```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
```

```python
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Define EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,  # Number of epochs with no improvement after which tra
                               restore_best_weights=True)  # Restores model to best weights with t

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=100,  # Set a high number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[early_stopping])  # Add the early stopping callback
```

```
Epoch 38/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1394 - val_loss: 0.1380
Epoch 39/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1394 - val_loss: 0.1378
Epoch 40/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1391 - val_loss: 0.1380
Epoch 41/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1392 - val_loss: 0.1377
Epoch 42/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1393 - val_loss: 0.1377
Epoch 43/100
```

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import TerminateOnNaN

# Define the TerminateOnNaN callback
terminate_on_nan = TerminateOnNaN()

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)
```

```python
# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Set the number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[terminate_on_nan])  # Add the TerminateOnNaN callback
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.4361 - val_loss: 0.2379
Epoch 2/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 0.2309 - val_loss: 0.2058
Epoch 3/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.2039 - val_loss: 0.1912
Epoch 4/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.1907 - val_loss: 0.1818
Epoch 5/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.1805 - val_loss: 0.1697
Epoch 6/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1688 - val_loss: 0.1615
Epoch 7/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1618 - val_loss: 0.1564
Epoch 8/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.1564 - val_loss: 0.1521
Epoch 9/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.1528 - val_loss: 0.1505
Epoch 10/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 0.1510 - val_loss: 0.1491
Epoch 11/30
```

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the ModelCheckpoint callback
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras',  # File path to save the model
                             monitor='val_loss',  # Metric to monitor
                             save_best_only=True,  # Save only the best model (based on the monito
                             mode='min',  # Minimize the monitored metric (e.g., validation loss)
                             save_weights_only=False,  # Save the entire model (set to True to sav
                             verbose=1)  # Print a message when saving the model


# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
```

```python
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[checkpoint])  # Add the ModelCheckpoint callback
```

```
Epoch 16: val_loss improved from 0.14921 to 0.14856, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1509 - val_loss: 0.1486
Epoch 17/30
229/235 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1497
Epoch 17: val_loss improved from 0.14856 to 0.14777, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1497 - val_loss: 0.1478
Epoch 18/30
232/235 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1487
Epoch 18: val_loss improved from 0.14777 to 0.14735, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1487 - val_loss: 0.1473
Epoch 19/30
231/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1488
Epoch 19: val_loss improved from 0.14735 to 0.14669, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.1488 - val_loss: 0.1467
Epoch 20/30
```

```
Epoch 23: val_loss improved from 0.14538 to 0.14441, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1463 - val_loss: 0.1444
Epoch 24/30
230/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1455
Epoch 24: val_loss improved from 0.14441 to 0.14363, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.1455 - val_loss: 0.1436
Epoch 25/30
232/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1448
Epoch 25: val_loss improved from 0.14363 to 0.14285, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1448 - val_loss: 0.1429
Epoch 26/30
228/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1439
Epoch 26: val_loss improved from 0.14285 to 0.14240, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1439 - val_loss: 0.1424
Epoch 27/30
229/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1436
Epoch 27: val_loss improved from 0.14240 to 0.14187, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1436 - val_loss: 0.1419
Epoch 28/30
232/235 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1432
Epoch 28: val_loss improved from 0.14187 to 0.14148, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1432 - val_loss: 0.1415
Epoch 29/30
229/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1429
Epoch 29: val_loss improved from 0.14148 to 0.14118, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1429 - val_loss: 0.1412
Epoch 30/30
234/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1425
Epoch 30: val_loss improved from 0.14118 to 0.14081, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1425 - val_loss: 0.1408
<keras.src.callbacks.history.History at 0x7cbadd270c40>
```

```
[ ] import numpy as np
    from tensorflow.keras.layers import Input, Dense
    from tensorflow.keras.models import Model
    from tensorflow.keras.datasets import mnist
    from tensorflow.keras.callbacks import ReduceLROnPlateau

    # Define the ReduceLROnPlateau callback
    reduce_lr = ReduceLROnPlateau(monitor='val_loss',  # Metric to monitor
```

