the plot have a stronger influence on the reconstruction). It can be seen that the feature- proto(Transaction protocol) is heavily influencing the reconstruction.

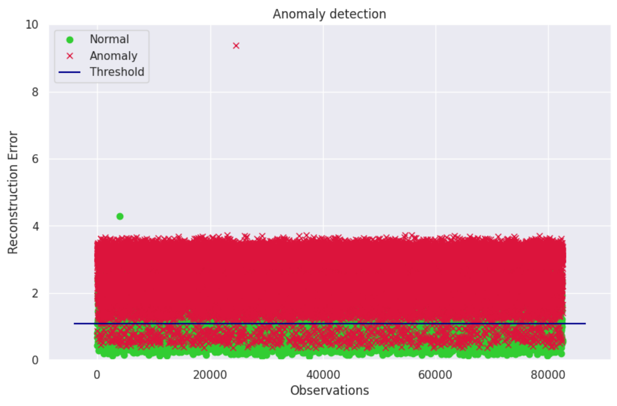
It can also be observed that the features- sttl, dwin, dtcpb, swin and dttl are still influencing the reconstruction on a global scale. These features have a significant influence on the reconstruction of the data, contributing to the accurate identification of anomalies. We can refer to Appendix B.1 for a comprehensive list of all feature names and their detailed description.

5.4.2 Reconstruction of NSL-KDD dataset

From Figure 8, it can be seen that AE model(8a) reconstructed most of the normal observations successfully and have a reconstruction error close to zero. Although, it is also evident that some anomalous cases have a reconstruction errors similar to those of normal instances. It implies that the AE model may have some challenges in accurately identifying the patterns of certain type of anomalies.



(a) Anomaly detection using AE(γ=0.0084)



(b) Anomaly detection using VAE(γ=1.102)

Figure 8: Anomaly detection comparison(NSL-KDD dataset)

In the case of VAE(8b), the model identifies most of the anomalies but some of them have reconstruction error close to normal cases; it also indicates that the patterns of some certain types of anomalies are not efficiently captured by the model.

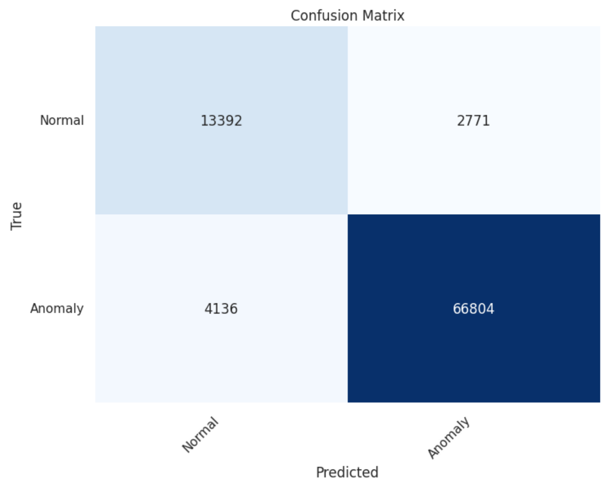
Table 5 shows the detailed comparison between AE and VAE performance metrics based on different threshold values(highlighted results indicate the optimal values).

Table 5: AE and VAE Model Comparison(NSL-KDD dataset)

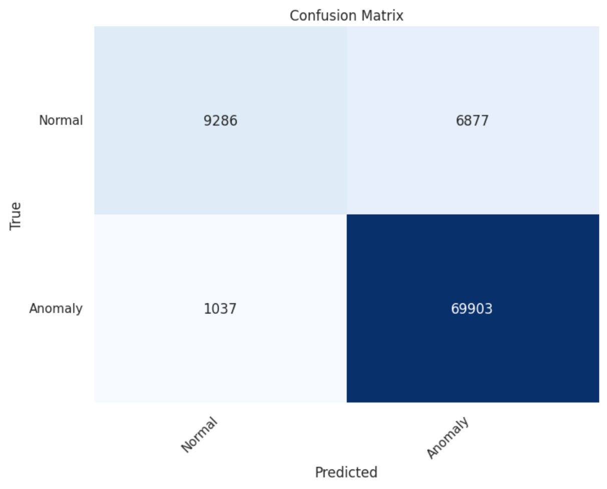
Model	Threshold (γ)	Performance Metrics			
		F1-Score	Precision	Recall	AUC-ROC
AE	0.0169	0.95	0.96	0.94	0.88
AE	0.0084	0.94	0.91	0.97	0.78
AE	0.0168	0.95	0.95	0.94	0.88
VAE	1.096	0.94	0.90	0.98	0.77
VAE	1.095	0.94	0.90	0.98	0.77
VAE	1.102	0.94	0.91	0.98	0.78

