Practical No.4

```
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


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

Output:

0123456789101112131415161718192021222324252627282930313233343536373839...101
102103104105106107108109110111112113114115116117118119120121122123124125126
127128129130131132133134135136137138139140 0-0.112522-2.827204-3.773897-
4.349751-4.376041-3.474986-2.181408-1.818286-1.250522-0.477492-0.363808-0.491957-
0.421855-0.309201-0.495939-0.342119-0.355336-0.367913-0.316503-0.412374-0.471672-
0.413458-0.364617-0.449298-0.471419-0.424777-0.462517-0.552472-0.475375-0.694200-
0.701868-0.593812-0.660684-0.713831-0.769807-0.672282-0.653676-0.639406-0.559302-
0.591670...1.2581791.4337891.7005331.9990432.1253411.9932911.9322461.7974371.5222
841.2511680.9987300.4837220.023132-0.194914-0.220917-0.243737-0.254695-0.291136-
0.256490-0.227874-0.322423-0.289286-0.318170-0.363654-0.393456-0.266419-0.256823-
0.288694-
0.1623380.1603480.7921680.9335410.7969580.5786210.2577400.2280770.1234310.925286
0.1931371.01-1.100878-3.996840-4.285843-4.506579-4.022377-3.234368-1.566126-
0.992258-
0.7546800.0423210.1489510.1835270.2948760.1902330.2355750.2534870.2217420.050233
0.1780420.1395630.0467940.0430070.1065440.0126540.0039950.045724-0.045999-
0.072667-0.071078-0.153866-0.227254-0.249270-0.253489-0.332835-0.264330-0.345825-
0.310781-0.334160-0.306178-
0.174563...1.8084282.1643462.0707471.9036141.7644551.5077691.2934280.8945620.5780
160.244343-0.286443-0.515881-0.732707-0.832465-0.803318-0.836252-0.777865-
0.774753-0.733404-0.721386-0.832095-0.711982-0.751867-0.757720-0.853120-0.766988-
0.688161-
0.5199230.0394060.5603270.5383560.6568810.7874900.7240460.5557840.4763330.773820
1.119621-1.4362501.02-0.567088-2.593450-3.874230-4.584095-4.187449-3.151462-
1.742940-1.490659-1.183580-0.394229-0.282897-0.356926-0.287297-0.399489-0.473244-
0.379048-0.399039-0.178594-0.339522-0.498447-0.337251-0.425480-0.423952-0.463170-
0.493253-0.549749-0.529831-0.530935-0.502365-0.417368-0.526346-0.471005-0.676784-

0.898612-0.610571-0.530164-0.765674-0.581937-0.537848-
0.556386...1.8109882.1853982.2629852.0529201.8904881.7930331.5647841.2346190.9003
020.5519570.258222-0.128587-0.092585-0.168606-0.495989-0.395034-0.328238-0.448138-
0.268230-0.456415-0.357867-0.317508-0.434112-0.549203-0.324615-0.268082-0.220384-
0.1174290.6140591.2848250.8860730.5314520.311377-0.021919-0.713683-
0.5321970.3210970.904227-0.4217971.030.490473-1.914407-3.616364-4.318823-4.268016-
3.881110-2.993280-1.671131-1.333884-0.965629-0.183319-0.101657-0.273874-0.127818-
0.195983-0.213523-0.176473-0.156932-0.149172-0.181510-0.180074-0.246151-0.274260-
0.140960-0.277449-0.382549-0.311937-0.360093-0.405968-0.571433-0.524106-0.537886-
0.606778-0.661446-0.683375-0.746683-0.635662-0.625231-0.540094-
0.674995...1.7721552.0007691.9250031.8984261.7209531.5017111.4224921.0232250.7763
410.5044260.056382-0.233161-0.406388-0.327528-0.460868-0.402536-0.345752-0.354206-
0.439959-0.425326-0.439789-0.451835-0.395926-0.448762-0.391789-0.376307-0.461069-
0.2535240.2130060.4911730.3508160.4991110.6003450.8420690.9520740.9901331.086798
1.403011-0.3835641.040.800232-0.874252-2.384761-3.973292-4.338224-3.802422-
2.534510-1.783423-1.594450-0.753199-0.298107-0.428928-0.491351-0.361304-0.339296-
0.324952-0.290113-0.363051-0.525684-0.597423-0.575523-0.567503-0.504555-0.618406-
0.682814-0.743849-0.815588-0.826902-0.782374-0.929462-0.999672-1.060969-1.007877-
1.028735-1.122629-1.028650-1.046515-1.063372-1.122423-
0.983242...1.1553631.3362541.6275341.7175941.6964871.7416861.6740781.5469281.3317
381.1101680.9222100.5217770.154852-0.123861-0.202998-0.247956-0.219122-0.214695-
0.319215-0.198597-0.151618-0.129593-0.074939-0.196807-0.174795-0.208833-0.210754-
0.1004850.1974460.9666061.1488840.9584341.0590251.3716821.2773920.9603040.971020
1.6143921.4214561.0

```
# 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, stratify=target
)

