Assignment 4: ECG Anomaly detection using Autoencoders

Name: Zahir Mulla

In [11]:
```
#Rollno: B512012
#Class: BE-IT(B)
```

In [2]:
```
#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()
```

Out[2]:

	0	1	2	3	4	5	6	7	8
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450

5 rows × 141 columns

In [3]:
```
#finding shape of the dataset
data.shape
```

Out[3]:
```
(4998, 141)
```

In [4]:
```
#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]
```

In [5]:
```
#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())
```

In [6]:
```
#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
```

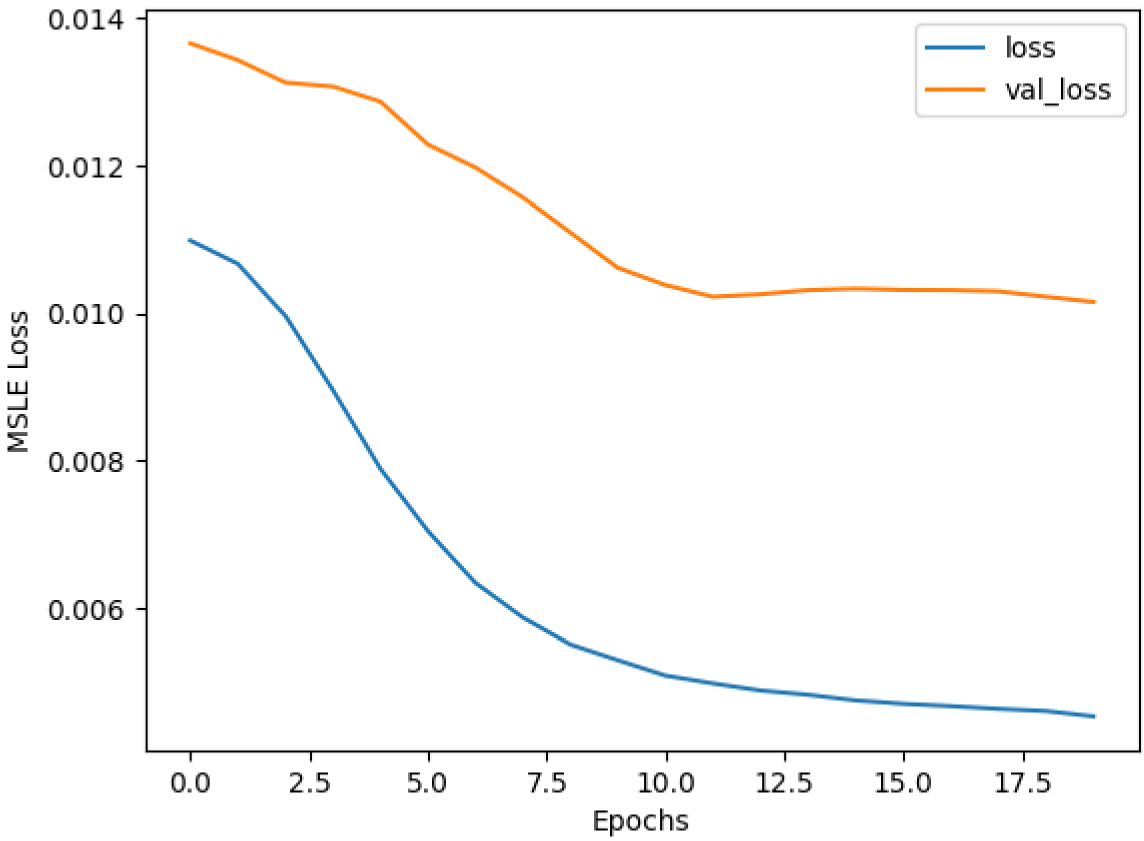
In [7]:
```
#model configuration
model = AutoEncoder(output_units=x_train_scaled.shape[1])
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
epochs = 20

history = model.fit(
    x_train_scaled,
    x_train_scaled,
    epochs=epochs,
    batch_size=512,
    validation_data=(x_test_scaled, x_test_scaled)
)
```

```
Epoch 1/20
5/5 [==============================] - 4s 108ms/step - loss: 0.0110 - mse: 0.0248
- val_loss: 0.0137 - val_mse: 0.0318
Epoch 2/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0107 - mse: 0.0241 -
val_loss: 0.0134 - val_mse: 0.0312
Epoch 3/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0100 - mse: 0.0225 -
val_loss: 0.0131 - val_mse: 0.0305
Epoch 4/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0090 - mse: 0.0201 -
val_loss: 0.0131 - val_mse: 0.0302
Epoch 5/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0079 - mse: 0.0177 -
val_loss: 0.0129 - val_mse: 0.0297
Epoch 6/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0070 - mse: 0.0158 -
val_loss: 0.0123 - val_mse: 0.0284
Epoch 7/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0063 - mse: 0.0142 -
val_loss: 0.0120 - val_mse: 0.0277
Epoch 8/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0059 - mse: 0.0131 -
val_loss: 0.0116 - val_mse: 0.0268
Epoch 9/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0055 - mse: 0.0123 -
val_loss: 0.0111 - val_mse: 0.0257
Epoch 10/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0053 - mse: 0.0118 -
val_loss: 0.0106 - val_mse: 0.0246
Epoch 11/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0051 - mse: 0.0114 -
val_loss: 0.0104 - val_mse: 0.0241
Epoch 12/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0050 - mse: 0.0111 -
val_loss: 0.0102 - val_mse: 0.0238
Epoch 13/20
5/5 [==============================] - 0s 24ms/step - loss: 0.0049 - mse: 0.0109 -
val_loss: 0.0103 - val_mse: 0.0239
Epoch 14/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0048 - mse: 0.0108 -
val_loss: 0.0103 - val_mse: 0.0240
Epoch 15/20
5/5 [==============================] - 0s 21ms/step - loss: 0.0048 - mse: 0.0106 -
val_loss: 0.0103 - val_mse: 0.0241
Epoch 16/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0047 - mse: 0.0105 -
val_loss: 0.0103 - val_mse: 0.0240
Epoch 17/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0047 - mse: 0.0105 -
val_loss: 0.0103 - val_mse: 0.0240
Epoch 18/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0046 - mse: 0.0104 -
val_loss: 0.0103 - val_mse: 0.0240
Epoch 19/20
5/5 [==============================] - 0s 23ms/step - loss: 0.0046 - mse: 0.0103 -
val_loss: 0.0102 - val_mse: 0.0238
Epoch 20/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0045 - mse: 0.0102 -
val_loss: 0.0102 - val_mse: 0.0237
```

In [8]:

```
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()
```



In [9]:
```
#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}")
```

```
73/73 [==============================] - 0s 3ms/step
Threshold: 0.009868882315032265
```

In [10]:
```
#getting accuracy score
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
```

```
32/32 [==============================] - 0s 2ms/step
```

Out[10]:
```
0.932
```

In [ ]: