In [1]:
```
# import warnings
# warnings.filterwarnings("ignore")

import pandas as pd
import sys
import numpy as np
cols=["srcip","sport","dstip","dsport","proto","state","dur","sbytes","dbytes","s
      "Dload","Spkts","Dpkts","swin","dwin","stcpb","dtcpb","smeansz","dmeansz","
      "Stime","Ltime","Sintpkt","Dintpkt","tcprtt","synack","ackdat","is_sm_ips_p
      "is_ftp_login","ct_ftp_cmd","ct_srv_src","ct_srv_dst","ct_dst_ltm","ct_src_
      "ct_dst_src_ltm","attack_cat","Label"]
# UNSW-NB15
unsw = pd.read_csv("E:/final_prj/dataset/UNSW-NB15/UNSW-NB15_1.csv",low_memory=Fa
# unsw_test = pd.read_csv("E:/final_prj/dataset/UNSW-NB15/set/UNSW_NB15_testing-s
```

In [2]:
```
unsw.head(3)
```

Out[2]:

	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	...	ct_ftp_
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	...	
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	...	
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	...	

3 rows × 49 columns

In [3]:
```
unsw.isnull().sum()
```

Out[3]:
```
srcip                  0
sport                  0
dstip                  0
dsport                 0
proto                  0
state                  0
dur                    0
sbytes                 0
dbytes                 0
sttl                   0
dttl                   0
sloss                  0
dloss                  0
service                0
Sload                  0
Dload                  0
Spkts                  0
Dpkts                  0
swin                   0
dwin                   0
stcpb                  0
dtcpb                  0
smeansz                0
dmeansz                0
trans_depth            0
res_bdy_len            0
Sjit                   0
Djit                   0
Stime                  0
Ltime                  0
Sintpkt                0
Dintpkt                0
tcprtt                 0
synack                 0
ackdat                 0
is_sm_ips_ports        0
ct_state_ttl           0
ct_flw_http_mthd       0
is_ftp_login           0
ct_ftp_cmd             0
ct_srv_src             0
ct_srv_dst             0
ct_dst_ltm             0
ct_src_ ltm            0
ct_src_dport_ltm       0
ct_dst_sport_ltm       0
ct_dst_src_ltm         0
attack_cat        677786
Label                  0
dtype: int64
```

In [4]:
```
unsw.attack_cat = unsw.attack_cat.fillna('Normal')
unsw.head()
```

Out[4]:

	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	...	ct_ftp_
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	...	
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	...	
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	...	
3	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	...	
4	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	...	

5 rows × 49 columns

In [5]:
```
for col_name in unsw.columns:
    if unsw[col_name].dtypes == 'object' :
        unique_cat = len(unsw[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=

print()
print('Distribution of categories in service:')
print(unsw['attack_cat'].value_counts().sort_values(ascending=False))
```

```
Feature 'srcip' has 40 categories
Feature 'sport' has 64541 categories
Feature 'dstip' has 44 categories
Feature 'dsport' has 62222 categories
Feature 'proto' has 135 categories
Feature 'state' has 16 categories
Feature 'service' has 13 categories
Feature 'attack_cat' has 10 categories

Distribution of categories in service:
Normal            677786
Generic             7522
Exploits            5409
 Fuzzers            5051
Reconnaissance      1759
DoS                 1167
Backdoors            534
Analysis             526
Shellcode            223
Worms                 24
Name: attack_cat, dtype: int64
```

In [6]:
```
unsw['attack_cat'].value_counts()
```

Out[6]:
```
Normal            677786
Generic             7522
Exploits            5409
 Fuzzers            5051
Reconnaissance      1759
DoS                 1167
Backdoors            534
Analysis             526
Shellcode            223
Worms                 24
Name: attack_cat, dtype: int64
```

In [7]:
```
from sklearn.preprocessing import LabelEncoder

# for unsw
categorical_columns=['srcip','sport','dstip','dsport','proto', 'state', 'service'
unsw_cat_values = unsw[categorical_columns]
# unsw_test_cat_values =unsw_test[categorical_columns_t]
print(unsw_cat_values.head(2))
# print(unsw_test_cat_values.head(2))
```

```
        srcip  sport          dstip dsport proto state service
0  59.166.0.0   1390  149.171.126.6     53   udp   CON     dns
1  59.166.0.0  33661  149.171.126.9   1024   udp   CON       -
```

