Configuration Manual

MSc Research Project
Cyber Security

Baran Diloglu
Student ID: 20221142

School of Computing
National College of Ireland

Supervisor: Imran Khan

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Baran Diloglu
Student ID:	20221142
Programme:	Cyber Security
Year:	2021
Module:	MSc Research Project
Supervisor:	Imran Khan
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	490
Page Count:	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Baran Diloglu
20221142

1 KDD'99 Dataset Unsupervised Deep Learning Implementation

Before starting the experimental process of the research, there some requirements as software and library. Users need to have software and libraries as can be seen in Table 1.

Software & Library Names
Python 3
Jupyter-Lab
pandas
numpy
tensorflow
sklearn

Table 1: Software & Library Names Requirements for KDD'99

```
import pandas as pd
import numpy as np

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

from sklearn import metrics
from sklearn.model_selection import train_test_split
```

Figure 1: Importing Libraries

```
df = pd.read_csv('kddcup.data_10_percent_corrected', header=None)

df.dropna(inplace=True, axis=1)
```

Figure 2: Importing Dataset

After importing libraries before adding the feature columns to the dataset, null columns are dropped.

```python
df.columns = [
    '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',
    'outcome'
]
```

Figure 3: KDD'99 Dataset Feature Setup

```python
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
```

Figure 4: KDD'99 Dataset Encoding Algorithm

Before creating deep learning model, dataset need to be encoded to numbers. After setting up encoding algorithm, all features need to be encoded with following Figure 10.

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

Figure 5: KDD'99 Dataset Encoding Features

```python
normal_mask = df['outcome']=='normal.'
attack_mask = df['outcome']!='normal.'

df.drop('outcome',axis=1,inplace=True)

df_Normal_Data = df[normal_mask]
df_Attack_Data = df[attack_mask]

print(f"Normal count: {len(df_Normal_Data)}")
print(f"Attack count: {len(df_Attack_Data)}")
```

Figure 6: KDD'99 Dataset Seperating Attacks

Normal tagged and attack tagged features need to be seperated into different dataframe to train the deep learning model. While doing that 'outcome' label has been dropped in order to create unsupervised deep learning model.

```python
x_normal_train, x_normal_test = train_test_split(x_normal, test_size=0.25, random_state=42)
```

```python
print(f"Normal Train Count: {len(x_normal_train)}")
print(f"Normal Test Count: {len(x_normal_test)}")
```

```
Normal Train Count: 72958
Normal Test Count: 24320
```

```python
monitor = EarlyStopping(monitor='loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)

model = Sequential()
model.add(Dense(25, input_dim=x_normal.shape[1], activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(x_normal.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_normal_train, x_normal_train, verbose=1, epochs=100, callbacks=[monitor])
```

Figure 7: KDD'99 Dataset Training First Deep Learning Model

```
prediction = model.predict(x_normal_test)
predScore_1 = np.sqrt(metrics.mean_squared_error(prediction,x_normal_test))

prediction = model.predict(x_normal)
predScore_2 = np.sqrt(metrics.mean_squared_error(prediction,x_normal))

prediction = model.predict(x_attack)
predScore_3 = np.sqrt(metrics.mean_squared_error(prediction,x_attack))

print(f"Out of Sample Normal Score (RMSE): {predScore_1}".format(predScore_1))
print(f"All Sample Normal Score (RMSE): {predScore_2}")
print(f"Attack Underway Score (RMSE): {predScore_3}")
760/760 [==============================] - 1s 643us/step
3040/3040 [==============================] - 2s 792us/step
12399/12399 [==============================] - 8s 643us/step
Out of Sample Normal Score (RMSE): 0.42966980763720186
All Sample Normal Score (RMSE): 0.415528409012631
Attack Underway Score (RMSE): 0.5196021720180707
```

Figure 8: KDD'99 Dataset Deep Learning Model Results

After training dataset, it can be seen with using RMSE how accurate the first autoencoder model.

```
model = Sequential()

model.add(Dense(60, input_dim=x_normal.shape[1], activation='relu'))
model.add(BatchNormalization())
model.add(Dense(20, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(60, activation='relu'))
model.add(Dense(x_normal.shape[1]))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_normal_train,x_normal_train,verbose=1,epochs=100,callbacks=[monitor])
```

