```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model


df=pd.read_csv('/content/SDN_DDoS_.csv')


# Splitting dataset into features and label
X= df.drop('Label', axis =1)
y = df['Label']

# Splitting the dataset into the training set and the test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


# scale data
t = MinMaxScaler()
t.fit(X_train)
X_train = t.transform(X_train)
X_test = t.transform(X_test)


# AutoEncoder Model Preparation
n_inputs = X.shape[1]
# define encoder
input_data_shape= Input(shape=(n_inputs,))
# encoder level 1
encoder= Dense(n_inputs*2)(input_data_shape)
encoder = BatchNormalization()(encoder)
encoder= LeakyReLU()(encoder)
# encoder level 2
encoder= Dense(n_inputs)(encoder)
encoder= BatchNormalization()(encoder)
encoder= LeakyReLU()(encoder)
# bottleneck
n_bottleneck = round(float(n_inputs) / 2.0)
bottleneck = Dense(n_bottleneck)(encoder)
# define decoder, level 1
decoder = Dense(n_inputs)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
# decoder level 2
decoder = Dense(n_inputs*2)(decoder)
```

```
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
# output layer
output = Dense(n_inputs, activation='linear')(decoder)
# define autoencoder model
model = Model(inputs=input_data_shape, outputs=output)
# compile autoencoder model
model.compile(optimizer='adam', loss='mse')


model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 66)]              0

 dense (Dense)               (None, 132)               8844

 batch_normalization (BatchN  (None, 132)              528
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 132)               0

 dense_1 (Dense)             (None, 66)                8778

 batch_normalization_1 (Batc  (None, 66)               264
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 66)                0

 dense_2 (Dense)             (None, 33)                2211

 dense_3 (Dense)             (None, 66)                2244

 batch_normalization_2 (Batc  (None, 66)               264
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 66)                0

 dense_4 (Dense)             (None, 132)               8844

 batch_normalization_3 (Batc  (None, 132)              528
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 132)               0

 dense_5 (Dense)             (None, 66)                8778

=================================================================
Total params: 41,283
Trainable params: 40,491
Non-trainable params: 792
_________________________________________________________________
```

```
# fit the autoencoder model to reconstruct input
history = model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=2, validation_data=(
```

```
Epoch 72/100
3802/3802 - 10s - loss: 1.7640e-04 - val_loss: 6.6507e-04 - 10s/epoch - 3ms/step
Epoch 73/100
3802/3802 - 11s - loss: 1.5097e-04 - val_loss: 6.1739e-04 - 11s/epoch - 3ms/step
Epoch 74/100
3802/3802 - 10s - loss: 1.2466e-04 - val_loss: 8.0289e-04 - 10s/epoch - 3ms/step
Epoch 75/100
3802/3802 - 11s - loss: 1.2532e-04 - val_loss: 9.3148e-04 - 11s/epoch - 3ms/step
Epoch 76/100
3802/3802 - 11s - loss: 2.3296e-04 - val_loss: 5.5499e-04 - 11s/epoch - 3ms/step
Epoch 77/100
3802/3802 - 11s - loss: 1.6846e-04 - val_loss: 8.0853e-04 - 11s/epoch - 3ms/step
Epoch 78/100
3802/3802 - 10s - loss: 2.0552e-04 - val_loss: 2.9493e-04 - 10s/epoch - 3ms/step
Epoch 79/100
3802/3802 - 10s - loss: 1.9249e-04 - val_loss: 3.6369e-04 - 10s/epoch - 3ms/step
Epoch 80/100
3802/3802 - 10s - loss: 1.5481e-04 - val_loss: 7.1519e-04 - 10s/epoch - 3ms/step
Epoch 81/100
3802/3802 - 11s - loss: 1.4743e-04 - val_loss: 8.3820e-04 - 11s/epoch - 3ms/step
Epoch 82/100
3802/3802 - 10s - loss: 1.6440e-04 - val_loss: 0.0012 - 10s/epoch - 3ms/step
Epoch 83/100
3802/3802 - 10s - loss: 1.6820e-04 - val_loss: 4.1108e-04 - 10s/epoch - 3ms/step
Epoch 84/100
3802/3802 - 11s - loss: 2.1986e-04 - val_loss: 0.0018 - 11s/epoch - 3ms/step
Epoch 85/100
3802/3802 - 10s - loss: 1.5675e-04 - val_loss: 0.0029 - 10s/epoch - 3ms/step
Epoch 86/100
3802/3802 - 10s - loss: 1.3213e-04 - val_loss: 0.0011 - 10s/epoch - 3ms/step
Epoch 87/100
3802/3802 - 11s - loss: 2.1554e-04 - val_loss: 4.7409e-04 - 11s/epoch - 3ms/step
Epoch 88/100
3802/3802 - 11s - loss: 1.4872e-04 - val_loss: 4.7926e-04 - 11s/epoch - 3ms/step
Epoch 89/100
3802/3802 - 11s - loss: 2.1318e-04 - val_loss: 3.4333e-04 - 11s/epoch - 3ms/step
Epoch 90/100
3802/3802 - 10s - loss: 1.4490e-04 - val_loss: 5.6028e-04 - 10s/epoch - 3ms/step
Epoch 91/100
3802/3802 - 11s - loss: 1.4984e-04 - val_loss: 5.6425e-04 - 11s/epoch - 3ms/step
Epoch 92/100
3802/3802 - 11s - loss: 1.2632e-04 - val_loss: 6.1842e-04 - 11s/epoch - 3ms/step
Epoch 93/100
3802/3802 - 11s - loss: 1.7509e-04 - val_loss: 5.9471e-04 - 11s/epoch - 3ms/step
Epoch 94/100
3802/3802 - 11s - loss: 1.8509e-04 - val_loss: 8.8566e-04 - 11s/epoch - 3ms/step
Epoch 95/100
3802/3802 - 10s - loss: 1.0618e-04 - val_loss: 0.0241 - 10s/epoch - 3ms/step
Epoch 96/100
3802/3802 - 11s - loss: 1.2687e-04 - val_loss: 3.9471e-04 - 11s/epoch - 3ms/step
Epoch 97/100
3802/3802 - 11s - loss: 7.3496e-05 - val_loss: 3.6882e-04 - 11s/epoch - 3ms/step
```

```
Epoch 98/100
3802/3802 - 10s - loss: 2.1742e-04 - val_loss: 4.8208e-04 - 10s/epoch - 3ms/step
Epoch 99/100
3802/3802 - 10s - loss: 1.7656e-04 - val_loss: 4.2022e-04 - 10s/epoch - 3ms/step
Epoch 100/100
3802/3802 - 10s - loss: 1.3446e-04 - val_loss: 0.0010 - 10s/epoch - 3ms/step
```

```
# define an encoder model (without the decoder)
encoder = Model(inputs=input_data_shape, outputs=bottleneck)
# save the encoder to file
encoder.save('encoder.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be b
```

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
#Compressing the input data using Encoder Model and fitting it on the Logistic Regression moc
# load the model from file
encoder = load_model('encoder.h5')

# encode the train data
X_train_encode = encoder.predict(X_train)
# encode the test data
X_test_encode = encoder.predict(X_test)
# define the model
model = XGBClassifier(max_iter=100)
# fit the model on the training set
model.fit(X_train_encode, y_train)
# make predictions on the test set
yhat = model.predict(X_test_encode)
# calculate classification accuracy
acc = accuracy_score(y_test, yhat)
print(acc)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *
0.9999342364855978
```

✓ 10s completed at 9:55 PM