VAEs are performing consistently better than AE on all of the performance metric. It may indicate that the underlying data distribution is precisely captured by VAEs and is able to generate a more informative latent-space representations, resulting in improved anomaly detection performance as compared to AE.

Figure 9 represents the confusion matrix for both the AE and VAE models(using the optimal threshold values).



(a) Confusion matrix for AE



(b) Confusion matrix for VAE

Figure 9: Confusion matrix(NSL-KDD dataset)

From the figure, Out of the total 70,940 anomalies, the AE model (9a) correctly identified 66,804 cases, while the VAE model (9b) correctly identified 69,903 observations, which explains the high recall score of the VAE.

Comparison with other models

Table 6 shows the comparison of AE and VAE with other anomaly detection methods on the NSL-KDD data.

Table 6: Model Performance Comparison(NSL-KDD dataset)

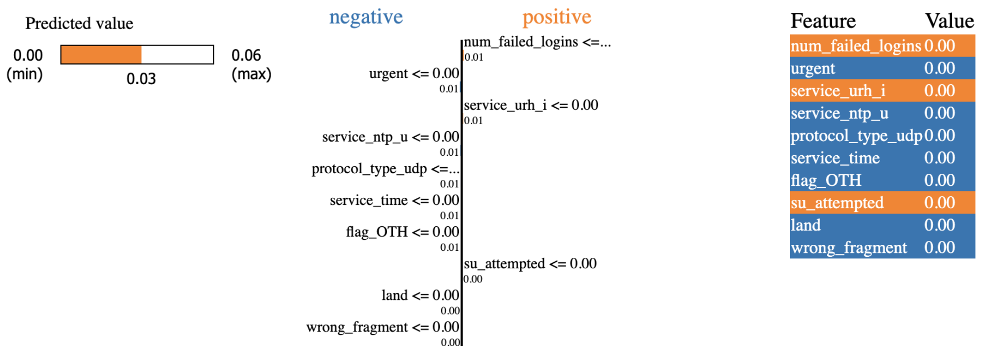
Model	Performance Metrics			
	F1-Score	Precision	Recall	AUC-ROC
KNN	0.62	0.55	0.72	0.58
One-class SVM	0.92	0.91	0.94	0.77
Isolation Forest	0.90	0.99	0.82	0.89
AE($\gamma = 0.0084$)	0.94	0.91	0.97	0.78
VAE($\gamma = 1.102$)	0.94	0.91	0.98	0.78

For the given dataset, the KNN model displays average performance across all metrics. Based on the recall score, the AE and VAE models continues to outperform other models. Although, isolation forest have a higher precision indicating the higher AUC-ROC score than the AE and VAE. But it falls behind in identifying the true anomalies.

Based on the performance table, VAE model demonstrates higher performance in detecting true anomalies accurately. Hence, LIME and SHAP is used to identify the important features that helped VAE in detecting the anomalies.

Interpreting VAE model

LIME is used to explain the reconstructed output for the anomalous observation as shown in figure 10.



Feature	Value
num_failed_logins	0.00
urgent	0.00
service_urh_i	0.00
service_ntp_u	0.00
protocol_type_udp	0.00
service_time	0.00
flag_OTH	0.00
su_attempted	0.00
land	0.00
wrong_fragment	0.00

Figure 10: LIME explanation for anomalous observation(NSL-KDD data)

The right part in the Figure 10 shows the 10 most important features contributing to the reconstruction. The features having positive influence locally on the reconstruction are- num_failed_logins(number of failed login attempts), service_urh_i(unkown service) and su_attempted(substitute user command was attempted in the connection). However, the remaining features have a negative impact on the reconstruction, potentially increasing the error.