Figure 9: KDD'99 Dataset Deep Learning Model with Batch Normalization

Batch Normalization added to the model.

```
prediction = model.predict(x_normal_test)
predScore_1 = np.sqrt(metrics.mean_squared_error(prediction,x_normal_test))

prediction = model.predict(x_normal)
predScore_2 = np.sqrt(metrics.mean_squared_error(prediction,x_normal))

prediction = model.predict(x_attack)
predScore_3 = np.sqrt(metrics.mean_squared_error(prediction,x_attack))

print(f"Out of Sample Normal Score (RMSE): {predScore_1}".format(predScore_1))
print(f"All Sample Normal Score (RMSE): {predScore_2}")
print(f"Attack Underway Score (RMSE): {predScore_3}")
760/760 [==============================] - 3s 4ms/step
3040/3040 [==============================] - 8s 3ms/step
12399/12399 [==============================] - 26s 2ms/step
Out of Sample Normal Score (RMSE): 0.3766195983323462
All Sample Normal Score (RMSE): 0.394510005420685
Attack Underway Score (RMSE): 0.5214187512340556
```

Figure 10: KDD'99 Dataset Deep Learning Model Scores with Batch Normalization

2 KDD'99 Dataset Supervised Deep Learning Implementation

```python
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn import metrics

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
```

```python
features = [
    "duration",
    "protocol_type",
    "service",
    "flag",
    "src_bytes",
    "dst_bytes",
    "land",
    "wrong_fragment",
```

Figure 11: Importing Libraries

Importing libraries and dataset features are same with Section 1.

```python
for name in df.columns:
  if name == 'outcome':
    pass
  elif name in ['protocol_type','service','flag','land','logged_in',
                'is_host_login','is_guest_login']:
    encode_text_dummy(df,name)
  else:
    encode_numeric_zscore(df,name)

df.dropna(inplace=True,axis=1)

x_columns = df.columns.drop('outcome')
x = df[x_columns].values
dummies = pd.get_dummies(df['outcome'])
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values
df
```

	duration	src_bytes	dst_bytes	...	is_host_login-0	is_guest_login-0	is_guest_login-1
0	-0.067792	-0.002879	0.138664	...	1	1	0
1	-0.067792	-0.002820	-0.011578	...	1	1	0
...	...	...	...	...	...	...	...
494019	-0.067792	-0.002767	0.010032	...	1	1	0
494020	-0.067792	-0.002840	0.011061	...	1	1	0

Figure 12: KDD'99 Dataset Encoding Features

Encoding features are same with the first section, the main difference will be keeping outcome label in the dataset to create supervised deep learning model, which is artificial neural network in this case.

```python
model = Sequential()

model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], activation='relu'))
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.add(Dense(y.shape[1],activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
model.fit(X_train,y_train,validation_data=(X_test,y_test),callbacks=[monitor],verbose=2,epochs=1000)
```

```python
pred = model.predict(X_test)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test,axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation Score: {}".format(score))
```

```
3860/3860 [==============================] - 6s 1ms/step
Validation Score: 0.9940731624374525
```

Figure 13: KDD'99 Dataset ANN Training and Results

3 UNSW-NB15 Dataset Unsupervised Deep Learning Implementation

Before starting the experimental process of this section, there some requirements as software and library. Users need to have software and libraries as can be seen in Table 2.

