ap-ex4-2

November 5, 2024

1 Import required libraries

2 Upload / access the dataset

3 Encoder converts it into latent representation

4 Decoder networks convert it back to the original input

5 Compile the models with Optimizer, Loss, and Evaluation Metrics

[ ]:
```
# Importing libraries
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
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

# Define the path to the dataset. You can change this to your local file path␣
↪if needed.
path = 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'

# Read the ECG dataset into a Pandas DataFrame
data = pd.read_csv(path, header=None)
```

[ ]:
```
data.head()
```

[ ]:
```
          0         1         2         3         4         5         6    \
0 -0.112522 -2.827204 -3.773897 -4.349751 -4.376041 -3.474986 -2.181408
1 -1.100878 -3.996840 -4.285843 -4.506579 -4.022377 -3.234368 -1.566126
2 -0.567088 -2.593450 -3.874230 -4.584095 -4.187449 -3.151462 -1.742940
3  0.490473 -1.914407 -3.616364 -4.318823 -4.268016 -3.881110 -2.993280
```

```
4  0.800232 -0.874252 -2.384761 -3.973292 -4.338224 -3.802422 -2.534510

          7         8         9  …       131       132       133       134  \
0 -1.818286 -1.250522 -0.477492  …  0.792168  0.933541  0.796958  0.578621
1 -0.992258 -0.754680  0.042321  …  0.538356  0.656881  0.787490  0.724046
2 -1.490659 -1.183580 -0.394229  …  0.886073  0.531452  0.311377 -0.021919
3 -1.671131 -1.333884 -0.965629  …  0.350816  0.499111  0.600345  0.842069
4 -1.783423 -1.594450 -0.753199  …  1.148884  0.958434  1.059025  1.371682

        135       136       137       138       139  140
0  0.257740  0.228077  0.123431  0.925286  0.193137  1.0
1  0.555784  0.476333  0.773820  1.119621 -1.436250  1.0
2 -0.713683 -0.532197  0.321097  0.904227 -0.421797  1.0
3  0.952074  0.990133  1.086798  1.403011 -0.383564  1.0
4  1.277392  0.960304  0.971020  1.614392  1.421456  1.0

[5 rows x 141 columns]
```

[ ]:
```python
# Get information about the dataset, such as column data types and non-null␣
↪counts
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4998 entries, 0 to 4997
Columns: 141 entries, 0 to 140
dtypes: float64(141)
memory usage: 5.4 MB
```

[ ]:
```python
# Splitting the dataset into features and target
features = data.drop(140, axis=1)  # Features are all columns except the last␣
↪(column 140)
target = data[140]  # Target is the last column (column 140)

# Split the data into training and testing sets (80% training, 20% testing)
x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2
)

# Get the indices of the training data points labeled as "1" (anomalies)
train_index = y_train[y_train == 1].index

# Select the training data points that are anomalies
train_data = x_train.loc[train_index]
```

[ ]:
```python
# Initialize the Min-Max Scaler to scale the data between 0 and 1
min_max_scaler = MinMaxScaler(feature_range=(0, 1))
```

```python
# Scale the training data
x_train_scaled = min_max_scaler.fit_transform(train_data.copy())

# Scale the testing data using the same scaler
x_test_scaled = min_max_scaler.transform(x_test.copy())
```

[ ]:
```python
# Creating an Autoencoder model by extending the Model class from Keras
class AutoEncoder(Model):
    def __init__(self, output_units, ldim=8):
        super().__init__()
        # Define the encoder part of the Autoencoder
        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')
        ])
        # Define the decoder part of the Autoencoder
        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):
        # Forward pass through the Autoencoder
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded
```

[ ]:
```python
# Create an instance of the AutoEncoder model with the appropriate output units
model = AutoEncoder(output_units=x_train_scaled.shape[1])

# Compile the model with Mean Squared Logarithmic Error (MSLE) loss and Mean␣
↪Squared Error (MSE) metric
model.compile(loss='msle', metrics=['mse'], optimizer='adam')

# Train the model using the scaled training data
history = model.fit(
    x_train_scaled,  # Input data for training
```

