Artificial Neural Network for Intrusion Detection

Submitted by : Aakruti, Mudit, Rubayat

In [1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
ce_remount=True).
```

In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import models,layers
from tensorflow import keras
from tensorflow import feature_column
from sklearn import metrics
from sklearn.model_selection import train_test_split
import seaborn as sb
```

In [3]:
```python
path = "/content/drive/MyDrive/Colab Notebooks/Final Dataset.csv"
initial = pd.read_csv(path)
initial
```

Out[3]:

	Node	Mobility	Mobility_speed	packet_Received	packet_Sent	Forwarding_Difference	Energy	num_Received	num_Sent	pack
0	R0	RandomWayPoint	4	37	10	161	-1.138122	1217	1056	
1	R1	RandomWayPoint	4	26	2	0	-0.198993	0	0	
2	R2	RandomWayPoint	4	34	4	0	-0.206068	0	0	
3	R3	RandomWayPoint	4	21	2	0	-0.164678	0	0	
4	R4	RandomWayPoint	4	17	2	0	-0.168908	0	0	
...	...	...	...	...	...	...	...	...	...	
2270	R20	Mass	12	1190	208	207	-0.355682	600	393	
2271	R21	Mass	12	933	233	67	-0.549195	633	566	
2272	R22	Mass	12	885	233	40	-0.431493	326	286	
2273	A	Mass	12	1064	2204	-477	-0.939987	88	565	
2274	B	Mass	12	770	229	72	-0.338136	120	48	

2275 rows × 13 columns

In [4]:
```python
print(initial.columns)
```

```
Index(['Node', 'Mobility', 'Mobility_speed', 'packet_Received', 'packet_Sent',
       'Forwarding_Difference', 'Energy', 'num_Received', 'num_Sent',
       'packet_Drop', 'Transmission_Power', 'Attack_type', 'Type_of_node'],
      dtype='object')
```

Checking whether there is any null values in dataset.

In [5]:
```python
initial.isnull().sum()
```

Out[5]:
```
Node                     0
Mobility                 0
Mobility_speed           0
packet_Received          0
packet_Sent              0
Forwarding_Difference    0
Energy                   0
num_Received             0
num_Sent                 0
packet_Drop              0
Transmission_Power       0
Attack_type              0
Type_of_node             0
dtype: int64
```

Data preprocessing

In this, categorical features are converted into numerical dataset.

In [6]:
```python
initial.Type_of_node = initial.Type_of_node.replace({'Normal':0, 'Attacker':1})
```

In [7]:
```python
print(initial['Attack_type'].unique())
print(initial['Mobility'].unique())
print(initial.dtypes)
```

```
['Normal' 'Blackhole' 'Hello Flooding' 'UDP Flooding DOS']
['RandomWayPoint' 'Mass']
Node                      object
Mobility                  object
Mobility_speed             int64
packet_Received            int64
packet_Sent                int64
Forwarding_Difference      int64
Energy                   float64
num_Received               int64
num_Sent                   int64
packet_Drop                int64
Transmission_Power       float64
Attack_type               object
Type_of_node               int64
dtype: object
```

In [8]:
```python
initial.Attack_type = initial.Attack_type.replace({'Normal':0, 'Blackhole':1, 'Hello
Flooding':2, 'UDP Flooding DOS':3})
initial.Mobility = initial.Mobility.replace({'Mass':0, 'RandomWayPoint':1})
```

In [9]:
```python
initial
```

Out[9]:

	Node	Mobility	Mobility_speed	packet_Received	packet_Sent	Forwarding_Difference	Energy	num_Received	num_Sent	packet_Drop
0	R0	1	4	37	10	161	-1.138122	1217	1056	0
1	R1	1	4	26	2	0	-0.198993	0	0	0
2	R2	1	4	34	4	0	-0.206068	0	0	0
3	R3	1	4	21	2	0	-0.164678	0	0	0
4	R4	1	4	17	2	0	-0.168908	0	0	0
...	...	...	...	...	...	...	...	...	...	...
2270	R20	0	12	1190	208	207	-0.355682	600	393	13
2271	R21	0	12	933	233	67	-0.549195	633	566	21
2272	R22	0	12	885	233	40	-0.431493	326	286	12
2273	A	0	12	1064	2204	-477	-0.939987	88	565	36
2274	B	0	12	770	229	72	-0.338136	120	48	18