Software & Library Names
Python 3
Jupyter-Lab
pandas
numpy
seaborn, matplotlib
keras
sklearn
xgboost, catboost, lightgbm, hdbscan
pyod

Table 2: Software & Library Names Requirements for UNSW-NB15

```
%config IPCompleter.greedy=True
import pandas as pd
import seaborn as sns
import numpy as np

import matplotlib as matplot
import matplotlib.pyplot as plt
%matplotlib inline

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.filterwarnings("ignore")

from keras import Sequential
from keras.models import Model, load_model
from keras.layers import *
from keras.callbacks import ModelCheckpoint
from keras import regularizers

from sklearn.metrics import *
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,normalize


import xgboost, lightgbm
```

```
train = pd.read_csv('UNSW-NB15 - CSV Files/a part of training and testing set/UNSW_NB15_training-set.csv')
test = pd.read_csv('UNSW-NB15 - CSV Files/a part of training and testing set/UNSW_NB15_testing-set.csv')
df = pd.concat([train, test]).drop(['id'],axis=1)
```

Figure 14: Importing Libraries & Dataset

Dataset file were combined because of they have been sent into two parts as 'training' and 'testing'. After this process contamination was checked to see if there are any overlaps or data corruptions after combination of these two files (28).

```
tmp = train.where(train['attack_cat'] == "Normal").dropna()
contamination = round(1 - len(tmp)/len(train), 2)
print("Contamination of Train --> ", contamination)

tmp = test.where(test['attack_cat'] == "Normal").dropna()
print("Contamination of Test --> ", round(1 - len(tmp)/len(test),2),'\n')

if contamination > 0.5:
    print(f'Contamination is {contamination}, it is greater than 0.5. Reducing the contamination.')
    contamination = round(1-contamination,2)
    print(f'Reduced Contamination --> {contamination}')
```

```
Contamination of Train -->  0.55
Contamination of Test -->  0.68

Contamination is 0.55, it is greater than 0.5. Reducing the contamination.
Reduced Contamination --> 0.45
```

```
le1 = LabelEncoder()
le = LabelEncoder()

vector = df['attack_cat']

print("Attack Cat:", set(list(vector)))

df['attack_cat'] = le1.fit_transform(vector)
df['proto'] = le.fit_transform(df['proto'])
df['service'] = le.fit_transform(df['service'])
df['state'] = le.fit_transform(df['state'])

vector = df['attack_cat']
print('\n Description of attack_type: ')
print('------------------------')
print("Min", vector.min())
print("Max", vector.max())
print("Mode",vector.mode(), "\nWhich is,", le1.inverse_transform(vector.mode()))
print("Mode", len(np.where(vector.values==6)[0])/len(vector),"%")
```

```
Attack Cat: {'Exploits', 'Fuzzers', 'DoS', 'Backdoor', 'Normal', 'Worms', 'Shellcode', 'Reconnaissance', 'Generic', 'Analysis'}
```

Figure 15: Contamination Check & Fixing

```
from pyod.models import lof, cblof, cof, pca, iforest, knn, ocsvm, abod
contamination = 0.4
threshold = 0.75
```

Local Outlier Factor

```
lof_clf = lof.LOF(contamination=contamination, n_jobs=-1)
lof_clf = lof_clf.fit(X_train[:50000])

predictions = lof_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = lof_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.52235
F1_weighted of train: 0.52897
Acc of test: 0.52797
F1_weighted of train: 0.53507
```

Cluster-based Local Outlier Factor

```
cblof_clf = cblof.CBLOF(contamination=contamination, n_jobs=-1, n_clusters=45)
cblof_clf = cblof_clf.fit(X_train)

predictions = cblof_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = cblof_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.37370
F1_weighted of train: 0.37962
Acc of test: 0.37324
F1_weighted of train: 0.37898
```

Figure 16: Importing PYOD Library & Testing PYOD Algorithms

Covalent Organic Frameworks

```
cof_clf = cof.COF(contamination=contamination)
cof_clf = cof_clf.fit(X_train[:5000])

predictions = cof_clf.predict(X_train[:5000])
print(f'Acc of train: {accuracy_score(y_train[:5000], predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train[:5000], predictions, average='weighted'):.5f}")

predictions = cof_clf.predict(X_test[:10000])
print(f'Acc of test: {accuracy_score(y_test[:10000], predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test[:10000], predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.55080
F1_weighted of train: 0.55548
Acc of test: 0.53100
F1_weighted of train: 0.53140
```