In [8]:
```
# changing categorical values to numeric for unsw
# train set
unsw_values_enc=unsw_cat_values.apply(LabelEncoder().fit_transform)
print(unsw_cat_values.head(5))
print('--------------------')
print(unsw_values_enc.head(3))

# # test set
# print()
# unsw_test_values_enc=unsw_test_cat_values.apply(LabelEncoder().fit_transform)
# print(unsw_test_values_enc.head(3))
```

```
        srcip  sport          dstip dsport proto state service
0  59.166.0.0   1390  149.171.126.6     53   udp   CON     dns
1  59.166.0.0  33661  149.171.126.9   1024   udp   CON       -
2  59.166.0.6   1464  149.171.126.7     53   udp   CON     dns
3  59.166.0.5   3593  149.171.126.5     53   udp   CON     dns
4  59.166.0.3  49664  149.171.126.0     53   udp   CON     dns
--------------------
   srcip  sport  dstip  dsport  proto  state  service
0     30   4272     22   45600    120      2        2
1     30  26018     25     239    120      2        0
2     36   5087     23   45600    120      2        2
```

In [9]:
```
# unsw train data set
# print(unsw_train)
unsw.drop(columns=['srcip'],inplace=True)
unsw.drop(columns=['sport'],inplace=True)
unsw.drop(columns=['dstip'],inplace=True)
unsw.drop(columns=['dsport'],inplace=True)
unsw.drop(columns=['proto'],inplace=True)
unsw.drop(columns=['state'],inplace=True)
unsw.drop(columns=['service'],inplace=True)
unsw.drop(columns=['Label'],inplace=True)
print(unsw)
```

```
             dur  sbytes   dbytes  sttl  dttl  sloss  dloss         Sload  \
0       0.001055     132      164    31    29      0      0  5.004739e+05
1       0.036133     528      304    31    29      0      0  8.767609e+04
2       0.001119     146      178    31    29      0      0  5.218945e+05
3       0.001209     132      164    31    29      0      0  4.367246e+05
4       0.001169     146      178    31    29      0      0  4.995722e+05
...          ...     ...      ...   ...   ...    ...    ...           ...
699996  0.020383     320     1874    31    29      1      2  1.047932e+05
699997  1.402957   19410  1087890    31    29      2    370  1.103783e+05
699998  0.007108    2158     2464    31    29      6      6  2.328644e+06
699999  0.004435     568      304    31    29      0      0  7.684329e+05
700000  0.072974    4238    60788    31    29      7     30  4.582454e+05

              Dload  Spkts  ...  is_ftp_login  ct_ftp_cmd  ct_srv_src  \
0       6.218009e+05      2  ...             0           0           3
1       5.048017e+04      4  ...             0           0           2
2       6.362824e+05      2  ...             0           0          12
3       5.425972e+05      2  ...             0           0           6
4       6.090676e+05      2  ...             0           0           7
...              ...    ...  ...           ...         ...         ...
699996  6.436736e+05      6  ...             0           0           8
699997  6.195098e+06    364  ...             0           0           1
699998  2.658413e+06     24  ...             0           0          13
699999  4.112740e+05      4  ...             0           0          10
700000  6.571546e+06     72  ...             0           0          13

        ct_srv_dst  ct_dst_ltm  ct_src_ ltm  ct_src_dport_ltm  \
0                7           1            3                 1
1                4           2            3                 1
2                8           1            2                 2
3                9           1            1                 1
4                9           1            1                 1
...            ...         ...          ...               ...
699996          20           7            5                 1
699997           1           2            7                 2
699998          13           6            7                 2
699999          13           6            5                 1
700000          13           6            7                 1

        ct_dst_sport_ltm  ct_dst_src_ltm  attack_cat
0                      1               1      Normal
1                      1               2      Normal
2                      1               1      Normal
3                      1               1      Normal
```

```
4                     1                1      Normal
...                 ...              ...         ...
699996                1                4      Normal
699997                2                2      Normal
699998                1                2      Normal
699999                1                3      Normal
700000                1                2      Normal

[700001 rows x 41 columns]
```