2275 rows × 13 columns

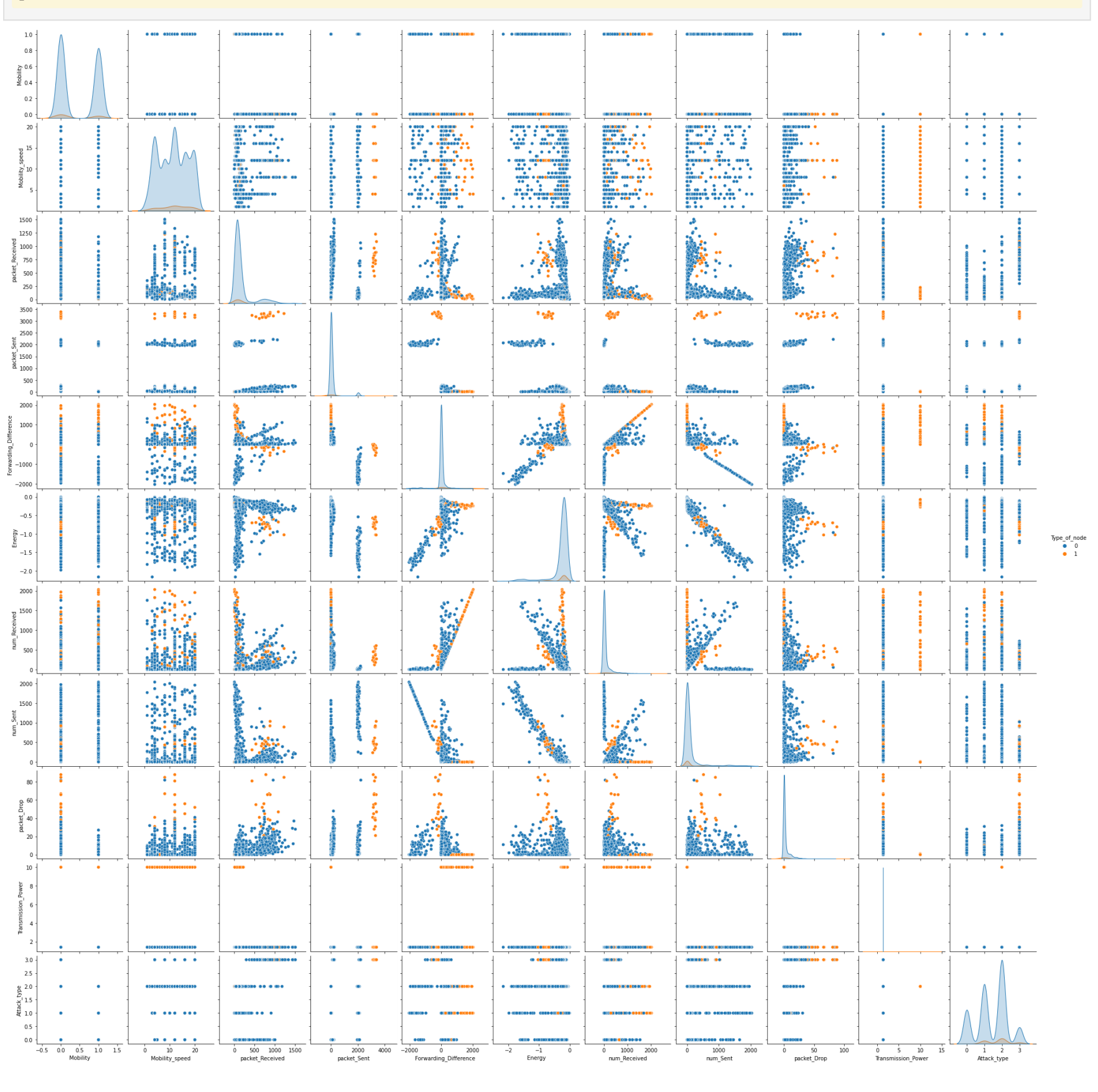
Splitted the dataset. Training dataset is 90% and Test dataset is 10%

In [10]:
```python
# Partitioning data
# Setting x and y
# features=list(initial.columns)
# cols=list(set(features)-set(['Node','Position', 'Mobility_model', 'Type_of_node']))
# print(cols)
cols = ['Mobility', 'Mobility_speed', 'packet_Received', 'packet_Sent',
       'Forwarding_Difference', 'Energy', 'num_Received', 'num_Sent',
       'packet_Drop', 'Transmission_Power', 'Attack_type']

x=np.array(initial[cols].values)
y=np.array(initial['Type_of_node'].values)

#Splitting data into training set and test set
train_data,test_data,train_target,test_target =
train_test_split(x,y,test_size=0.1,random_state=1)
print(train_data.shape,"\t",train_target.shape)
print(test_data.shape,"\t",test_target.shape)
```

```
(2047, 11)        (2047,)
(228, 11)         (228,)
```

Plotting all the features into pairplot to observe the correlation between the features. We can also observe that whether the attacker nodes and normal nodes are linearly seprable using any combination of features.

In [11]:
```python
# Inspecting the dataset
sb.pairplot(initial,hue='Type_of_node')
plt.show()
```



Check the exact correlation between the values between all the features.

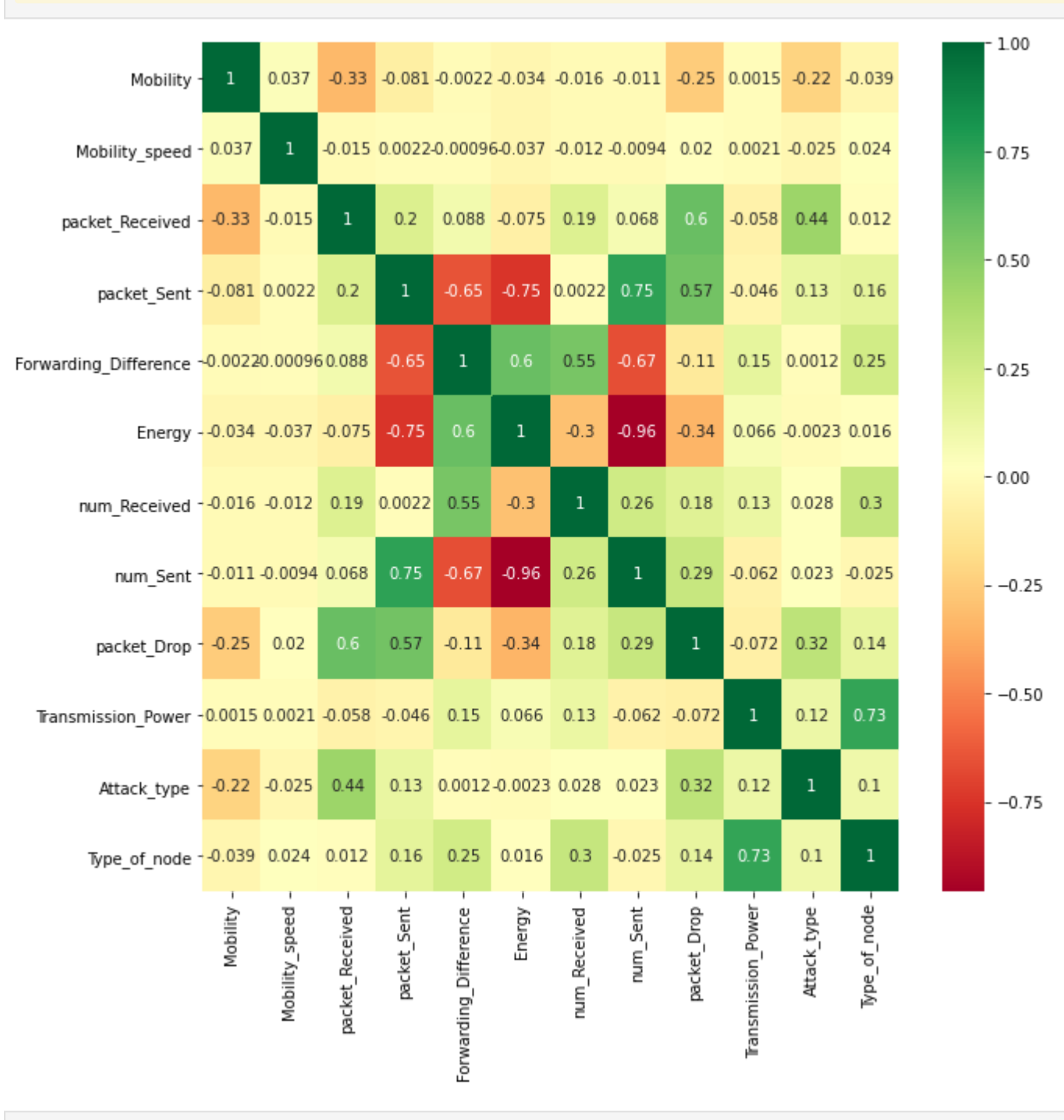
If the value is greater than 0.5 then feature has strong positive corelation with other feature. If the value is between -0.5 and -1 then that feature have strong negative corelation with other feature. Like Energy has strong negative correlation with num_Sent. It means Energy value will decrease if value of number of packets forwarded increase. Other values can be said as, have weak correlation. And if the value is 0 then that features are completely independent of each other mean not correlated at all.