Principal Component Analysis

```
pca_clf = pca.PCA(contamination=contamination)
pca_clf = pca_clf.fit(X_train)

predictions = pca_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = pca_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.45786
F1_weighted of train: 0.46299
Acc of test: 0.45947
F1_weighted of train: 0.46453
```

Figure 17: COF & PCA Testing

IsolationForest Outlier Detector

```
iforest_clf = iforest.IForest(contamination=contamination, n_estimators=300, max_samples= 1028
iforest_clf = iforest_clf.fit(X_train)

predictions = iforest_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = iforest_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.35531
F1_weighted of train: 0.36140
Acc of test: 0.35731
F1_weighted of train: 0.36320
```

K-Nearest Neighbour

```
knn_clf = knn.KNN(contamination=contamination, radius=1.5, n_neighbors=20, n_jobs=-1)
knn_clf = knn_clf.fit(X_train)

predictions = knn_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = knn_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.40278
F1_weighted of train: 0.40787
Acc of test: 0.39641
F1_weighted of train: 0.40189
```

Figure 18: IForest & KNN Testing

Hierarchical Density-Based Spatial Clustering of Applications with Noise

```
hdbScan_clf = HDBSCAN(10, leaf_size=80)
hdbScan_clf = hdbScan_clf.fit(X_train)

hdbScan_clf_outliers = hdbScan_clf.outlier_scores_ > 0.15
print(accuracy_score(y_train, hdbScan_clf_outliers))

print('')

hdbScan_clf_outliers = hdbScan_clf.outlier_scores_ > 0.45
print(accuracy_score(y_train, hdbScan_clf_outliers))
```

```
13.0728 % outliers
0.32991976248920624

3.0009 % outliers
0.3609960317845327
```

Figure 19: OCSVM & ABOD Testing

One-Class Support Vector Machines

```python
ocsvm_clf = ocsvm.OCSVM(contamination=contamination)
ocsvm_clf = ocsvm_clf.fit(X_train[:2000])

predictions = ocsvm_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_train, predictions, average='weighted'):.5f}")

predictions = ocsvm_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions):.5f}')
print(f"F1_weighted of train: {f1_score(y_test, predictions, average='weighted'):.5f}")
```

```
Acc of train: 0.55653
F1_weighted of train: 0.47694
Acc of test: 0.54994
F1_weighted of train: 0.46759
```

Angle-based Outlier Detection (ABOD) evaluates the degree of outlierness on the variance of the angles (VOA) between a point and all other pairs of points in the data set.

```python
X_temp = X_train.astype(np.float)
abod_clf = abod.ABOD(contamination=contamination, n_neighbors=10, )
abod_clf = abod_clf.fit(X_temp)

predictions = abod_clf.predict(X_train)
print(f'Acc of train: {accuracy_score(y_train, predictions)}')

predictions = abod_clf.predict(X_test)
print(f'Acc of test: {accuracy_score(y_test, predictions)}')
```

```
Acc of train: 0.3605594310607457
Acc of test: 0.3623750848937615
```

Figure 20: HDBSCAN Testing

A deep neural VAE is quite similar in architecture to a regular AE. The main difference is that the core of a VAE has a layer of data means and standard deviations. These means and standard deviations are used to generate the core representations values.

```python
from pyod.models import vae
```

```python
data = X_train
dim = X_train.shape[1]

model = vae.VAE(contamination=contamination, encoder_neurons=[dim, dim-1],decoder_neurons=[dim-1,dim], epochs=30)
model = model.fit(data)
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 42)]	0	[]
dense (Dense)	(None, 42)	1806	['input_1[0][0]']
dense_1 (Dense)	(None, 42)	1806	['dense[0][0]']
dropout (Dropout)	(None, 42)	0	['dense_1[0][0]']
dense_2 (Dense)	(None, 41)	1763	['dropout[0][0]']
dropout_1 (Dropout)	(None, 41)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 2)	84	['dropout_1[0][0]']
dense_4 (Dense)	(None, 2)	84	['dropout_1[0][0]']
lambda (Lambda)	(None, 2)	0	['dense_3[0][0]', 'dense_4[0][0]']

Figure 21: Creating Autoencoder with VAE

