Deep Learning methods applied on intrusion detection

SYSC 5801W Advanced Network Routing Techn Winter 2024 final report

Name: Yifei Zhang

1. Abstract

This report investigates the application of deep learning techniques in intrusion detection systems (IDS), highlighting the adaptability and broad applicability of these models across diverse datasets. Despite existing research achieving high accuracy in identifying various attack types, this study introduces a novel IDS approach that extends these models' utility beyond specific contexts. Utilizing a combination of Convolutional Neural Networks (CNN) and self-attention mechanisms, this model processes and classifies network traffic data transformed from packet captures (Pcap) files. In the initial stage, CNN layers extract key features at the byte level from each packet. These features are then compiled into a comprehensive packet-level vector, with an attention layer emphasizing crucial attributes for targeted classification. The classification phase uses additional CNN layers to adjust the feature vectors to fit the target classes accurately. This report proposes three distinct training methods—supervised, semi-supervised, and self-supervised learning—to enhance the model's ability to discern latent features in normal and abnormal traffic.

Key words: *Convolutional Neural Networks, self-attention mechanisms, supervised, latent features*

2. Introduction

Intrusion Detection Systems (IDS) are essential for network security, designed to identify unauthorized access and malicious activities within networks. Traditional IDS methods often find it challenging to keep up with the evolving and complex nature of modern cyber threats. To address this, there is increasing interest in using advanced computational methods to improve these systems' detection capabilities. Machine learning, especially deep learning that employs sophisticated neural network architectures, has proven effective in enhancing both the accuracy and efficiency of intrusion detection by automatically recognizing patterns of both normal and suspicious activities. Thus, the key innovation of my project can be generalized as follow:

- **Adaptive Multi-Flow with Packet-based Neural Network:** This report introduces a new neural network architecture that seamlessly adapts to the network structure. It is uniquely designed to leverage data characteristics, automatically extracting abstract features. This approach surpasses the limitations of models reliant on handcrafted features, thereby achieving superior classification results by accuracy and F-1 scores.

- **Diverse Training Modalities:** The experiment is designed to explore various training modes to develop a representative digital vector for different types of network traffic. Through difference characters of supervised and unsupervised learning, the proposed model enhances the ability to generalize across various situations, ensuring its effectiveness regardless of the various context.

- **Model Generalization by Across Datasets:** By evaluating the model's performance on a wide range of datasets, in other words, I am going to use the same model on different dataset to prove the robustness and flexibility of the models. This addresses a critical challenge in applying machine learning and deep learning technologies in network environments, focusing on adapting to new and unseen data effectively.

The rest of report is organized as follows: **Section 3** will talk about the dataset I use and some results researchers have done in applying deep learning methods for Intrusion detection. **Section 4** is dedicated to detailing the preprocessing steps required to adapt raw data formats for machine learning applications, **Section 5** followed by an in-depth discussion of the methodologies and technologies implemented in this project. The **Section 6** provides a detailed description of the experiments design and training procedures used. The **Section 7** give the conclude with an analysis of the experimental results.

3. Related work

3.1 IDS dataset

Sharafaldin and colleagues[3] at the Canadian Institute of Cybersecurity developed a new IDS dataset known as CIC-IDS, which incorporates the most prevalent attack types alongside genuine benign traffic, without concerns of privacy or ethics. In collaboration with the Communications Security Establishment (CSE), the subsequent datasets—CIC-IDS2017 and CSE-CIC-IDS2018—expanded on the earlier ISCXIDS2012 by introducing additional types of attacks, including Quadratic Discriminate, *Slowloris*, *Hulk*, *Goldeneye*, *LOIT*, *Heartbleed*, and various forms of *LOIC (UDP, TCP, HTTP)*, as detailed in Table1 of the Appendix. The ISCXIDS 2012 dataset initially categorized four types of intrusion attacks, whereas CIC-IDS2017 broadened to eight categories with twelve sub-categories, and CSE-CIC-IDS2018 included six categories with fourteen sub-categories. Given the more extensive array of network intrusion types and the more recent publication dates of CIC-IDS2017 and CSE-CIC-IDS2018, alongside the substantial data volume of CSE-CIC-IDS2018, reaching up to around 445GB, this paper's experiments focus on these two datasets with experiments.