In [12]:
```python
# To check the corealtion between the features
initial.corr()
```

Out[12]:

	Mobility	Mobility_speed	packet_Received	packet_Sent	Forwarding_Difference	Energy	num_Received	num_Sent	pa
Mobility	1.000000	0.036856	-0.329774	-0.080579	-0.002238	-0.034268	-0.015754	-0.011488	
Mobility_speed	0.036856	1.000000	-0.015383	0.002169	-0.000962	-0.037185	-0.011727	-0.009365	
packet_Received	-0.329774	-0.015383	1.000000	0.200353	0.087693	-0.074573	0.189838	0.068121	
packet_Sent	-0.080579	0.002169	0.200353	1.000000	-0.648007	-0.746082	0.002227	0.752177	
Forwarding_Difference	-0.002238	-0.000962	0.087693	-0.648007	1.000000	0.597733	0.550387	-0.665852	
Energy	-0.034268	-0.037185	-0.074573	-0.746082	0.597733	1.000000	-0.296670	-0.957092	
num_Received	-0.015754	-0.011727	0.189838	0.002227	0.550387	-0.296670	1.000000	0.256436	
num_Sent	-0.011488	-0.009365	0.068121	0.752177	-0.665852	-0.957092	0.256436	1.000000	
packet_Drop	-0.246313	0.020399	0.595976	0.569524	-0.110029	-0.340782	0.178645	0.287017	
Transmission_Power	0.001503	0.002084	-0.058204	-0.045888	0.151132	0.065996	0.126784	-0.061667	
Attack_type	-0.215638	-0.025178	0.435036	0.132300	0.001240	-0.002340	0.027602	0.023231	
Type_of_node	-0.039196	0.024328	0.011693	0.158042	0.249909	0.015589	0.295442	-0.025306	

In [13]:
```python
heatmap = initial.corr()
top_feat = heatmap.index
plt.figure(figsize=(10,10))
p = sb.heatmap(initial[top_feat].corr(),annot=True,cmap ='RdYlGn')
```



In [14]:
```python
print(train_data[0])
print(train_data.shape[1])
```

```
[ 1.        1.       57.       16.        0.       -0.07286  0.        0.
  0.        1.4       2.      ]
11
```

Designed Artificial Neural Network model

In [15]:
```python
#Created model
model=tf.keras.Sequential([
  layers.Dense(32,activation='relu',input_shape=[train_data.shape[1]]),
  layers.Dense(64,activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(32,activation='relu'),
  layers.Dense(8,activation='relu'),
  layers.Dense(1,activation='sigmoid')])
model.summary()
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['mse','accuracy'])
# print(train_1)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                384
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2112
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 264
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9
=================================================================
Total params: 4,849
Trainable params: 4,849
Non-trainable params: 0
_________________________________________________________________
```

Checking whether the model is predicting the output in the way or format we expected.

In [16]:
```python
example_batch = train_data[:10]
example_result = model.predict(example_batch)
print(example_result)
```

```
[[0.9018006]
 [1.       ]
 [0.961885 ]
 [1.       ]
 [0.9832674]
 [1.       ]
 [0.9575359]
 [0.9997722]
 [0.8104565]
 [1.       ]]
```

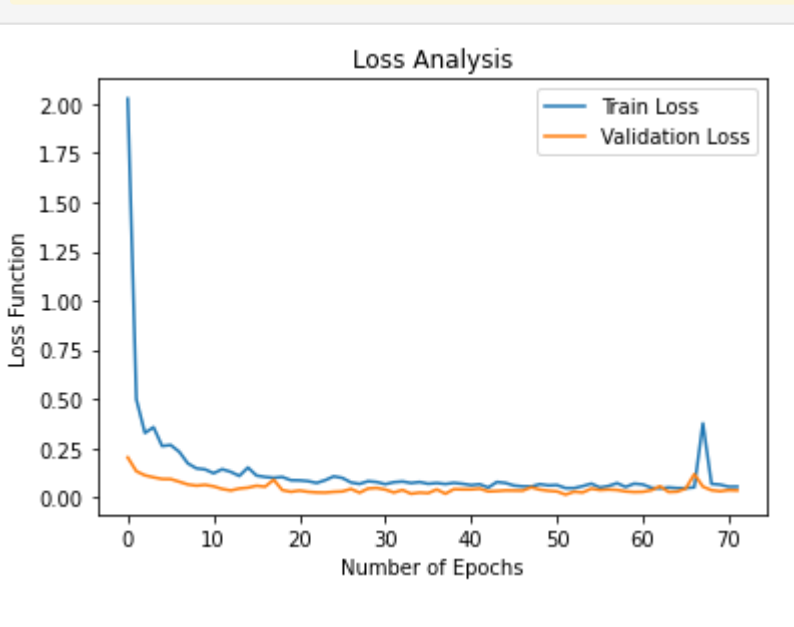
Training of dataset with all the hyper-parameters specified.