Globally the most significant features affecting the reconstruction of the data can be visualised in Figure 11.

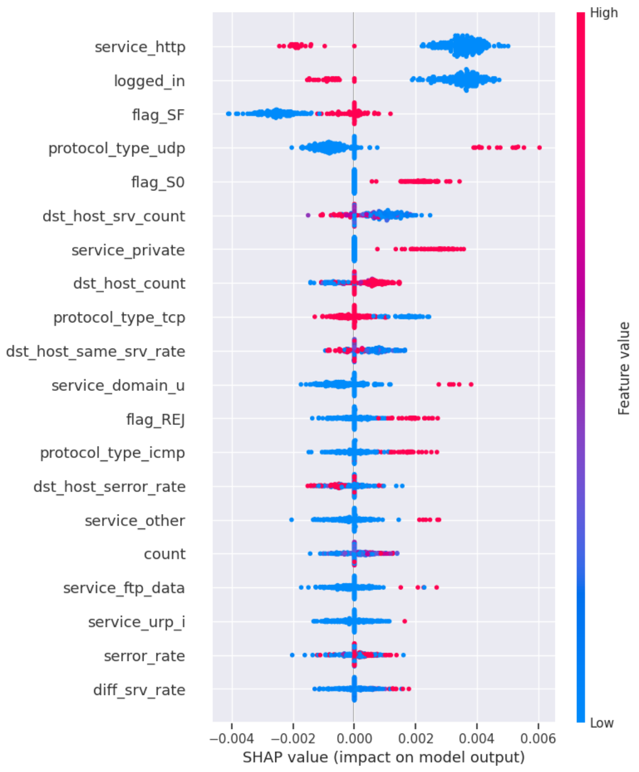


Figure 11: SHAP global explanation for NSL-KDD dataset

It is evident that feature named service_http is the most important feature. However, feature protocol_type_udp is still influencing the reconstruction on the global scale. Furthermore, feature logged_in(if the user is logged in to the network) also played a significant role on the reconstruction. We can refer to Appendix B.2 for a comprehensive list of all feature names and their detailed description.

5.5 Limitations of AE and VAE in Anomaly Detection

The **threshold** is the most important hyperparameter that significantly influences the performance of AEs and VAEs. Inadequate tuning of the threshold values can lead to the misclassification of the true anomalies for both models. Moreover, in the case of NSL-KDD dataset both models encountered difficulties in identifying the complex patterns and could not effectively separate the normal and anomalous observations based on the reconstruction error as seen in Figure 8, which indicate that these models have had trouble detecting the patterns of certain type of attacks. Additionally, we are training our models only on the normal observations, and training over these observations could potentially lead to over-fitting to normal patterns and may hinder the models ability to generalise effectively to true anomalies.

6 Conclusion

This report thoroughly investigated the utilisation of Autoencoders and Variational Autoencoders(VAEs) for anomaly detection in the field of cybersecurity. Considering the specific characteristics of the datasets analysed in this study, the choice of the preferred model for anomaly detection between autoencoders(AEs) and variational Autoencoders(VAEs) can vary depending on the data. In the case of the first dataset, AE demonstrated superior performance in accurately identifying true anomalies. However, for the second dataset, VAE showed better capabilities. Moreover, the results achieved by both models demonstrate their superior performance as compared to other traditional anomaly detection approaches.

However, the performance of AE and VAE can vary depending on different types of anomalies present in the data and the architecture of encoder and decoder. Further research is needed to address the performance of these models with different types of attacks(anomalies) in the real-world data with varying characteristics. This may help us to identify the strengths and limitations of these models in handling specific types of anomalies.

References

[1] S. Alla and S. K. Adari. *Beginning anomaly detection using python-based deep learning.* Springer, 2019.

[2] C. A. Charu. *OUTLIER ANALYSIS.* springer, 2019.

[3] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[4] S. Keydana. Adding uncertainty to deep learning with tensorflow probability. Image retrieved from `https://rpubs.com/zkajdan/533047`.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] D. P. Kingma, M. Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[7] S. Lundberg. Shap: A unified approach to explain the output of any machine learning model. `https://github.com/slundberg/shap`, n.d.

[8] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[9] 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)*, pages 1–6, 2015.

[10] M. T. C. Ribeiro. Lime: Explaining the predictions of any machine learning classifier. `https://github.com/marcotcr/lime`.

[11] J. Rocca. Understanding variational autoencoders. Image retrieved from `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

[12] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6. IEEE, 2009.

Appendices

A Data: Insights and Preprocessing steps

There were two datasets provided for our investigation- UNSW-NB15 and NSL-KDD. Both are made up of network traffic data that replicates real-world network data. Initially UNSW-NB15 consists of 700,001 observations and NSL-KDD consisted of 148517 observations.

UNSW-NB15 dataset: Initially, dataset consisted of 22,215 anomalous observations out of 700,001. But preprocessing revealed that it consists of 59,213 duplicate observations(i.e duplicate rows), which were removed by keeping their first occurrence only.

However, there were no missing observations. Some of the features such as sport(Source port number) and dsport(destination port number) consists of 6 and 56 hexadecimal values respectively which were also removed. This resulted in the fewer number of observation (i.e 640,726) as compared to initial observations. After the modifications, we were left with 14,278 anomalous cases only. Also, feature named **service** contains most of the observation with '-' as a unique value, so it was replaced with "other" category. We were not able to remove these observation because most of the observation(i.e 388,508) were assigned to this value. Based on the difference between the Stime(record start time) and Ltime(record last time), a new feature time difference was created, and both Stime and Ltime features were removed.

There were some variables which had less than 10 unique values, so we also used them as categorical variables (such as trans_depth(represents the pipe-lined depth into the connection) with 6 unique values, sport and dsport with approximately 65,000 unique values each). In total, eleven categorical features were identified and to encode these categorical features we first attempted one hot encoding. However, system crashed due to the high volume of observation. To reduce the load, we also attempted to encode the features which had less than 50 unique values but the results were unsatisfactory due to the sparsity of the dataset. Instead frequency encoding was used, it involves the encoding categorical variables based on frequency of occurrence of each category within the dataset. The main disadvantage was that if a new or unseen category appeared in the new data set or during the inference, the model would behave unusually. To address this issue, we can apply a special label to these unseen categories.

Numerical features were scaled using Min-max scaling as the expected output range was between 0 and 1. This scaling choice was motivated by the use of a sigmoid activation function in the last layer of the autoencoder and variational autoencoder model.

NSL-KDD dataset: NSL-KDD dataset contained 610 duplicate observations, which were also removed by keeping their first occurrence only. We cannot find any missing values in the dataset. After the removal of unwanted observations, we were left with 147,907 observations. And in those observation 70,940 were anomalies.

Feature named attack was divided into subcategories of various attack namely- DoS(Denial of service), U2R(User to root), R2L(Remote to local). Detailed subcategories of these attacks are shown in table 7.

Table 7: **Attack Categories** (Reference from https://www.kaggle.com/code/timgoodfellow/nsl-kdd-explorationshttps)

Attack Type	**Attacks**
DoS Attacks	apache2, back, land, neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm
Probe Attacks	ipsweep, mscan, nmap, portsweep, saint, satan
U2R	buffer_overflow, loadmdoule, perl, ps, rootkit, sqlattack, xterm
R2L	ftp_write, guess_passwd, http_tunnel, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, xclock, xsnoop

Based on these categories, 75% cases were related to the DoS attacks, 18% related to R2L and rest 7% was related to U2R attacks.We identified three categorical features and to prepare them for the model one hot encoding method was used to encode them. This encoding will help the model to utilise these categorical features effectively. The training data contained both normal and anomalous cases, despite the fact that this dataset had already been divided into training and testing data. However, because we only wanted to train our model on normal observations, we combined these two sets of data and used 70% of the normal observations to train all of the models. The remaining 15% was joined with anomalous cases to create test data, and the remaining 15% normal cases were used as validation data.

B Important feature names and their Description

B.1 Most Important features in UNSW-NB15 dataset for Anomaly detection

Table 8: Important features using LIME

Feature Name	**Description**
sintpkt	Duration (in seconds) between the current and previous packet sent by the source
swin	Size of the TCP window advertisement in the TCP handshake process
dintpkt	Duration (in seconds) between the current and previous packet received by the destination
sttl	Time to live (TTL) value of the packet sent by the source
dwin	Size of the TCP window advertisement in the TCP handshake process
dttl	Time to live (TTL) value of the packet received by the destination
dtcpb	Number of data bytes in the TCP payload received by the destination
sjit	Jitter (variance in packet inter-arrival time) of packets sent by the source
stcpb	Number of data bytes in the TCP payload sent by the source

Table 9: Important features (in decreasing order) using SHAP

Feature Name	**Description**
proto	Protocol used in the network communication
ct_state_ttl	Connection state and Time-to-Live (TTL) value of the packet received by the destination
dmeansz	Mean size of the data payload received by the destination
service	Type of service associated with the network connection
trans_depth	Depth of HTTP transactions within the connection
sttl	Time to live (TTL) value of the packet received by the source
dwin	Size of the TCP window advertisement in the TCP process received by the destination
dtcpb	Number of data bytes in the TCP payload received by the destination
Dpkts	Number of packets received during the connection
ct_srv_src	Number of connections with the same service and source IP address in a certain time frame
ct_dst_ltm	Number of connections with the same destination IP address in a certain time frame
res_bdy_len	Length of the response body in an HTTP transaction
Spkts	Number of packets sent during the connection
ct_srv_dst	Number of connections with the same service and destination IP address in a certain time
smeansz	Mean size of the data payload sent by the source
swin	Size of the TCP window advertisement in the TCP handshake process sent by the source
dstip	Destination IP address
ct_src_ltm	Number of connections with the same source IP address in a certain time frame
stcpb	Number of data bytes in the TCP payload sent by the source
dttl	Time to live (TTL) value of the packet received by the destination

B.2 Most Important features in NSL-KDD dataset for Anomaly detection

Table 10: Important features using LIME

Feature Name	**Description**
urgent	Indicates the presence of urgent data in the network connection
service_ntp_u	Network Time Protocol (NTP) service, UDP connection
service_time	Duration of the network connection
flag_OTH	Flag value indicating other (unexpected) connection state
land	Indicates if the source and destination IP addresses and port numbers are equal
wrong_fragment	Indicates the presence of incorrect or malformed IP fragments
num_failed_logins	Number of failed login attempts
service_urh_i	Service is an unknown or unrecognized protocol (URH-I)
su_attempted	Indicates if the user's superuser (root) privileges were attempted to be gained

Table 11: Important features (in decreasing order) using SHAP

Feature Name	**Description**
service_http	HTTP protocol used for communication
logged_in	Indicates whether the user is logged in
flag_SF	Normal packet (no flags set)
flag_S0	Indication of a connection attempt
dst_host_srv_count	Number of connections to the same service on the destination host
service_private	Private service used for communication
dst_host_count	Number of connections to the same host
protocol_type_tcp	TCP protocol used for communication
dst_host_same_srv_rate	Percentage of connections to the same service on the destination host
service_domain_u	Service used for communication is unknown or undefined
flag_REj	Connection rejected (SYN received, RST sent)
protocol_type_icmp	ICMP protocol used for communication
dst_host_serror_rate	Percentage of connections with a SYN error on the destination host
service_other	Other service used for communication
count	Number of connections to the same host as the current connection
service_ftp_data	FTP data transfer service
service_urp_i	URP protocol used for communication
serror_rate	Percentage of connections with a SYN error
diff_srv_rate	Percentage of different services used among connections

C Anomaly detection using other models

C.1 K-nearest neighbours(KNN)

KNN is a machine learning technique that can be applied to both supervised and unsupervised learning tasks. The fundamental basis of KNN is that the similar data points are in close proximity with each other, while outliers are typically located far from the vicinity of similar observations, which is determined based on the distance metrics such as Euclidean distance. If the distance between the points is greater than a specific threshold, we can label that as an anomaly. However, since we don't have access to labels in the actual world for anomaly detection, we are employing an unsupervised method instead and creating labels depending on the threshold.

The algorithm 5 illustrates an unsupervised method of anomaly detection using KNN.