In [10]:
```
unsw_data= pd.concat([unsw_values_enc,unsw],axis=1)
print(unsw_data.head(3))
```

```
   srcip  sport  dstip  dsport  proto  state  service       dur  sbytes  \
0     30   4272     22   45600    120      2        2  0.001055     132
1     30  26018     25     239    120      2        0  0.036133     528
2     36   5087     23   45600    120      2        2  0.001119     146

   dbytes  ...  is_ftp_login  ct_ftp_cmd  ct_srv_src  ct_srv_dst  ct_dst_ltm
\
0     164  ...             0           0           3           7           1
1     304  ...             0           0           2           4           2
2     178  ...             0           0          12           8           1

   ct_src_ ltm  ct_src_dport_ltm  ct_dst_sport_ltm  ct_dst_src_ltm  attack_ca
t
0            3                 1                 1               1      Norma
l
1            3                 1                 1               2      Norma
l
2            2                 2                 1               1      Norma
l

[3 rows x 48 columns]
```

In [11]:
```
labeldf=unsw_data['attack_cat']
newlabeldf=labeldf.replace({ 'Normal' : 0, 'Generic' : 1 ,'Exploits': 2, ' Fuzzer
                            'Analysis': 6,'Backdoors': 7, 'Shellcode': 8, 'Worms'
unsw_data['attack_cat'] = newlabeldf
```

In [12]:
```
x=unsw_data.drop(['attack_cat'],1)  # features
y=newlabeldf
y=y.astype('int')
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_8548\2988123074.py:1: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the ar
gument 'labels' will be keyword-only.
  x=unsw_data.drop(['attack_cat'],1)  # features
```

In [13]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_
```

In [14]:
```
x_train
```

Out[14]:

	srcip	sport	dstip	dsport	proto	state	service	dur	sbytes	dbytes	...	ct_flw_
669239	36	23159	22	43798	114	5	0	0.375464	4376	3080	...	
689590	31	28051	18	37708	114	5	0	0.272304	8928	320	...	
542354	36	32753	18	45600	120	2	2	0.001045	146	178	...	
441092	37	42444	22	45600	120	2	2	0.001095	146	178	...	
309437	34	31172	22	42543	120	2	0	0.001772	528	304	...	
...	...	...	...	...	...	...	...	...	...	...	...	
371403	39	23605	18	60303	114	5	5	1.126003	1580	7488	...	
491263	38	33499	22	45600	120	2	2	0.001361	146	178	...	
470924	39	50494	20	28347	114	5	0	0.021267	3182	38534	...	
491755	38	6699	6	15932	114	5	9	0.193172	37184	3276	...	
128037	36	27370	19	12756	114	5	11	0.141949	9608	12194	...	

490000 rows × 47 columns

In [15]:
```
y_train
```

Out[15]:
```
669239    0
689590    0
542354    0
441092    0
309437    0
         ..
371403    0
491263    0
470924    0
491755    0
128037    0
Name: attack_cat, Length: 490000, dtype: int32
```

In [16]:
```
num_col = list(unsw_data.select_dtypes(include='number').columns)
# num_col.remove('id')
num_col.remove('attack_cat')
print(num_col)
```

```
['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'service', 'dur', 'sbyt
es', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'Sload', 'Dload', 'Spkts', 'Dp
kts', 'swin', 'dwin', 'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'r
es_bdy_len', 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt', 'Dintpkt', 'tcprtt',
'synack', 'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_
ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ lt
m', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm']
```

In [17]:
```
# train
from sklearn.preprocessing import MinMaxScaler

minmax_scale = MinMaxScaler(feature_range=(0, 1))
def normalization(df,col):
  for i in col:
    arr = df[i]
    arr = np.array(arr)
    df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))
  return df

print("data after normalisation")
x_train = normalization(x_train,num_col)
print(x_train.head(2))
```

