**Assignment4**

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

In [1]:

*#importing libraries and dataset*

**import** numpy **as** np

**import** pandas **as** pd

**import** tensorflow **as** tf

**import** matplotlib.pyplot **as** plt

**from** sklearn.metrics **import** accuracy\_score

**from** tensorflow.keras.optimizers **import** Adam

**from** sklearn.preprocessing **import** MinMaxScaler

**from** tensorflow.keras **import** Model, Sequential

**from** tensorflow.keras.layers **import** Dense, Dropout

**from** sklearn.model\_selection **import** train\_test\_split

**from** tensorflow.keras.losses **import** MeanSquaredLogarithmicError

PATH\_TO\_DATA **=** 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'

data **=** pd**.**read\_csv(PATH\_TO\_DATA, header**=None**)

data**.**head()

*#finding shape of the dataset*

data**.**shape

*#splitting training and testing dataset*

features **=** data**.**drop(140, axis**=**1)

target **=** data[140]

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(

features, target, test\_size**=**0.2, stratify**=**target

)

train\_index **=** y\_train[y\_train **==** 1]**.**index

train\_data **=** x\_train**.**loc[train\_index]

*#scaling the data using MinMaxScaler*

min\_max\_scaler **=** MinMaxScaler(feature\_range**=**(0, 1))

x\_train\_scaled **=** min\_max\_scaler**.**fit\_transform(train\_data**.**copy())

x\_test\_scaled **=** min\_max\_scaler**.**transform(x\_test**.**copy())

*#creating autoencoder subclass by extending Model class from keras*

**class** AutoEncoder(Model):

**def** \_\_init\_\_(self, output\_units, ldim**=**8):

super()**.**\_\_init\_\_()

self**.**encoder **=** Sequential([

Dense(64, activation**=**'relu'),

Dropout(0.1),

Dense(32, activation**=**'relu'),

Dropout(0.1),

Dense(16, activation**=**'relu'),

Dropout(0.1),

Dense(ldim, activation**=**'relu')

])

self**.**decoder **=** Sequential([

Dense(16, activation**=**'relu'),

Dropout(0.1),

Dense(32, activation**=**'relu'),

Dropout(0.1),

Dense(64, activation**=**'relu'),

Dropout(0.1),

Dense(output\_units, activation**=**'sigmoid')

])

**def** call(self, inputs):

encoded **=** self**.**encoder(inputs)

decoded **=** self**.**decoder(encoded)

**return** decoded

*#model configuration*

model **=** AutoEncoder(output\_units**=**x\_train\_scaled**.**shape[1])

model**.**compile(loss**=**'msle', metrics**=**['mse'], optimizer**=**'adam')

epochs **=** 20

history **=** model**.**fit(

x\_train\_scaled,

x\_train\_scaled,

epochs**=**epochs,

batch\_size**=**512,

validation\_data**=**(x\_test\_scaled, x\_test\_scaled)

)

plt**.**plot(history**.**history['loss'])

plt**.**plot(history**.**history['val\_loss'])

plt**.**xlabel('Epochs')

plt**.**ylabel('MSLE Loss')

plt**.**legend(['loss', 'val\_loss'])

plt**.**show()

*#finding threshold for anomaly and doing predictions*

**def** find\_threshold(model, x\_train\_scaled):

reconstructions **=** model**.**predict(x\_train\_scaled)

reconstruction\_errors **=** tf**.**keras**.**losses**.**msle(reconstructions, x\_train\_scaled)

threshold **=** np**.**mean(reconstruction\_errors**.**numpy()) \

**+** np**.**std(reconstruction\_errors**.**numpy())

**return** threshold

**def** get\_predictions(model, x\_test\_scaled, threshold):

predictions **=** model**.**predict(x\_test\_scaled)

errors **=** tf**.**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(model, x\_train\_scaled)

print(f"Threshold: {threshold}")

*#getting accuracy score*

predictions **=** get\_predictions(model, x\_test\_scaled, threshold)

accuracy\_score(predictions, y\_test)