EEC 289A Final Report
Ejiroghene Uwhuba

Unsupervised Network Anomaly Detection

1. Introduction:

1.1 The Problem

The detection of anomalies in network traffic is crucial for ensuring network security. Modern networks handle vast amounts of data, making it challenging to identify patterns or behaviors that signal potential threats such as cyberattacks, data breaches, or unauthorized access. Packet spoofing is one such cyberattack technique where a hacker can make a corrupted packet outwardly appear as coming from a trusted source. Detecting such anomalies manually takes too much time, as dozens of data packets are sent back and forth per second on most networks. Therefore, a well trained machine learning model is the most efficient way of addressing this problem.

1.2 The Motivation

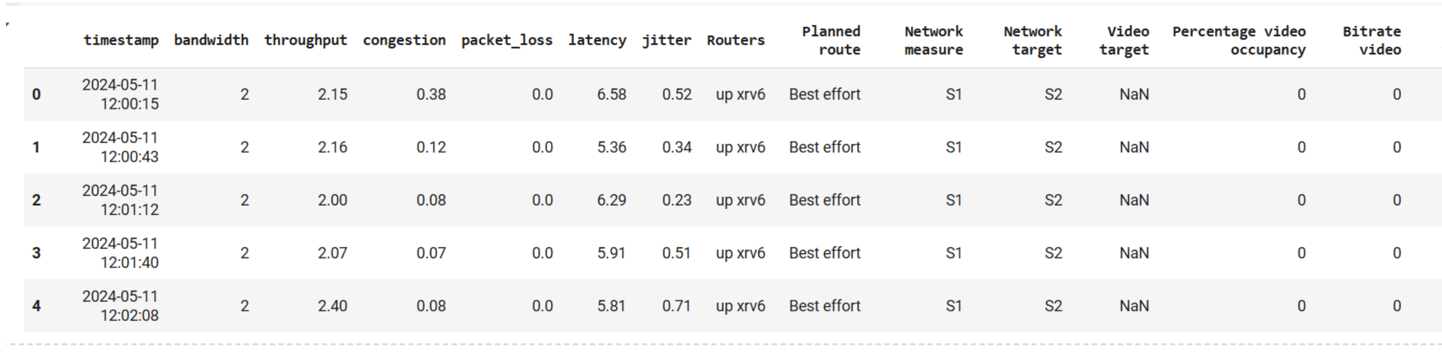
Bolstering the defenses of a laptop from such attacks is of utmost importance in the field of cybersecurity. Anomaly detection in network traffic is essential for proactive cybersecurity. Identifying threats early helps prevent data breaches and ensure smooth network operations. With the rise of sophisticated attacks, this project contributes to advancing techniques for safeguarding digital infrastructure. A machine learning model allows for easy finetuning and hyperparameter editing to keep a machine learning model up to date for the most current hacking attacks. Lastly, the field of cybersecurity is a field that personally interests me and could have a future working in for my career advancement as a Software Engineer.

1.3 Introduction:

While building this final project, the main goal that I wanted to achieve was to build a completely unsupervised machine learning model, with a variational autoencoder as the core training model. No labeled data was used in the process of training this model. A variational autoencoder is an advanced autoencoder that goes beyond just converting input into latent spaces for reconstruction, but also generates entirely new data. The latent space representation allows for easier identification of anomalous items as similar inputs are mapped onto similar points inside the latent space. Through these reconstructed objects, I was able to evaluate my model by evaluating and graphically reporting these core components: training/testing loss of the VAE over a range of epochs, and the calculation of the reconstruction error formed by the VAE to name a few. I will go into great detail at each method that I used to formulate my anomaly detection model at the methodology section.

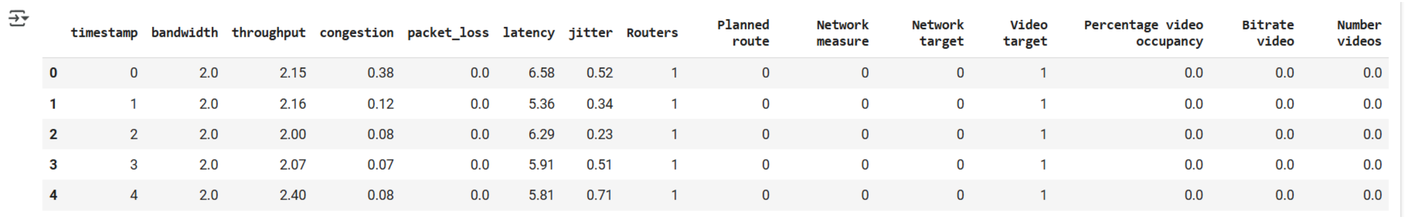
1.4 Dataset

The project will utilize a publicly available network traffic dataset from Kaggle. This dataset includes features such as throughput/congestion/packet-loss/latency/jitter, but no labels for determining anomaly as my project will be unsupervised in its entirety. In the python notebook final output of my project, the inside of the dataset will periodically be shown by using the panda class for dataset preprocessing.



	timestamp	bandwidth	throughput	congestion	packet_loss	latency	jitter	Routers	Planned route	Network measure	Network target	Video target	Percentage video occupancy	Bitrate video
0	2024-05-11 12:00:15	2	2.15	0.38	0.0	6.58	0.52	up xrv6	Best effort	S1	S2	NaN	0	0
1	2024-05-11 12:00:43	2	2.16	0.12	0.0	5.36	0.34	up xrv6	Best effort	S1	S2	NaN	0	0
2	2024-05-11 12:01:12	2	2.00	0.08	0.0	6.29	0.23	up xrv6	Best effort	S1	S2	NaN	0	0
3	2024-05-11 12:01:40	2	2.07	0.07	0.0	5.91	0.51	up xrv6	Best effort	S1	S2	NaN	0	0
4	2024-05-11 12:02:08	2	2.40	0.08	0.0	5.81	0.71	up xrv6	Best effort	S1	S2	NaN	0	0

My first task for this project was to complete the preprocessing of this dataset for usage in my project; the separation of columns into both categorical and numeric types was an important first step for streamlining this process. I used the label encoding class of the sklearn package to transform the categorical columns into integers, mapping each unique category into distinct values.



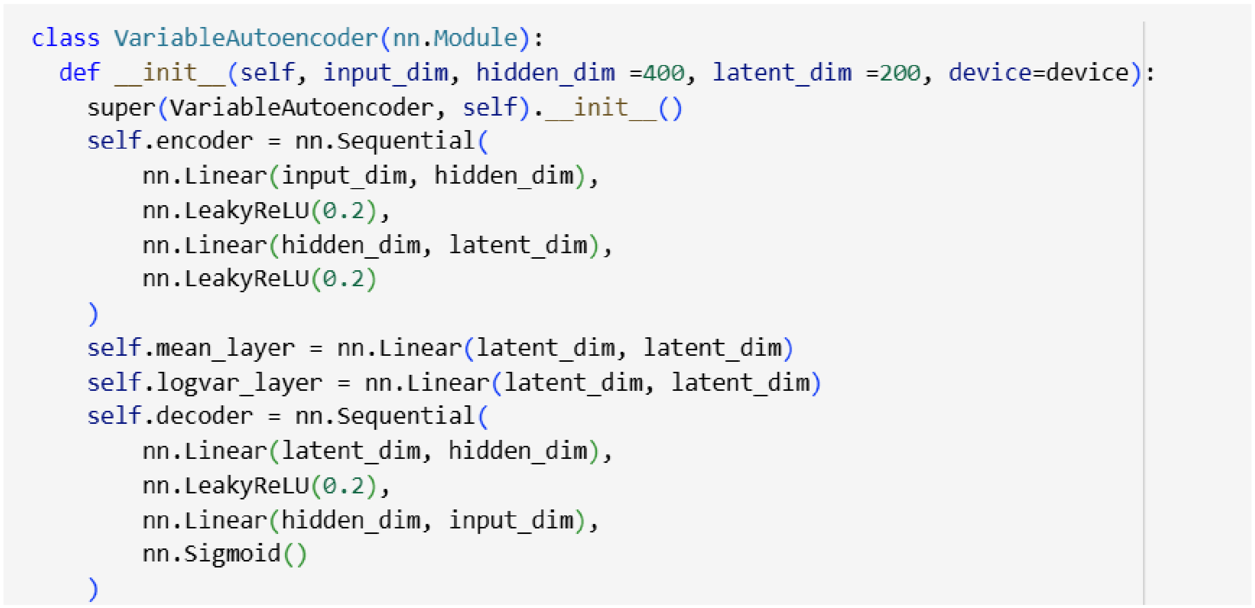
	timestamp	bandwidth	throughput	congestion	packet_loss	latency	jitter	Routers	Planned route	Network measure	Network target	Video target	Percentage video occupancy	Bitrate video	Number videos
0	0	2.0	2.15	0.38	0.0	6.58	0.52	1	0	0	0	1	0.0	0.0	0.0
1	1	2.0	2.16	0.12	0.0	5.36	0.34	1	0	0	0	1	0.0	0.0	0.0
2	2	2.0	2.00	0.08	0.0	6.29	0.23	1	0	0	0	1	0.0	0.0	0.0
3	3	2.0	2.07	0.07	0.0	5.91	0.51	1	0	0	0	1	0.0	0.0	0.0
4	4	2.0	2.40	0.08	0.0	5.81	0.71	1	0	0	0	1	0.0	0.0	0.0