```
data after normalisation
           srcip     sport     dstip    dsport     proto     state  service  \
669239  0.923077  0.358822  0.511628  0.703910  0.850746  0.333333      0.0
689590  0.794872  0.434621  0.418605  0.606033  0.850746  0.333333      0.0

              dur    sbytes    dbytes  ...  ct_flw_http_mthd  is_ftp_login  \
669239  0.000043  0.000318  0.000210  ...               0.0           0.0
689590  0.000031  0.000651  0.000022  ...               0.0           0.0

        ct_ftp_cmd  ct_srv_src  ct_srv_dst  ct_dst_ltm  ct_src_ ltm  \
669239         0.0    0.139535    0.390244    0.146341     0.061224
689590         0.0    0.139535    0.317073    0.048780     0.040816

        ct_src_dport_ltm  ct_dst_sport_ltm  ct_dst_src_ltm
669239               0.0               0.0        0.054054
689590               0.0               0.0        0.027027

[2 rows x 47 columns]
```

In [18]:
```
# test
from sklearn.preprocessing import MinMaxScaler

minmax_scale = MinMaxScaler(feature_range=(0, 1))
def normalization(df,col):
  for i in col:
    arr = df[i]
    arr = np.array(arr)
    df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))
  return df

print("data after normalisation")
x_test = normalization(x_test,num_col)
print(x_test.head(2))
```

```
data after normalisation
           srcip    sport     dstip    dsport     proto     state   service  \
511046  0.897436  0.90423  0.488372  0.552498  0.850746  0.333333  0.000000
381461  0.897436  0.23322  0.558140  0.428874  0.850746  0.333333  0.333333

             dur    sbytes    dbytes  ...  ct_flw_http_mthd  is_ftp_login  \
511046  0.000005  0.000210  0.002169  ...               0.0           0.0
381461  0.000025  0.000768  0.000033  ...               0.0           0.0

        ct_ftp_cmd  ct_srv_src  ct_srv_dst  ct_dst_ltm  ct_src_ ltm  \
511046         0.0    0.023256    0.097561    0.097561     0.122449
381461         0.0    0.046512    0.048780    0.097561     0.081633

        ct_src_dport_ltm  ct_dst_sport_ltm  ct_dst_src_ltm
511046               0.0               0.0        0.054054
381461               0.0               0.0        0.081081

[2 rows x 47 columns]
```

In [19]:
```
entropy = -(1 * np.sum(np.log2(x_train) * x_train))
entropy
```

```
C:\Users\DELL\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas
\core\internals\blocks.py:402: RuntimeWarning: divide by zero encountered in lo
g2
  result = func(self.values, **kwargs)
```

Out[19]:
```
srcip                81992.845266
sport               172102.356118
dstip               233369.108128
dsport              146266.070964
proto                90161.044771
state               240244.235856
service              82883.042018
dur                    450.176691
sbytes                1692.963035
dbytes                8451.179035
sttl                174573.284630
dttl                165432.059731
sloss                 5160.175023
dloss                11021.491555
Sload                 2944.561932
Dload                72296.622606
Spkts                13106.369314
Dpkts                14603.900317
swin                     4.300240
dwin                     0.042621
stcpb               124812.357332
dtcpb               124665.793339
smeansz             100483.524166
dmeansz             162431.699618
trans_depth          16528.530639
res_bdy_len           2649.430586
Sjit                  4650.231849
Djit                  4012.832878
Stime               182338.098709
Ltime               182097.565514
Sintpkt               2724.732045
Dintpkt               1952.375176
tcprtt                3543.087721
synack                2640.606288
ackdat                2324.614840
is_sm_ips_ports         -0.000000
ct_state_ttl          9722.098618
ct_flw_http_mthd      8274.192781
is_ftp_login            -0.000000
ct_ftp_cmd            3814.629798
ct_srv_src          146391.787368
ct_srv_dst          146609.921991
ct_dst_ltm          111901.575219
ct_src_ ltm         111835.224854
ct_src_dport_ltm     30710.666552
ct_dst_sport_ltm     18811.426957
ct_dst_src_ltm       52899.670022
dtype: float64
```

In [20]:
```
def entropy(y_train):
    probs = [] # Probabilities of each class label
    for c in set(y_train): # Set gets a unique set of values. We're iterating ove
        num_same_class = sum(y_train == c)  # Remember that true == 1, so we can
        p = num_same_class / len(y) # Probability of this class label
        probs.append(p)
    return np.sum(-p * np.log2(p) for p in probs)

