```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model
```

```
import numpy as np
import pandas as pd
train = pd.read_csv('UNSW_NB15_training-set.csv')
test = pd.read_csv('UNSW_NB15_testing-set.csv')
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
```

```
train['proto_code'] = ord_enc.fit_transform(train[['proto']])
train[['proto','proto_code']].head(175341)
train['state_code'] = ord_enc.fit_transform(train[['state']])
train[['state','state_code']].head(175341)
train['service_code'] = ord_enc.fit_transform(train[['service']])
train[['service','service_code']].head(175341)
```

	service	service_code
0	-	0.0
1	-	0.0
2	-	0.0
3	ftp	3.0
4	-	0.0
...	...	...
175336	dns	2.0
175337	-	0.0
175338	dns	2.0
175339	dns	2.0
175340	dns	2.0

175341 rows × 2 columns

```
test['proto_code'] = ord_enc.fit_transform(test[['proto']])
test[['proto','proto_code']].head(82332)
test['state_code'] = ord_enc.fit_transform(test[['state']])
test[['state','state_code']].head(82332)
test['service_code'] = ord_enc.fit_transform(test[['service']])
test[['service','service_code']].head(82332)
```

	service	service_code
0	-	0.0
1	-	0.0
2	-	0.0
3	-	0.0
4	-	0.0
...	...	...
82327	-	0.0
82328	-	0.0
82329	-	0.0
82330	-	0.0
82331	-	0.0