3.2 Deep-learning based models

Lan et al., 2022 [4] present a novel structure combined CNN and self-attention with high accuracy on CIC-IDS2017. Xie et al., 2022[5] developed a packet-level online network flow classification technique utilizing a self-attention mechanism. Taking advantage of balanced dataset, it reaches a high score both in F-1 and accuracy. Li et al., 2022[6] incorporates a distribution dynamic adjustment mechanism at the flow level and a multiscale spatial mapping mechanism, combing with an adjustment in both hyperparameter and loss function, reaching an incredible score on test dataset. Yu et al., 2021[1] indicate the importance of byte-based information of network packets and construct a multi-layer CNN model to extract the information.

4. Data preprocess

Data preprocessing serves two main goals: removing redundant data to prevent interference with feature learning, and transforming network flows into a format suitable for a feature extraction model. This involves anonymizing unnecessary data, vectorizing strings, and standardizing the sample's bytes, packets, and flows.

Parsing Network Traffic: The initial step bypasses pre-processed CSV files in favor of raw Pcap files, which provide more granular data including nanosecond-precision timestamps. Each packet within the Pcap files is parsed to extract crucial data fields according to network protocols. (For example, in CIC-IDS2018, according to the time span and attack ip address provided by official, I divided the whole Pcap into attack Pcap and benign Pcap)

Organizing Network Traffic Flows: Utilizing the concept of "bi-flows" or sessions to organize the data, where network packets are grouped based on quintuple data (source IP and port, destination IP and port, protocol). Sessions are segmented using a combination of flags from TCP connections and timeout strategies to manage data continuity and integrity.

Session Segmentation Strategies: Different strategies for session segmentation are evaluated, including using TCP flags and timeout-based segmentation. The timeout strategy is straightforward and easy to implement. It operates under the assumption that a session has concluded if there has been a significant lapse in data exchange. Consequently, this approach serves as a key criterion for segmenting sessions within this research. The threshold for determining session inactivity is aligned with previous studies and is established at 64 seconds.

Data Transformation: The transformation of raw data into a format suitable for input into deep learning models. This includes anonymizing sensitive information, encoding data into a numerical format, and normalizing data to ensure effective model training. Lastly, raw data in byte form is converted into numerical values from 0 to 255, suitable for model. The model limits the number of flows, packets per flow, and bytes per packet to maintain effective filtering and achieve optimal accuracy, specifically with configurations of, 20 packets per flow, and 256 bytes per packet. This procedure of preprocessing original Pcap files strictly follow the paper Yu et al., 2021[1].

5. Methodology

In this section, I will overview the methodologies employed in this project, covering aspects from model training and components to loss functions and evaluation metrics.

The model utilizes three training modes: a basic Encoder-Decoder structure for direct target prediction, a Variational Auto-Encoder (VAE) that learns distributions over latent spaces to generalize complex traffic patterns, and Constructive Learning that enhances class differentiation by diversifying feature representations in the encoder. Key components include Convolutional Neural Networks (CNNs) for robust feature extraction from network traffic, and an attention mechanism that prioritizes critical features to improve detection accuracy.

5.1 Activation function

A. ReLU (Rectified Linear Unit) Activation Function

ReLU is a piecewise linear function that outputs the input directly if it is positive; otherwise, it will output zero. It is defined mathematically as:

- $\text{ReLU}(x) = \max(0, x)$

B. Softmax Activation Function

Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the exponentials of the input numbers. It is defined as:

- $Softmax\ (z_i) = e^{z_i} / \sum_j e^{z_j}$

Softmax is typically used in the output layer of a classifier to provide a probabilistic interpretation of class membership.

5.2 Batch Normalization (BatchNorm)

Batch normalization is a method to normalize the activations of a layer across the current mini-batch. It adjusts and scales the activations to have a mean of zero and a standard deviation of one, which are then scaled and shifted by two learnable parameters per feature. The transformation is defined as follows:

$$\text{BN}\ (x_i) = \gamma\ ((x_i - \mu_B)/(\sigma_B^2 + \epsilon)^{0.5}) + \beta$$

where $xixi$ are the inputs, μ_B and σ_B^2 are the mean and variance computed over the mini-batch, γ and β are parameters to be learned, and ϵ is a small constant added for numerical stability.