The next step in preparing this dataset was then to split the numerical columns into a training, testing, and validation portion. Twenty percent of the dataset is reserved for testing, and the remaining eighty percent reserved for training and validation (training-75% of 80, validation-25% of 80) For my final step of preprocessing, I used StandardScaler in Python to apply scaling transformation on the newly split dataset in accordance to the mean and standard deviation. These scaled data points will be used in building the tensors that will hold the train/test/validation dataset and loader.

2 Methodology & Approach:

2.1 Variational Autoencoder (VAE)

Anomaly detection through variational autoencoders is achieved by using the model's generative capabilities; a trained VAE model that can accurately reconstruct normal data points from a dataspace should in turn struggle to generate anomalous data points distinct from its trained model. The inability to reconstruct such data can then be used as a way to identify anomalies in a space. The VAE, with its major components being the encoder and decoder, compresses the dataset to address the curse of dimensionality and transform the distribution from a high dimensional space into a lower one. Passed into my VAE's architecture is the dimensionality of the input, which the encoder portion of the autoencoder will be tasked with compressing. Furthermore, the hidden dimensions are accessed by the decoder to do transformations on the input, and reduce the linearity. Lastly the latent dimensions determines the capacity of the latent representation.



```
class VariableAutoencoder(nn.Module):
  def __init__(self, input_dim, hidden_dim =400, latent_dim =200, device=device):
    super(VariableAutoencoder, self).__init__()
    self.encoder = nn.Sequential(
        nn.Linear(input_dim, hidden_dim),
        nn.LeakyReLU(0.2),
        nn.Linear(hidden_dim, latent_dim),
        nn.LeakyReLU(0.2)
    )
    self.mean_layer = nn.Linear(latent_dim, latent_dim)
    self.logvar_layer = nn.Linear(latent_dim, latent_dim)
    self.decoder = nn.Sequential(
        nn.Linear(latent_dim, hidden_dim),
        nn.LeakyReLU(0.2),
        nn.Linear(hidden_dim, input_dim),
        nn.Sigmoid()
    )
```

2.2 VAE Architecture

The encoder method takes as input a linear layer composed of the aforementioned parameters, and then computes the mean and log variance of the latent distribution. The decoder method implements the forward pass of the decoder by creating a latent vector z, produced by taking the mean and log variance into the reparametrize method. This method samples from a standard normal distribution to separate randomness, and allow for backpropagation during the training. The decoder then passes this vector through its linear layers to produce a reconstructed data point (x_hat) trained by the VAE. Outside the VAE class is a helper method containing the loss function, which takes the mean square error and Kullback-Leibler Divergence of the mean/log variance/x-hat, and minimizes the sum of these two values to train the VAE.