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import ReduceLROnPlateau

# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss',  # Metric to monitor
                              factor=0.5,  # Factor by which the learning rate will be reduced (ne
                              patience=3,  # Number of epochs with no improvement after which lear
                              min_lr=1e-6,  # Lower bound for the learning rate
                              verbose=1)  # Print message when the learning rate is reduced


# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)
```

```python
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[reduce_lr])  # Add the ReduceLROnPlateau callback
```

```
Epoch 2/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.2272 - val_loss: 0.1941 - learning_rate: 0.0010
Epoch 3/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1900 - val_loss: 0.1770 - learning_rate: 0.0010
Epoch 4/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.1763 - val_loss: 0.1686 - learning_rate: 0.001
Epoch 5/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - loss: 0.1678 - val_loss: 0.1605 - learning_rate: 0.0010
Epoch 6/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1609 - val_loss: 0.1562 - learning_rate: 0.0010
Epoch 7/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1570 - val_loss: 0.1531 - learning_rate: 0.0010
```

```
<keras.src.callbacks.history.History at 0x7cbade30a530>
```

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TerminateOnNaN, ReduceLROnP

# EarlyStopping callback to stop training if validation loss stops improving
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# ModelCheckpoint callback to save the best model based on validation loss
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras', monitor='val_loss', save_best_only

# TerminateOnNaN callback to stop training if the loss becomes NaN
terminate_on_nan = TerminateOnNaN()

# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,   patience=3, min_lr=1e-6, verbose=1

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features
```

```python
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Training with multiple callbacks
autoencoder.fit(x_train, x_train,
                epochs=30,  # You can set a high number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[reduce_lr, early_stopping, checkpoint, terminate_on_nan])  # Using mult
```

```
Epoch 1/30
231/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.4351
Epoch 1: val_loss improved from inf to 0.24041, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.4326 - val_loss: 0.2404 - learning_rate: 0.001
Epoch 2/30
230/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.2276
Epoch 2: val_loss improved from 0.24041 to 0.19593, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.2273 - val_loss: 0.1959 - learning_rate: 0.0010
Epoch 3/30
229/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1929
Epoch 3: val_loss improved from 0.19593 to 0.17989, saving model to autoencoder_best.keras
```

```
233/235 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1372
Epoch 29: val_loss improved from 0.13590 to 0.13543, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1372 - val_loss: 0.1354 - learning_rate: 0.001
Epoch 30/30
233/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1370
Epoch 30: val_loss improved from 0.13543 to 0.13522, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.1370 - val_loss: 0.1352 - learning_rate: 0.0010
<keras.src.callbacks.history.History at 0x7cbade1dcd00>
```

```python
from tensorflow.keras.models import load_model

# Load the entire model
best_autoencoder = load_model('autoencoder_best.keras')

# Let's look at the encoded representations
encoded_data = best_autoencoder.predict(x_test)
print(encoded_data)
print(encoded_data.shape)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
[[1.9807577e-15 1.5152339e-14 3.9891061e-13 ... 7.5100016e-15
  6.1764503e-15 2.0991466e-16]
 [1.0935202e-10 4.1839920e-10 2.3050595e-10 ... 1.8456821e-10
  2.4213373e-10 2.5670152e-10]
 [1.9581055e-16 1.8931988e-14 5.0116181e-16 ... 3.8398740e-16
  8.2408886e-15 2.9677345e-15]
 ...
 [9.4068260e-16 6.2475350e-16 3.9129981e-14 ... 9.5997547e-15
  1.2012526e-15 2.2314454e-17]
 [2.7297345e-10 8.1712576e-10 3.1884095e-09 ... 1.1436131e-09
  4.0451742e-10 3.5425576e-11]
 [6.7035170e-14 3.0569222e-15 1.2275606e-14 ... 1.4145321e-13
  1.8933355e-15 2.8076197e-15]]
(10000, 784)
```

my GitHub link:https://github.com/nithin1086/BDA