Batch normalization is applied as a layer within neural networks between the linear (or convolutional) layer and its activation function.

5.3 Model Components

In a single network flow, the feature extraction process has three components: Spatially Bytes-Encoded extraction, dynamic feature adjustment and Spatially Packet-Encoded extraction. In this report, three neural networks, namely, a CNN, a self-attention mechanism network and another CNN, are used to implement the three components of the network flow feature extraction process. The general process is as follows: after a single flow X is preprocessed, n_p packets in flow X have the same length n_b, the matrix of $n_p \times n_b$ is then go through the one-hot encoding which mined for spatial consistency features by the CNN, with the number of input sequences n_b kept constant by padding or truncate. Then through applying self-attention mechanism network on X, the representative of byte-encoded will be refine. Finally, the Packet level representative will be got from another CNN which used as our final feature.

A. Byte-encoded extraction (CNNs)

The initial layer receives the one-hot encoded byte data with 256 channels (corresponding to the 256 possible byte values in a one-hot encoded format). Subsequent layers decrease the channel depth from 256 to 8 in the first convolution layer, then converting 8 to 16 keeping the other same and finally 16 to 32.

Each convolution layer employs a kernel size of 3, stride of 1, and padding of 1 to ensure the spatial dimensions are preserved after each convolution operation, allowing for a continuous and stable feature extraction without loss of border data. Each convolution operation is followed by a ReLU (Rectified Linear Unit) activation function which introduces non-linearity to the model, helping it to learn more complex patterns in the data. Following the ReLU activation, batch normalization is applied to stabilize the learning process by normalizing the output of the previous layer. This speeds up training and reduces the sensitivity to network initialization. A dropout rate of 35% is included after batch normalization to prevent overfitting by randomly omitting a subset of features during training. A max pooling layer with a kernel size of 2 and a stride of 2 follows the convolution layers. Max pooling is used to reduce the dimensionality of the data, which helps in decreasing the computational cost and also in extracting dominant features that are robust to small variations in the input data. The whole procedure of byte-encoded extraction is showed in figure1.

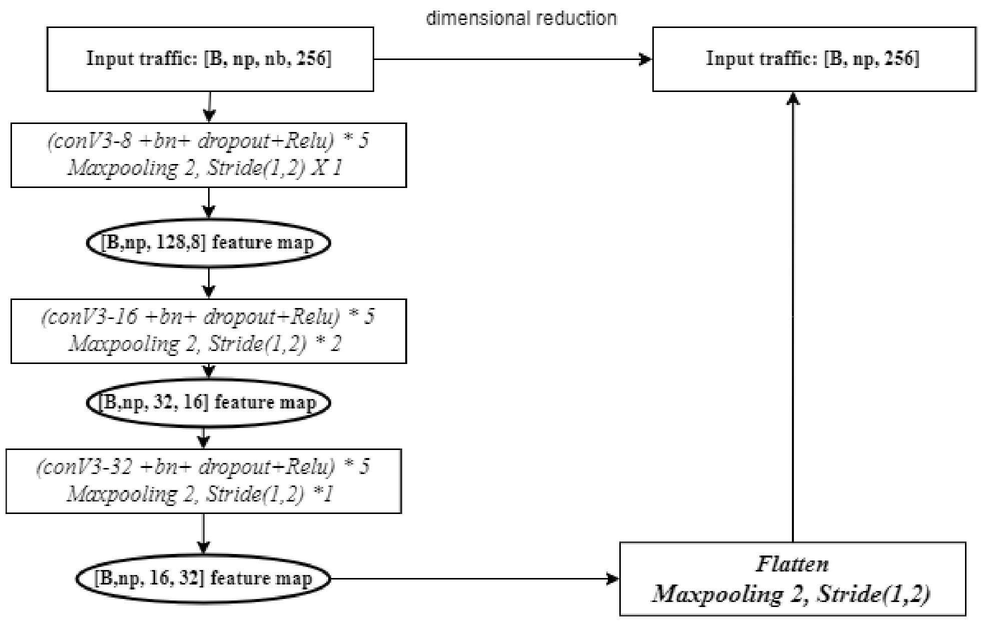

dimensional reduction
Input traffic: [B, np, nb, 256]
Input traffic: [B, np, 256]
(conV3-8 +bn+ dropout+Relu) * 5
Maxpooling 2, Stride(1,2) X 1
[B,np, 128,8] feature map
(conV3-16 +bn+ dropout+Relu) * 5
Maxpooling 2, Stride(1,2) * 2
[B,np, 32, 16] feature map
(conV3-32 +bn+ dropout+Relu) * 5
Maxpooling 2, Stride(1,2) *1
[B,np, 16, 32] feature map
Flatten
Maxpooling 2, Stride(1,2)