# What is the entropy of the entire set?
print("Entire set entropy = %.2f" % entropy(y_train))
```

```
Entire set entropy = 0.56

C:\Users\DELL\AppData\Local\Temp\ipykernel_8548\213859896.py:7: DeprecationWarn
ing: Calling np.sum(generator) is deprecated, and in the future will give a dif
ferent result. Use np.sum(np.fromiter(generator)) or the python sum builtin ins
tead.
  return np.sum(-p * np.log2(p) for p in probs)
```

In [21]:
```
def class_probability(feature, y_train):
    """Calculates the proportional length of each value in the set of instances"'
    # This is doc string, used for documentation
    probs = []
    for value in set(feature):
        select = feature == value # Split by feature value into two classes
        y_new = y_train[select]         # Those that exist in this class are now
        probs.append(float(len(y_new))/len(x))  # Convert to float, because ints
    return probs

def class_entropy(feature, y_train):
    """Calculates the entropy for each value in the set of instances"""
    ents = []
    for value in set(feature):
        select = feature == value # Split by feature value into two classes
        y_new = y_train[select]         # Those that exist in this class are now
        ents.append(entropy(y_new))
    return ents

def proportionate_class_entropy(feature, y_train):
    """Calculatates the weighted proportional entropy for a feature when splittir
    probs = class_probability(feature, y_train)
    ents = class_entropy(feature, y_train)
    return np.sum(np.multiply(probs, ents))
```

In [22]:
```
new_entropy = proportionate_class_entropy(x_train["service"], y_train)
print("Information gain of %.2f" % (entropy(y_train) - new_entropy))
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_8548\213859896.py:7: DeprecationWarn
ing: Calling np.sum(generator) is deprecated, and in the future will give a dif
ferent result. Use np.sum(np.fromiter(generator)) or the python sum builtin ins
tead.
  return np.sum(-p * np.log2(p) for p in probs)

Information gain of 0.21
```

In [23]:
```
# for c in x_train.columns:
#     new_entropy = proportionate_class_entropy(x_train[c], y_train)
#     print("%s %.2f" % (c, entropy(y_train) - new_entropy))
```

In [24]:
```
from sklearn.feature_selection import mutual_info_classif
# determine the mutual information
mutual_info = mutual_info_classif(x_train, y_train)
mutual_info
```

Out[24]:
```
array([0.13784631, 0.05997042, 0.14450284, 0.07732835, 0.07859762,
       0.07812514, 0.02467788, 0.08036717, 0.14194329, 0.10717312,
       0.13662893, 0.11944812, 0.03210586, 0.03680214, 0.1107323 ,
       0.0888083 , 0.04428285, 0.07742279, 0.07114304, 0.07022973,
       0.01562296, 0.01561756, 0.09586413, 0.09770715, 0.00115057,
       0.00633781, 0.04016852, 0.03815808, 0.06796831, 0.06860988,
       0.04811182, 0.08425639, 0.07013409, 0.06831455, 0.06923162,
       0.        , 0.12776874, 0.00115691, 0.00054125, 0.00026961,
       0.01412524, 0.01924026, 0.01757164, 0.01270474, 0.02643115,
       0.02653004, 0.01768953])
```

In [25]:
```
mutual_info = pd.Series(mutual_info)
mutual_info.index = x_train.columns
mutual_info.sort_values(ascending=False)
```

Out[25]:
```
dstip               0.144503
sbytes              0.141943
srcip               0.137846
sttl                0.136629
ct_state_ttl        0.127769
dttl                0.119448
Sload               0.110732
dbytes              0.107173
dmeansz             0.097707
smeansz             0.095864
Dload               0.088808
Dintpkt             0.084256
dur                 0.080367
proto               0.078598
state               0.078125
Dpkts               0.077423
dsport              0.077328
swin                0.071143
dwin                0.070230
tcprtt              0.070134
ackdat              0.069232
Ltime               0.068610
synack              0.068315
Stime               0.067968
sport               0.059970
Sintpkt             0.048112
Spkts               0.044283
Sjit                0.040169
Djit                0.038158
dloss               0.036802
sloss               0.032106
ct_dst_sport_ltm    0.026530
ct_src_dport_ltm    0.026431
service             0.024678
ct_srv_dst          0.019240
ct_dst_src_ltm      0.017690
ct_dst_ltm          0.017572
stcpb               0.015623
dtcpb               0.015618
ct_srv_src          0.014125
ct_src_ ltm         0.012705
res_bdy_len         0.006338
ct_flw_http_mthd    0.001157
trans_depth         0.001151
is_ftp_login        0.000541
ct_ftp_cmd          0.000270
is_sm_ips_ports     0.000000
dtype: float64
```

In [26]:
```
# from sklearn.feature_selection import SelectKBest
# sel_cols = SelectKBest(mutual_info_classif, k=10)
# sel_cols.fit(x_train, y_train)
# x.columns[sel_cols.get_support()]
```

In [27]:
```
from sklearn.feature_selection import SelectKBest
sel_cols = SelectKBest(mutual_info_classif, k=10)
sel_cols.fit(x_train, y_train)
xtrain=sel_cols.transform(x_train)
xtest=sel_cols.transform(x_test)
# x.columns[sel_cols.get_support()]
```

In [28]:
```
print(xtrain.shape)
print(xtest.shape)
```