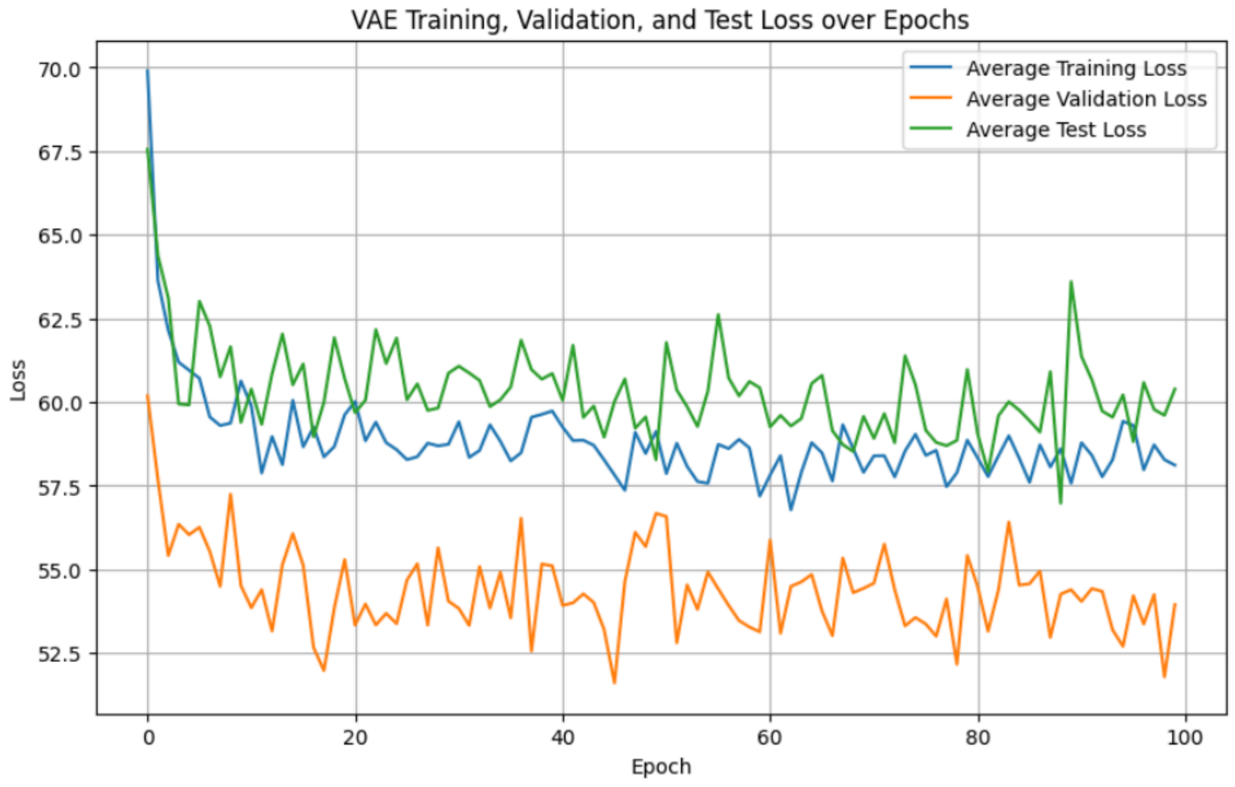
2.3 The Training Loop

My training loop traverses through the training samples at batches of 8 across a time range of 100 epochs. The x_hat, mean, and log variance computed in the helper loss function are logged to continually update the model's loss as it passes through each batched training sample. Then the Adam optimizer facilitates the processing of gradient descent, backpropagation, and the updating of parameters. The training loop also maintains a consistent count of training/testing/validation loss so that the average can be reported numerically and graphically after the last epoch is finished. Refer to the results section of the paper to see the numerical and graphical output of these averages.

3 Experiments, Results, Analysis:

3.1 Training Loop Results

```
VAE Final Average Training Loss over 100 epochs: 58.8559156791687
VAE Final Average Validation Loss over 100 epochs: 54.385867943573
VAE Final Average Test Loss over 100 epochs: 60.33000109461637
```



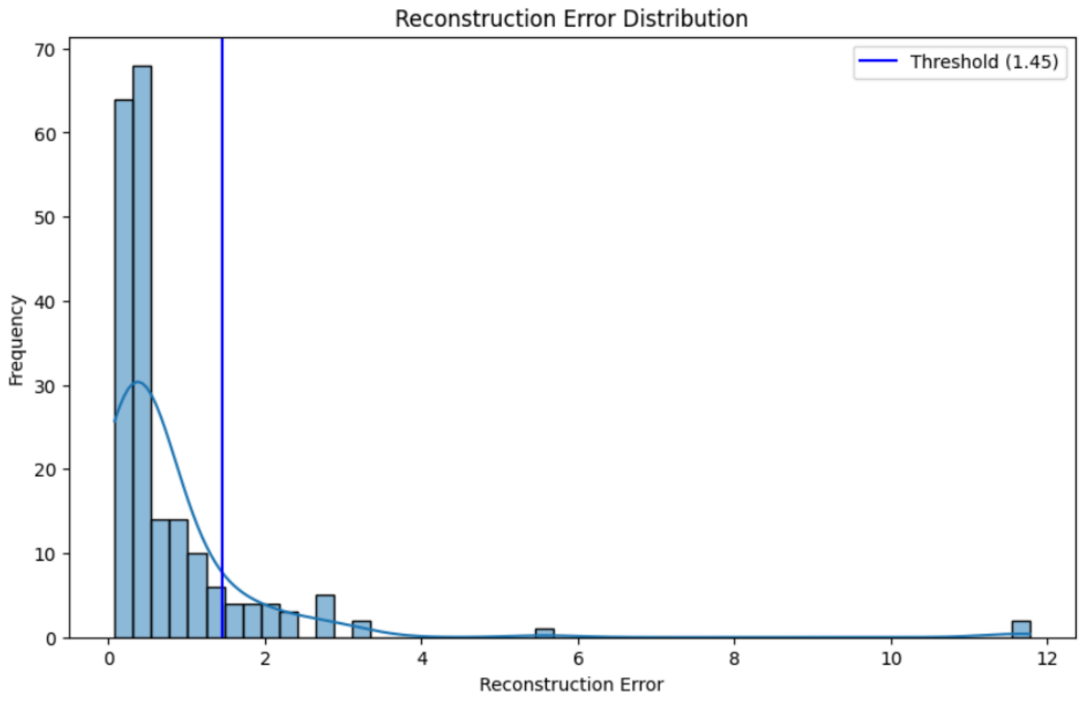
The average training and testing stabilized between the low 60's and 50's across 100 epochs. The validation error neither increasing or plateauing is a sign that no overfitting occurs in the model. I experimented with certain hyper-parameters such as learning rate to determine the best model fit, and came to initialize my numbers as is. In the development of evaluation metrics

I had many attempts at trying to create a cross-validation of the model, until realizing it's better utilized in supervised machine learning. I instead used these three evaluation metrics: reconstruction error visualization, the latent space through t-SNE, and One-Class SVM Anomaly detection.

