Intrusion Detection system with GAN to address class imbalance issues.

Table of Contents:

Special thanks to BITS Examiner Mithun, for his valuable feedback and guidance.

List of Acronyms and Abbreviations

Abbreviations	Meaning
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
IDS	Intrusion Detection System
DDos	Distributed Denial of Service
SMOTE	Synthetic Minority Oversampling Technique

List of Figures

List of Tables

Introduction

Problem Statement:

An Intrusion Detection System (IDS) monitors the network traffic for malicious traffic and alerts the administrator. Its mainly used for detecting vulnerable exploits against target application or hardware which includes highly available servers such as bank servers which handle transactions. There are different types of attacks possible these days such as DDoS, Neptune (SYN flood) - which causes unavailability of servers due to high traffic flooding the systems. With the advancement and usage of internet across, detecting malicious traffic is still being a challenge as new types of attacks are identified each day. According to recent studies, about 4000 attacks occur each day despite having security infrastructure available. An effective IDS is still being a necessary research topic, considering the pace in which world is shifting to cloud based from on-premise technologies.

Existing work and Challenges:

There has been a significant amount of work with respect to IDS with Neural networks and other machine learning algorithms. Due to the large variety of network attacks, class imbalance problems have always been an issue when it comes to training the model. This leads to overfitting issues and hence the model fails in detecting the actual attack traffic. There are traditional techniques for class imbalance problems which uses random oversampling to substitute random samples for minority class, SMOTE. These are proven to provide better results than just classification, however there is definitely more scope of improvement.

Objective:

Generative modeling is popular these days in Machine learning, which involves training a model to produce new data like the given dataset. And its counterpart Discriminative model aims in identifying/classifying the samples. A GAN is a model in which both Generator and Discriminator models compete with each other to and comes up with samples that is similar to the real dataset.

In this work, we have used GAN for encountering the overfitting problem, by introducing generated samples similar to the actual samples from the dataset. Initially CNN is applied on the NSL-KDD dataset, followed by CNN after SMOTE. Later GAN is used for generating samples which are added to the data-set before feeding into CNN.

Proposed Model and Dataset:

Model:

Three Different approaches are taken in this work and their results are compared and GAN model provides better accuracy than the other 2 models. The primary model is just with CNN, secondary model uses SMOTE before CNN and final model has GAN before CNN.

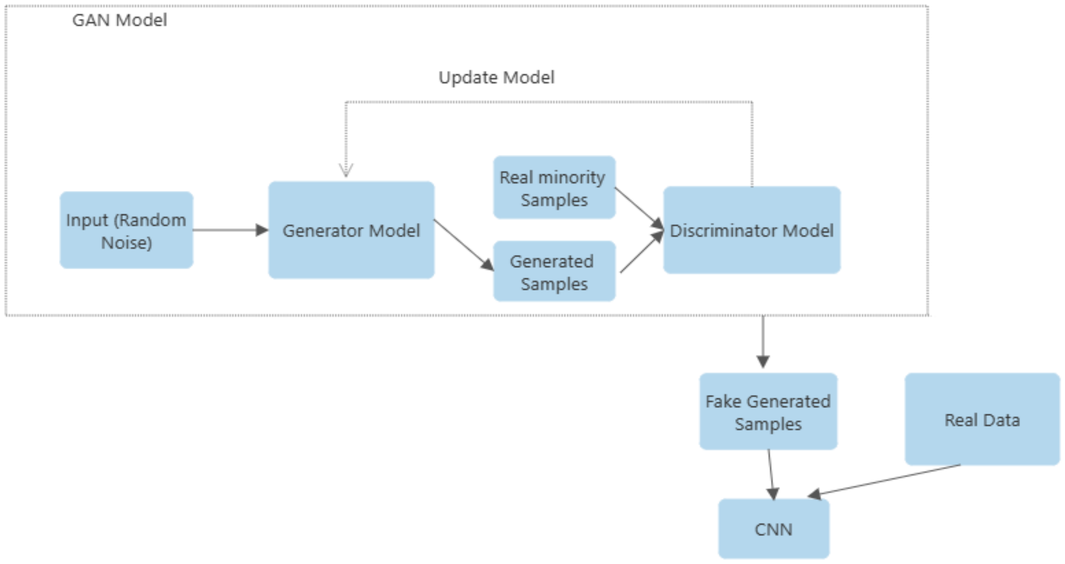

Fig 2.1 GAN Model For Intrusion Detection

Dataset overview:

NSL-KDD dataset has been used in this work and it is one of the most popular dataset for IDS. There are 42 columns in the dataset with 41 features and a label column which labels the type of traffic as attack or normal. There are 126973 samples in the training dataset and 22544 samples in the test set. The available feature columns include,

```
duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell', 'su_attempted', 'num_root',
'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds', 'is_host_login',
'is_guest_login', 'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate',
'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
'dst_host_srv_rerror_rate'
```

There are multiple types of attacks as well as mentioned below.

Attack Type	Count
normal	67343
neptune	41214
satan	3633
ipsweep	3599
portsweep	2931
smurf	2646
nmap	1493
back	956
teardrop	892
warezclient	890
pod	201
guess_passwd	53
buffer_overflow	30
warezmaster	20
land	18
imap	11
rootkit	10
loadmodule	9
ftp_write	8
multihop	7
phf	4
perl	3
spy	2

Table 2.1 Types of attacks in the dataset

We have altered the data to have normal and attack, hence it's a binary classification problem rather than multi-label classification problem.

There is a separate test and training dataset and here we have used training set for training the model and test set for evaluation purpose. Hence the model has not seen a test set before hand for training.

Experiment and Analysis

Data Pre-processing

Missing Values Check:

There are no missing values or NAN in the dataset. Hence no imputation techniques are required. Only alteration that has been made is for the attack column values to make it binary classification problem.

Handling categorical variables:

There are three categorical features, protocol_type, service, flag. Label_Encoder from sklearn has been used to substitute the same with numbers. This has been applied to training and test set.

Normalization:

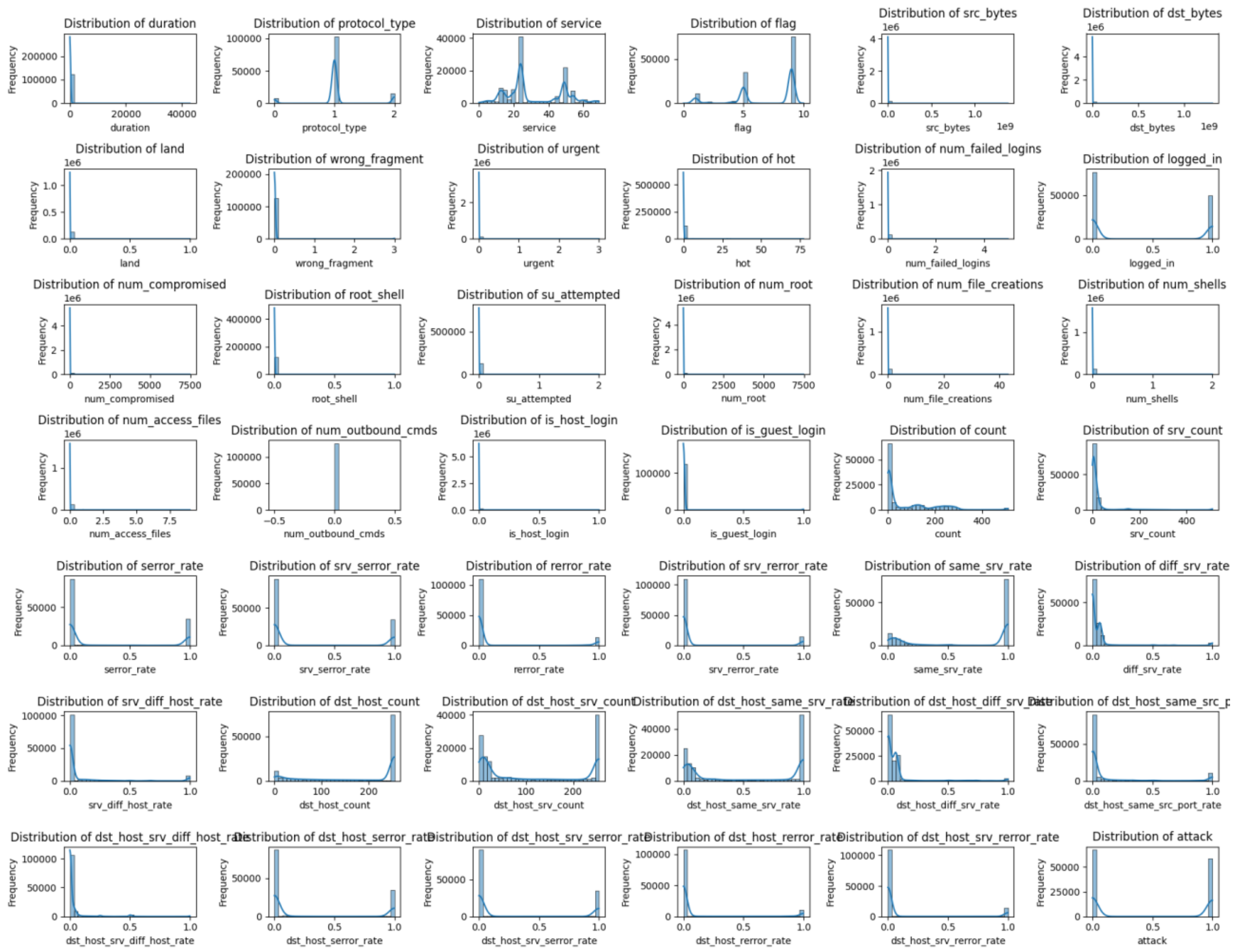

Fig: 3.1 Data distribution range.

The data distribution has been across with different scales. Columns like

dst_host_srv_count has values ranging between 0 to 200, however other attributes like dst_host_rerror_rate ranges between 0 to 1.

Hence normalization is required. We have used Standard Scaler from sklearn preprocessing module to normalize the data distribution.

Basic Classification with CNN:

Preprocessed data has been fed into a CNN model. The CNN model has 2 hidden layer with relu used as activation function. Sigmoid is used as an activation function for the output layer as its for binary classification. Adam optimizer is used, with loss function as binary cross-entropy. The model summary is provided below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)              Output Shape              Param #
=================================================================
 dense (Dense)              (None, 64)               2688