```
(490000, 10)
(210001, 10)
```

In [29]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_dim=xtrain.shape[1], activation='relu'))
model.add(Dense(75, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=
model.fit(xtrain, y_train,validation_split=0.33, batch_size=50, epochs=20)
model.summary()
```

```
Epoch 1/20
6566/6566 [==============================] - 26s 4ms/step - loss: 0.0622 - ac
curacy: 0.9815 - val_loss: 0.0455 - val_accuracy: 0.9841
Epoch 2/20
6566/6566 [==============================] - 24s 4ms/step - loss: 0.0454 - ac
curacy: 0.9838 - val_loss: 0.0501 - val_accuracy: 0.9818
Epoch 3/20
6566/6566 [==============================] - 24s 4ms/step - loss: 0.0432 - ac
curacy: 0.9842 - val_loss: 0.0402 - val_accuracy: 0.9837
Epoch 4/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0404 - ac
curacy: 0.9844 - val_loss: 0.0384 - val_accuracy: 0.9846
Epoch 5/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0385 - ac
curacy: 0.9849 - val_loss: 0.0365 - val_accuracy: 0.9857
Epoch 6/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0371 - ac
curacy: 0.9855 - val_loss: 0.0379 - val_accuracy: 0.9850
Epoch 7/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0368 - ac
curacy: 0.9857 - val_loss: 0.0348 - val_accuracy: 0.9859
Epoch 8/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0361 - ac
curacy: 0.9857 - val_loss: 0.0340 - val_accuracy: 0.9863
Epoch 9/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0362 - ac
curacy: 0.9858 - val_loss: 0.0346 - val_accuracy: 0.9861
Epoch 10/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0357 - ac
curacy: 0.9859 - val_loss: 0.0352 - val_accuracy: 0.9860
Epoch 11/20
6566/6566 [==============================] - 24s 4ms/step - loss: 0.0349 - ac
curacy: 0.9862 - val_loss: 0.0339 - val_accuracy: 0.9860
Epoch 12/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0348 - ac
curacy: 0.9861 - val_loss: 0.0369 - val_accuracy: 0.9856
Epoch 13/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0346 - ac
curacy: 0.9864 - val_loss: 0.0337 - val_accuracy: 0.9866
Epoch 14/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0348 - ac
curacy: 0.9860 - val_loss: 0.0365 - val_accuracy: 0.9861
```

```
Epoch 15/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0342 - ac
curacy: 0.9863 - val_loss: 0.0335 - val_accuracy: 0.9869
Epoch 16/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0342 - ac
curacy: 0.9865 - val_loss: 0.0341 - val_accuracy: 0.9862
Epoch 17/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0343 - ac
curacy: 0.9866 - val_loss: 0.0338 - val_accuracy: 0.9869
Epoch 18/20
6566/6566 [==============================] - 25s 4ms/step - loss: 0.0339 - ac
curacy: 0.9868 - val_loss: 0.0345 - val_accuracy: 0.9868
Epoch 19/20
6566/6566 [==============================] - 26s 4ms/step - loss: 0.0335 - ac
curacy: 0.9869 - val_loss: 0.0328 - val_accuracy: 0.9872
Epoch 20/20
6566/6566 [==============================] - 26s 4ms/step - loss: 0.0333 - ac
curacy: 0.9869 - val_loss: 0.0329 - val_accuracy: 0.9869
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               1100

 dense_1 (Dense)             (None, 75)                7575

 dense_2 (Dense)             (None, 50)                3800

 dense_3 (Dense)             (None, 30)                1530

 dense_4 (Dense)             (None, 20)                620

 dense_5 (Dense)             (None, 15)                315

 dense_6 (Dense)             (None, 10)                160

=================================================================
Total params: 15,100
Trainable params: 15,100
Non-trainable params: 0
_________________________________________________________________
```