82332 rows × 2 columns

```
x1 = train[['dur','spkts','dpkts','proto_code','state_code','service_code',
          'sbytes','dbytes','rate','sttl','dttl','sload','dload','sloss','dloss',
        'sinpkt','dinpkt','sjit','djit','swin','stcpb',
        'dtcpb','dwin','tcprtt','synack','ackdat','smean','dmean','trans_depth','response_body_len','ct_srv_src','ct_state_t
```

```
       ,'ct_dst_ltm','ct_src_dport_ltm','ct_dst_sport_ltm','ct_dst_src_ltm','is_ftp_login',
        'ct_ftp_cmd','ct_flw_http_mthd','ct_src_ltm','ct_srv_dst','is_sm_ips_ports']]
y1 = train['label']
x2 = test[['dur','spkts','dpkts','proto_code','state_code','service_code',
          'sbytes','dbytes','rate','sttl','dttl','sload','dload','sloss','dloss',
         'sinpkt','dinpkt','sjit','djit','swin','stcpb',
         'dtcpb','dwin','tcprtt','synack','ackdat','smean','dmean','trans_depth','response_body_len','ct_srv_src','ct_state_t
       ,'ct_dst_ltm','ct_src_dport_ltm','ct_dst_sport_ltm','ct_dst_src_ltm','is_ftp_login',
        'ct_ftp_cmd','ct_flw_http_mthd','ct_src_ltm','ct_srv_dst','is_sm_ips_ports']]
y2 = test['label']
```

```
model = MinMaxScaler()
model.fit(x1)
x1 = model.transform(x1)
x2 = model.transform(x2)
```

```
n_inputs = x1.shape[1]
n_inputs = x2.shape[1]
```

```
input_data_shape = Input(shape=(n_inputs,))
encoder = Dense(n_inputs*2)(input_data_shape)
encoder = BatchNormalization()(encoder)
encoder = LeakyReLU()(encoder)
encoder = Dense(n_inputs)(encoder)
encoder = BatchNormalization()(encoder)
encoder = LeakyReLU()(encoder)
n_bottleneck = round(float(n_inputs)/2.0)
bottleneck = Dense(n_bottleneck)(encoder)
decoder = Dense(n_inputs)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
decoder = Dense(n_inputs*2)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
```

```
output = Dense(n_inputs,activation = 'linear')(decoder)
model_AE = Model(inputs = input_data_shape,outputs = output)
model_AE.compile(optimizer = 'adam',loss='mse')
model_AE.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 42)]              0

 dense_12 (Dense)            (None, 84)                3612

 batch_normalization_8 (Batc  (None, 84)               336
 hNormalization)

 leaky_re_lu_8 (LeakyReLU)   (None, 84)                0

 dense_13 (Dense)            (None, 42)                3570

 batch_normalization_9 (Batc  (None, 42)               168
 hNormalization)

 leaky_re_lu_9 (LeakyReLU)   (None, 42)                0

 dense_14 (Dense)            (None, 21)                903

 dense_16 (Dense)            (None, 84)                1848

 batch_normalization_11 (Bat  (None, 84)               336
 chNormalization)

 leaky_re_lu_11 (LeakyReLU)  (None, 84)                0

 dense_17 (Dense)            (None, 42)                3570

=================================================================
Total params: 14,343
Trainable params: 13,923
Non-trainable params: 420
_________________________________________________________________
```

```
history = model_AE.fit(x1,x1,epochs = 50,batch_size = 16,verbose = 2,validation_data = (x2,x2))
```

```
10959/10959 - 30s - loss: 3.9145e-04 - val_loss: 7.2702e-04 - 30s/epoch - 3ms/step
Epoch 22/50
10959/10959 - 29s - loss: 3.9789e-04 - val_loss: 6.7395e-04 - 29s/epoch - 3ms/step
Epoch 23/50
```

```
Epoch 23/50
10959/10959 - 29s - loss: 3.9491e-04 - val_loss: 6.7184e-04 - 29s/epoch - 3ms/step
Epoch 24/50
10959/10959 - 30s - loss: 3.7757e-04 - val_loss: 6.2764e-04 - 30s/epoch - 3ms/step
Epoch 25/50
10959/10959 - 35s - loss: 3.8318e-04 - val_loss: 7.8785e-04 - 35s/epoch - 3ms/step
Epoch 26/50
10959/10959 - 29s - loss: 3.7458e-04 - val_loss: 6.7092e-04 - 29s/epoch - 3ms/step
Epoch 27/50
10959/10959 - 30s - loss: 3.6294e-04 - val_loss: 7.2804e-04 - 30s/epoch - 3ms/step
Epoch 28/50
10959/10959 - 34s - loss: 3.5356e-04 - val_loss: 6.1226e-04 - 34s/epoch - 3ms/step
Epoch 29/50
10959/10959 - 29s - loss: 3.6016e-04 - val_loss: 8.6791e-04 - 29s/epoch - 3ms/step
Epoch 30/50
10959/10959 - 35s - loss: 3.6324e-04 - val_loss: 7.0834e-04 - 35s/epoch - 3ms/step
Epoch 31/50
10959/10959 - 35s - loss: 3.4638e-04 - val_loss: 8.2770e-04 - 35s/epoch - 3ms/step
Epoch 32/50
10959/10959 - 30s - loss: 3.4158e-04 - val_loss: 7.3669e-04 - 30s/epoch - 3ms/step
Epoch 33/50
10959/10959 - 35s - loss: 3.4655e-04 - val_loss: 7.7535e-04 - 35s/epoch - 3ms/step
Epoch 34/50
10959/10959 - 30s - loss: 3.5163e-04 - val_loss: 6.9589e-04 - 30s/epoch - 3ms/step
Epoch 35/50
10959/10959 - 36s - loss: 3.6533e-04 - val_loss: 7.1176e-04 - 36s/epoch - 3ms/step
Epoch 36/50
10959/10959 - 35s - loss: 3.3477e-04 - val_loss: 6.7198e-04 - 35s/epoch - 3ms/step
Epoch 37/50
10959/10959 - 29s - loss: 3.4120e-04 - val_loss: 6.8567e-04 - 29s/epoch - 3ms/step
Epoch 38/50
10959/10959 - 36s - loss: 3.3053e-04 - val_loss: 7.2224e-04 - 36s/epoch - 3ms/step
Epoch 39/50
10959/10959 - 36s - loss: 3.3720e-04 - val_loss: 7.7018e-04 - 36s/epoch - 3ms/step
Epoch 40/50
10959/10959 - 35s - loss: 3.3583e-04 - val_loss: 8.1989e-04 - 35s/epoch - 3ms/step
Epoch 41/50
10959/10959 - 35s - loss: 3.2631e-04 - val_loss: 7.4394e-04 - 35s/epoch - 3ms/step
Epoch 42/50
10959/10959 - 35s - loss: 3.2230e-04 - val_loss: 8.2110e-04 - 35s/epoch - 3ms/step
Epoch 43/50
10959/10959 - 34s - loss: 3.2935e-04 - val_loss: 8.8571e-04 - 34s/epoch - 3ms/step
Epoch 44/50