In [17]:
```python
store=model.fit(train_data, train_target, validation_split=0.15, epochs=100, batch_size=32,
                callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)])
model.summary()
```

```
Epoch 1/100
55/55 [==============================] - 1s 5ms/step - loss: 2.0276 - mse: 0.1434 - accuracy: 0.8453 - val_los
s: 0.2027 - val_mse: 0.0315 - val_accuracy: 0.9708
Epoch 2/100
55/55 [==============================] - 0s 2ms/step - loss: 0.4988 - mse: 0.0663 - accuracy: 0.9385 - val_los
s: 0.1325 - val_mse: 0.0296 - val_accuracy: 0.9740
Epoch 3/100
55/55 [==============================] - 0s 2ms/step - loss: 0.3270 - mse: 0.0607 - accuracy: 0.9425 - val_los
s: 0.1116 - val_mse: 0.0247 - val_accuracy: 0.9773
Epoch 4/100
55/55 [==============================] - 0s 2ms/step - loss: 0.3565 - mse: 0.0559 - accuracy: 0.9419 - val_los
s: 0.1015 - val_mse: 0.0244 - val_accuracy: 0.9740
Epoch 5/100
55/55 [==============================] - 0s 2ms/step - loss: 0.2606 - mse: 0.0485 - accuracy: 0.9488 - val_los
s: 0.0931 - val_mse: 0.0217 - val_accuracy: 0.9773
Epoch 6/100
55/55 [==============================] - 0s 2ms/step - loss: 0.2659 - mse: 0.0457 - accuracy: 0.9523 - val_los
s: 0.0927 - val_mse: 0.0218 - val_accuracy: 0.9740
Epoch 7/100
55/55 [==============================] - 0s 2ms/step - loss: 0.2314 - mse: 0.0499 - accuracy: 0.9442 - val_los
s: 0.0792 - val_mse: 0.0195 - val_accuracy: 0.9740
Epoch 8/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1724 - mse: 0.0385 - accuracy: 0.9500 - val_los
s: 0.0652 - val_mse: 0.0171 - val_accuracy: 0.9773
Epoch 9/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1470 - mse: 0.0360 - accuracy: 0.9551 - val_los
s: 0.0592 - val_mse: 0.0146 - val_accuracy: 0.9773
Epoch 10/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1419 - mse: 0.0347 - accuracy: 0.9546 - val_los
s: 0.0632 - val_mse: 0.0129 - val_accuracy: 0.9903
Epoch 11/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1223 - mse: 0.0292 - accuracy: 0.9574 - val_los
s: 0.0557 - val_mse: 0.0133 - val_accuracy: 0.9838
Epoch 12/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1426 - mse: 0.0310 - accuracy: 0.9603 - val_los
s: 0.0425 - val_mse: 0.0114 - val_accuracy: 0.9838
Epoch 13/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1294 - mse: 0.0296 - accuracy: 0.9569 - val_los
s: 0.0346 - val_mse: 0.0099 - val_accuracy: 0.9838
Epoch 14/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1087 - mse: 0.0260 - accuracy: 0.9632 - val_los
s: 0.0445 - val_mse: 0.0124 - val_accuracy: 0.9805
Epoch 15/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1514 - mse: 0.0308 - accuracy: 0.9586 - val_los
s: 0.0489 - val_mse: 0.0121 - val_accuracy: 0.9903
Epoch 16/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1105 - mse: 0.0267 - accuracy: 0.9643 - val_los
s: 0.0587 - val_mse: 0.0180 - val_accuracy: 0.9578
Epoch 17/100
55/55 [==============================] - 0s 3ms/step - loss: 0.1033 - mse: 0.0250 - accuracy: 0.9643 - val_los
s: 0.0539 - val_mse: 0.0170 - val_accuracy: 0.9578
Epoch 18/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0994 - mse: 0.0249 - accuracy: 0.9626 - val_los
s: 0.0892 - val_mse: 0.0209 - val_accuracy: 0.9545
Epoch 19/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1034 - mse: 0.0244 - accuracy: 0.9672 - val_los
s: 0.0366 - val_mse: 0.0100 - val_accuracy: 0.9870
Epoch 20/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0868 - mse: 0.0211 - accuracy: 0.9770 - val_los
s: 0.0281 - val_mse: 0.0075 - val_accuracy: 0.9903
Epoch 21/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0860 - mse: 0.0195 - accuracy: 0.9758 - val_los
s: 0.0343 - val_mse: 0.0083 - val_accuracy: 0.9903
Epoch 22/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0822 - mse: 0.0189 - accuracy: 0.9793 - val_los
s: 0.0279 - val_mse: 0.0079 - val_accuracy: 0.9870
Epoch 23/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0729 - mse: 0.0169 - accuracy: 0.9804 - val_los
s: 0.0248 - val_mse: 0.0064 - val_accuracy: 0.9903
Epoch 24/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0870 - mse: 0.0201 - accuracy: 0.9747 - val_los