```
Epoch 23/30
5798/5798 [==============================] - 10s 2ms/step - loss: 42.0917 - val_loss: 41.1609
Epoch 24/30
5798/5798 [==============================] - 11s 2ms/step - loss: 42.0881 - val_loss: 41.1609
Epoch 25/30
5798/5798 [==============================] - 10s 2ms/step - loss: 42.0880 - val_loss: 41.1609
Epoch 26/30
5798/5798 [==============================] - 10s 2ms/step - loss: 42.0878 - val_loss: 41.1609
Epoch 27/30
5798/5798 [==============================] - 10s 2ms/step - loss: 42.0885 - val_loss: 41.1609
Epoch 28/30
5798/5798 [==============================] - 11s 2ms/step - loss: 42.0885 - val_loss: 41.1609
Epoch 29/30
5798/5798 [==============================] - 11s 2ms/step - loss: 42.0878 - val_loss: 41.1609
Epoch 30/30
5798/5798 [==============================] - 10s 2ms/step - loss: 42.0890 - val_loss: 41.1609
6442/6442 [==============================] - 6s 907us/step
```

Figure 22: Loss of Autoencoders with this Dataset

4 UNSW-NB15 Dataset Supervised Deep Learning Implementation

Dataset file were combined because of they have been sent into two parts as 'training' and 'testing'. After this process contamination was checked to see if there are any overlaps or data corruptions after combination of these two files (28).

```
%config IPCompleter.greedy=True
import pandas as pd
import seaborn as sns
import numpy as np

import matplotlib as matplot
import matplotlib.pyplot as plt
%matplotlib inline

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.filterwarnings("ignore")

from keras import Sequential
from keras.models import Model, load_model
from keras.layers import *
from keras.callbacks import ModelCheckpoint
from keras import regularizers

from sklearn.metrics import *
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,normalize


import xgboost, lightgbm
```

```
train = pd.read_csv('UNSW-NB15 - CSV Files/a part of training and testing set/UNSW_NB15_training-set.csv')
test = pd.read_csv('UNSW-NB15 - CSV Files/a part of training and testing set/UNSW_NB15_testing-set.csv')
df = pd.concat([train, test]).drop(['id'],axis=1)
```

Figure 23: Importing Libraries & Dataset

```
tmp = train.where(train['attack_cat'] == "Normal").dropna()
contamination = round(1 - len(tmp)/len(train), 2)
print("Contamination of Train --> ", contamination)

tmp = test.where(test['attack_cat'] == "Normal").dropna()
print("Contamination of Test --> ", round(1 - len(tmp)/len(test),2),'\n')

if contamination > 0.5:
    print(f'Contamination is {contamination}, it is greater than 0.5. Reducing the contamination.')
    contamination = round(1-contamination,2)
    print(f'Reduced Contamination --> {contamination}')
```

```
Contamination of Train -->  0.55
Contamination of Test -->  0.68

Contamination is 0.55, it is greater than 0.5. Reducing the contamination.
Reduced Contamination --> 0.45
```

```
le1 = LabelEncoder()
le = LabelEncoder()

vector = df['attack_cat']

print("Attack Cat:", set(list(vector)))

df['attack_cat'] = le1.fit_transform(vector)
df['proto'] = le.fit_transform(df['proto'])
df['service'] = le.fit_transform(df['service'])
df['state'] = le.fit_transform(df['state'])

vector = df['attack_cat']
print('\n Description of attack_type: ')
print('---------------------------')
print("Min", vector.min())
print("Max", vector.max())
print("Mode",vector.mode(), "\nWhich is,", le1.inverse_transform(vector.mode()))
print("Mode", len(np.where(vector.values==6)[0])/len(vector),"%")
```

```
Attack Cat: {'Exploits', 'Fuzzers', 'DoS', 'Backdoor', 'Normal', 'Worms', 'Shellcode', 'Reconnaissance', 'Generic', 'Analysis'}
```

Figure 24: Contamination Check & Fixing

With using Label Encoder attack categories were fitted into another data frame.