Figure 1. B: batch size, n_p: packet length, n_b: byte length

B. Packet-level Self-attention Mechanism

Given an input tensor X of shape [batch-size, n_p, 256] where n_p is the number of packets and 256 represents the feature dimension per packet, it is linearly projected to form queries (Q), keys (K), and values (V) using parameter matrices W^Q, W^K, W^V respectively. Each of these matrices transforms the input to facilitate a different aspect of the attention mechanism.

- $Q = X'W^Q$
- $K = X'W^K$
- $V = X'W^V$

Where W^Q, W^K, and W^V are weight matrices of dimensions [256, d_q], [256, d_k], and [256, d_v] respectively, with d_k and d_v being the dimensions for keys/queries and values.

Compute the dot product of Q and the transpose of K to determine the alignment scores between different packets, then scale by the square root of d_k to stabilize gradients during training.

- $A = \mathit{softmax}\ (QK^T/d_k{}^{0.5})$

Multiply the attention scores A with V to get the output of the attention mechanism. This operation aggregates information from different positions, weighted by the computed attention scores.

- $Z=AV$

In practice, the attention mechanism is applied multiple times in parallel to the input, allowing the model to jointly attend to information from different representational subspaces at different positions. Let's denote each attention head's output as Z_i for head *i*. Concatenate the outputs from all heads and project it once more to return to the original feature space size: $Z_{multi\text{-}head} = \text{concat}\ (Z_1, Z_2,..., Z_h)\ W^O$. Where W^O is the final linear transformation applied to the concatenated output of all heads, and h is the number of heads. The output shape will remain the same as X, but get different number each position. More details about self-attention can be found in Vaswani, Ashish, et al., 2017[7].

C. Packet-encoded extraction (CNNs)

Given the input X which has aggregate the information among sequence, a similar structure has been applied to this level of data. Since the previous two layers of model has been extracting so many information, this CNN is going to use relatively low layers and Maxpooling to reduce dimension for the final representative vector. First Convolutional Layer takes the input with 256 channels and applies a convolution with 128 output channels using a kernel size of 3 and padding of 1. This convolution helps to capture local dependencies and reduces the feature dimension. Second Convolutional Layer: Following the first convolution and activation sequence, a second convolutional layer further processes the data, reducing the channels from 128 to 64. This step enhances the ability to capture more abstract features in the data. Finally flatten the last two dimensional of the data and connected to Fully Connected Layers to get a 128-dimensional latent representative in my model. Each convolutional layer is combined with ReLU activation function, batch normalization and dropout.

5.4 Evaluation matrix

To quantify the effectiveness of the intrusion detection model, we computed several key performance metrics:

- **True Positives (TP):** True Positives are the cases where the model correctly predicts the positive class. In the context of an intrusion detection system, a TP would be correct predictions of an actual attack.
- **True Negatives (TN):** True Negatives occur when the model correctly predicts the negative class. For an intrusion detection system, this would be the Correct predictions where no attack is present.

- **False Positives (FP):** False Positives occur when the model incorrectly predicts the positive class. In the context of an intrusion detection system, an FP would be incorrect predictions that an attack has occurred when it hasn't.
- **False Negatives (FN):** False Negatives happen when the model fails to detect the positive class. For an intrusion detection system, an FN would be incorrect predictions where an actual attack is missed.