10959/10959 - 35s - loss: 3.2474e-04 - val_loss: 6.9232e-04 - 35s/epoch - 3ms/step
Epoch 45/50
10959/10959 - 30s - loss: 3.1958e-04 - val_loss: 6.3976e-04 - 30s/epoch - 3ms/step
Epoch 46/50
10959/10959 - 32s - loss: 3.1371e-04 - val_loss: 7.3848e-04 - 32s/epoch - 3ms/step
Epoch 47/50
10959/10959 - 30s - loss: 3.1649e-04 - val_loss: 7.8478e-04 - 30s/epoch - 3ms/step
Epoch 48/50
10959/10959 - 30s - loss: 3.2053e-04 - val_loss: 7.2122e-04 - 30s/epoch - 3ms/step
Epoch 49/50
10959/10959 - 34s - loss: 3.1754e-04 - val_loss: 6.6753e-04 - 34s/epoch - 3ms/step
Epoch 50/50
10959/10959 - 35s - loss: 3.1484e-04 - val_loss: 7.7300e-04 - 35s/epoch - 3ms/step
```

```
encoder = Model(inputs = input_data_shape,outputs = bottleneck)
encoder.save('encoder.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` w
```

```
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
import xgboost as xgb
params = {
            'booster':'gblinear',
            'objective':'binary:logistic',
            'max_depth': 5, 'min_child_weight': 1, 'gamma': 0.1, 'subsample': 0.8,
            'colsample_bytree': 0.8, 'scale_pos_weight': 1,
            'alpha': 10,
            'learning_rate': 0.01,
            'n_estimators':1000,'nthread': 4,'seed': 27
         }
xgb_clf = XGBClassifier(**params)
xgb_clf.fit(x1, y1)
y_pred=xgb_clf.predict(x2)
print(y_pred)
accuracy = accuracy_score(y2, y_pred)*100
print("Accuracy before feature selection:-", accuracy)
from sklearn.metrics import f1_score
from sklearn import metrics
print("Confusion Matrix =", metrics.confusion_matrix(y2, y_pred, labels=None,
                                                     sample_weight=None))
print("Recall =", metrics.recall_score(y2, y_pred, labels=None,
```

```
                                            pos_label=1, average='weighted',
                                            sample_weight=None))
print("Precision =", metrics.precision_score(y2, y_pred, labels=None,
                                            pos_label=1, average='weighted',
                                            sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y2, y_pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                 digits=2,
                                                                 output_dict=False))

print("F1 Score = ",f1_score(y2, y_pred, average='macro'))
```

```
[1 1 1 ... 1 1 1]
Accuracy before feature selection:- 55.06000097167566
Confusion Matrix = [[    0 37000]
 [    0 45332]]
Recall = 0.5506000097167566
Precision = 0.30316037070009244
Classification Report =
              precision    recall  f1-score   support

           0       0.00      0.00      0.00     37000
           1       0.55      1.00      0.71     45332

    accuracy                           0.55     82332
   macro avg       0.28      0.50      0.36     82332
weighted avg       0.30      0.55      0.39     82332

F1 Score =  0.35508835693695956
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and
  _warn_prf(average, modifier, msg_start, len(result))
```

```
encoder = load_model('encoder.h5')
x1_encode = encoder.predict(x1)
x2_encode = encoder.predict(x2)
params = {
            'booster':'gblinear',
            'objective':'binary:logistic',
            'max_depth': 5, 'min_child_weight': 1, 'gamma': 0.1, 'subsample': 0.8,
            'colsample_bytree': 0.8, 'scale_pos_weight': 1,
            'alpha': 10,
            'learning_rate': 0.01,
            'n_estimators':1000,'nthread': 4,'seed': 27
         }
model_final = XGBClassifier(**params)
model_final.fit(x1_encode, y1)
y_pred_new = model_final.predict(x2_encode)
accuracy = accuracy_score(y2, y_pred_new)*100
print("Accuracy after feature selection:-", accuracy)
from sklearn.metrics import f1_score
from sklearn import metrics
print("Confusion Matrix =", metrics.confusion_matrix(y2, y_pred_new, labels=None,
                                             sample_weight=None))
print("Recall =", metrics.recall_score(y2, y_pred_new, labels=None,
                                            pos_label=1, average='weighted',
                                            sample_weight=None))
print("Precision =", metrics.precision_score(y2, y_pred_new, labels=None,
                                            pos_label=1, average='weighted',
                                            sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y2, y_pred_new,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                 digits=2,
                                                                 output_dict=False))

print("F1 Score = ",f1_score(y2, y_pred_new, average='macro'))
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manua
Accuracy after feature selection:- 71.33799737647574
Confusion Matrix = [[16787 20213]
 [ 3385 41947]]
Recall = 0.7133799737647574
Precision = 0.745545159489587
Classification Report =
```

```
              precision    recall  f1-score   support

           0       0.83      0.45      0.59     37000
           1       0.67      0.93      0.78     45332

    accuracy                           0.71     82332
   macro avg       0.75      0.69      0.68     82332
weighted avg       0.75      0.71      0.69     82332

F1 Score =  0.683856444205901
```

✓ 1m 0s completed at 5:39 PM