Assignment No.:-04

Title:-Use Autoencoder to implement anomaly detection. Build the model by using:

a. Import required libraries

b. Upload / access the dataset

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Batch:-B

Roll No.:-37

class:-BEIT

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

```
dataset = pd.read_csv("/content/creditcard.csv")
```

```
dataset.size
```

```
185194
```

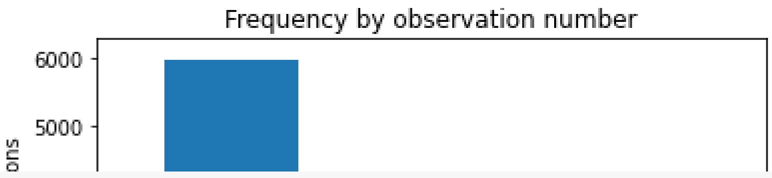
```
dataset.shape
```

```
(5974, 31)
```

```
#check for any  nullvalues
print("Any nulls in the dataset ",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ",dataset.Class.unique())
#0 is for normal credit card transaction
#1 is for fraudulent credit card transaction
print('-------')
print("Break down of the Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'], sort = True) )
```

```
Any nulls in the dataset  True
-------
No. of unique labels  3
Label values  [ 0.  1. nan]
-------
Break down of the Normal and Fraud Transactions
0.0    5970
1.0       3
Name: Class, dtype: int64
```

```
#Visualizing the imbalanced dataset
#plotting the number of normal and fraud transactions in the dataset.
count_classes = pd.value_counts(dataset['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```




```
#The last column in the dataset is our target variable.
raw_data = dataset.values
# The last element contains if the transaction is normal which is represented by a 0 and if fraud then 1
labels = raw_data[:, -1]
# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=2021
)
```

```
labels
```

```
array([ 0.,  0.,  0., ...,  0.,  0., nan])
```

```
# Use only normal transactions to train the Autoencoder.
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
#creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train Data=",len(fraud_train_data))
print(" No. of records in Normal Train data=",len(normal_train_data))
print(" No. of records in Fraud Test Data=",len(fraud_test_data))
print(" No. of records in Normal Test data=",len(normal_test_data))
```

```
 No. of records in Fraud Train Data= 3
 No. of records in Normal Train data= 4776
 No. of records in Fraud Test Data= 1
 No. of records in Normal Test data= 1194
```

```
#Set the training parameter values
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1] #num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7
```

```
#Create the Autoencoder

#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh")(input_layer)
encoder=tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_2, activation='relu')(encoder)
# Decoder
decoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
decoder=tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder)
#Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 30)]              0

 dense_18 (Dense)            (None, 14)                434

 dropout_6 (Dropout)         (None, 14)                0

 dense_19 (Dense)            (None, 7)                 105

 dense_20 (Dense)            (None, 4)                 32

 dense_21 (Dense)            (None, 7)                 35

 dropout_7 (Dropout)         (None, 7)                 0
```

```
 dense_22 (Dense)            (None, 14)                112

 dense_23 (Dense)            (None, 30)                450

=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

```
# Define the callbacks for checkpoints and early stopping

cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",save_best_only=True)

# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True)
```

```
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')
```

```
history = autoencoder.fit(normal_train_data, normal_train_data,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_data, test_data),
                    verbose=1,
                    callbacks=[cp, early_stop]
                    ).history
```

```
Epoch 1/50
75/75 [==============================] - 0s 4ms/step - loss: 340437.3750 - accuracy: 0.9296 - val_loss: nan - val_accuracy: 0.985
Epoch 2/50
75/75 [==============================] - 0s 3ms/step - loss: 340430.3125 - accuracy: 0.9215 - val_loss: nan - val_accuracy: 0.985
Epoch 3/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.7500 - accuracy: 0.9242 - val_loss: nan - val_accuracy: 0.985
Epoch 4/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.5625 - accuracy: 0.9269 - val_loss: nan - val_accuracy: 0.985
Epoch 5/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.6562 - accuracy: 0.9200 - val_loss: nan - val_accuracy: 0.985
Epoch 6/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.3438 - accuracy: 0.9280 - val_loss: nan - val_accuracy: 0.985
Epoch 7/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.3438 - accuracy: 0.9294 - val_loss: nan - val_accuracy: 0.985
Epoch 8/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.4688 - accuracy: 0.9225 - val_loss: nan - val_accuracy: 0.985
Epoch 9/50
75/75 [==============================] - 0s 3ms/step - loss: 340429.2188 - accuracy: 0.9236 - val_loss: nan - val_accuracy: 0.985
Epoch 10/50
64/75 [========================>.....] - ETA: 0s - loss: 341630.9688 - accuracy: 0.9241Restoring model weights from the end of th
75/75 [==============================] - 0s 3ms/step - loss: 340429.2812 - accuracy: 0.9227 - val_loss: nan - val_accuracy: 0.985
Epoch 10: early stopping
```

```
# Detect Anomalies on test data
test_x_predictions = autoencoder.predict(test_data)
test_x_predictions
```

```
38/38 [==============================] - 0s 1ms/step
array([[ 0.99959546, -0.8194607 ,  0.21586528, ...,  0.0033375 ,
        -0.06662563,  0.9929175 ],
       [ 0.99971944, -0.8357152 ,  0.22561987, ...,  0.01192524,
        -0.08088943,  0.99463356],
       [ 0.9973886 , -0.7217145 ,  0.17370744, ..., -0.05207426,
         0.01875337,  0.97195125],
       ...,
       [ 0.99971944, -0.8357152 ,  0.22561987, ...,  0.01192524,
        -0.08088943,  0.99463356],
       [ 0.99971944, -0.8357152 ,  0.22561987, ...,  0.01192524,
        -0.08088943,  0.99463356],
       [ 0.99971944, -0.8357152 ,  0.22561987, ...,  0.01192524,
        -0.08088943,  0.99463356]], dtype=float32)
```

```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
```

```
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```