```
lowSTD = list(df.std().to_frame().nsmallest(6, columns=0).index)
lowCORR = list(df.corr().abs().sort_values('attack_cat')['attack_cat'].nsmallest(3).index)

lowSTD
lowCORR
```

```
['ackdat', 'synack', 'tcprtt', 'is_ftp_login', 'ct_ftp_cmd', 'is_sm_ips_ports']

['sjit', 'response_body_len', 'djit']
```

```
drop = set( lowCORR + lowSTD)
drop = {'ackdat', 'ct_ftp_cmd', 'djit', 'is_ftp_login', 'is_sm_ips_ports', 'response_body_len', 'sjit', 'synack', 'tcprtt'}
```

Figure 25: Low Correlated Feature Removal

```
RFC = RandomForestClassifier(n_estimators=150, random_state=42, n_jobs=-1)

RFC = RFC.fit(X_train,y_train)
pred = RFC.score(X_test, y_test)
name = str(type(RFC)).split(".")[-1][:-2]
print("Acc: %0.5f for the %s" % (pred, name))
```

```
Acc: 0.95166 for the RandomForestClassifier
```

```
ETC = ExtraTreesClassifier(n_estimators=200, random_state=42, n_jobs=-1)

ETC = ETC.fit(X_train,y_train)
pred = ETC.score(X_test, y_test)
name = str(type(ETC)).split(".")[-1][:-2]
print("Acc: %0.5f for the %s" % (pred, name))
```

```
Acc: 0.94947 for the ExtraTreesClassifier
```

```
XGB = xgboost.XGBClassifier(n_estimators=150, n_jobs=-1)

XGB = XGB.fit(X_train,y_train)
pred = XGB.score(X_test, y_test)
name = str(type(XGB)).split(".")[-1][:-2]
print("Acc: %0.5f for the %s" % (pred, name))
```

```
Acc: 0.95019 for the XGBClassifier
```

```
GBM = lightgbm.LGBMClassifier(objective='binary', n_estimators= 500, n_jobs=-1)

GBM = GBM.fit(X_train,y_train)
pred = GBM.score(X_test, y_test)
name = str(type(GBM)).split(".")[-1][:-2]
print("Acc: %0.5f for the %s" % (pred, name))
```

```
Acc: 0.95100 for the LGBMClassifier
```

Figure 26: Machine Learning Classifiers & Results

```
import catboost

CBC = catboost.CatBoostClassifier(iterations=3000, eval_metric='AUC', use_best_model=True, task_type="CPU", devices='0:1', random_seed=42, verbose=False)
CBC = CBC.fit(X_train,y_train, eval_set=(X_test, y_test))
pred = CBC.score(X_test,y_test)
name = str(type(CBC)).split(".")[-1][:-2]
print("Acc: %0.5f for the %s" % (pred, name))
```

```
Acc: 0.95306 for the CatBoostClassifier
```

Figure 27: Using CatBoost Machine Learning

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
dim = X_train.shape[1]
```

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train = mms.fit_transform(X_train)
X_test = mms.transform(X_test)
```

```
classifier = Sequential()

classifier.add(Dense(42, activation='relu', input_dim=dim))

classifier.add(Dense(64, activation='relu'))
classifier.add(Dropout(0.07))
classifier.add(Dense(42, activation='relu' ))
classifier.add(Dropout(0.07))
classifier.add(Dense(25, activation='relu'))

classifier.add(Dense(1, activation='sigmoid'))

classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
```

```
history = classifier.fit(X_train,y_train, batch_size=64, epochs=15, validation_data=(X_test,y_test)).history
```

Figure 28: ANN Creation

5 CIC-IDS2017 Dataset Supervised Deep Learning Implementation

Software & Library Names
Python 3
Jupyter-Lab
pandas
numpy
seaborn, matplotlib
sklearn
keras
tensorflow

Table 3: Software & Library Names Requirements for CIC-IDS-2017

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import random
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.tree import export_text
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import GridSearchCV

import base64
from facets_overview.generic_feature_statistics_generator import GenericFeatureStatisticsGenerator

from keras import Sequential
from keras.models import Model, load_model
from keras.layers import *
from keras.callbacks import ModelCheckpoint
from keras import regularizers

from sklearn.metrics import *
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import normalize
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
```

```
df = pd.read_csv('MachineLearningCVE/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv', engine='python')
```