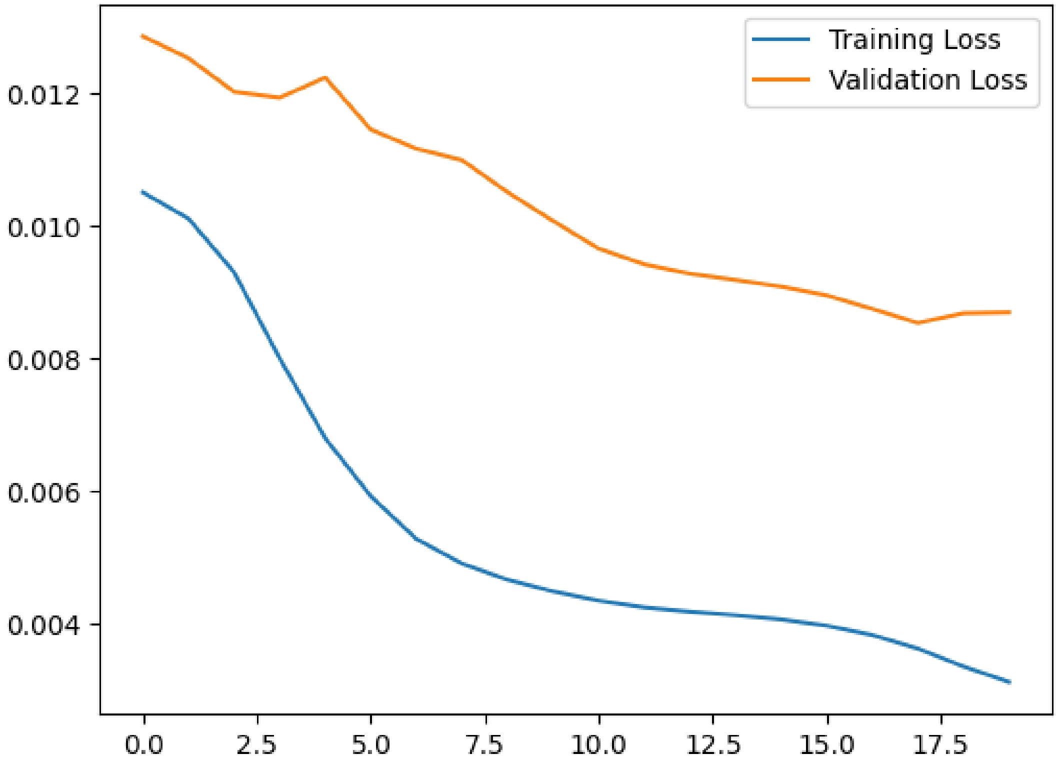
```
    x_train_scaled,  # Target data for training (autoencoder reconstructs the␣
 ↪input)
    epochs=20,         # Number of training epochs
    batch_size=512,    # Batch size
    validation_data=(x_test_scaled, x_test_scaled),  # Validation data
    shuffle=True      # Shuffle the data during training
)
```

```
Epoch 1/20
5/5               5s 171ms/step - loss:
0.0106 - mse: 0.0238 - val_loss: 0.0128 - val_mse: 0.0300
Epoch 2/20
5/5               0s 30ms/step - loss:
0.0102 - mse: 0.0229 - val_loss: 0.0125 - val_mse: 0.0292
Epoch 3/20
5/5               0s 45ms/step - loss:
0.0094 - mse: 0.0211 - val_loss: 0.0120 - val_mse: 0.0281
Epoch 4/20
5/5               0s 61ms/step - loss:
0.0081 - mse: 0.0181 - val_loss: 0.0119 - val_mse: 0.0278
Epoch 5/20
5/5               0s 43ms/step - loss:
0.0070 - mse: 0.0157 - val_loss: 0.0122 - val_mse: 0.0283
Epoch 6/20
5/5               0s 30ms/step - loss:
0.0060 - mse: 0.0134 - val_loss: 0.0114 - val_mse: 0.0266
Epoch 7/20
5/5               0s 47ms/step - loss:
0.0053 - mse: 0.0118 - val_loss: 0.0112 - val_mse: 0.0259
Epoch 8/20
5/5               0s 40ms/step - loss:
0.0050 - mse: 0.0112 - val_loss: 0.0110 - val_mse: 0.0255
Epoch 9/20
5/5               0s 40ms/step - loss:
0.0047 - mse: 0.0105 - val_loss: 0.0105 - val_mse: 0.0245
Epoch 10/20
5/5               0s 53ms/step - loss:
0.0044 - mse: 0.0099 - val_loss: 0.0101 - val_mse: 0.0235
Epoch 11/20
5/5               1s 42ms/step - loss:
0.0043 - mse: 0.0096 - val_loss: 0.0097 - val_mse: 0.0226
Epoch 12/20
5/5               0s 50ms/step - loss:
0.0044 - mse: 0.0097 - val_loss: 0.0094 - val_mse: 0.0221
Epoch 13/20
5/5               0s 11ms/step - loss:
0.0042 - mse: 0.0093 - val_loss: 0.0093 - val_mse: 0.0218
```

```
Epoch 14/20
5/5              0s 14ms/step - loss:
0.0041 - mse: 0.0092 - val_loss: 0.0092 - val_mse: 0.0216
Epoch 15/20
5/5              0s 11ms/step - loss:
0.0041 - mse: 0.0091 - val_loss: 0.0091 - val_mse: 0.0214
Epoch 16/20
5/5              0s 11ms/step - loss:
0.0039 - mse: 0.0088 - val_loss: 0.0089 - val_mse: 0.0211
Epoch 17/20
5/5              0s 11ms/step - loss:
0.0038 - mse: 0.0085 - val_loss: 0.0087 - val_mse: 0.0206
Epoch 18/20
5/5              0s 11ms/step - loss:
0.0037 - mse: 0.0084 - val_loss: 0.0085 - val_mse: 0.0201
Epoch 19/20
5/5              0s 15ms/step - loss:
0.0034 - mse: 0.0075 - val_loss: 0.0087 - val_mse: 0.0203
Epoch 20/20
5/5              0s 11ms/step - loss:
0.0031 - mse: 0.0071 - val_loss: 0.0087 - val_mse: 0.0202
```

