Name: Jay Goyal

Roll No: C017

Experiment 9

Anomaly Detection using Autoencoders

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#Downloading the dataset
PATH_TO_DATA = 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.cs
data = pd.read_csv(PATH_TO_DATA,header=None)
```

```
data.shape
```

```
(4998, 141)
```

```
data.head()
```

	0	1	2	3	4	5	6
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510

5 rows × 141 columns

```
TARGET=140
```

```
#separating features and labels
features = data.drop(TARGET,axis=1)
target=data[TARGET]
```

```
#split the data into tarin and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features, target, test_size=0.2,


#autoencoders are trained on normal data samples
train_index=y_train[y_train==1].index
train_data=X_train.loc[train_index]


#preprocessing the data
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler(feature_range=(0,1))
X_train_scaled = min_max_scaler.fit_transform(train_data)
X_test_scaled = min_max_scaler.transform(X_test)


X_train_scaled

     array([[0.53032863, 0.35732197, 0.13136741, ..., 0.58989718, 0.72968416,
             0.64554917],
            [0.71774313, 0.49932166, 0.16360805, ..., 0.54700548, 0.56321967,
             0.47713917],
            [0.71985295, 0.59654539, 0.36649653, ..., 0.70882181, 0.801835  ,
             0.64562095],
            ...,
            [0.43994267, 0.42297589, 0.32084589, ..., 0.49412713, 0.44742198,
             0.37968101],
            [0.7966341 , 0.67969636, 0.42505844, ..., 0.81916794, 0.92264633,
             0.60376345],
            [0.58172221, 0.53164252, 0.28278679, ..., 0.75069722, 0.70575296,
             0.501402  ]])


X_test_scaled

     array([[0.61902289, 0.6062941 , 0.47763508, ..., 0.13389981, 0.18302171,
             0.40544082],
            [0.52399443, 0.40571699, 0.10669065, ..., 0.6252717 , 0.73008626,
             0.67931036],
            [0.68440074, 0.61789448, 0.40990557, ..., 0.81496046, 0.77815655,
             0.83723282],
            ...,
            [0.70551343, 0.64090179, 0.40273765, ..., 0.43347687, 0.47604965,
             0.36215781],
            [0.7429811 , 0.69943577, 0.4935516 , ..., 0.57782585, 0.55981495,
             0.47831898],
            [0.57186399, 0.59417708, 0.50420384, ..., 0.738499  , 0.72851827,
             0.79537012]])


from keras.layers import Dense, Dropout
import keras
```

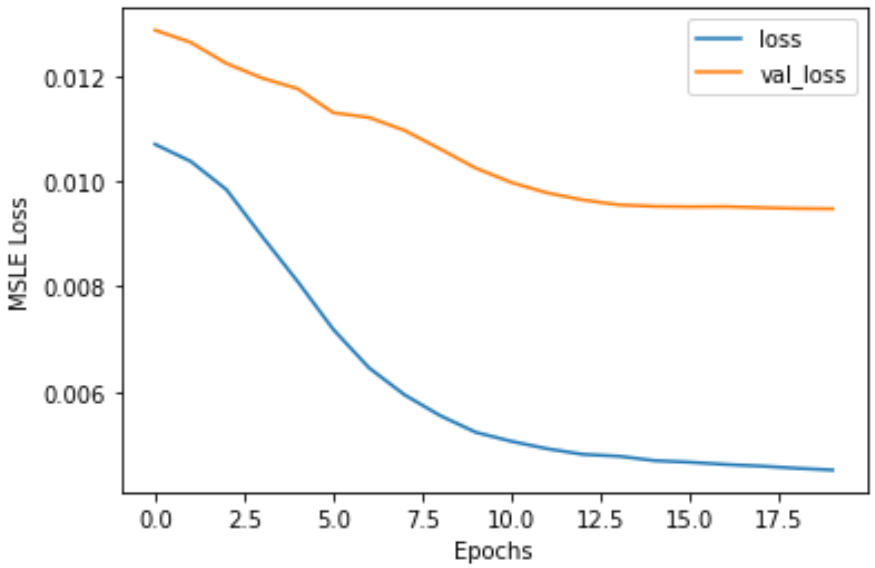
```
#Building an encoder network
input_shape=keras.Input(shape=(X_train_scaled.shape[1],))
x=Dense(64, activation='relu')(input_shape)
x=Dropout(0.1)(x)
x=Dense(32, activation='relu')(x)
x=Dropout(0.1)(x)
x=Dense(16, activation='relu')(x)
x=Dropout(0.1)(x)
encoder_layer=Dense(8, activation='relu')(x)


#Building the decoder network
x=Dense(16, activation='relu')(encoder_layer)
x=Dropout(0.1)(x)
x=Dense(32, activation='relu')(x)
x=Dropout(0.1)(x)
x=Dense(64, activation='relu')(x)
x=Dropout(0.1)(x)
decoder_layer = Dense(X_train_scaled.shape[1], activation='sigmoid')(x)


#defining the autoencoder
autoencoder = keras.Model(input_shape,decoder_layer)


#compiling the model
autoencoder.compile(loss='msle', metrics=['mse'], optimizer='adam')


#fit the model
history = autoencoder.fit(
    X_train_scaled,X_train_scaled,
    epochs=20,batch_size=512,
    validation_data=(X_test_scaled,X_test_scaled))
```

```
 Epoch 1/20
 5/5 [==============================] - 1s 48ms/step - loss: 0.0107 - mse: 0
 Epoch 2/20
 5/5 [==============================] - 0s 12ms/step - loss: 0.0104 - mse: 0
 Epoch 3/20
 5/5 [==============================] - 0s 13ms/step - loss: 0.0098 - mse: 0
 Epoch 4/20
 5/5 [==============================] - 0s 12ms/step - loss: 0.0090 - mse: 0
 Epoch 5/20
 5/5 [==============================] - 0s 13ms/step - loss: 0.0081 - mse: 0
 Epoch 6/20
 5/5 [==============================] - 0s 13ms/step - loss: 0.0072 - mse: 0
 Epoch 7/20
 5/5 [==============================] - 0s 19ms/step - loss: 0.0065 - mse: 0
 Epoch 8/20
 5/5 [==============================] - 0s 13ms/step - loss: 0.0059 - mse: 0
```

```
Epoch 9/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0055 - mse: 0
Epoch 10/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0052 - mse: 0
Epoch 11/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0051 - mse: 0
Epoch 12/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0049 - mse: 0
Epoch 13/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0048 - mse: 0
Epoch 14/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0048 - mse: 0
Epoch 15/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0047 - mse: 0
Epoch 16/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0047 - mse: 0
Epoch 17/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0046 - mse: 0
Epoch 18/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0046 - mse: 0
Epoch 19/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0045 - mse: 0
Epoch 20/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0045 - mse: 0
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('MSLE Loss')
plt.legend(['loss','val_loss'])
```

```
<matplotlib.legend.Legend at 0x7efe17826a90>
```




```
def find_threshold(model, X_train_scaled):
  reconstructions=model.predict(X_train_scaled)
  reconstructions_errors=keras.losses.msle(reconstructions,X_train_scaled)
```

```
  #threshold for anomaly scores
  threshold = np.mean(reconstructions_errors.numpy()) + np.std(reconstructions_er
  return threshold


def get_prediction(model, X_test_scaled, threshold):
  predictions=model.predict(X_test_scaled)
  errors=keras.losses.msle(predictions,X_test_scaled)
  anomaly_mask = pd.Series(errors)>threshold
  preds = anomaly_mask.map(lambda x:0.0 if x==True else 1.0)
  return preds


threshold=find_threshold(autoencoder, X_train_scaled)
threshold
```

```
0.01000504511446505
```

```
from sklearn.metrics import accuracy_score
preds = get_prediction(autoencoder,X_test_scaled,threshold)
accuracy_score(preds,y_test)
```

```
0.949
```

✓ 0s completed at 5:22 PM