SELF-SUPERVISED NETWORK ANOMALY DETECTION AND ATTACK CLASSIFICATION

A PROJECT REPORT BY

APAR GARG (A0231539E)

SUBMITTED TO

INSTITUTE OF SYSTEMS SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE
21 LOWER KENT RIDGE RD, SINGAPORE 119077

in

Partial fulfillment of the requirements
for the degree of

MASTER OF TECHNOLOGY

MAY 2022

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Background

Every organization in today's information age is attempting to establish a presence on the Internet. The Internet enables an organization to deploy a web application like e-commerce in order to provide timely services to customers [1]. Therefore, quick and easy access to services via web applications enables organizations to raise their revenue.

1.2. Problem

Apart from the aforementioned advantages, the number of attacks is exploding overwhelmingly as the number of web applications grows [2]. A resource containing a vulnerability might be exploited by the attackers, and consequently can jeopardize the confidentiality, integrity, and availability properties of the organization's crucial resources. This could result in a significant financial loss as well as irreversible damage to an organization.

1.3. Solution

In this project, we developed an intrusion detection system using artificial intelligence which could detect and classify various types of attacks in a network.

Even though network traffic generates a huge amount of data to fuel deep learning, one difficulty to overcome is the imbalanced nature of network data - network intrusions and malicious behavior often account for a small portion of total network traffic. Several recent studies [3,4] have looked into ways to resample training data and make deep learning resilient to intrusion imbalance. To cope with the imbalanced data, down-sampling and over-sampling approaches are frequently combined with deep neural networks [5,6]. However, down-sampling causes loss of information, while generating artificial samples may cause overfitting, noise, or class overlap. In this project, we investigated unsupervised and self-supervised learning methods to learn better representations for features in data and as a means to deal with the phenomenon of intrusion class imbalance.

2. DATASET DESCRIPTION

I worked on CICIDS-2017 [7] and UNSW-NB15 [8] datasets. Details about the datasets are mentioned below.

2.1. CICIDS-2017

CICIDS-2017 is an open-source dataset. It contains benign and up-to-date common attacks that are similar to real-world data packet capture files. The data capturing period started on Monday, July 3, 2017, at 09:00 and ended on Friday, July 7, 2017, at 17:00, for a total of 5 days. Monday was a normal day and included only benign traffic. As shown in Figure 1, the dataset has more than 2.8 million rows and has 85 columns (84 features and 1 label). The dataset consists of 15 classes/categories which are listed in Figure 2.

```
dataset.shape
```

```
(2830436, 85)
```

Figure 1: Shape of CICIDS-2017 dataset.

```
dataset[" Label"].value_counts()
```

```
BENIGN                          2272790
DoS Hulk                         231073
PortScan                         158930
DDoS                             128027
DoS GoldenEye                     10293
FTP-Patator                        7938
SSH-Patator                        5897
DoS slowloris                      5796
DoS Slowhttptest                   5499
Bot                                1966
Web Attack ? Brute Force           1507
Web Attack ? XSS                    652
Infiltration                         36
Web Attack ? Sql Injection           21
Heartbleed                           11
Name:  Label, dtype: int64
```

Figure 2: Classes with their sample frequency in CICIDS-2017 dataset.

2.2. UNSW-NB15

UNSW-NB15 is an open-source dataset. It was created by the Australian Centre for Cyber Security (ACCS) in 2015. It is a comprehensive network attack traffic dataset. As shown in Figure 3, the dataset has more than 2.5 million rows and has 49 columns (47 features and 2 labels). 1 label contains attack classes/categories (9 different categories of attack) and 1 label contains attack (1)/normal category (0). The 9 attack categories are listed in Figure 4.

```
print(df.shape)
```

```
(2540047, 49)
```

Figure 3: Shape of UNSW-NB15 dataset.

```
df['attack_cat'].value_counts()
```

```
normal            2218764
generic            215481
exploits            44525
fuzzers             24246
dos                 16353
reconnaissance      13987
analysis             2677
backdoor             2329
shellcode            1511
worms                 174
Name: attack_cat, dtype: int64
```

Figure 4: Classes with their sample frequency in UNSW-NB15 dataset.

3. MULTICLASS CLASSIFICATION ON CICIDS-2017 WITHOUT ENSEMBLE

3.1. Methodology

3.1.1. Dataset Preparation

Several preprocessing steps - removing whitespaces in column names, replacing special characters in class names with an underscore, grouping classes, removing non-finite values (positive infinity and negative infinity), removing NULL (NaN) values, and dropping duplicate records were applied to the CICIDS-2017 dataset. Table 1 shows the old class names and corresponding new class names. The dataset was split into 80-20 train-test (stratified). Columns having float datatype, less than 85% correlation, and more than 1 unique value, were selected for further modeling. The class imbalance was handled using resampling (random undersampling and SMOTE oversampling). Standardization was applied to scale the features. After much consideration, I saved 6 CSV files and later used them for modeling. Table 2 shows the characteristics of each file. Table 3 shows the number of records corresponding to each class in the files.

Table 1: Map for grouping attack classes used for multiclass classification on CICIDS-2017 without ensemble.

New Class	Old Class
Normal	Benign
PortScan	PortScan
Bot	Bot
Infiltration	Infiltration
DDoS/DoS	DDoS, DoS_slowloris, DoS_Slowhttptest, DoS_Hulk, DoS_GoldenEye, Heartbleed
Brute_Force	FTPPatator, SSHPatator
Web_Attack	Web_Attack_Brute_Force, Web_Attack_XSS, Web_Attack_Sql_Injection

Table 2: Description of saved CSV file used for multiclass classification on CICIDS-2017 without ensemble.

File Name	Resampling (Y/N)	Scaling (Y/N)	No. of rows	No. of columns
train_original.csv	N	N	1801970	37
train_scaled.csv	N	Y	1801970	37
train_original_resampled_smote.csv	Y	N	1803585	37
train_scaled_resampled_smote.csv	Y	Y	1803585	37
test_original.csv	N	N	473602	37
test_scaled.csv	N	Y	473602	37

Table 3: Class distribution in saved CSV file used for multiclass classification on CICIDS-2017 without ensemble.

File Name	Normal	DDoS/ DoS	Port Scan	Brute_ Force	Web_ Attack	Bot	Infiltration
train_original.csv	1532360	257655	1649	7396	1719	1162	29
train_scaled.csv	1532360	257655	1649	7396	1719	1162	29
train_original_resampled_smote.csv	257655	257655	257655	257655	257655	257655	257655
train_scaled_resampled_smote.csv	257655	257655	257655	257655	257655	257655	257655
test_original.csv	405193	64987	608	2038	435	334	7
test_scaled.csv	405193	64987	608	2038	435	334	7

3.1.2. Transfer Learning on DAE

Vanilla autoencoder's requirement for a bottleneck layer forces the model to do some summarization. This tampers the model's ability to learn good representations. Unlike Vanilla autoencoders, Denoising Autoencoder (DAE) does not learn the identity function. A DAE tries to reconstruct the noisy version of the input. By corrupting the input, we're taking examples and artificially "pushing" them away from the manifold. By using the unaltered input example as a target, it can approximate a function that projects points in the feature space onto the feature distribution manifold. This removes the need for a bottleneck layer in DAE. That's why DAE is a better choice. Instead of compressing our examples, we can expand each sample into a higher-dimensional space. This will give better and more useful nonlinear representations of the data [9].

I trained a DAE on **train_original.csv** (without labels). The architecture of the proposed DAE model is illustrated in Figure 5. I used the dfencoder [10] library for this purpose. It is a python library for building DAE autoencoders with tabular data. Instead of setting values to zero or adding some Gaussian noise to them, the dfencoder library uses a different type of noise called "swap noise". It randomly picks 20% of the cells in the dataframe and replaces their values with values from the same column but randomly sampled rows. This provides a computationally cheap way to sample values from the distribution of a column while avoiding the need to model these distributions. The library also uses GaussRank scaling to scale the features. It took around 7 min to train the DAE model on the CPU.

After training the DAE, the latent features from all 3 hidden layers were extracted and stacked for each sample in **train_original_resampled_smote.csv** as well as **test_original.csv**. It took around 13 sec to extract the latent features from **train_original_resampled_smote.csv** and around 4 sec to extract the latent features from **test_original.csv**. Then ANN (Artificial Neural Network) and 1DCNN (1D Convolutional Neural Networks) were trained on stacked latent features (input size=50+50+50=150) of **train_original_resampled_smote.csv** (with labels) for classification. Keras Tuner library [11] was used for tuning the hyperparameter in both the classifiers. It took around 58 min to train the ANN and around 1.2 hr to train the 1DCNN on the CPU. Figure 6 illustrates the model architecture for ANN and 1DCNN.

This technique of coupling DAE with various classifiers was used by a Kaggle competition winner in 2017 [12]. The key to his winning solution was GaussRank scaling and swap noise.