Figure 29: Importing Libraries & Dataset

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
df[string_features] = df[string_features].apply(lambda col: le.fit_transform(col))
```

```
benign_total = len(df[df['Label'] == "BENIGN"])
benign_total
```

```
168186
```

```
attack_total = len(df[df['Label'] != "BENIGN"])
attack_total
```

```
2180
```

Figure 30: Encoding Features & Checking Number of Features

To get a better result, benign number are limited to 2.5 times of attack values (31).

```
enlargement = 1.1
benign_included_max = attack_total / 30 * 70
benign_inc_probability = (benign_included_max / benign_total) * enlargement
print(benign_included_max, benign_inc_probability)
```

```
5086.666666666667 0.03326872232726466
```

```
indexes = []
benign_included_count = 0
for index, row in df.iterrows():
    if (row['Label'] != "BENIGN"):
        indexes.append(index)
    else:
        if random.random() > benign_inc_probability: continue
        if benign_included_count > benign_included_max: continue
        benign_included_count += 1
        indexes.append(index)
df_balanced = df.loc[indexes]
```

```
df_balanced['Label'].value_counts()
```

```
BENIGN                        5087
Web Attack � Brute Force      1507
Web Attack � XSS               652
Web Attack � Sql Injection      21
Name: Label, dtype: int64
```

Figure 31: Balancing Benign & Attack Values

Unrelevant features are removed from the dataset before looking at correlation matrix (39).

```
excluded = ['Flow ID', 'Source IP', 'Source Port', 'Destination IP', 'Destination Port', 'Protocol', 'Timestamp']
df = df.drop(columns=excluded, errors='ignore')
```

```
excluded2 = ['Init_Win_bytes_backward', 'Init_Win_bytes_forward']
df = df.drop(columns=excluded2, errors='ignore')
```

```
y = df['Label'].values
X = df.drop(columns=['Label'])
print(X.shape, y.shape)
```

```
(7267, 76) (7267,)
```

Figure 32: Feature Removal

For feature elimination, feature importance library was used to decide.

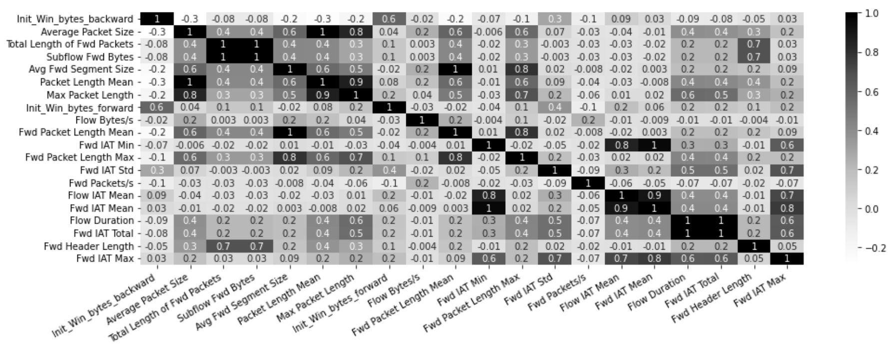
```python
features = X.columns
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
webattack_features = []

for index, i in enumerate(indices[:20]):
    webattack_features.append(features[i])
    print('{}.\t#{}\t{:.3f}\t{}'.format(index + 1, i, importances[i], features[i]))
```