s: 0.0240 - val_mse: 0.0058 - val_accuracy: 0.9903
Epoch 25/100
55/55 [==============================] - 0s 2ms/step - loss: 0.1064 - mse: 0.0201 - accuracy: 0.9753 - val_los
s: 0.0277 - val_mse: 0.0069 - val_accuracy: 0.9935
Epoch 26/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0985 - mse: 0.0198 - accuracy: 0.9781 - val_los
s: 0.0298 - val_mse: 0.0070 - val_accuracy: 0.9935
Epoch 27/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0754 - mse: 0.0174 - accuracy: 0.9753 - val_los
s: 0.0425 - val_mse: 0.0139 - val_accuracy: 0.9610
Epoch 28/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0681 - mse: 0.0172 - accuracy: 0.9753 - val_los
s: 0.0234 - val_mse: 0.0052 - val_accuracy: 0.9935
Epoch 29/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0818 - mse: 0.0175 - accuracy: 0.9724 - val_los
s: 0.0446 - val_mse: 0.0140 - val_accuracy: 0.9610
Epoch 30/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0776 - mse: 0.0181 - accuracy: 0.9712 - val_los
s: 0.0474 - val_mse: 0.0130 - val_accuracy: 0.9643
Epoch 31/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0668 - mse: 0.0165 - accuracy: 0.9770 - val_los
s: 0.0393 - val_mse: 0.0108 - val_accuracy: 0.9935
Epoch 32/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0768 - mse: 0.0164 - accuracy: 0.9787 - val_los
s: 0.0242 - val_mse: 0.0053 - val_accuracy: 0.9935
Epoch 33/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0803 - mse: 0.0163 - accuracy: 0.9816 - val_los
s: 0.0372 - val_mse: 0.0102 - val_accuracy: 0.9870
Epoch 34/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0726 - mse: 0.0168 - accuracy: 0.9822 - val_los
s: 0.0182 - val_mse: 0.0046 - val_accuracy: 0.9935
Epoch 35/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0775 - mse: 0.0170 - accuracy: 0.9799 - val_los
s: 0.0238 - val_mse: 0.0049 - val_accuracy: 0.9968
Epoch 36/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0688 - mse: 0.0165 - accuracy: 0.9822 - val_los
s: 0.0213 - val_mse: 0.0050 - val_accuracy: 0.9935
Epoch 37/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0726 - mse: 0.0166 - accuracy: 0.9810 - val_los
s: 0.0400 - val_mse: 0.0107 - val_accuracy: 0.9935
Epoch 38/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0676 - mse: 0.0150 - accuracy: 0.9839 - val_los
s: 0.0181 - val_mse: 0.0042 - val_accuracy: 0.9968
Epoch 39/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0733 - mse: 0.0155 - accuracy: 0.9845 - val_los
s: 0.0414 - val_mse: 0.0111 - val_accuracy: 0.9935
Epoch 40/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0687 - mse: 0.0144 - accuracy: 0.9856 - val_los
s: 0.0402 - val_mse: 0.0111 - val_accuracy: 0.9935
Epoch 41/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0620 - mse: 0.0145 - accuracy: 0.9873 - val_los
s: 0.0403 - val_mse: 0.0119 - val_accuracy: 0.9870
Epoch 42/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0662 - mse: 0.0169 - accuracy: 0.9816 - val_los
s: 0.0438 - val_mse: 0.0126 - val_accuracy: 0.9870
Epoch 43/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0496 - mse: 0.0122 - accuracy: 0.9873 - val_los
s: 0.0300 - val_mse: 0.0069 - val_accuracy: 0.9935
Epoch 44/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0777 - mse: 0.0158 - accuracy: 0.9839 - val_los
s: 0.0318 - val_mse: 0.0098 - val_accuracy: 0.9968
Epoch 45/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0728 - mse: 0.0169 - accuracy: 0.9833 - val_los
s: 0.0348 - val_mse: 0.0097 - val_accuracy: 0.9935
Epoch 46/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0597 - mse: 0.0134 - accuracy: 0.9879 - val_los
s: 0.0343 - val_mse: 0.0093 - val_accuracy: 0.9903
Epoch 47/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0554 - mse: 0.0126 - accuracy: 0.9885 - val_los
s: 0.0337 - val_mse: 0.0093 - val_accuracy: 0.9935
Epoch 48/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0548 - mse: 0.0121 - accuracy: 0.9873 - val_los
s: 0.0507 - val_mse: 0.0112 - val_accuracy: 0.9903
Epoch 49/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0666 - mse: 0.0164 - accuracy: 0.9816 - val_los
s: 0.0391 - val_mse: 0.0119 - val_accuracy: 0.9903