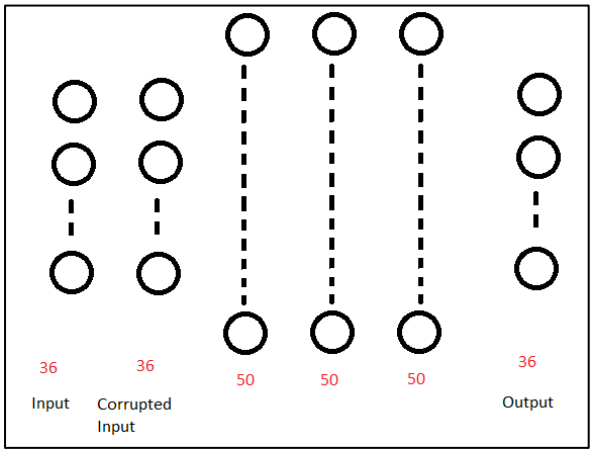


Figure 5: Proposed deep stack DAE architecture used for multiclass classification on CICIDS-2017 without ensemble.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	38656
batch_normalization (BatchNormalization)	(None, 256)	1024
activation (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_1 (BatchNormalization)	(None, 128)	512
activation_1 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
batch_normalization_2 (BatchNormalization)	(None, 64)	256
activation_2 (Activation)	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2080
batch_normalization_3 (BatchNormalization)	(None, 32)	128
activation_3 (Activation)	(None, 32)	0
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 7)	231

Total params: 84,039
Trainable params: 83,079
Non-trainable params: 960

ANN

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 146, 8)	48
max_pooling1d (MaxPooling1D)	(None, 73, 8)	0
dropout (Dropout)	(None, 73, 8)	0
conv1d_1 (Conv1D)	(None, 69, 16)	656
max_pooling1d_1 (MaxPooling1D)	(None, 34, 16)	0
dropout_1 (Dropout)	(None, 34, 16)	0
conv1d_2 (Conv1D)	(None, 32, 32)	1568
max_pooling1d_2 (MaxPooling1D)	(None, 16, 32)	0
dropout_2 (Dropout)	(None, 16, 32)	0
conv1d_3 (Conv1D)	(None, 14, 64)	6208
max_pooling1d_3 (MaxPooling1D)	(None, 7, 64)	0
dropout_3 (Dropout)	(None, 7, 64)	0
flatten (Flatten)	(None, 448)	0
dense (Dense)	(None, 128)	57472
batch_normalization (BatchNormalization)	(None, 128)	512
activation (Activation)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
batch_normalization_1 (BatchNormalization)	(None, 64)	256
activation_1 (Activation)	(None, 64)	0
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
batch_normalization_2 (BatchNormalization)	(None, 32)	128
activation_2 (Activation)	(None, 32)	0
dropout_6 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 7)	231

Total params: 77,415
Trainable params: 76,967
Non-trainable params: 448

1DCNN

Figure 6: Proposed ANN and 1DCNN model architecture trained on DAE features used for multiclass classification on CICIDS-2017 without ensemble.

3.1.3. Transfer Learning on VIME

The authors [13] proposed a novel self-supervised learning model for tabular data, which they refer to as VIME (Value Imputation and Mask Estimation). VIME introduces two pretext tasks: feature vector estimation and mask vector estimation.

- Mask vector estimation - Predict which features have been masked.
- Feature vector estimation - Predict the values of the features that have been corrupted.

This helps the encoder learn the correlation among the features. Figure 7 illustrates the entire self-supervised learning framework proposed by the authors.

I trained the VIME on **train_scaled.csv** (without labels). The architecture of the proposed VIME model is illustrated in Figure 8. I used VIME [14] framework for this purpose. It took around 23 min to train the VIME model on the CPU. After training the VIME, the latent features from the last hidden layer were extracted for each sample in **train_scaled_resampled_smote.csv** as well as **test_scaled.csv**. It took around 25 sec to extract the latent features from **train_scaled_resampled_smote.csv** and around 7 sec to extract the latent features from **test_scaled.csv**. Then ANN and 1DCNN were trained on the **train_scaled_resampled_smote.csv** latent features (with labels) for classification. Keras Tuner library [11] was used for tuning the hyperparameter in both the classifiers. It took around 24 min to train the ANN and around 33 min to train the 1DCNN on the CPU. Figure 9 shows the model architecture for ANN and 1DCNN.

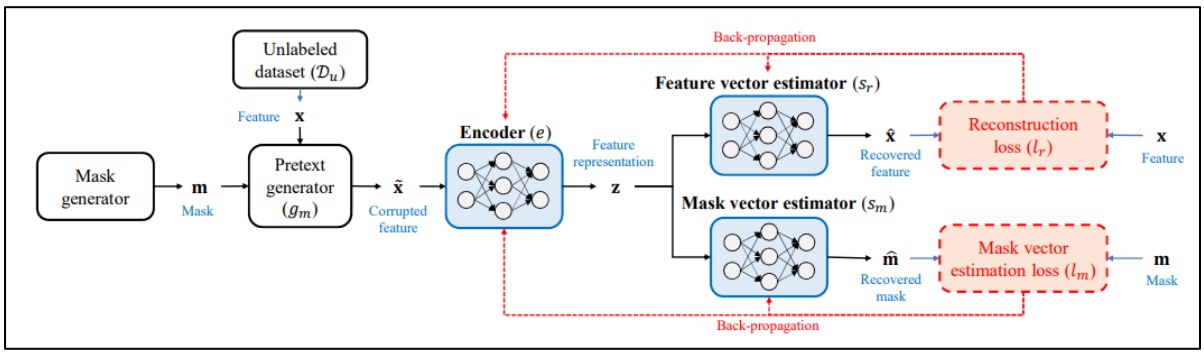
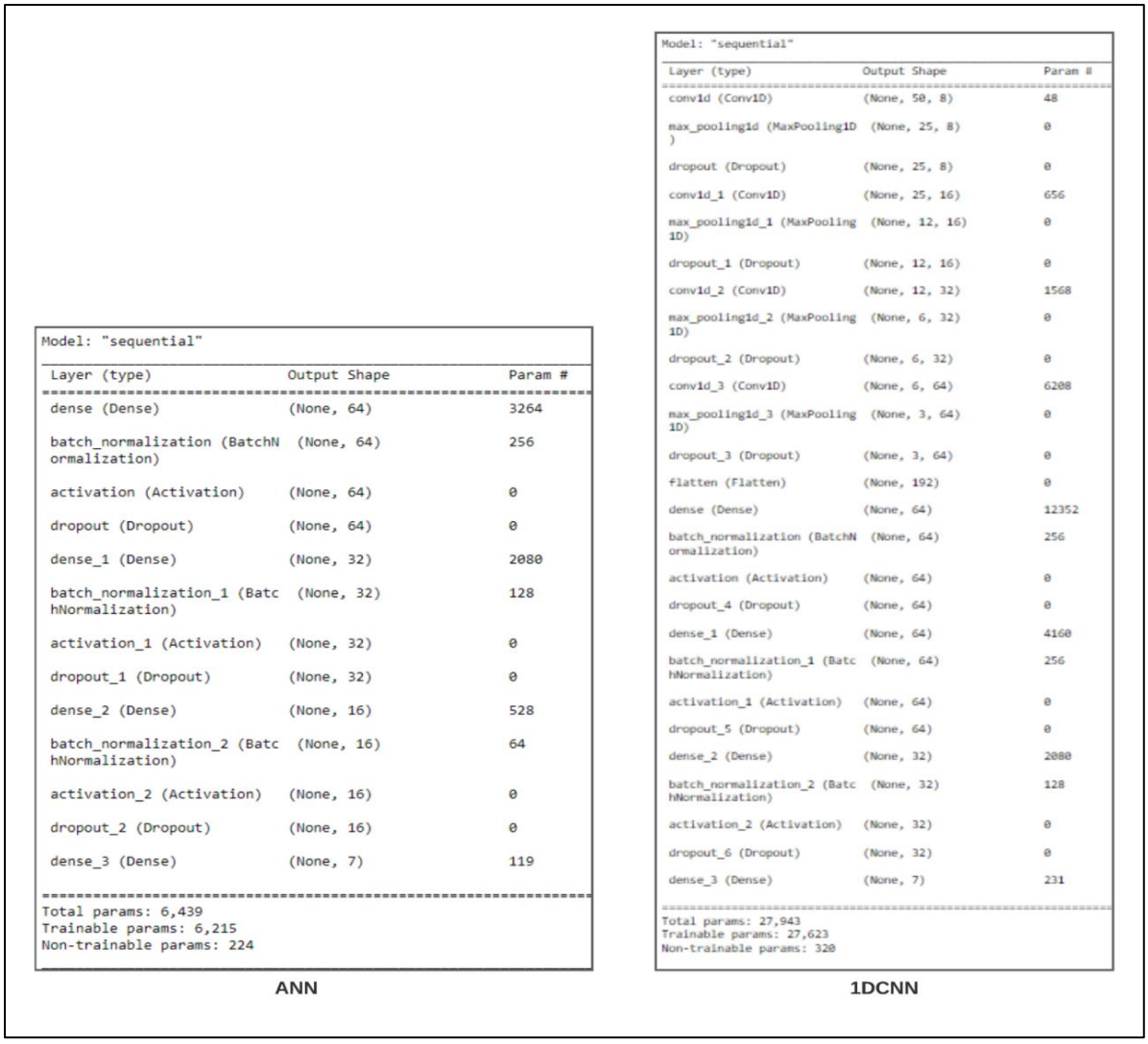


Figure 7: Block diagram of the proposed self-supervised learning framework on tabular data. (1) Mask generator generates binary mask vector (**m**) which is combined with an input sample (**x**) to create a masked and corrupted sample ($\tilde{\mathbf{x}}$), (2) Encoder (e) transforms $\tilde{\mathbf{x}}$ into a latent representation (**z**), (3) Mask vector estimator (s_m) is trained by minimizing the cross-entropy loss with **m**, feature vector estimator (s_r) is trained by minimizing the reconstruction loss with **x**, (4) Encoder (e) is trained by minimizing the weighted sum of both losses.

```
Model: "model"
_________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================
 input_1 (InputLayer)        [(None, 36)]              0           []

 dense (Dense)               (None, 50)                1850        ['input_1[0][0]']

 dense_1 (Dense)             (None, 50)                2550        ['dense[0][0]']

 dense_2 (Dense)             (None, 50)                2550        ['dense_1[0][0]']

 mask (Dense)                (None, 36)                1836        ['dense_2[0][0]']

 feature (Dense)             (None, 36)                1836        ['dense_2[0][0]']

=================================================================================
Total params: 10,622
Trainable params: 10,622
Non-trainable params: 0
```

Figure 8: Proposed VIME model architecture used for multiclass classification on CICIDS-2017 without ensemble.



Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	3264
batch_normalization (BatchNormalization)	(None, 64)	256
activation (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
batch_normalization_1 (BatchNormalization)	(None, 32)	128
activation_1 (Activation)	(None, 32)	0
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
batch_normalization_2 (BatchNormalization)	(None, 16)	64
activation_2 (Activation)	(None, 16)	0
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 7)	119

Total params: 6,439
Trainable params: 6,215
Non-trainable params: 224

ANN

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 50, 8)	48
max_pooling1d (MaxPooling1D)	(None, 25, 8)	0
dropout (Dropout)	(None, 25, 8)	0
conv1d_1 (Conv1D)	(None, 25, 16)	656
max_pooling1d_1 (MaxPooling1D)	(None, 12, 16)	0
dropout_1 (Dropout)	(None, 12, 16)	0
conv1d_2 (Conv1D)	(None, 12, 32)	1568
max_pooling1d_2 (MaxPooling1D)	(None, 6, 32)	0
dropout_2 (Dropout)	(None, 6, 32)	0
conv1d_3 (Conv1D)	(None, 6, 64)	6208
max_pooling1d_3 (MaxPooling1D)	(None, 3, 64)	0
dropout_3 (Dropout)	(None, 3, 64)	0
flatten (Flatten)	(None, 192)	0
dense (Dense)	(None, 64)	12352
batch_normalization (BatchNormalization)	(None, 64)	256
activation (Activation)	(None, 64)	0
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4160
batch_normalization_1 (BatchNormalization)	(None, 64)	256
activation_1 (Activation)	(None, 64)	0
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
batch_normalization_2 (BatchNormalization)	(None, 32)	128
activation_2 (Activation)	(None, 32)	0
dropout_6 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 7)	231

Total params: 27,943
Trainable params: 27,623
Non-trainable params: 320

1DCNN

Figure 9: Proposed ANN and 1DCNN model architecture trained on VIME features used for multiclass classification on CICIDS-2017 without ensemble.

3.2. Results

3.2.1. DAE Unsupervised Learning

Figure 10 presents the reconstruction loss vs epoch curve for DAE trained on **train_original.csv**.

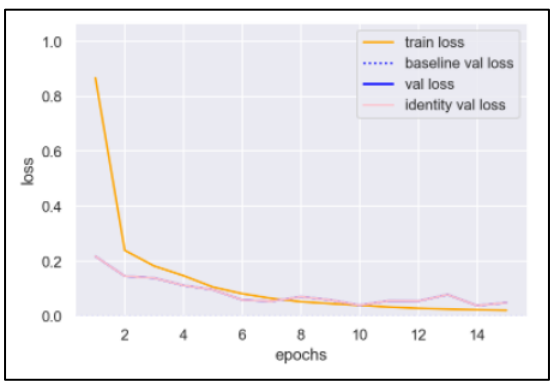


Figure 10: Loss curve for DAE unsupervised training for multiclass classification on CICIDS-2017 without ensemble.

3.2.2. VIME Unsupervised Learning

Figure 11 presents the mask loss vs epoch, feature (reconstruction) loss vs epoch, and total loss vs epoch curves for VIME trained on **train_scaled.csv**.

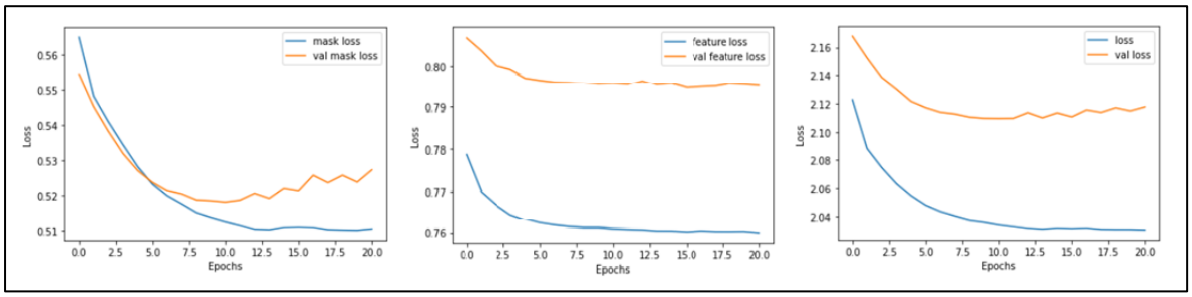


Figure 11: Loss curves for VIME unsupervised training for multiclass classification on CICIDS-2017 without ensemble.

3.2.3. DAE + ANN Supervised Learning

The ANN trained on stacked latent features of **train_original_resampled_smote.csv** was tested on stacked latent features of **test_original.csv.** The training and validation loss curves for ANN trained on DAE latent features are shown in Figure 12. Figure 13 shows the classification report and confusion matrix on the testing dataset. The classification report shows the main classification metrics precision, recall, and f1-score on a per-class basis. The confusion matrix summarizes the number of correct and incorrect predictions with count values and breaks them down by each class. ANN took around 17 sec to return results on the testing dataset.

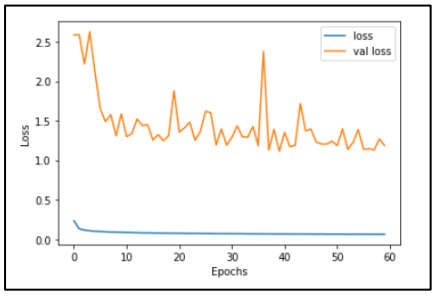
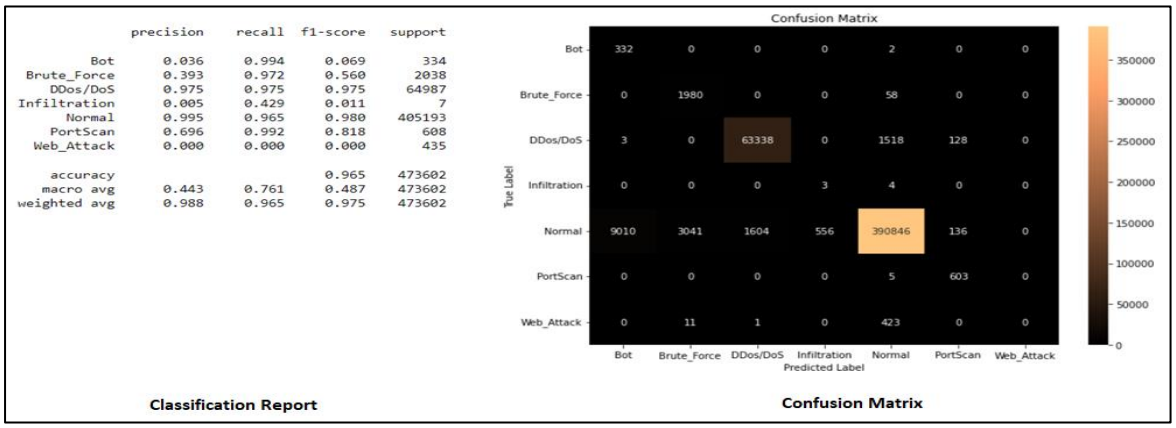


Figure 12: Loss curve for ANN trained on DAE features for multiclass classification on CICIDS-2017 without ensemble.



	precision	recall	f1-score	support
Bot	0.036	0.994	0.069	334
Brute_Force	0.393	0.972	0.560	2038
DDos/DoS	0.975	0.975	0.975	64987
Infiltration	0.005	0.429	0.011	7
Normal	0.995	0.965	0.980	405193
PortScan	0.696	0.992	0.818	608
Web_Attack	0.000	0.000	0.000	435
accuracy			0.965	473602
macro avg	0.443	0.761	0.487	473602
weighted avg	0.988	0.965	0.975	473602

Figure 13: Performance of DAE + ANN on testing dataset for multiclass classification on CICIDS-2017 without ensemble.

3.2.4. DAE + 1DCNN Supervised Learning

The 1DCNN trained on stacked latent features of **train_original_resampled_smote.csv** was tested on stacked latent features of **test_original.csv.** The training and validation loss curves for 1DCNN trained on DAE latent features are shown in Figure 14. Figure 15 shows the classification report and confusion matrix on the testing dataset. 1DCNN took around 20 sec to return results on the testing dataset.

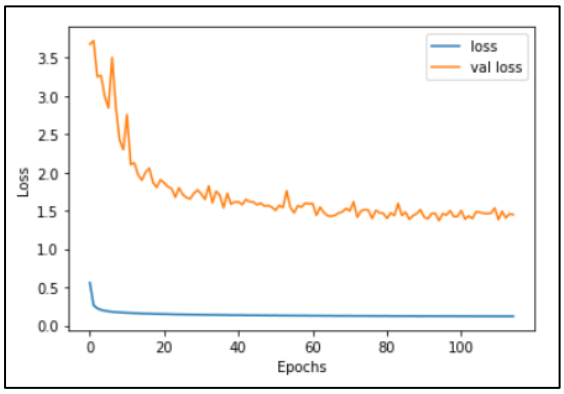
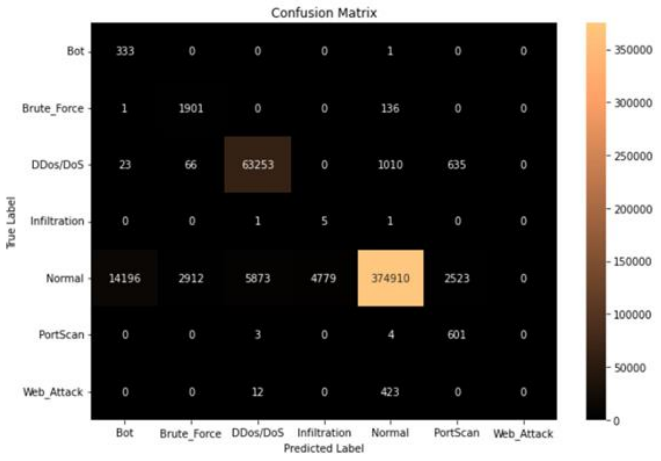


Figure 14: Loss curve for 1DCNN trained on DAE features for multiclass classification on CICIDS-2017 without ensemble.

	precision	recall	f1-score	support
Bot	0.023	0.997	0.045	334
Brute_Force	0.390	0.933	0.550	2038
DDos/DoS	0.915	0.973	0.943	64987
Infiltration	0.001	0.714	0.002	7
Normal	0.996	0.925	0.959	405193
PortScan	0.160	0.988	0.275	608
Web_Attack	0.000	0.000	0.000	435
accuracy			0.931	473602
macro avg	0.355	0.790	0.396	473602
weighted avg	0.979	0.931	0.953	473602



Classification Report **Confusion Matrix**

Figure 15: Performance of DAE + 1DCNN on testing dataset for multiclass classification on CICIDS-2017 without ensemble.

3.2.5. VIME + ANN Supervised Learning

The ANN trained on latent features of **train_scaled_resampled_smote.csv** was tested on latent features of **test_scaled.csv.** The training and validation loss curves for ANN trained on VIME latent features are shown in Figure 16. Figure 17 shows the classification report and confusion matrix on the testing dataset. ANN took around 13 sec to return results on the testing dataset.