3.2 Reconstruction Error

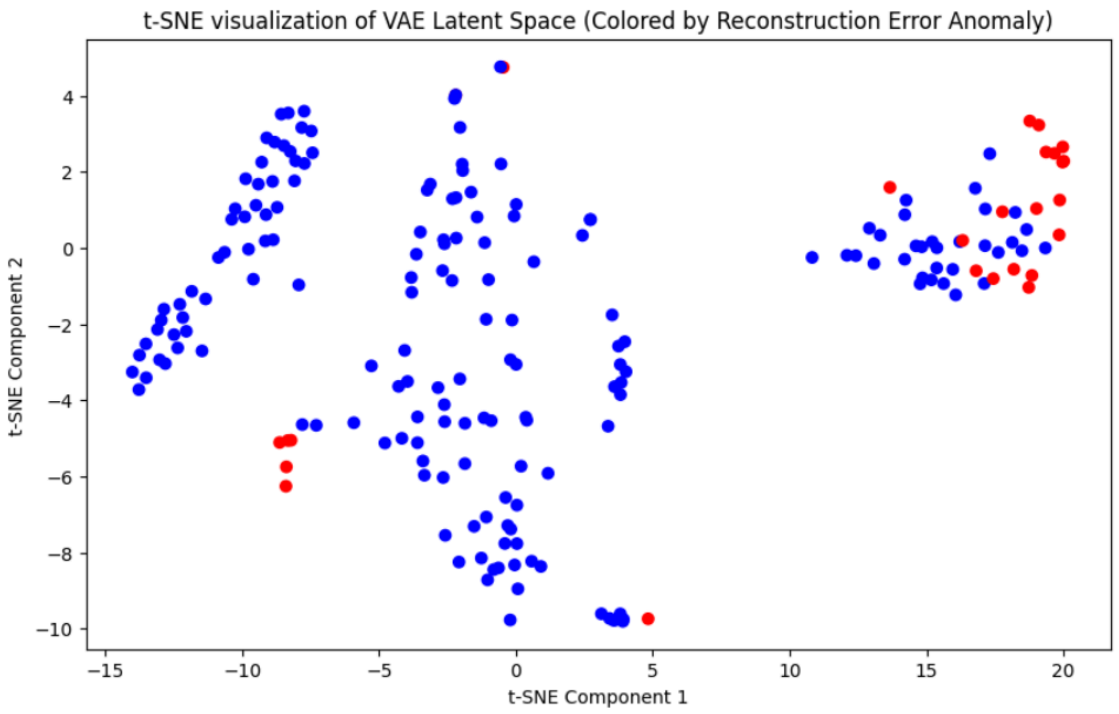
The VAE model allows for easy calculation of reconstruction and utilizes the torch.grad looping function used for testing and validating in the dataset in training loop. The 87th percentile was chosen after hyperparameter testing in the One-class SVM anomaly, and is the threshold for which anomalies are considered. With this threshold 26 anomalies were found.

```
Reconstruction error threshold for anomalies (87th percentile): 1.4484
Number of anomalies detected using reconstruction error: 26
```