Accuracy: Measures the overall correctness of the model and is defined as the ratio of correctly predicted observations (both true positives and true negatives) to the total observations:

- Accuracy = $(TP+TN)/(TP+FN+FP+TN)$

Precision measures the accuracy of positive predictions. Formulated as the ratio of true positives to all positive predictions:

- Precision = $TP/(TP+FP)$

Recall (or Sensitivity) measures the ability of the model to find all the relevant cases (positives). It is defined as the ratio of true positives to the actual number of positives:

- Recall = $TP/(TP+FN)$

F1-Score: The F1-score is the harmonic mean of precision and recall, offering a balance between the two when an equal weight is of concern:

- F1-Score = 2·Precision·Recall / (Precision + Recall)

Weighted F1-Score: Particularly useful in scenarios of class imbalance, the weighted F1-score calculates metrics for each label, and finds their average weighted by support (the number of true instances for each label). This adjusts the score to account for the influence of each class:

- Weighted F1-Score = $\sum_i^l (w_i \cdot \text{F1-Score}_i)$

where w_i is the proportion of the number of samples in class i and F1-Score$_i$ is the F1-score for class i. In my experiments, accuracy, F-1 score and weighted F-1 scores are the evaluation performance of model.

5.5 Variational Autoencoder (VAE)

The objective of a VAE is to maximize the likelihood of observing the training data, but directly computing this likelihood can be computationally infeasible. Instead, VAEs optimize the ELBO, which is a lower bound to the log likelihood of the observed data. Here's how ELBO is formulated mathematically:

$$\text{ELBO} = \text{E}q_\phi\ (z|x)\ [\log_{p\theta}(x|z)] - \text{KL}\ (q_\phi(z|x)\|p(z))$$

Where:

- $q_\phi(z|x)$ is the approximate posterior distribution (encoded by the encoder) of the latent variables z given the input x, parameterized by ϕ and here refers to the encoder of my model.
- $p_\theta(x|z)$ is the likelihood of the data x given the latent variables z, modeled by the decoder and parameterized by θ and here refers to the decoder of my model.
- $p(z)$ is the prior distribution over the latent variables, typically assumed to be a standard normal distribution $N\ (0, I)$.
- KL denotes the Kullback-Leibler divergence, which measures how one probability distribution diverges from another probability distribution.

$\mathrm{E}q_{\phi}(z|x)[\log_{p\theta}(x|z)]$: This term is the expected log likelihood of the data given the latent variables. It encourages the decoder to accurately reconstruct the input data from the latent variables. It is often computed as the negative reconstruction loss (e.g., negative mean squared error for continuous data or cross-entropy for binary data).

$\mathrm{KL}(q_{\phi}(z|x)\|p(z))$: The KL divergence between the approximate posterior $q_{\phi}(z|x)$ and the prior $p(z)$. This term acts as a punishment, ensuring that the distribution of the latent variables doesn't deviate too much from the prior. In brief, it prevents the model from making the variance of the latent distribution approaching to 0 for better accuracy. For a standard Gaussian prior, this term can be analytically solved as:

$$\mathrm{KL}(q_{\phi}(z|x)\|p(z)) = ½ \sum_{j=1}^{J}(1+\log((\sigma_j^2))-\mu_j^2-\sigma_j^2)$$

Here, μ and σ are the mean and standard deviation of the Gaussian distribution $q_{\phi}(z|x)$, and J is the dimensionality of the latent space.

The goal in training a VAE is to maximize the ELBO. Since most optimization algorithms are designed to minimize rather than maximize, it's common to minimize the negative ELBO:

$$\min -\mathrm{ELBO} = \min\left(- \mathrm{E}q_{\phi}(z|x)[\log_{p\theta}(x|z)] + \mathrm{KL}(q_{\phi}(z|x)\|p(z))\right)$$

This formulation ensures that while the model learns to encode the input data into a compact representation (latent space) that captures the essential features of the data, it also regularizes these representations to not stray too far from a simple Gaussian, promoting robustness and generalization. More details can be found in Kingma et al., 2013[8]

6. Experiments

6.1 Experiments set up

In this study, the CSE-CIC-IDS2018 dataset was utilized to be the pre-trained model, with a particular focus on the encoder component. By splitting the dataset equally, I used the first 50% for training and the remaining 50% for evaluating the model's intrusion detection capabilities. The robustness of the encoder was further tested by applying its learned representations to a different dataset, where a new classifier was re-trained using these features. This approach tries to demonstrate that the encoder's ability to extract meaningful and generalizable features that are not only effective on the original dataset but also adaptable to other datasets with minimal data requirements. This proves the potential of using our pre-trained model to fine-tune or construct new classifiers on different datasets, significantly reducing the need for large amounts of training data and accelerating the deployment of effective intrusion detection systems across various network environments. Specially, I use 20% of the CICIDS-2017 as training dataset and the rest to test the performance of models. The figure 2 shows the whole procedure of training and evaluation.

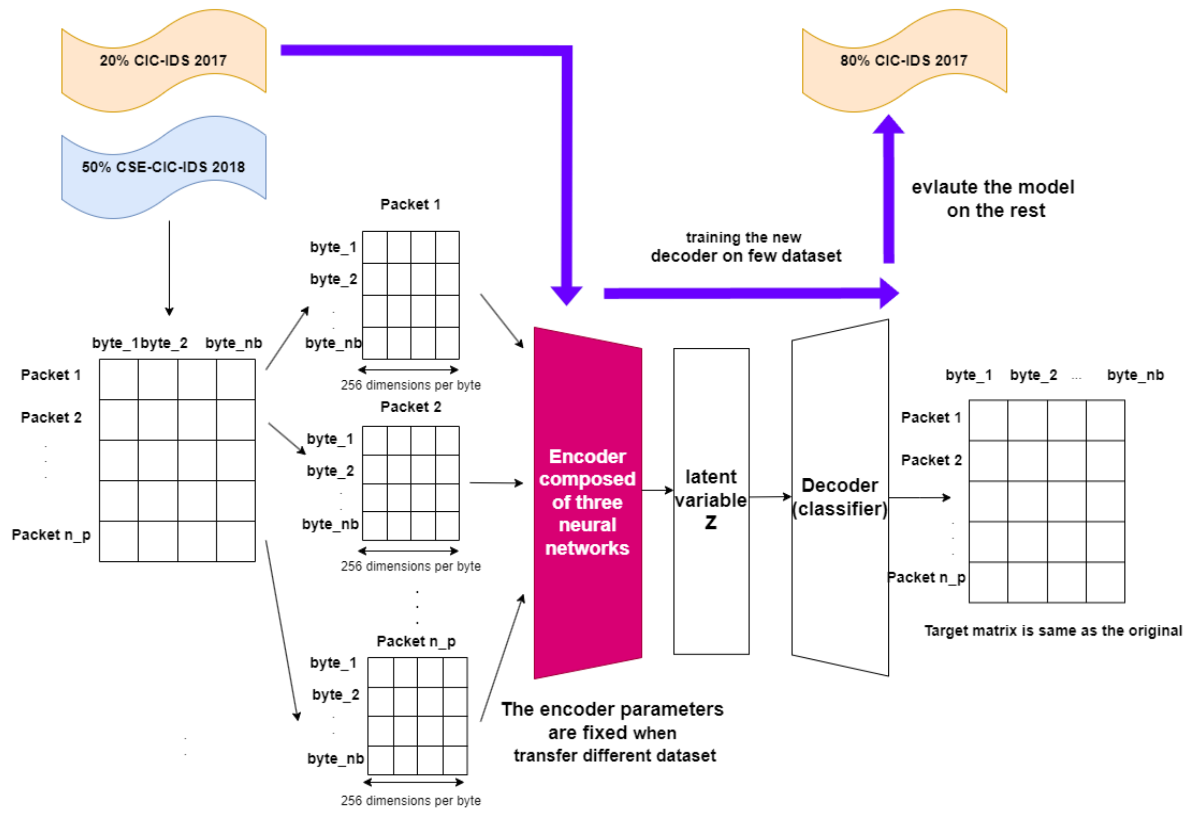

20% CIC-IDS 2017
50% CSE-CIC-IDS 2018
80% CIC-IDS 2017
evlaute the model on the rest
training the new decoder on few dataset
Packet 1
Packet 2
Packet n_p
byte_1
byte_2
byte_nb
256 dimensions per byte
Encoder composed of three neural networks
latent variable z
Decoder (classifier)
Target matrix is same as the original
The encoder parameters are fixed when transfer different dataset

Figure 2 training and evaluation procedure

7. Final results

In my experiments, I implemented two distinct training approaches using the CSE-CIC-IDS2018 dataset:

Encoder-Decoder Structure:

- Training: I trained the encoder-decoder model using 30% of the CSE-CIC-IDS2018 dataset.
- Performance: The model achieved high performance on the entire dataset, reaching 98% accuracy and an F1-score of 0.82.
- Transferability: However, when the encoder was transferred and applied to the CIC-IDS2017 dataset, the performance dropped, achieving only 86% accuracy and a lower F1-score, indicating limited transferability.