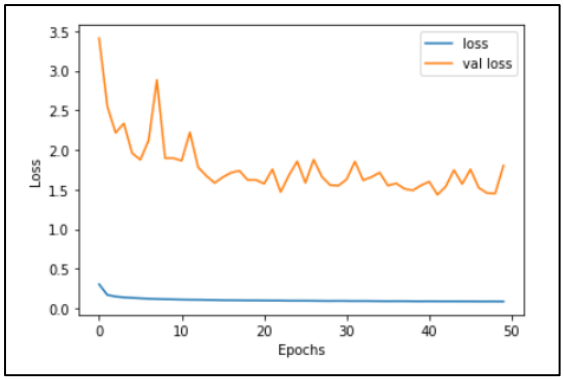
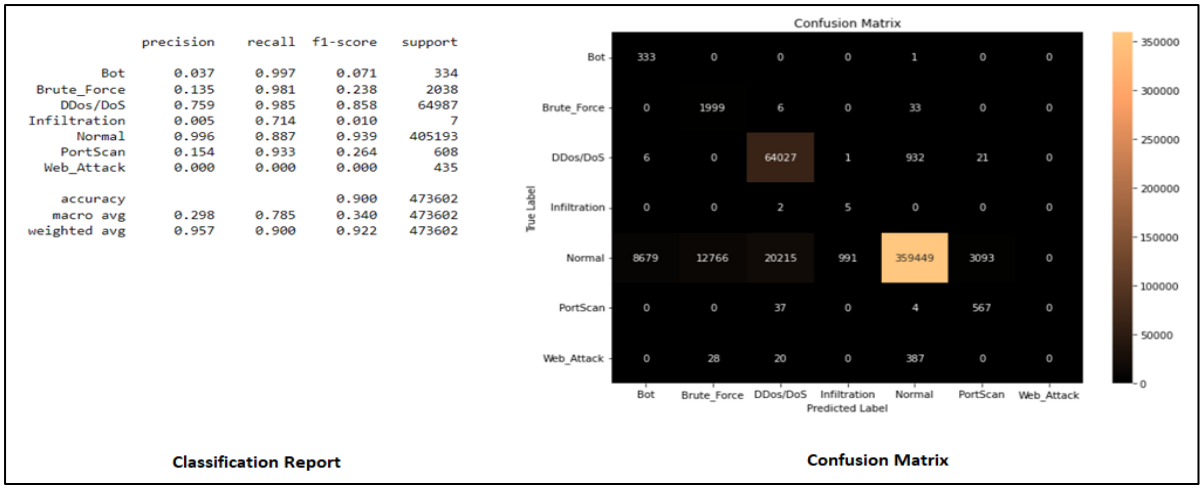


Figure 16: Loss curve for ANN trained on VIME features for multiclass classification on CICIDS-2017 without ensemble.



	precision	recall	f1-score	support
Bot	0.037	0.997	0.071	334
Brute_Force	0.135	0.981	0.238	2038
DDos/DoS	0.759	0.985	0.858	64987
Infiltration	0.005	0.714	0.010	7
Normal	0.996	0.887	0.939	405193
PortScan	0.154	0.933	0.264	608
Web_Attack	0.000	0.000	0.000	435
accuracy			0.900	473602
macro avg	0.298	0.785	0.340	473602
weighted avg	0.957	0.900	0.922	473602

Classification Report **Confusion Matrix**

Figure 17: Performance of VIME + ANN on testing dataset for multiclass classification on CICIDS-2017 without ensemble.

3.2.6. VIME + 1DCNN Supervised Learning

The 1DCNN trained on latent features of **train_scaled_resampled_smote.csv** was tested on latent features of **test_scaled.csv.** The training and validation loss curves for 1DCNN trained on VIME latent features are shown in Figure 18. Figure 19 shows the classification report and confusion matrix on the testing dataset. 1DCNN took around 20 sec to return results on the testing dataset.

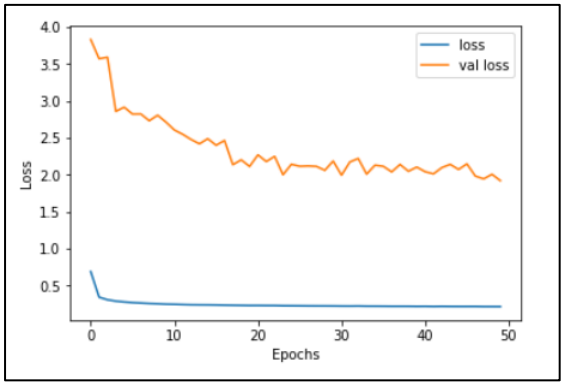


Figure 18: Loss curve for 1DCNN trained on VIME features for multiclass classification on CICIDS-2017 without ensemble.

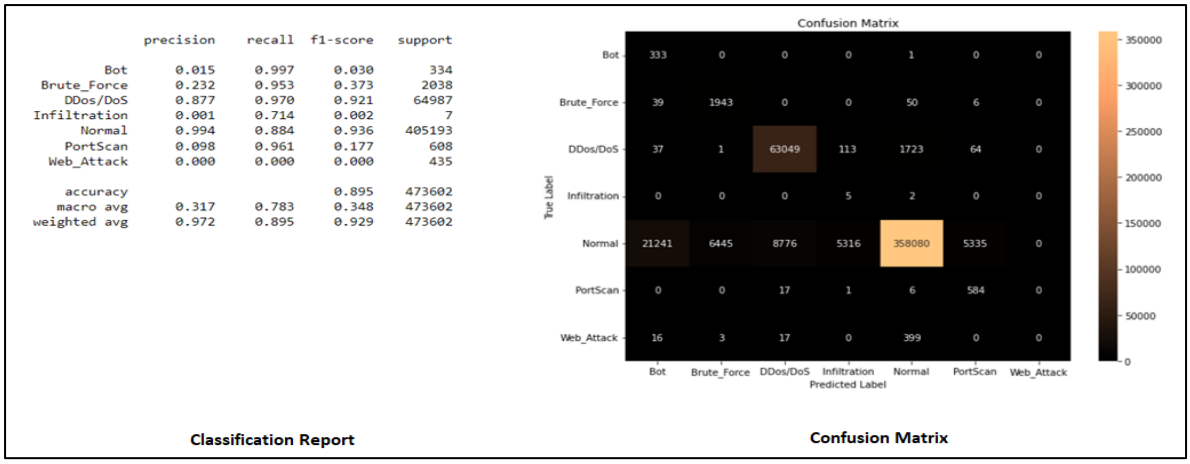


Figure 19: Performance of VIME + 1DCNN on testing dataset for multiclass classification on CICIDS-2017 without ensemble.

4. MULTICLASS CLASSIFICATION ON CICIDS-2017 WITH ENSEMBLE

4.1. Methodology

4.1.1. Dataset Preparation

Several preprocessing steps - removing whitespaces in column names, replacing special characters in class names with an underscore, grouping classes, removing non-finite values (positive infinity and negative infinity), removing NULL (NaN) values, and dropping duplicate records were applied to the CICIDS-2017 dataset. Table 1 shows the old class names and corresponding new class names. The dataset was split into 80-20 train-test (stratified). Features having float datatype, less than 85% correlation, and more than 1 unique value, were selected for further modeling. Standardization was applied to scale the features. The 7-class classification problem was further decomposed into 7 binary classification sub-problems using the One Against All (OAA) strategy [15,16]. The OAA decomposition method converts a multi-class classification problem into a set of subproblems and each of them aims at classifying one class against all the other classes. Consequently, the number of subproblems is equal to the number of classes. After much consideration, I saved 4 CSV files and later used them for modeling. Table 4 shows the number of records and columns in the training set (**train_original.csv**) for each subproblem. The testing set (**test_original.csv**) has 473716 records. Figure 20 shows the number of records corresponding to each class in the training files.

Table 4: Description of saved CSV file used for multiclass classification on CICIDS-2017 with ensemble.

Subproblem	**No. of rows in train_original.csv**	**No. of columns in all CSV files**
Normal vs Not-Normal	1802355	37
Bot vs Not-Bot	1802144	37
Brute_Force vs Not-Brute_Force	1802124	37
DDoS/DoS vs Not-DDoS/DoS	1802234	37
Infiltration vs Not-Infiltration	1802124	37
PortScan vs Not-PortScan	1802225	37
Web_Attack vs Not-Web_Attack	1802124	37

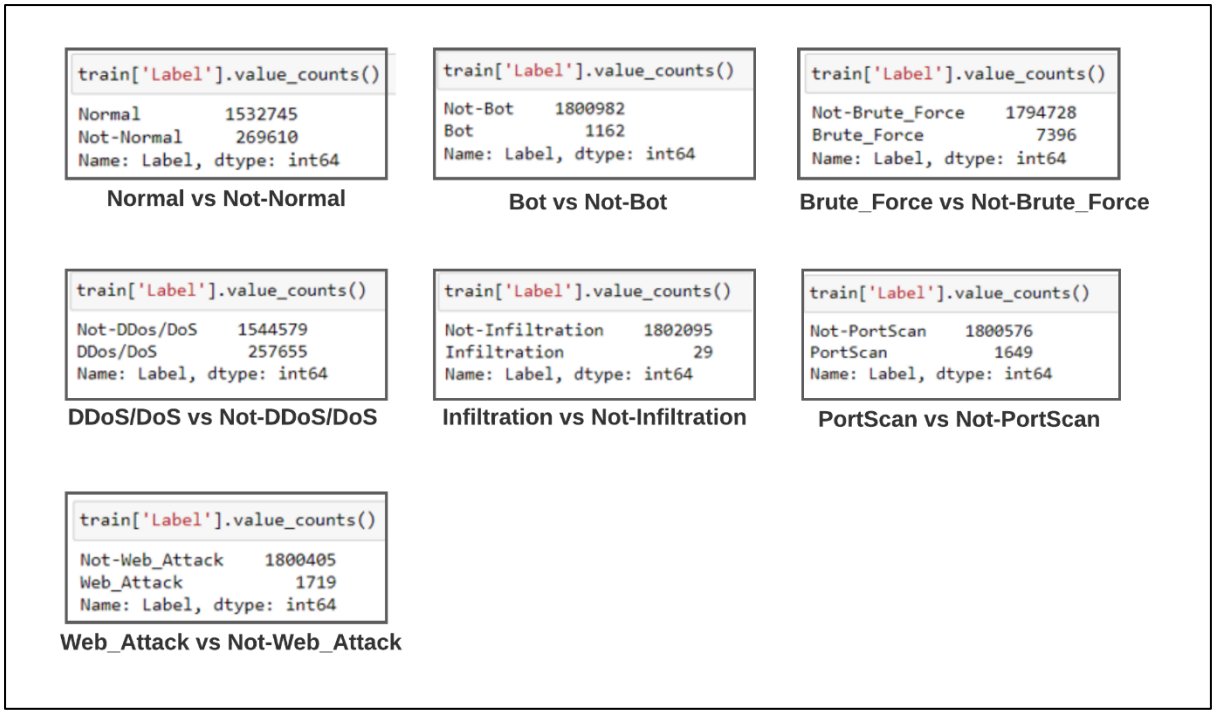


Figure 20: Class distribution in saved training CSV file used for multiclass classification on CICIDS-2017 with ensemble.

4.1.2. Transfer Learning on DAE

For every subproblem, I trained a DAE on the respective **train_original.csv** (without labels). It took 5-7 min to train every DAE model on the CPU. After training the DAE, the latent features from all 3 hidden layers were extracted and stacked for each sample in **train_original.csv** as well as **test_original.csv**. Then ANN was trained on stacked latent features (input size=50+50+50=150) of **train_original.csv** (with labels) for classification. It took 8-23 min to train every ANN on the CPU. For more details about the DAE model architecture and training process, refer to section 3.1.2.

4.1.3. Ensemble Model

I created an ensemble of all 7 ANN trained on DAE features. As shown in Figure 22, during the testing/inference phase, the ANN models take **test_original.csv** latent features (by DAE models) as input and output the binary class probabilities. The decision by the Arbitrator is taken using the maximum confidence strategy [15]. The maximum confidence strategy is the most common and simple used method for aggregation. As shown in Figure 21, the output class is simply taken from the classifier with the greatest response. Z_i is the output of the classifier corresponding to the subproblem: class *i* against all the other classes.

$$\text{Class}(X) = \arg \max_{i=1\ldots K} Z_i$$

Figure 21: Maximum confidence aggregation strategy formula.

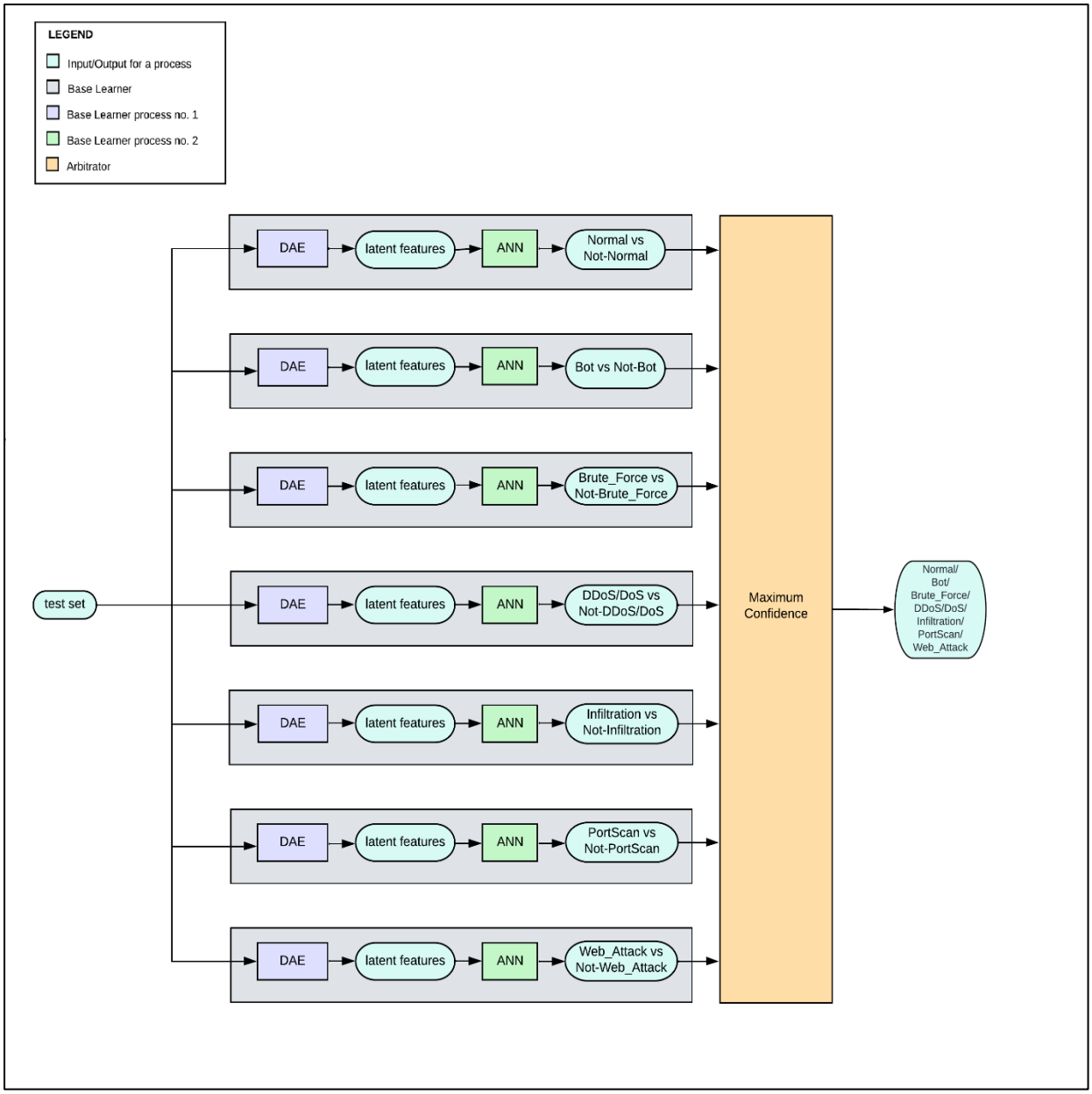


Figure 22: Block diagram of proposed ensemble model used for multiclass classification on CICIDS-2017 with ensemble.

4.2. Results

4.2.1. DAE Unsupervised Learning

Figure 23 presents the reconstruction loss vs epoch curve for DAE models trained on respective **train_original.csv**.

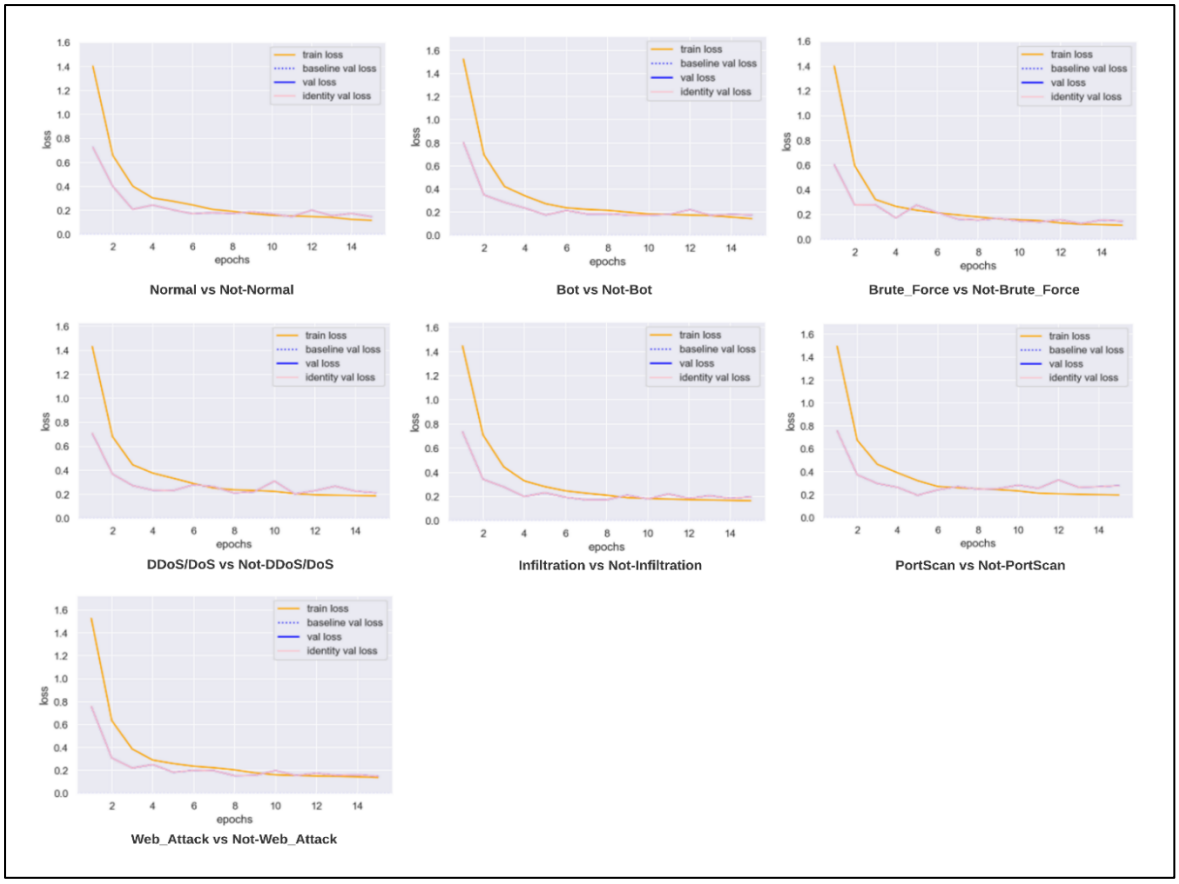


Figure 23: Loss curves for DAE unsupervised training for multiclass classification on CICIDS-2017 with ensemble.

4.2.2. DAE + ANN Supervised Learning

The training and validation loss curves for ANN models trained on DAE latent features are shown in Figure 24.

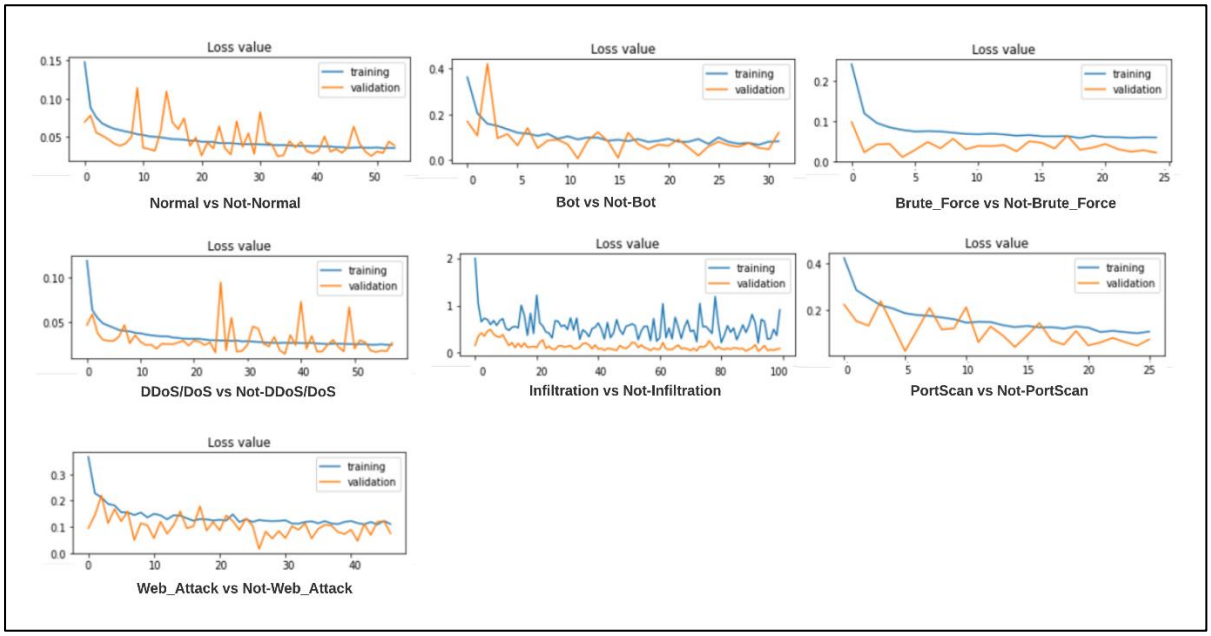
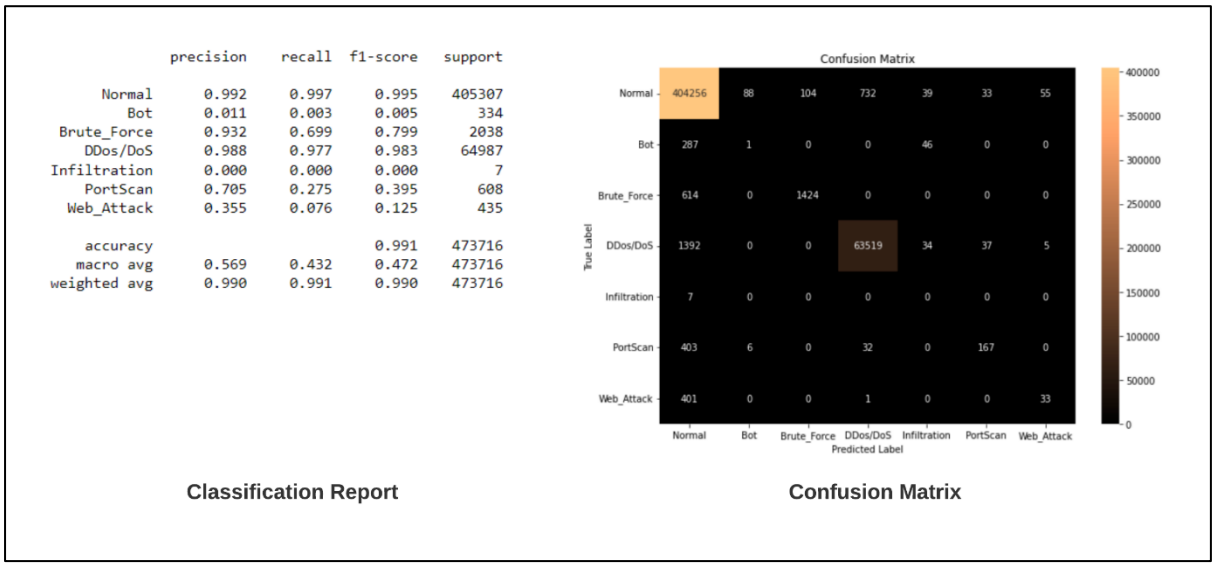


Figure 24: Loss curve for ANN trained on DAE features for multiclass classification on CICIDS-2017 with ensemble.

4.2.3. Ensemble Model

The classification report and confusion matrix of the proposed ensemble model applied on the **test_original.csv** are shown in Figure 25.



	precision	recall	f1-score	support
Normal	0.992	0.997	0.995	405307
Bot	0.011	0.003	0.005	334
Brute_Force	0.932	0.699	0.799	2038
DDos/DoS	0.988	0.977	0.983	64987
Infiltration	0.000	0.000	0.000	7
PortScan	0.705	0.275	0.395	608
Web_Attack	0.355	0.076	0.125	435
accuracy			0.991	473716
macro avg	0.569	0.432	0.472	473716
weighted avg	0.990	0.991	0.990	473716

Figure 25: Performance of proposed ensemble model on testing dataset for multiclass classification on CICIDS-2017 with ensemble.

5. ANOMALY SCORE BASED BINARY CLASSIFICATION ON CICIDS-2017

5.1. Methodology

5.1.1. Dataset Preparation

The dataset prepared in section 3.1.1. was also used for this task.

5.1.2. DAE Unsupervised Learning

By employing a reconstruction-based technique, autoencoders can be used for unsupervised network anomaly detection. It is reasonable to assume that if the autoencoders have successfully learned the data distribution (i.e., normal network traffic behavior), the models will be able to generate samples that are similar to those in the training set, resulting in small reconstruction errors between the original and generated samples. In contrast, given an anomalous sample, which the models have not seen during the training, the sample might be poorly reconstructed, resulting in a large reconstruction error. Further, by learning patterns from reconstruction errors of normal and anomalous samples, we can discriminate whether the given data sample is anomalous or not [17].

I trained a DAE on **train_original.csv** (without labels). For more details about the DAE model architecture and training process, refer to section 3.1.2. It took around 8 min to train the DAE on the CPU. After training the DAE, samples in **train_original_resampled_smote.csv** and **test_original.csv** were input to the model, and the reconstruction error was noted. In this work, Mean Squared Error (MSE) was used to compute the reconstruction error of a sample.

5.1.3. VIME Unsupervised Learning

I trained a VIME on **train_scaled.csv** (without labels). For more details about the VIME model architecture and training process, refer to section 3.1.3. It took around 23 min to train the VIME on the CPU. After training the VIME, samples in **train_scaled_resampled_smote.csv** and **test_scaled.csv** were input to the model, and the reconstruction error was noted. In this work, MSE was used to compute the reconstruction error of a sample.

5.1.4. Ensemble Model

I created a novel ensemble model which takes in reconstruction errors from DAE and VIME (base learners) and outputs normal/anomaly with the help of an ANN (arbitrator). It took around 15 min to train the ANN on reconstruction error features on the CPU. Keras Tuner library [11] was used for tuning the ANN hyperparameters. The proposed ensemble model architecture and its working during the testing/inference phase are illustrated in Figure 26. Figure 27 illustrates the model architecture for ANN.

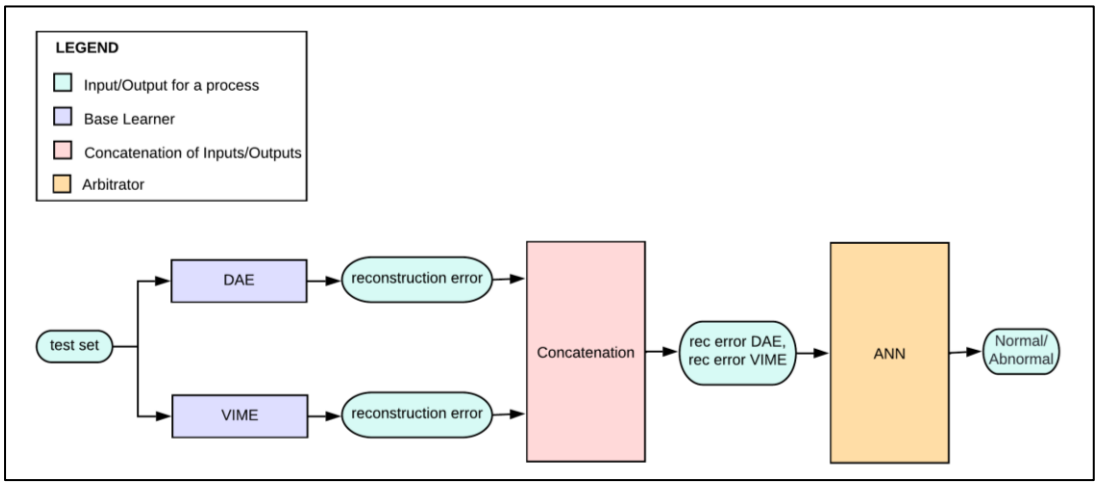


Figure 26: Block diagram of proposed ensemble network architecture used for anomaly score based binary classification on CICIDS-2017.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                48