In [30]:

```
prediction = model.predict(xtest[121:122])
print(prediction)
print(y_test[121:122])
```

```
1/1 [==============================] - 0s 182ms/step
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
247947    0
Name: attack_cat, dtype: int32
```

In [31]:
```
score,acc = model.evaluate(xtest,y_test,batch_size=20)
print("score",score)
print("accuracy",acc)

scores=model.evaluate(xtest,y_test)
print("\n%s: %.2f%%" %(model.metrics_names[1],scores[1]*100))
```

```
10501/10501 [==============================] - 23s 2ms/step - loss: 0.0807 - ac
curacy: 0.9758
score 0.08073591440916061
accuracy 0.975809633731842
6563/6563 [==============================] - 15s 2ms/step - loss: 0.0807 - accu
racy: 0.9758

accuracy: 97.58%
```

In [ ]:
```

```

In [32]:
```
from sklearn.decomposition import PCA

pca = PCA(0.95)

x_train = pca.fit_transform(x_train)
print(x_train)
```

```
[[-0.29947903 -0.61137577  0.00324559 ...  0.01239593 -0.07331234
  -0.00721189]
 [-0.48175033 -0.65840863  0.13866073 ... -0.02162046 -0.16762255
   0.08476253]
 [ 1.12048788 -0.40665596 -0.13967629 ... -0.03897949  0.01826431
  -0.07456957]
 ...
 [-0.19386131 -0.2487166  -0.04934532 ...  0.05166665 -0.01280445
  -0.0629275 ]
 [-0.64655435 -0.12735705  0.54818602 ... -0.16304227 -0.2387596
  -0.10666505]
 [-0.46740832  0.51039781  0.38801364 ...  0.05881903 -0.07766548
  -0.03760659]]
```

In [33]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(490000, 13)
(490000,)
```

In [34]:
```
# DNN
from keras.models import Sequential
from keras.layers import Dense

dnn_model = Sequential()
dnn_model.add(Dense(100, input_dim=x_train.shape[1], activation='relu'))
dnn_model.add(Dense(75, activation='relu'))
dnn_model.add(Dense(50, activation='relu'))
dnn_model.add(Dense(30, activation='relu'))
dnn_model.add(Dense(20, activation='relu'))
dnn_model.add(Dense(15, activation='relu'))
dnn_model.add(Dense(10, activation='softmax'))
dnn_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metri
dnn_model.fit(x_train, y_train, batch_size=50, epochs=20)
dnn_model.summary()
```

```
Epoch 1/20
9800/9800 [==============================] - 32s 3ms/step - loss: 0.0457 - accu
racy: 0.9839
Epoch 2/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0309 - accu
racy: 0.9877
Epoch 3/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0297 - accu
racy: 0.9880
Epoch 4/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0290 - accu
racy: 0.9882
Epoch 5/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0284 - accu
racy: 0.9885
Epoch 6/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0280 - accu
racy: 0.9885
Epoch 7/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0274 - accu
racy: 0.9887
Epoch 8/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0269 - accu
racy: 0.9890
Epoch 9/20
9800/9800 [==============================] - 29s 3ms/step - loss: 0.0266 - accu
racy: 0.9890
Epoch 10/20
9800/9800 [==============================] - 29s 3ms/step - loss: 0.0264 - accu
racy: 0.9891
Epoch 11/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0262 - accu
racy: 0.9893
Epoch 12/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0261 - accu
racy: 0.9893
Epoch 13/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0257 - accu
racy: 0.9893
Epoch 14/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0257 - accu
```

