

## Assignment No-4

**Aim:** Use Autoencoder to implement anomaly detection.

**Build the model by using the following:**

- a. Import required libraries
- b. Upload/access the dataset
- c. The encoder converts it into a latent representation
- d. Decoder networks convert it back to the original input
- e. Compile the models with Optimizer, Loss, and Evaluation Metrics

**Objective:** Student will learn:

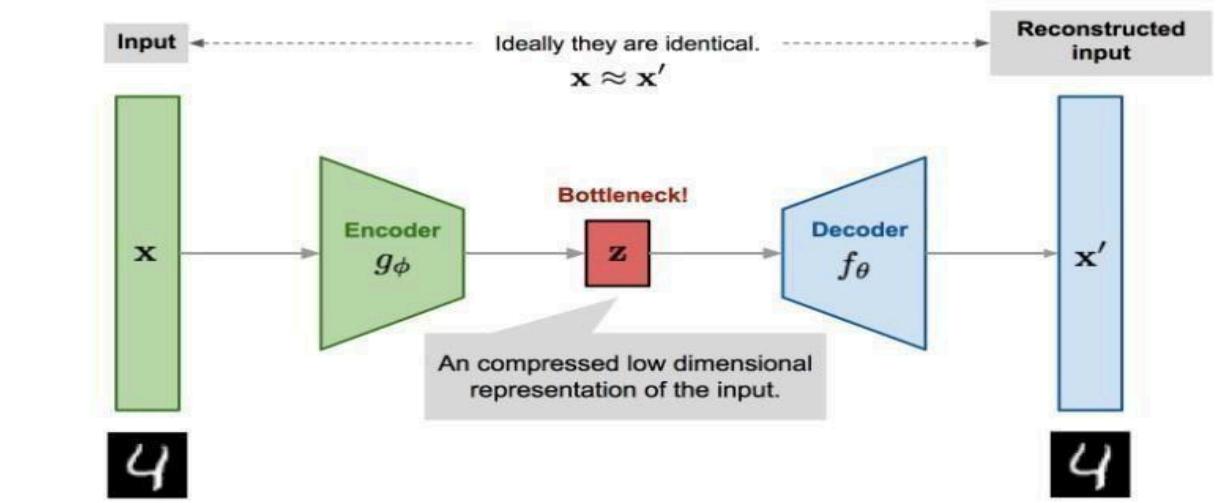
- 1) Use Autoencoder to implement anomaly detection.

**Theory:**

**What is an autoencoder?**

An autoencoder is a type of [neural network](#) architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation.

AutoEncoder is a generative unsupervised deep learning algorithm used for reconstructing high-dimensional input data using a neural network with a narrow bottleneck layer in the middle which contains the latent representation of the input data.



**An autoencoder consists of the following two primary components:**

Autoencoders discover latent variables by passing input data through a “bottleneck” before it reaches the decoder. This forces the encoder to learn to extract and pass through only the information most conducive to accurately reconstructing the original input.

1. Encoder-The encoder comprises layers that *encode* a compressed representation of the input data through *dimensionality reduction*. In a typical autoencoder, the hidden layers of the neural network contain a progressively smaller number of nodes than the input layer: as data traverses the encoder layers, it is compressed by the process of “squeezing” itself into fewer dimensions.
2. Bottleneck- The bottleneck (or “code”) contains the most compressed representation of the input: it is both the output layer of the encoder network and the input layer of the *decoder* network. A fundamental goal of the design and training of an autoencoder is discovering the minimum number of important features (or *dimensions*) needed for effective reconstruction of the input data. The latent space representation—that is, the *code*—emerging from this layer is then fed into the decoder.
3. Decoder-The decoder comprises hidden layers with a progressively larger number of nodes that decompress (or *decode*) the encoded representation of data, ultimately reconstructing the data back to its original, pre-encoding form. This reconstructed output is then compared to the “ground truth”—which in most cases is simply the original input—to gauge the efficacy of the autoencoder. The difference between the output and ground truth is called the *reconstruction error*.

The key idea is that AutoEncoders are trained to minimize reconstruction errors, which makes them efficient in learning the distribution of the input data.