```
1.      #65     0.119   Init_Win_bytes_backward
2.      #51     0.057   Average Packet Size
3.      #3      0.057   Total Length of Fwd Packets
4.      #61     0.054   Subflow Fwd Bytes
5.      #52     0.054   Avg Fwd Segment Size
6.      #39     0.050   Packet Length Mean
7.      #38     0.049   Max Packet Length
8.      #64     0.046   Init_Win_bytes_forward
9.      #13     0.046   Flow Bytes/s
10.     #7      0.038   Fwd Packet Length Mean
11.     #23     0.030   Fwd IAT Min
12.     #5      0.030   Fwd Packet Length Max
13.     #21     0.027   Fwd IAT Std
14.     #35     0.024   Fwd Packets/s
15.     #15     0.022   Flow IAT Mean
16.     #20     0.022   Fwd IAT Mean
17.     #0      0.019   Flow Duration
18.     #19     0.017   Fwd IAT Total
19.     #33     0.017   Fwd Header Length
20.     #22     0.016   Fwd IAT Max
```

Figure 33: Feature Importance

After getting results of feature elimination, correlation matrix was checked to make sure.

```python
corr_matrix = df[webattack_features].corr()
plt.rcParams['figure.figsize'] = (16, 5)
g = sns.heatmap(corr_matrix, annot=True, fmt='.1g', cmap='Greys')
g.set_xticklabels(g.get_xticklabels(), verticalalignment='top', horizontalalignment='right', rotation=30);
plt.savefig('cic_corr_heatmap.png', dpi=300, bbox_inches='tight')
```




```python
selected_features = {'Packet Length Mean', 'Avg Fwd Segment Size', 'Subflow Fwd Bytes',
                     'Fwd Packets/s', 'Fwd IAT Total', 'Fwd IAT Max'}
webattack_features = [item for item in webattack_features if item not in selected_features]
webattack_features = webattack_features[:10]
webattack_features
```

Figure 34: Feature Selection

```
df = pd.read_csv('web_attacks_balanced.csv')
df['Label'] = df['Label'].apply(lambda x: 0 if x == 'BENIGN' else 1)
y = df['Label'].values
X = df[webattack_features]
print(X.shape, y.shape)
```

```
(7267, 10) (7267,)
```

```
rfc = RandomForestClassifier(random_state=1)
rfc.get_params().keys()
```

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrease', 'mi
n_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_start'])
```

Figure 35: Creating X & y Values for Neural Network

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(5086, 10) (5086,)
(2181, 10) (2181,)
```

Figure 36: Artificial Neural Network Fitting Train & Test Values

Before training the neural network precision scores and f1-scores were checked to see if data is fit correctly.

```
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred)
print('Accuracy =', accuracy)
print('Precision =', precision)
print('Recall =', recall)
print('F1 =', f1)
```

```
Accuracy = 0.994956441999083
Precision = 0.9939117199391172
Recall = 0.9893939393939394
F1 = 0.9916476841305998
```

Figure 37: Accuracy & F1 Score

Values are scaled with using MinMaxScaler before ANN training.

```
mms = MinMaxScaler()
X_train = mms.fit_transform(X_train)
X_test = mms.transform(X_test)
```

```
dim = X_train.shape[1]
```

```
model = Sequential()

model.add(Dense(42, activation='relu', input_dim=dim))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.02))
model.add(Dense(42, activation='relu' ))
model.add(Dropout(0.02))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.02))
model.add(Dense(12, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])

monitor = EarlyStopping(monitor='loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
```

```
history = model.fit(X_train,y_train, batch_size=64, epochs=100, validation_data=(X_test,y_test), callbacks=[monitor]).history
```

Figure 38: Scaling & Training Network

```
eval_model = model.evaluate(X_train, y_train)
print(eval_model)

eval_model = model.evaluate(X_test, y_test)
print(eval_model)

predictions = model.predict(X_test)
predictions =(predictions>0.80)

mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse,'true_class': y_test.reshape(1,-1)[0]})
error_df.describe()
```

```
159/159 [==============================] - 0s 870us/step - loss: 0.0590 - accuracy: 0.9752
[0.05902234464883804, 0.975226104259491]
228/228 [==============================] - 0s 755us/step - loss: 0.0611 - accuracy: 0.9756
[0.06108376383781433, 0.9756433367729187]
228/228 [==============================] - 0s 616us/step
```

Figure 39: Printing Accuracy Score of Network