 dense_1 (Dense)            (None, 32)               2080

 dense_2 (Dense)            (None, 1)                33

=================================================================
Total params: 4801 (18.75 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 0 (0.00 Byte)
```

Accuracy and loss during training and test dataset

Metric	Training data	Test Data
Accuracy	0.9957	0.7837
Loss	0.0128	0.7988

Table 3.1 CNN metrics

The accuracy is low for test dataset due to overfitting.

Applying SMOTE for oversampling:

SMOTE technique has been used to add generated samples of minority class and the same data is fed into CNN with the same model parameters. Here we notice some improvement in accuracy of the test data but not very significantly.

Metric	Training data	Test Data
Accuracy	0.9968	0.8054
Loss	0.0122	1.5632

Table 3.2 SMOTE with CNN metrics

To improve on the same, we are introducing GAN to generate samples and the same is fed into neural network.

GAN for oversampling:

As discussed earlier, GAN has two models combined, Generator and Discriminator. Generator module is used to generate fake samples with random input vector and discriminator will classify the same as fake and hence improves the samples generator makes to be like the actual samples. Here samples from minority class alone is taken into

Generator Model:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)             Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)       [(None, 100)]             0         []

 dense (Dense)              (None, 8)                 808       ['input_1[0][0]']

 re_lu (ReLU)               (None, 8)                 0         ['dense[0][0]']

 dense_1 (Dense)            (None, 16)                144       ['re_lu[0][0]']

 re_lu_1 (ReLU)             (None, 16)                0         ['dense_1[0][0]']

 dense_2 (Dense)            (None, 32)                544       ['re_lu_1[0][0]']

 dense_7 (Dense)            (None, 64)                1088      ['re_lu_1[0][0]']

 re_lu_2 (ReLU)             (None, 32)                0         ['dense_2[0][0]']

 re_lu_4 (ReLU)             (None, 64)                0         ['dense_7[0][0]']

 dense_3 (Dense)            (None, 64)                2112      ['re_lu_2[0][0]']

 dense_8 (Dense)            (None, 128)               8320      ['re_lu_4[0][0]']

 re_lu_3 (ReLU)             (None, 64)                0         ['dense_3[0][0]']

 re_lu_5 (ReLU)             (None, 128)               0         ['dense_8[0][0]']

 dense_4 (Dense)            (None, 3)                 195       ['re_lu_3[0][0]']
 dense_5 (Dense)            (None, 66)                 4290      ['re_lu_3[0][0]']

 dense_6 (Dense)            (None, 11)                 715       ['re_lu_3[0][0]']

 dense_9 (Dense)            (None, 38)                 4902      ['re_lu_5[0][0]']

 concatenate (Concatenate)   (None, 118)                0        ['dense_4[0][0]',
                                                                  'dense_5[0][0]',
                                                                  'dense_6[0][0]',
                                                                  'dense_9[0][0]']

==========================================================================
====================
Total params: 23118 (90.30 KB)
Trainable params: 23118 (90.30 KB)
Non-trainable params: 0 (0.00 Byte)
```

There have been two different neural networks created one for categorical and other for numerical. For categorical columns, softmax is used as activation function in output layer and this is because these are not binary classes. Each category has more than 2 unique values. For numerical columns, sigmoid activation function is used in the output layer. Hidden layer has Relu as an activation function.

Discriminator Model:

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 118)]             0

 dense_10 (Dense)            (None, 128)               15232

 re_lu_6 (ReLU)              (None, 128)               0

 batch_normalization (Batch  (None, 128)               512
 Normalization)

 dense_11 (Dense)            (None, 64)                8256

 re_lu_7 (ReLU)              (None, 64)                0

 batch_normalization_1 (Bat  (None, 64)                256
 chNormalization)

 dense_12 (Dense)            (None, 32)                2080

 re_lu_8 (ReLU)              (None, 32)                0
 batch_normalization_2 (Bat  (None, 32)              128
 chNormalization)

 dense_13 (Dense)           (None, 16)             528

 re_lu_9 (ReLU)            (None, 16)            0

 batch_normalization_3 (Bat  (None, 16)             64
 chNormalization)

 dense_14 (Dense)           (None, 8)             136

 re_lu_10 (ReLU)           (None, 8)             0

 batch_normalization_4 (Bat  (None, 8)             32
 chNormalization)

