In [2]:

```
Vaibhav Rokade
Roll No:54
```

In [ ]:

```
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
```

Load the data

In [3]:

```
# Download the dataset
path = '''http://storage.googleapis.com/
download.tensorflow.org/data/ecg.csv'''
data = pd.read_csv(path, header=None)
print(data.shape)
data.head()
```

```
(4998, 141)
```

Out[3]:

	0	1	2	3	4	5	6	7	8
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818287	-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	-1.490658	-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

Split the data for training and testing ¶

In [4]:

```
# last column is the target
# 0 = anomaly, 1 = normal
TARGET = 140

features = data.drop(TARGET, axis=1)
target = data[TARGET]

x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2,
    random_state = 0, stratify=target
)
```

In [ ]:

```
x_test.shape
```

In [ ]:

```
x_train.shape
```

In [ ]:

```
target.value_counts()
```

In [5]:

```
# use case is novelty detection so use only the normal data
# for training
train_index = y_train[y_train == 1].index
train_data = x_train.loc[train_index]
```

Scale the data using MinMaxScaler

In [6]:

```
min_max_scaler = MinMaxScaler()
x_train_scaled = min_max_scaler.fit_transform(
    train_data.copy())
x_test_scaled = min_max_scaler.transform(x_test.copy())
```

In [ ]:

```
x_train.describe()
```

In [ ]:

```
pd.DataFrame(x_train_scaled).describe()
```

Build an AutoEncoder model

In [7]:

```
# create a model by subclassing Model class in tensorflow
class AutoEncoder(Model):
  """
  Parameters
  ----------
  output_units: int
    Number of output units

  code_size: int
    Number of units in bottle neck
  """

  def __init__(self, output_units, code_size=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(code_size, 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 [8]:

```
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)
)
```

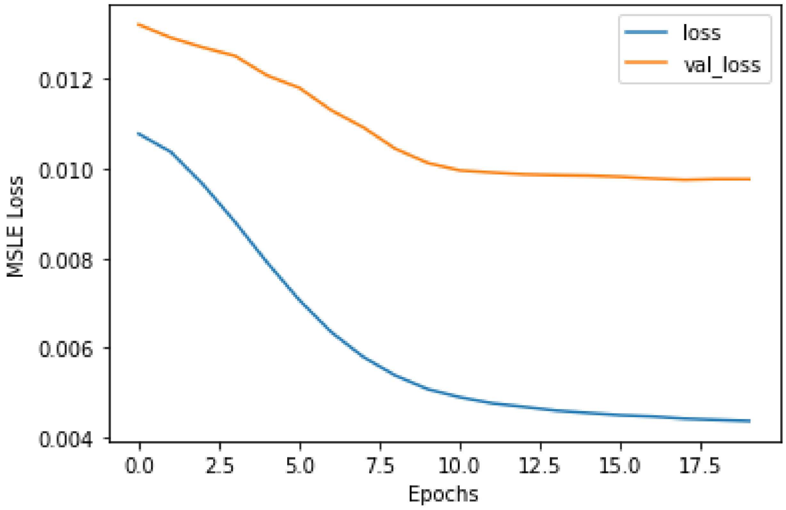
```
Epoch 1/20
5/5 [==============================] - 0s 51ms/step - loss: 0.0108 - mse: 0.0
243 - val_loss: 0.0132 - val_mse: 0.0307
Epoch 2/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0104 - mse: 0.02
33 - val_loss: 0.0129 - val_mse: 0.0300
Epoch 3/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0096 - mse: 0.02
17 - val_loss: 0.0127 - val_mse: 0.0294
Epoch 4/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0088 - mse: 0.01
97 - val_loss: 0.0125 - val_mse: 0.0289
Epoch 5/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0079 - mse: 0.01
76 - val_loss: 0.0121 - val_mse: 0.0279
Epoch 6/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0071 - mse: 0.01
57 - val_loss: 0.0118 - val_mse: 0.0272
Epoch 7/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0063 - mse: 0.01
41 - val_loss: 0.0113 - val_mse: 0.0261
Epoch 8/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0058 - mse: 0.01
28 - val_loss: 0.0109 - val_mse: 0.0252
Epoch 9/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0054 - mse: 0.01
19 - val_loss: 0.0104 - val_mse: 0.0242
Epoch 10/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0051 - mse: 0.01
13 - val_loss: 0.0101 - val_mse: 0.0235
Epoch 11/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0049 - mse: 0.01
09 - val_loss: 0.0100 - val_mse: 0.0231
Epoch 12/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0048 - mse: 0.01
06 - val_loss: 0.0099 - val_mse: 0.0230
Epoch 13/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0047 - mse: 0.01
04 - val_loss: 0.0099 - val_mse: 0.0230
Epoch 14/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0046 - mse: 0.0
103 - val_loss: 0.0099 - val_mse: 0.0229
Epoch 15/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0045 - mse: 0.0
101 - val_loss: 0.0098 - val_mse: 0.0229
Epoch 16/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0045 - mse: 0.0
100 - val_loss: 0.0098 - val_mse: 0.0229
Epoch 17/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0045 - mse: 0.0
100 - val_loss: 0.0098 - val_mse: 0.0228
Epoch 18/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0044 - mse: 0.0
099 - val_loss: 0.0097 - val_mse: 0.0227
Epoch 19/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0044 - mse: 0.00
98 - val_loss: 0.0098 - val_mse: 0.0228
```

```
Epoch 20/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0044 - mse: 0.00
98 - val_loss: 0.0098 - val_mse: 0.0228
```

Plot history

In [9]:

```
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()
```


loss
val_loss
MSLE Loss
Epochs