[ ]:
```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7bb15e653100>
```



[ ]:
```
# Function to find the threshold for anomalies based on the training data
def find_threshold(model, x_train_scaled):
    # Reconstruct the data using the model
    recons = model.predict(x_train_scaled)

    # Calculate the mean squared log error between reconstructed data and the␣
 ↪original data
    recons_error = tf.keras.metrics.msle(recons, x_train_scaled)

    # Set the threshold as the mean error plus one standard deviation
    threshold = np.mean(recons_error.numpy()) + np.std(recons_error.numpy())

    return threshold

# Function to make predictions for anomalies based on the threshold
def get_predictions(model, x_test_scaled, threshold):
    # Reconstruct the data using the model
    predictions = model.predict(x_test_scaled)

    # Calculate the mean squared log error between reconstructed data and the␣
 ↪original data
    errors = tf.keras.losses.msle(predictions, x_test_scaled)
```

```
    # Create a mask for anomalies based on the threshold
    anomaly_mask = pd.Series(errors) > threshold

    # Map True (anomalies) to 0 and False (normal data) to 1
    preds = anomaly_mask.map(lambda x: 0.0 if x == True else 1.0)

    return preds

# Find the threshold for anomalies
threshold = find_threshold(model, x_train_scaled)
print(f"Threshold: {threshold}")
```

```
73/73              1s 6ms/step
Threshold: 0.006778224756144374
```

[ ]:
```
# Get predictions for anomalies based on the model and threshold
predictions = get_predictions(model, x_test_scaled, threshold)

# Calculate the accuracy score by comparing the predicted anomalies to the true␣
↪labels
accuracy = accuracy_score(predictions, y_test)

# Print the accuracy score
print(f"Accuracy Score: {accuracy}")
```

```
32/32              0s 4ms/step
Accuracy Score: 0.955
```

[ ]:
```

```