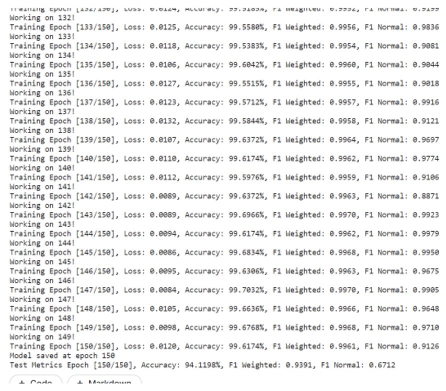
Variational Autoencoder (VAE):

- Training Setup: For the VAE model, I utilized a larger portion of the dataset, training with 20% of the CSE-CIC-IDS2018 dataset.
- Evaluation: The model was then tested on the remaining 80% of the data.

- Results: This approach yielded relatively high accuracy and good F1-scores, demonstrating the VAE's effectiveness in capturing and generalizing the underlying data distribution well across a larger dataset split.

These results underscore the importance of model selection and dataset proportioning in achieving optimal performance and transferability. The encoder-decoder model, while highly effective within its original dataset, shows limitations when applied to external data, suggesting a need for further tuning or a different model architecture, such as the VAE, which demonstrated better generalization capabilities.

Since the severe imbalance of CIC-IDS2017, the weighted F-1 score has relatively good scores to prove the model has certain generality. The following screenshot proves my experiments procedure. (there should be more visualization on checking which kind of attack has been misclassified, the time limitations)



```
Training Epoch [132/150], Loss: 0.0124, Accuracy: 99.5185%, F1 Weighted: 0.9952, F1 Normal: 0.9199
Working on 132!
Training Epoch [133/150], Loss: 0.0125, Accuracy: 99.5580%, F1 Weighted: 0.9956, F1 Normal: 0.9836
Working on 133!
Training Epoch [134/150], Loss: 0.0118, Accuracy: 99.5383%, F1 Weighted: 0.9954, F1 Normal: 0.9081
Working on 134!
Training Epoch [135/150], Loss: 0.0106, Accuracy: 99.6042%, F1 Weighted: 0.9960, F1 Normal: 0.9044
Working on 135!
Training Epoch [136/150], Loss: 0.0127, Accuracy: 99.5515%, F1 Weighted: 0.9955, F1 Normal: 0.9018
Working on 136!
Training Epoch [137/150], Loss: 0.0123, Accuracy: 99.5712%, F1 Weighted: 0.9957, F1 Normal: 0.9916
Working on 137!
Training Epoch [138/150], Loss: 0.0132, Accuracy: 99.5844%, F1 Weighted: 0.9958, F1 Normal: 0.9121
Working on 138!
Training Epoch [139/150], Loss: 0.0107, Accuracy: 99.6372%, F1 Weighted: 0.9964, F1 Normal: 0.9697
Working on 139!
Training Epoch [140/150], Loss: 0.0110, Accuracy: 99.6174%, F1 Weighted: 0.9962, F1 Normal: 0.9774
Working on 140!
Training Epoch [141/150], Loss: 0.0112, Accuracy: 99.5976%, F1 Weighted: 0.9959, F1 Normal: 0.9106
Working on 141!
Training Epoch [142/150], Loss: 0.0089, Accuracy: 99.6372%, F1 Weighted: 0.9963, F1 Normal: 0.8871
Working on 142!
Training Epoch [143/150], Loss: 0.0089, Accuracy: 99.6966%, F1 Weighted: 0.9970, F1 Normal: 0.9923
Working on 143!
Training Epoch [144/150], Loss: 0.0094, Accuracy: 99.6174%, F1 Weighted: 0.9962, F1 Normal: 0.9979
Working on 144!
Training Epoch [145/150], Loss: 0.0086, Accuracy: 99.6834%, F1 Weighted: 0.9968, F1 Normal: 0.9950
Working on 145!
Training Epoch [146/150], Loss: 0.0095, Accuracy: 99.6306%, F1 Weighted: 0.9963, F1 Normal: 0.9675
Working on 146!
Training Epoch [147/150], Loss: 0.0084, Accuracy: 99.7032%, F1 Weighted: 0.9970, F1 Normal: 0.9905
Working on 147!
Training Epoch [148/150], Loss: 0.0105, Accuracy: 99.6636%, F1 Weighted: 0.9966, F1 Normal: 0.9648
Working on 148!
Training Epoch [149/150], Loss: 0.0098, Accuracy: 99.6768%, F1 Weighted: 0.9968, F1 Normal: 0.9710
Working on 149!
Training Epoch [150/150], Loss: 0.0120, Accuracy: 99.6174%, F1 Weighted: 0.9961, F1 Normal: 0.9126
Model saved at epoch 150
Test Metrics Epoch [150/150], Accuracy: 94.1198%, F1 Weighted: 0.9391, F1 Normal: 0.6712
```