 dense_15 (Dense)           (None, 1)             9

=================================================================
Total params: 27233 (106.38 KB)
Trainable params: 26737 (104.44 KB)
Non-trainable params: 496 (1.94 KB)
_________________________________________________________________
```

Discriminator has 5 hidden layer with relu as activation function and output layer with sigmoid as activation function. Further binary_crossentrophy and adam are used as loss function and optimizer. Batch Normalization has been applied in the hidden layer to stabilize the training process of deep neural networks.

GAN with CNN:

Combining generator and discriminator and training and feeding one at a time will be the effective GAN model. The generator loss should decrease over time as it produces realistic data. Discriminator loss also should decrease over time as they differentiate the samples well. An equilibrium of 0.5 is ideal for perfect GAN model.

The GAN model ‘s loss values are after 7000 epochs

Generator Loss	0.693173
Discriminator Loss	0.693160

Table 3.3 GAN loss values

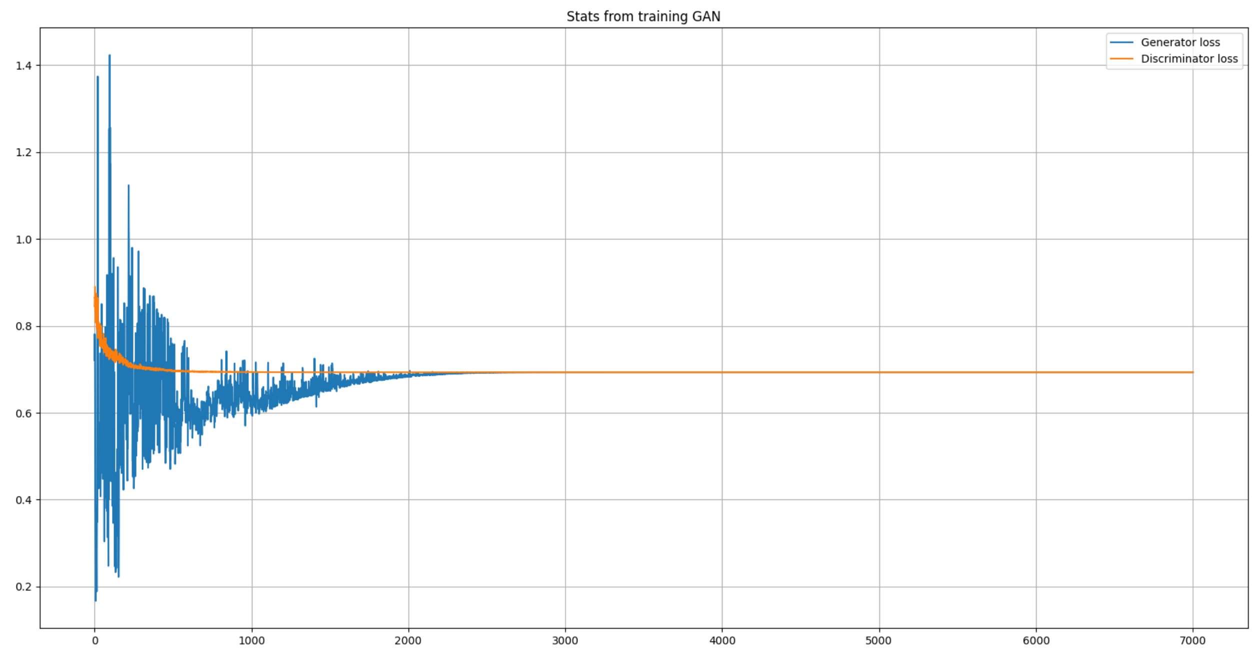

Fig 3.2 GAN loss chart

This model generates 8713 samples of minority class and hence making both the classes of same number.

Both the training data and generated data is fed into CNN and following are the results.

Metric	**Training data**	**Test Data**
Accuracy	0.9526	0.8302
Loss	279.52	6.316

Table 3.4 GAN with CNN metrics

Conclusion:

With GAN based over-sampling, there has been significant increase in the accuracy on classification of attack and normal samples. There can be more improvements done on the models by introducing GAN over specific category of attack traffic alone and work on multi label classification problem.

References:

1. Cloudwards.net. (n.d.). Ransomware Statistics, Trends and Facts for 2022 and Beyond. https://www.cloudwards.net/ransomware-statistics-trends-facts/
2. O'Reilly, David Foster, Generative Deep Learning. O'Reilly Media, 2023.
3. J. Li, "Network Intrusion Detection Algorithm and Simulation of Complex System in Internet Environment," 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2022, pp. 520-523, doi: 10.1109/ICIRCA54612.2022.9985720.
4. X. Quan, "A Neural Network Classifier Intrusion Detection Vulnerability System Based on HFE-CAMMLP Intrusion Detection Model and Weblog Analysis," 2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC), Qingdao, China, 2022, pp. 718-721, doi: 10.1109/ICFTIC57696.2022.10075290.
5. N. Zhu, G. Zhao, Y. Yang, H. Yang and Z. Liu, "AEC_GAN: Unbalanced Data Processing Decision-Making in Network Attacks Based on ACGAN and Machine Learning," in IEEE Access, vol. 11, pp. 52452-52465, 2023, doi: 10.1109/ACCESS.2023.3280421.