Algorithm 5 KNN-Based Unsupervised Anomaly Detection

```
Require: Dataset with N data points: $\{x_1, x_2, \ldots, x_N\}$, K (Number of clusters), threshold $\gamma$
Ensure: Anomalies: $\{anomaly_1, anomaly_2, \ldots, anomaly_m\}$
 1: procedure DetectAnomalies($\{x_1, x_2, \ldots, x_N\}$, K, $\gamma$)
 2:     for $i = 1$ to $N$ do
 3:         for $j = 1$ to $N$ do
 4:             $D[i][j] \leftarrow$ EuclideanDistance($x_i, x_j$)
 5:     Initialize Anomalies ← empty array
 6:     for $i = 1$ to $N$ do
 7:         Find the indices of the $K$ nearest neighbors of $x_i$ based on $D$
 8:         Calculate the average distance, $avg\_distance$, to the $K$ nearest neighbors
 9:         if $avg\_distance > \gamma$ then
10:             Classify $x_i$ as an anomaly
11:             Add $x_i$ to Anomalies
12:     return Anomalies

13: function EuclideanDistance($x, y$)
14:     return $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
```

In above algorithm 5, both K(number of clusters) and threshold γ are both considered as hyper parameters. The algorithm is trained on whole data.

C.2 One-class SVM

The algorithm 6 illustrates anomaly detection using one-class SVM.

Algorithm 6 One-Class SVM for Anomaly Detection

```
Require: Train data without anomalies: $\{(\mathbf{x}_1), (\mathbf{x}_2), \ldots, (\mathbf{x}_N)\}$, Test data:$\{(\mathbf{x}_{test1}), (\mathbf{x}_{test2}), \ldots, (\mathbf{x}_{testM})\}$,
    Hyperparameters: kernel, $\nu$
Ensure: Anomalies: $\{(\mathbf{x}_{test_i}) \mid \text{Anomaly detected}\}$
 1: procedure DetectAnomalies(TrainData, TestData, kernel, $\nu$)
 2:     Train the One-Class SVM model with kernel and $\nu$ parameters
 3:     Fit the One-Class SVM model on TrainData
 4:     Initialize Anomalies ← empty array
 5:     for each instance $\mathbf{x}_{test_i}$ in TestData do
 6:         Predict the anomaly score: $s_i$ = model.predict($\mathbf{x}_{test_i}$)
 7:         if $s_i < 0$ then
 8:             Add $\mathbf{x}_{test_i}$ to Anomalies
 9:     return Anomalies
```

C.3 Isolation Forest

The algorithm 7 illustrates anomaly detection using isolation-forest.

Algorithm 7 Anomaly Detection using Isolation Forest

```
Require: Train data with normal observations: ${(\mathbf{x}_1), (\mathbf{x}_2), \ldots, (\mathbf{x}_N)}$,
    Test data: ${(\mathbf{x}_{\text{test}1}), (\mathbf{x}_{\text{test}2}), \ldots, (\mathbf{x}_{\text{test}M})}$, Hyperparameters: n_trees, sample_size, max_depth
Ensure: Anomalies: ${(\mathbf{x}_{\text{test}_i}) \mid \text{Anomaly detected}}$
 1: procedure DetectAnomalies(TrainData, TestData, n_trees, sample_size, max_depth)
 2:     Initialize empty list: Anomalies $\leftarrow$ empty array
 3:     for each instance $\mathbf{x}_{\text{test}_i}$ in TestData do
 4:         Initialize isolation forest parameters: n_trees, sample_size, max_depth
 5:         Initialize empty list: anomaly_scores
 6:         for $j = 1$ to n_trees do
 7:             Randomly select a subsample of sample_size instances from TrainData
 8:             Build an isolation tree using the selected subsample with a maximum depth.
 9:             Calculate anomaly scores for each instance in TestData based on the isolation tree
10:             Append the anomaly scores to anomaly_scores
11:         Calculate the average anomaly score for the instance by aggregating the scores.
12:         if average anomaly score > threshold then
13:             Add $\mathbf{x}_{\text{test}_i}$ to Anomalies
14:     return Anomalies
```