# 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]

# min max scale the input data
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())


# create a model by subclassing Model class in tensorflow
```

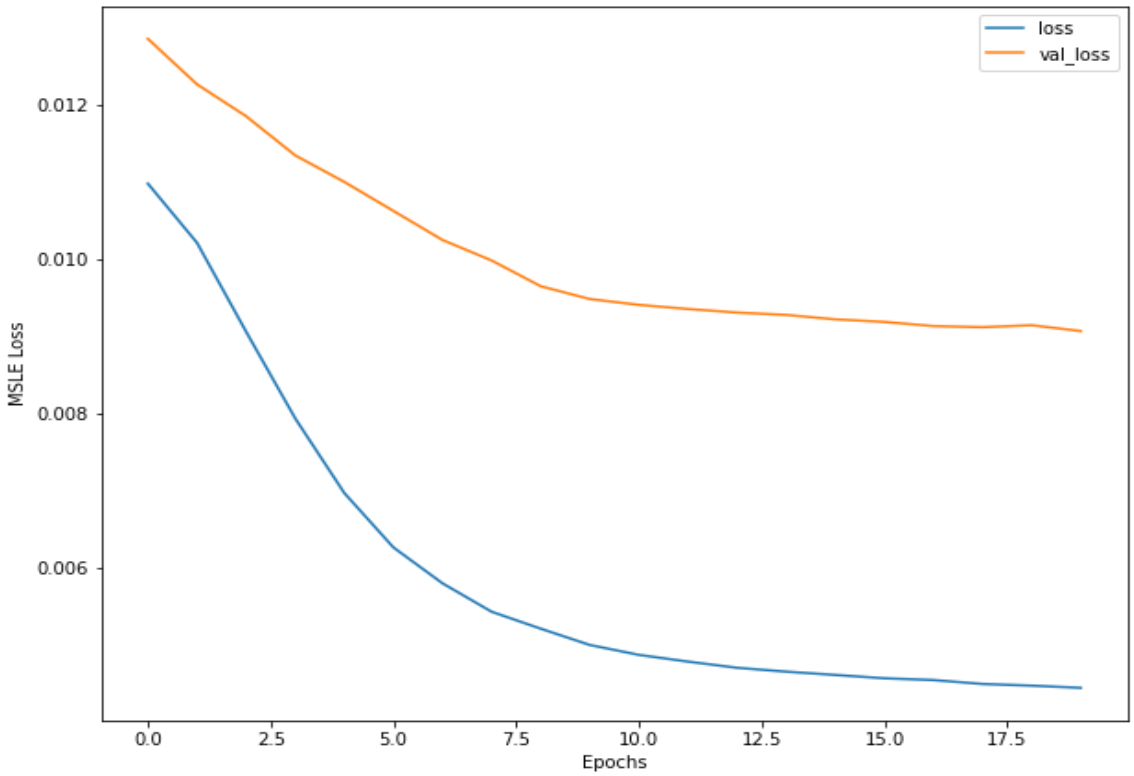
```
class AutoEncoder(Model):
  """
  Parameters
  ----------
  output_units: int
    Number of output units

  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

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)
)
```
Output:

```
Epoch 1/20
5/5 [==============================] - 1s 49ms/step - loss: 0.0110 - mse: 0.0247 - val_loss: 0.0128 - val_mse: 0.0295
Epoch 2/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0102 - mse: 0.0229 - val_loss: 0.0123 - val_mse: 0.0280
Epoch 3/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0091 - mse: 0.0203 - val_loss: 0.0118 - val_mse: 0.0270
Epoch 4/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0079 - mse: 0.0177 - val_loss: 0.0113 - val_mse: 0.0259
Epoch 5/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0070 - mse: 0.0156 - val_loss: 0.0110 - val_mse: 0.0251
Epoch 6/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0063 - mse: 0.0140 - val_loss: 0.0106 - val_mse: 0.0243
Epoch 7/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0058 - mse: 0.0129 - val_loss: 0.0102 - val_mse: 0.0235
Epoch 8/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0054 - mse: 0.0121 - val_loss: 0.0100 - val_mse: 0.0229
Epoch 9/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0052 - mse: 0.0116 - val_loss: 0.0096 - val_mse: 0.0222
Epoch 10/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0050 - mse: 0.0112 - val_loss: 0.0095 - val_mse: 0.0218
Epoch 11/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0049 - mse: 0.0109 - val_loss: 0.0094 - val_mse: 0.0216
Epoch 12/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0048 - mse: 0.0107 - val_loss: 0.0094 - val_mse: 0.0215
Epoch 13/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0047 - mse: 0.0105 - val_loss: 0.0093 - val_mse: 0.0214
Epoch 14/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0047 - mse: 0.0104 - val_loss: 0.0093 - val_mse: 0.0214
Epoch 15/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0046 - mse: 0.0103 - val_loss: 0.0092 - val_mse: 0.0212
Epoch 16/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0046 - mse: 0.0102 - val_loss: 0.0092 - val_mse: 0.0212
Epoch 17/20
```

```
5/5 [==============================] - 0s 14ms/step - loss: 0.0045 - mse: 0.0102 -
val_loss: 0.0091 - val_mse: 0.0210
Epoch 18/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0045 - mse: 0.0100 -
val_loss: 0.0091 - val_mse: 0.0210
Epoch 19/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0045 - mse: 0.0100 -
val_loss: 0.0091 - val_mse: 0.0211
Epoch 20/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0044 - mse: 0.0099 -
val_loss: 0.0091 - val_mse: 0.0209
```

```
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()
```

Output:




```
def 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

def get_predictions(model, x_test_scaled, threshold):
  predictions = model.predict(x_test_scaled)
  # provides losses of individual instances
  errors = tf.keras.losses.msle(predictions, x_test_scaled)
  # 0 = anomaly, 1 = normal
  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}")
# Threshold: 0.01001314025746261
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
# 0.944
```

Output:

```
73/73 [==============================] - 0s 990us/step
Threshold: 0.009627700998111052
32/32 [==============================] - 0s 981us/step
0.947
```