### **AutoEncoders for Anomaly Detection:**

They are trained on normal data to learn the representation of the normal state. During inference, if an input significantly deviates from this learned representation, the AutoEncoder will likely reconstruct it poorly. This poor reconstruction is a signal of an anomaly.

### **How Does It Work ?**

1. Training: The AutoEncoder is trained exclusively on normal data. The training process involves adjusting the weights to minimize the reconstruction error.
2. Inference: During inference, we feed new data to the Autoencoder. If the data is normal, the AutoEncoder will successfully reconstruct it with minimal error. However, if the data is anomalous, the reconstruction error will be significantly higher.
3. Thresholding: We set a threshold for the reconstruction error. If the error surpasses this threshold, the data point is flagged as an anomaly.

## **Steps/ Algorithm:**

### **1. Dataset link and libraries :**

Dataset : <http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv>

### **Libraries required :**

Import the required libraries and load the data. Here we are using the ECG data which consists of labels 0 and 1. Label 0 denotes the observation as an anomaly and label 1 denotes the observation as normal.

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

### **For Model building and evaluation following libraries:**

```
sklearn.metrics import accuracy_score  
tensorflow.keras.optimizers import Adam  
sklearn.preprocessing import MinMaxScaler  
tensorflow.keras import Model, Sequential  
tensorflow.keras.layers import Dense, Dropout  
tensorflow.keras.losses import MeanSquaredLogarithmicError  
Ref:https://www.analyticsvidhya.com/blog/2021/05/anomaly-detection-using-autoencoder s-a-walk-through-in-python/
```

#### **a) Import following libraries from SKlearn :**

- i) MinMaxscaler (sklearn.preprocessing)
- ii) Accuracy(sklearn.metrics) .
- iii) train\_test\_split (model\_selection)

#### **b) Import Following libraries from tensorflow.keras : models , layers, optimizers, datasets ,and set to respective values.**

#### **c) Grab to ECG.csv required dataset -**

```
PATH_TO_DATA = 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'
```

```
data = pd.read_csv(PATH_TO_DATA, header=None)
```

#### **d) Find shape of dataset- data.shape**

#### **e) Use train\_test\_split from sklearn to build model (e.g. train\_test\_split(features, target, test\_size=0.2, stratify=target)**

f) Take use case Novelty detection hence select training data set as Target class is 1 i.e.Normal class

g) Scale the data using MinMaxScaler.

The last column in the data is the target ( column name is 140). Split the data for training and testing and scale the data using MinMaxScaler.

h) Create Autoencoder Subclass by extending model class from keras.

i) Select parameters as -

1. Encoder : 4 layers
2. Decoder : 4 layers
3. Activation Function :Relu
4. Model : sequential.

j) Configure model with following parameters : epoch = 20 , batch size =512 and compile with Mean Squared Logarithmic loss and Adam optimizer.

```
e.g. model = AutoEncoder(output_units=x_train_scaled.shape[1])
# configurations of model
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
history=model.fit(x_train_scaled,x_train_scaled,epochs=20,batch_size=512,
validation_data=(x_test_scaled, x_test_scaled))
```

k) Plot loss,Val\_loss, Epochs and msle loss:

l) Find threshold for anomaly and do predictions :

Anomalies are data points where the reconstruction loss is higher.

To calculate the reconstruction loss on test data, predict the test data and calculate the mean square error between the test data and the reconstructed test data.

```
e.g. : find_threshold(model, x_train_scaled):
reconstructions = model.predict(x_train_scaled)
```

# provides losses of individual instances

```
reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)
# threshold for anomaly scores
threshold=np.mean(reconstruction_errors.numpy())                                     \+
np.std(reconstruction_errors.numpy())
return threshold
m) Get accuracy score-
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
```

## **Conclusion:**

Autoencoders can be used as an anomaly detection algorithm when we have an unbalanced dataset where we have a lot of good examples and only a few anomalies. Autoencoders are trained to minimize reconstruction error. When we train the autoencoders on normal data or good data, we can hypothesize that the anomalies will have higher reconstruction errors than the good or normal data.