Epoch 50/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0605 - mse: 0.0158 - accuracy: 0.9816 - val_los
s: 0.0331 - val_mse: 0.0103 - val_accuracy: 0.9935
Epoch 51/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0622 - mse: 0.0140 - accuracy: 0.9873 - val_los
s: 0.0299 - val_mse: 0.0086 - val_accuracy: 0.9935
Epoch 52/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0468 - mse: 0.0109 - accuracy: 0.9873 - val_los
s: 0.0135 - val_mse: 0.0030 - val_accuracy: 0.9968
Epoch 53/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0466 - mse: 0.0117 - accuracy: 0.9868 - val_los
s: 0.0284 - val_mse: 0.0093 - val_accuracy: 0.9935
Epoch 54/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0568 - mse: 0.0142 - accuracy: 0.9850 - val_los
s: 0.0246 - val_mse: 0.0069 - val_accuracy: 0.9935
Epoch 55/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0695 - mse: 0.0146 - accuracy: 0.9850 - val_los
s: 0.0440 - val_mse: 0.0110 - val_accuracy: 0.9968
Epoch 56/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0497 - mse: 0.0129 - accuracy: 0.9885 - val_los
s: 0.0370 - val_mse: 0.0100 - val_accuracy: 0.9935
Epoch 57/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0572 - mse: 0.0122 - accuracy: 0.9862 - val_los
s: 0.0396 - val_mse: 0.0117 - val_accuracy: 0.9903
Epoch 58/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0718 - mse: 0.0162 - accuracy: 0.9827 - val_los
s: 0.0372 - val_mse: 0.0093 - val_accuracy: 0.9935
Epoch 59/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0526 - mse: 0.0132 - accuracy: 0.9868 - val_los
s: 0.0299 - val_mse: 0.0087 - val_accuracy: 0.9935
Epoch 60/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0694 - mse: 0.0153 - accuracy: 0.9839 - val_los
s: 0.0268 - val_mse: 0.0081 - val_accuracy: 0.9968
Epoch 61/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0651 - mse: 0.0137 - accuracy: 0.9850 - val_los
s: 0.0279 - val_mse: 0.0079 - val_accuracy: 0.9968
Epoch 62/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0484 - mse: 0.0118 - accuracy: 0.9891 - val_los
s: 0.0358 - val_mse: 0.0100 - val_accuracy: 0.9935
Epoch 63/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0451 - mse: 0.0113 - accuracy: 0.9896 - val_los
s: 0.0567 - val_mse: 0.0150 - val_accuracy: 0.9838
Epoch 64/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0501 - mse: 0.0115 - accuracy: 0.9885 - val_los
s: 0.0271 - val_mse: 0.0046 - val_accuracy: 0.9968
Epoch 65/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0466 - mse: 0.0109 - accuracy: 0.9879 - val_los
s: 0.0287 - val_mse: 0.0079 - val_accuracy: 0.9968
Epoch 66/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0467 - mse: 0.0121 - accuracy: 0.9879 - val_los
s: 0.0445 - val_mse: 0.0104 - val_accuracy: 0.9935
Epoch 67/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0509 - mse: 0.0124 - accuracy: 0.9868 - val_los
s: 0.1175 - val_mse: 0.0091 - val_accuracy: 0.9903
Epoch 68/100
55/55 [==============================] - 0s 2ms/step - loss: 0.3765 - mse: 0.0138 - accuracy: 0.9862 - val_los
s: 0.0541 - val_mse: 0.0144 - val_accuracy: 0.9870
Epoch 69/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0681 - mse: 0.0171 - accuracy: 0.9804 - val_los
s: 0.0360 - val_mse: 0.0104 - val_accuracy: 0.9903
Epoch 70/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0644 - mse: 0.0167 - accuracy: 0.9822 - val_los
s: 0.0309 - val_mse: 0.0093 - val_accuracy: 0.9968
Epoch 71/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0537 - mse: 0.0150 - accuracy: 0.9839 - val_los
s: 0.0365 - val_mse: 0.0094 - val_accuracy: 0.9968
Epoch 72/100
55/55 [==============================] - 0s 2ms/step - loss: 0.0540 - mse: 0.0148 - accuracy: 0.9816 - val_los
s: 0.0351 - val_mse: 0.0095 - val_accuracy: 0.9968
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                384
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2112
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 264
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9
=================================================================
Total params: 4,849
Trainable params: 4,849
Non-trainable params: 0
_________________________________________________________________
```