8. Conclusion

In the process of training the models, I discovered that achieving high accuracy and F1-scores on the training dataset was straightforward, but these metrics significantly dropped on the testing dataset, indicating overfitting when using a complex Encoder-Decoder structure. To address this, I revised the model architecture by incorporating more dropout, applying batch normalization techniques, and simplifying the model structure. These modifications allowed the model to still achieve high scores even with minimal data usage. The test results suggest that the packet data exhibits a distinct and clear distribution.

However, simplifying the model also reduced its ability to capture the data distribution effectively. The Variational Autoencoder (VAE), which uses data itself as the target, addresses this issue by maintaining robustness against overfitting even as I added more layers in self-attention and CNN, demonstrating good performance on different datasets.

I believe the relative success of my model can be attributed to several factors:

- I uses all the class as target that I can go through once to get a good result. However, in some other researches paper, for example in [1] , they separate the dataset with imbalanced category attack and use few-slot to do another classification on these imbalanced categories, thus combine them and getting a high score both in F-1 and accuracy.

- Only using few of dataset but reaching relatively high score.
- Transferring the model on different dataset to evaluate, which taken the idea from the bert[9] to get a pre-trained model that can be applied on all packet data.

Given the strong inherent patterns within the data, which are well-suited for exploitation by machine learning models, there is potential to leverage statistical data augmentation techniques to enhance the stability and reliability of the results. Due to time constraints, I utilized only 50% of the CSE-CIC-IDS 2018 dataset for training the model. With the implementation of data augmentation technologies to enrich and balance the dataset, there is a significant opportunity to boost the model's performance. Such enhancements would make the model more robust and better prepared for further fine-tuning or for application in classification tasks across more diverse scenarios.

9. References

[1] Lian Yu, Jingtao Dong, Lihao Chen, Mengyuan Li, Bingfeng Xu, Zhao Li, Lin Qiao, Lijun Liu, Bei Zhao, Chen Zhang, PBCNN: Packet Bytes-based Convolutional Neural Network for Network Intrusion Detection, Computer Networks, Volume 194,2021,108117, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.108117.

[2] Kim, Jiyeon & Kim, Jiwon & Kim, Hyunjung & Shim, Minsun & Choi, Eunjung. (2020). CNN-Based Network Intrusion Detection against Denial-of-Service Attacks. Electronics. 9. 916. 10.3390/electronics9060916.

[3] Sharafaldin, I., Habibi Lashkari, A., & Ghorbani, A.A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *International Conference on Information Systems Security and Privacy*.

[4] J. Lan, Y. Li, B. Li and X. Liu, "MATTER: A Multi-Level Attention-Enhanced Representation Learning Model for Network Intrusion Detection," 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Wuhan, China, 2022, pp. 111-116, doi: 10.1109/TrustCom56396.2022.00026.

[5] Guorui Xie, Qing Li, Yong Jiang, Self-attentive deep learning method for online traffic classification and its interpretability, Computer Networks, Volume 196, 2021, 108267, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.108267.

[6] Yanan Li, Tao Qin, Yongzhong Huang, Jinghong Lan, ZanHao Liang, Tongtong Geng, HDFEF: A hierarchical and dynamic feature extraction framework for intrusion detection systems, Computers & Security, Volume 121, 2022, 102842, ISSN 0167-4048.

[7] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[8] Kingma D P, Welling M. Auto-encoding variational bayes[J]. arXiv preprint arXiv:1312.6114, 2013.

[9] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.