3.3 t-SNE Visualization of Latent Space

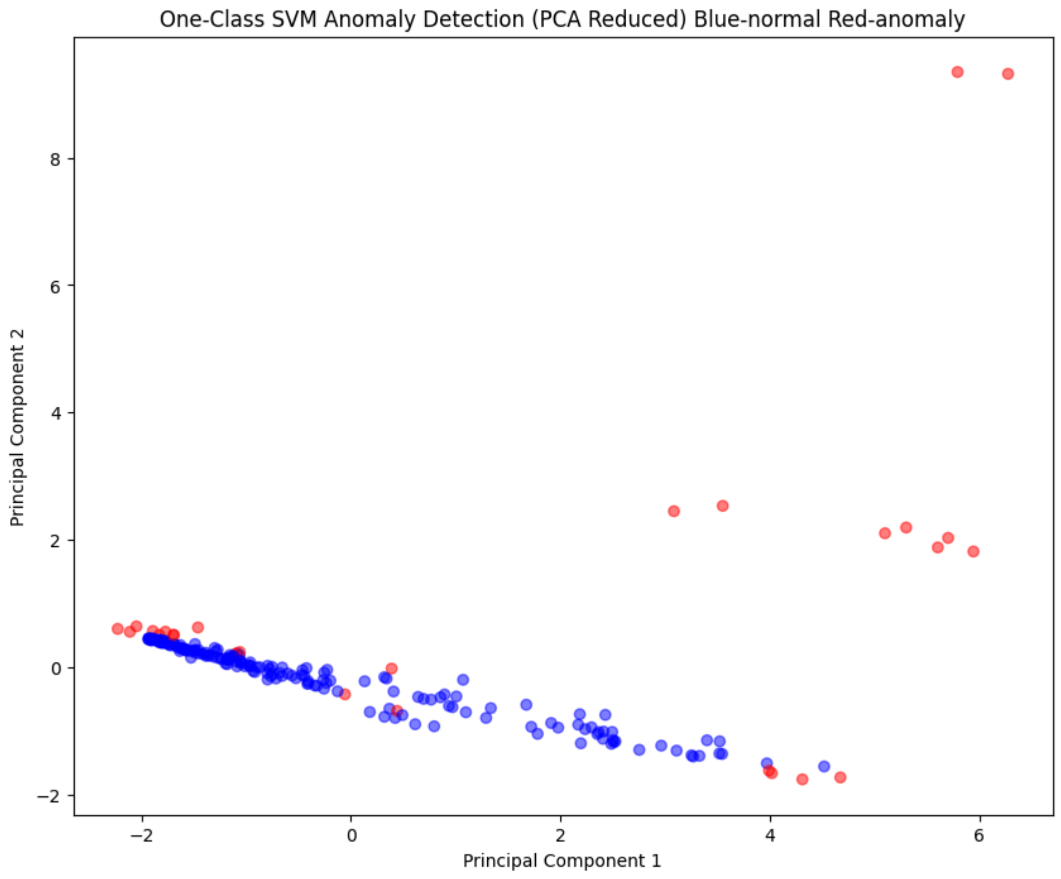
The t-SNE Visualization method is used to visualize the anomalous clusters found with the reconstruction error distribution and visualize them in a latent space. The red dots represent the data points above the threshold and are anomalous.



3.4 One-Class SVM Anomaly Detection

This distribution method excels at identifying data points with high deviance. It's different from the aforementioned reconstruction error, and is another way to view the anomalies. The 87th percentile used as the threshold in prior methods was to match the anomaly detection rate of this SVM architecture.

```
One-Class SVM detects network anomalies, with a maximum of 10% of the training data labeled as outliers detected totalling: 27
Percentage of anomalies detected: 13.43%
```



4. Post Analysis, Conclusion & Future Work

The comprehensive and rigorous evaluation of this network anomaly detection has showcased a great success rate at finding anomalies without falling for common setbacks such as overfitting. Some limitations that I came across with this project was finding a good balance between a dataset's ability to pack as much information as possible in a network packet, compared to the dataset size. At first I tried using wireshark to read network packet data coming from my laptop, but the amount of categorical and numerical data that could be processed was fewer than the smaller publicly available kaggle dataset that I chose for my project. In my future work, utilizing better network traffic analysis tools to create my own dataset could be a great way to expand on my current work. Right now my evaluational complexity is limited by the relatively small size of the dataset, but the high scalability of the code makes it easier to ramp up the model's scope with ease.

Word Count: 1458

5. References

Rekalantar, R. (n.d.). *Hands-on anomaly detection with variational autoencoders*. Medium. Retrieved from https://medium.com/data-science/hands-on-anomaly-detection-with-variational-autoencoders-d4044672acd5

GeeksForGeeks. (n.d.). *Variational Autoencoders*. Retrieved from https://www.geeksforgeeks.org/variational-autoencoders/

Rekalantar, R. (n.d.). *Variational Autoencoder (VAE) PyTorch tutorial*. Medium. Retrieved from https://medium.com/@rekalantar/variational-auto-encoder-vae-pytorch-tutorial-dce2d2fe0f5f

HackerNoon. (n.d.). *Latent space visualization - Deep learning bits*. Retrieved from https://medium.com/hackernoon/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Towards Data Science. (n.d.). *Difference between autoencoder (AE) and variational autoencoder (VAE)*. Retrieved from https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2/

Kaggle. (n.d.). *Network anomaly dataset*. Retrieved from https://www.kaggle.com/datasets/kaiser14/network-anomaly-dataset