```
racy: 0.9894
Epoch 15/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0255 - accu
racy: 0.9893
Epoch 16/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0254 - accu
racy: 0.9893
Epoch 17/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0253 - accu
racy: 0.9895
Epoch 18/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0251 - accu
racy: 0.9896
Epoch 19/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0252 - accu
racy: 0.9896
Epoch 20/20
9800/9800 [==============================] - 28s 3ms/step - loss: 0.0253 - accu
racy: 0.9895
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_7 (Dense)              (None, 100)               1400

 dense_8 (Dense)              (None, 75)                7575

 dense_9 (Dense)              (None, 50)                3800

 dense_10 (Dense)             (None, 30)                1530

 dense_11 (Dense)             (None, 20)                620

 dense_12 (Dense)             (None, 15)                315

 dense_13 (Dense)             (None, 10)                160

=================================================================
Total params: 15,400
Trainable params: 15,400
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
from sklearn.decomposition import PCA

pca = PCA(0.95)

x_test = pca.fit_transform(x_test)
x_test.shape
```

Out[35]:
```
(210001, 13)
```

In [36]:
```
print(y_test[121:122])
```

```
247947    0
Name: attack_cat, dtype: int32
```

In [37]:
```
dnn_prediction = dnn_model.predict(x_test[121:122])
print(dnn_prediction)
print(y_test[121:122])
```

```
1/1 [==============================] - 0s 121ms/step
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
247947    0
Name: attack_cat, dtype: int32
```

In [38]:
```
dnn_score,dnn_acc = dnn_model.evaluate(x_test,y_test,batch_size=20)
print("score",dnn_score)
print("accuracy",dnn_acc)

d_scores=dnn_model.evaluate(x_test,y_test)
print("\n%s: %.2f%%" %(dnn_model.metrics_names[1],d_scores[1]*100))
```

```
10501/10501 [==============================] - 23s 2ms/step - loss: 0.0262 - ac
curacy: 0.9897
score 0.026237310841679573
accuracy 0.9897095561027527
6563/6563 [==============================] - 15s 2ms/step - loss: 0.0262 - accu
racy: 0.9897

accuracy: 98.97%
```

In [39]:
```
pd.crosstab(y_test, dnn_prediction, rownames=['Actual attacks'], colnames=['Predi
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [39], in <cell line: 1>()
----> 1 pd.crosstab(y_test, dnn_prediction, rownames=['Actual attacks'], colnam
es=['Predicted attacks'])

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\re
shape\pivot.py:640, in crosstab(index, columns, values, rownames, colnames, agg
func, margins, margins_name, dropna, normalize)
    637     columns = [columns]
    639 common_idx = None
--> 640 pass_objs = [x for x in index + columns if isinstance(x, (ABCSeries, AB
CDataFrame))]
    641 if pass_objs:
    642     common_idx = get_objs_combined_axis(pass_objs, intersect=True, sort
=False)

ValueError: operands could not be broadcast together with shapes (1,210001) (1,
10)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train, dnn_prediction)
```

In [ ]:
```
from sklearn.metrics import precision_score, recall_score
precision_score(y_train, dnn_prediction)
recall_score(y_train, dnn_prediction)
```

In [ ]:
```
# from keras.models import Sequential
# from keras.layers import Dense

# ann_model = Sequential()
# ann_model.add(Dense(100, input_dim=x_train.shape[1], activation='relu'))
# ann_model.add(Dense(55, activation='relu'))
# ann_model.add(Dense(10, activation='softmax'))
# ann_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', met
# ann_model.fit(x_train, y_train, batch_size=20, epochs=25)
# ann_model.summary()


# ann_prediction = ann_model.predict(x_test[121:122])
# print(ann_prediction)
# print(y_test[121:122])


# ann_score,ann_acc = ann_model.evaluate(x_test,y_test,batch_size=20)
# print("score",ann_score)
# print("accuracy",ann_acc)

# a_scores=ann_model.evaluate(x_test,y_test)
# print("\n%s: %.2f%%" %(ann_model.metrics_names[1],a_scores[1]*100))
```