 activation (Activation)     (None, 16)                0

 dense_1 (Dense)             (None, 16)                272

 activation_1 (Activation)   (None, 16)                0

 dense_2 (Dense)             (None, 8)                 136

 activation_2 (Activation)   (None, 8)                 0

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 465
Trainable params: 465
Non-trainable params: 0
```

Figure 27: Proposed ANN model architecture trained on reconstruction error features used for anomaly score based binary classification on CICIDS-2017.

5.2. Results

5.2.1. DAE Unsupervised Learning

Figure 28 presents the reconstruction loss vs epoch curve for DAE trained on **train_original.csv**.

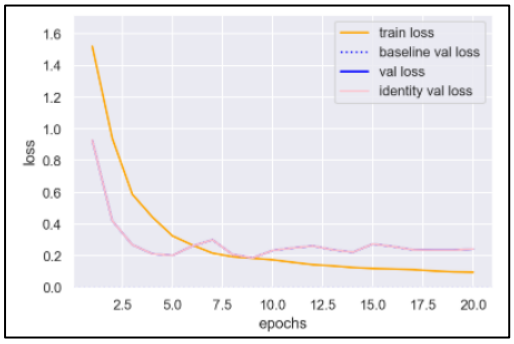


Figure 28: Loss curve for DAE unsupervised training for anomaly score based binary classification on CICIDS-2017.

5.2.2. VIME Unsupervised Learning

Figure 29 presents the mask loss vs epoch, feature/reconstruction loss vs epoch, and total loss vs epoch curves for the VIME model trained on **train_scaled.csv**.

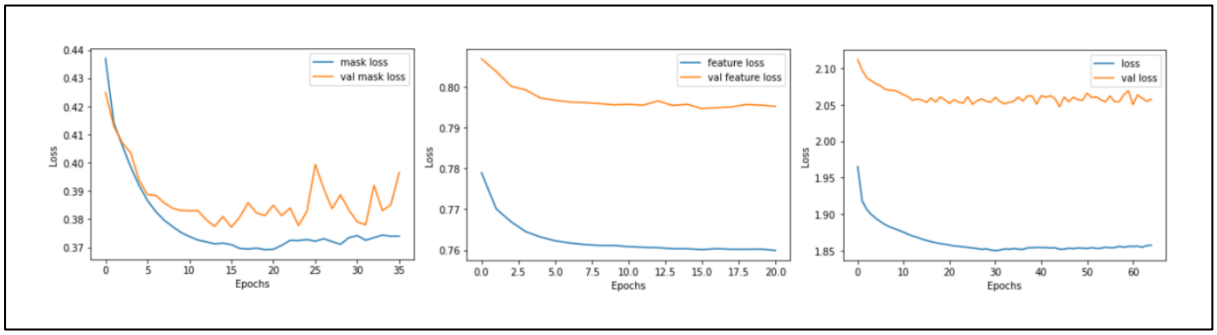


Figure 29: Loss curves for VIME unsupervised training for anomaly score based binary classification on CICIDS-2017.

5.2.3. Ensemble Model

The training and validation loss curves for the ANN model trained on DAE and VIME reconstruction error features are shown in Figure 30. The classification report and confusion matrix of the proposed ensemble model on the test set are shown in Figure 31.

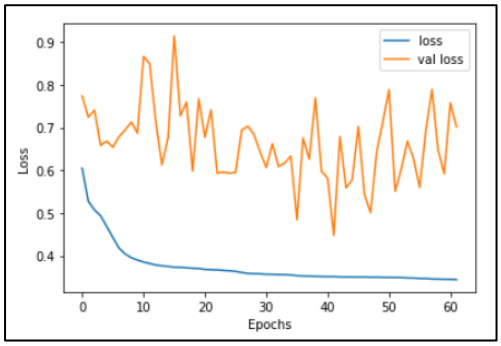


Figure 30: Loss curve for ANN trained on reconstruction error features for anomaly score based binary classification on CICIDS-2017.

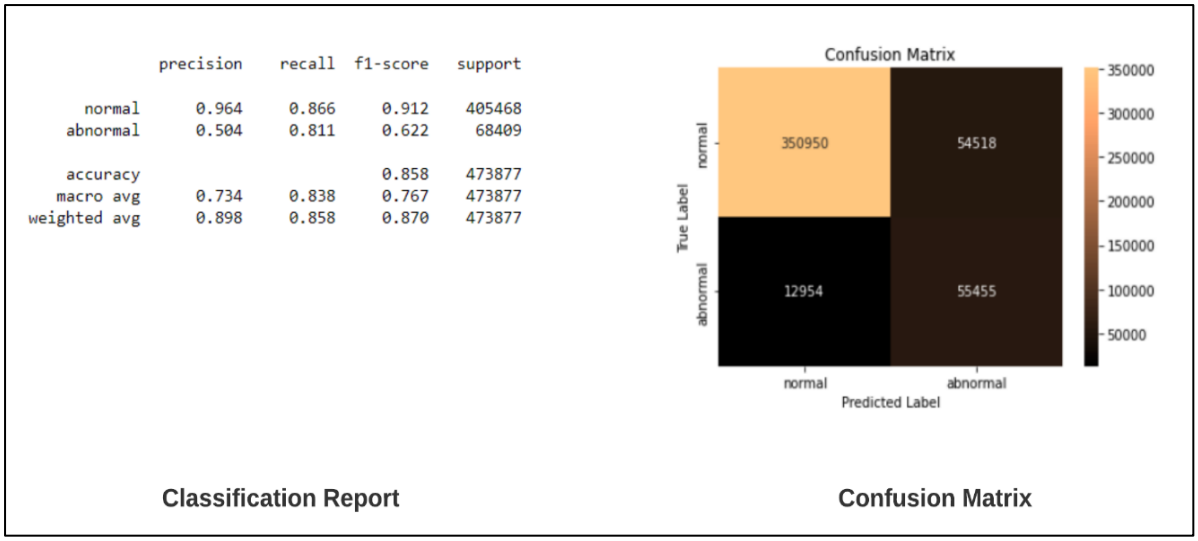


Figure 31: Performance of proposed ensemble model on testing dataset for anomaly score based binary classification on CICIDS-2017.

6. ANOMALY SCORE BASED BINARY CLASSIFICATION ON UNSW-NB15

6.1. Methodology

6.1.1. Dataset Preparation

Several preprocessing steps – dropping all timestamp and a few unuseful categorical features, changing datatype of features, lowercasing class names, removing non-finite values (positive infinity and negative infinity), encoding categorical features, removing NULL (NaN) values, and dropping duplicate records were applied to the UNSW-NB15 dataset. The dataset was split into 80-20 train-test (stratified). Columns having less than 85% correlation, and more than 1 unique value, were selected for further modeling. The class imbalance was handled using resampling (random undersampling and SMOTE oversampling). Standardization was applied to scale the features. After much consideration, I saved 6 CSV files and later used them for modeling. Table 5 shows the characteristics of each file. Table 6 shows the number of records corresponding to each class in the files.

Table 5: Description of saved CSV file used for anomaly score based binary classification on UNSW-NB15.

File Name	Resampling (Y/N)	Scaling (Y/N)	No. of rows	No. of columns
train_original.csv	N	N	1371712	169
train_scaled.csv	N	Y	1371712	169
train_original_resampled_smote.csv	Y	N	216500	169
train_scaled_resampled_smote.csv	Y	Y	216500	169
test_original.csv	N	N	365223	169
test_scaled.csv	N	Y	365223	169

Table 6: Class distribution in saved CSV file used for anomaly score based binary classification on UNSW-NB15.

File Name	normal	exploits	fuzzers	recon.	generic	dos	shellcode	analysis	backdoor	worms
train_original.csv	1317562	21650	14884	6351	4428	3728	1137	930	906	136
train_scaled.csv	1317562	21650	14884	6351	4428	3728	1137	930	906	136
train_original_ resampled_smote.csv	21650	21650	21650	21650	21650	21650	21650	21650	21650	21650
train_scaled_ resampled_smote.csv	21650	21650	21650	21650	21650	21650	21650	21650	21650	21650
test_original.csv	348893	6183	4203	1786	1797	1338	297	354	338	34
test_scaled.csv	348893	6183	4203	1786	1797	1338	297	354	338	34

6.1.2. DAE Unsupervised Learning

Just like in section 5.1.2., I trained a DAE on **train_original.csv** (without labels). The model architecture of the proposed DAE is illustrated in Figure 32. For more details about the training process, refer to section 3.1.2. It took around 6 min to train the DAE on the CPU. After training the DAE, samples in **train_original_resampled_smote.csv** and **test_original.csv** were input to the model, and the reconstruction error was noted. In this work, MSE was used to compute the reconstruction error of a sample.

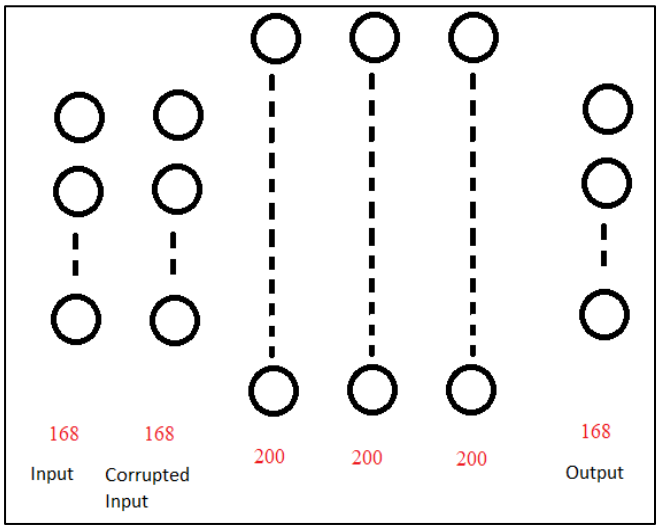


Figure 32: Proposed deep stack DAE architecture used for anomaly score based binary classification on UNSW-NB15.

6.1.3. VIME Unsupervised Learning

Just like in section 5.1.3., I trained a VIME on **train_scaled.csv** (without labels). For more details about model architecture and training process, refer to section 3.1.3. The model architecture of the proposed VIME is illustrated in Figure 33. It took around 27 min to train the VIME. After training the VIME, samples in **train_scaled_resampled_smote.csv** and **test_scaled.csv** were input to the model, and the reconstruction error was noted. In this work, MSE was used to compute the reconstruction error of a sample.

```
Model: "model"
_________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================
 input_1 (InputLayer)        [(None, 168)]             0           []

 dense (Dense)               (None, 200)               33800       ['input_1[0][0]']

 dense_1 (Dense)             (None, 200)               40200       ['dense[0][0]']

 dense_2 (Dense)             (None, 200)               40200       ['dense_1[0][0]']

 mask (Dense)                (None, 168)               33768       ['dense_2[0][0]']

 feature (Dense)             (None, 168)               33768       ['dense_2[0][0]']

=================================================================================
Total params: 181,736
Trainable params: 181,736
Non-trainable params: 0
```

Figure 33: Proposed VIME model architecture used for anomaly score based binary classification on UNSW-NB15.

6.1.4. Ensemble Model

The ensemble model architecture presented in section 5.1.4., was used here as well. It took around 5 min to train the ANN (arbitrator) on reconstruction error features on the CPU. Keras Tuner library [11] was used for tuning the ANN hyperparameters.

6.2. Results

6.2.1. DAE Unsupervised Learning

Figure 34 presents the reconstruction loss vs epoch curve for the DAE model trained on **train_original.csv**.

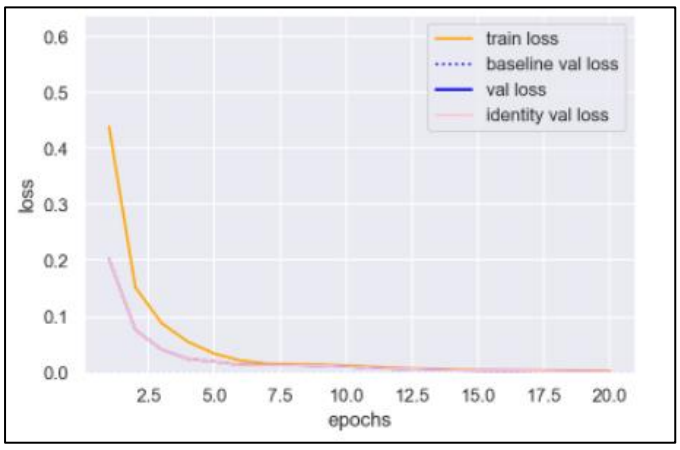


Figure 34: Loss curve for DAE unsupervised training used for anomaly score based binary classification on UNSW-NB15.

6.2.2. VIME Unsupervised Learning

Figure 35 presents the mask loss vs epoch, feature/reconstruction loss vs epoch, and total loss vs epoch curves for the unsupervised VIME model trained on **train_scaled.csv**.

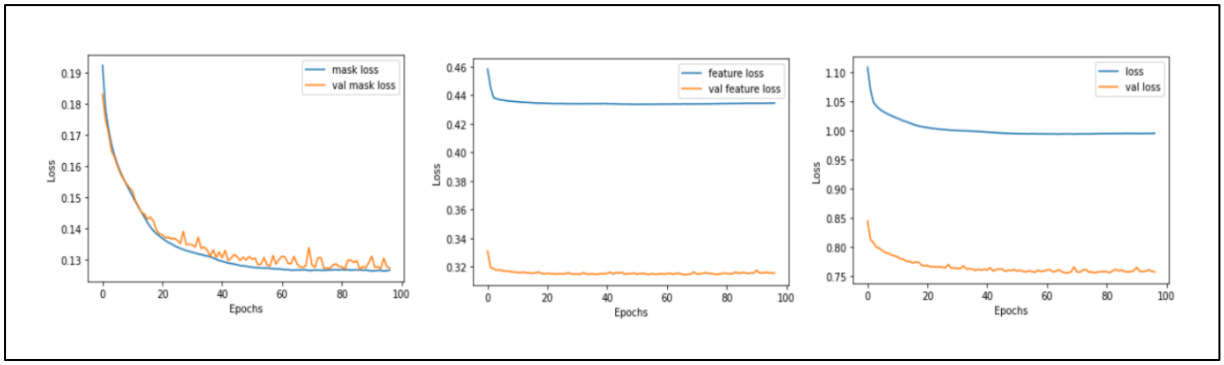


Figure 35: Loss curves for VIME unsupervised training for anomaly score based binary classification on UNSW-NB15.

6.2.3. Ensemble Model

The training and validation loss curves for the ANN model trained on DAE and VIME reconstruction error features are shown in Figure 36. The classification report and confusion matrix of the proposed ensemble model on the test set are shown in Figure 37.

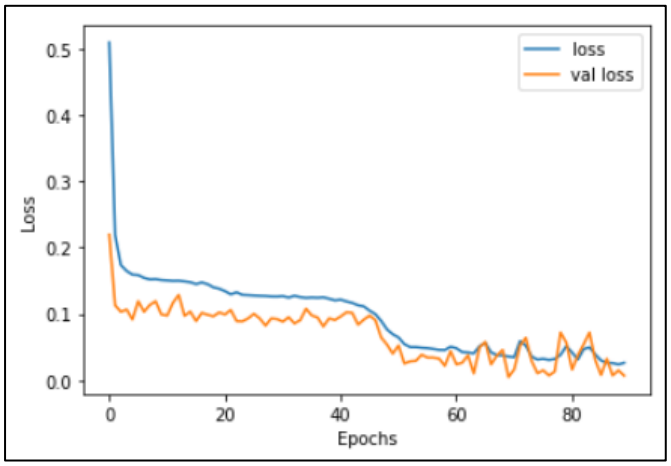
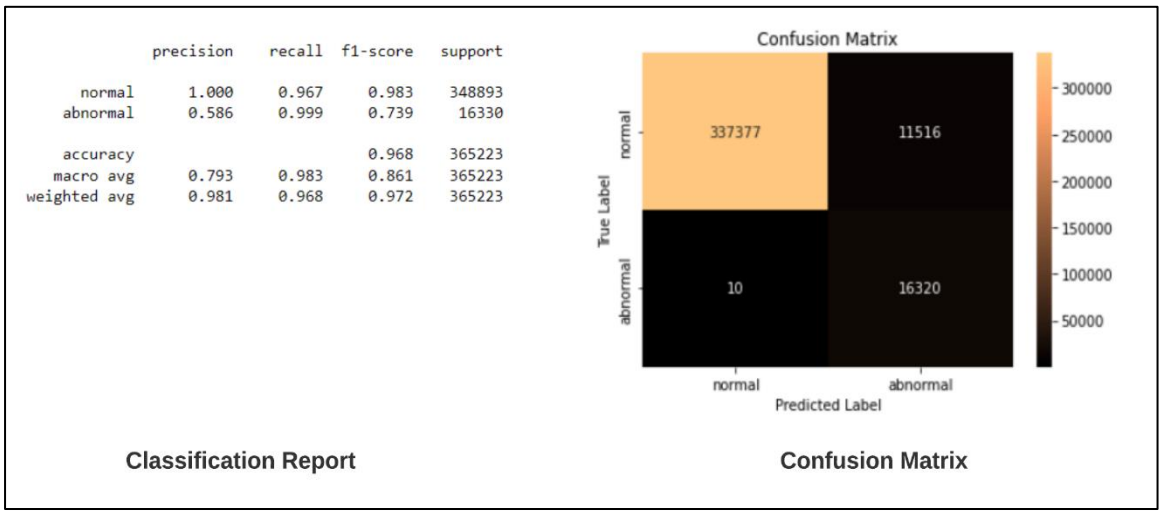


Figure 36: Loss curve for ANN trained on reconstruction error features for anomaly score based binary classification on UNSW-NB15.



	precision	recall	f1-score	support
normal	1.000	0.967	0.983	348893
abnormal	0.586	0.999	0.739	16330
accuracy			0.968	365223
macro avg	0.793	0.983	0.861	365223
weighted avg	0.981	0.968	0.972	365223

Figure 37: Performance of proposed ensemble model on testing dataset for anomaly score based binary classification on UNSW-NB15.

7. CONCLUSION AND FUTURE WORK

In this work, an intelligent intrusion detection system for network anomaly detection and attack classification was created with the help of unsupervised and self-supervised machine learning techniques. A novel ensemble framework for unsupervised network anomaly detection was proposed. For this, two autoencoders that can effectively detect network anomalies were developed and integrated into the ensemble framework. To determine the final prediction of a traffic sample, a weighting scheme was proposed using ANN that allows the framework to quantify each model's goodness and then predicts the label of the sample based on the output of the individual models. In addition, two novel label-free self-supervised learning methods were introduced to learn better representations for features in data and as a means to deal with the phenomenon of intrusion class imbalance. Extensive experiments were carried out on two widely used open-source datasets namely UNSW-NB15 and CICIDS-2017. In the future, I would like to experiment with other commonly used open-source datasets for network intrusion detection and attack classification such as IPS stratosphere, Kitsune, and KDDCUP99.

8. LEARNING OUTCOMES

I have learned a lot in this internship project:

8.1. Technical

- Differences between Supervised, Semi-supervised, and Self-supervised learning.
- Variations of autoencoder available and limitations and advantages of each.
- Different ways to handle class imbalance like class weights, focal loss, SMOTE oversampling, ADASYN oversampling, and adding more samples using VAE.
- Fundamentals of network security like Source IP, Destination IP, Source Port, Destination Port, packets, and various types of network attacks.
- Various types of error metrics like Mean Squared Error, Mean Absolute Error, and Mahalanobis distance available and limitations and advantages of each.

8.2. Non-Technical

- Work and build strong relationships with teammates from other cultures like Singapore, China, and Indonesia.
- Function effectively as a member of an organization.
- Write Minutes of Meeting (MoM).
- Write Status Update e-mail at the end of the day.
- Develop a timeline for a project using the Gantt chart and Waterfall Software Development Life Cycle (SDLC) model.

REFERENCES

1. https://www.sciencedirect.com/science/article/pii/S1877042813039554
2. https://search.ieice.org/bin/summary.php?id=e100-d_8_1729
3. https://www.sciencedirect.com/science/article/abs/pii/S1570870519311035
4. https://ieeexplore.ieee.org/document/9311173
5. https://link.springer.com/article/10.1007/s11042-017-4554-8
6. https://link.springer.com/article/10.1007/s10916-017-0814-4
7. https://www.unb.ca/cic/datasets/ids-2017.html
8. https://research.unsw.edu.au/projects/unsw-nb15-dataset
9. https://towardsdatascience.com/how-to-apply-self-supervision-to-tabular-data-introducing-dfencoder-eec21c4afaef
10. https://github.com/AlliedToasters/dfencoder
11. https://www.tensorflow.org/tutorials/keras/keras_tuner
12. https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/discussion/44629
13. https://proceedings.neurips.cc//paper/2020/file/7d97667a3e056acab9aaf653807b4a03-Paper.pdf
14. https://github.com/jsyoon0823/VIME
15. https://link.springer.com/article/10.1007/s13748-021-00243-5
16. https://www.sciencedirect.com/science/article/abs/pii/S002002552100462X#:~:text=It%20learns%20an%20intrusion%20detection,network%20flows%20and%20attacks%2C%20respectively.
17. https://ieeexplore.ieee.org/document/9527982