Plotting the Loss Analysis of Validation Set and Training Set and to check how Early Fitting works to avoid Overfitting.

In [18]:
```python
# Plotting validation loss and training loss

history_df = pd.DataFrame(store.history)
history_df['epoch']=store.epoch
plt.figure()
plt.title("Loss Analysis")
plt.xlabel("Number of Epochs")
plt.ylabel("Loss Function")
plt.plot(history_df['epoch'], history_df['loss'], label="Train Loss")
plt.plot(history_df['epoch'], history_df['val_loss'], label="Validation Loss")
plt.legend()
plt.show()
```



Evaluating the Test dataset and checking the accuracy on untrained/unknown dataset.

In [19]:
```python
test_loss, test_mse, test_acc1 = model.evaluate(test_data, test_target)
print('Test Accuracy:',test_acc1)
```

```
8/8 [==============================] - 0s 2ms/step - loss: 0.0671 - mse: 0.0189 - accuracy: 0.9781
Test Accuracy: 0.9780701994895935
```

To check which rows have predicted which class value to study how this model is acting.

In [20]:
```python
test_answers = model.predict_classes(test_data)
cols = ['Mobility', 'Mobility_speed', 'packet_Received', 'packet_Sent',
       'Forwarding_Difference', 'Energy', 'num_Received', 'num_Sent',
       'packet_Drop', 'Transmission_Power', 'Attack_type']

test_df = pd.DataFrame(test_data, columns = cols)
test_df['Type_of_node'] = test_answers
test_df
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.pr
edict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.pred
ict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer ac
tivation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if i
t uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

Out[20]:

	Mobility	Mobility_speed	packet_Received	packet_Sent	Forwarding_Difference	Energy	num_Received	num_Sent	packet_Drop	Transmi
0	0.0	17.0	81.0	2034.0	-1817.0	-1.904590	11.0	1828.0	14.0	
1	1.0	20.0	105.0	19.0	0.0	-0.256808	1.0	1.0	0.0	
2	0.0	8.0	144.0	17.0	0.0	-0.092380	0.0	0.0	0.0	
3	0.0	20.0	100.0	19.0	37.0	-0.241400	40.0	3.0	0.0	
4	1.0	9.0	204.0	30.0	6.0	-0.295570	19.0	13.0	4.0	
...	...	...	...	...	...	...	...	...	...	
223	0.0	4.0	755.0	129.0	3.0	-0.128647	78.0	75.0	4.0	
224	1.0	16.0	37.0	9.0	10.0	-0.450602	268.0	258.0	0.0	
225	0.0	8.0	157.0	34.0	1020.0	-0.623530	1203.0	183.0	22.0	
226	1.0	19.0	44.0	6.0	0.0	-0.171500	0.0	0.0	0.0	
227	0.0	15.0	50.0	7.0	0.0	-0.159180	1.0	1.0	0.0	

228 rows × 12 columns

Generated the confusion matrix for Test Dataset to get the exact values of number of datapoints which are correctly classified and incorrectly classified.

In [24]:
```python
# Confusion Matrix for test dataset
cf1 = metrics.confusion_matrix(test_target, test_answers)
print(cf1)
# Thus in binary classification, the count of true negatives is , false negatives is , true
positives is  and false positives is .
```

```
[[210   4]
 [  1  13]]
```