Ian Pope 700717 Big Data Analytics ICP 6

Ian Pope 700717419 Big Data Analytics ICP 6

Load data and set up base autoencoder

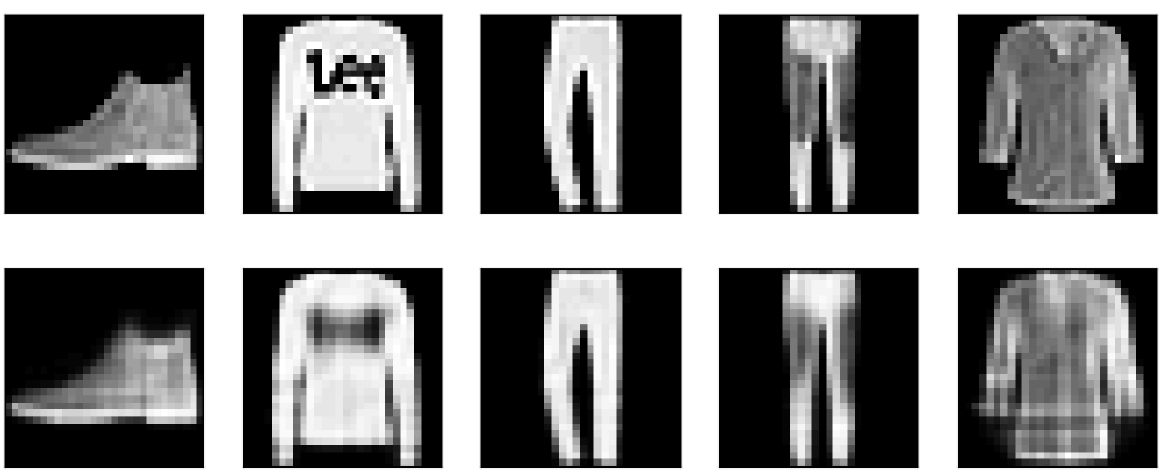
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.models import Model

# Load Fashion MNIST data
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize data to values between 0 and 1
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the 28x28 images into vectors of size 784
x_train = x_train.reshape((x_train.shape[0], 28 * 28))
x_test = x_test.reshape((x_test.shape[0], 28 * 28))

# Define the dimensions
input_dim = 28 * 28  # 784 input neurons
encoding_dim = 64    # Compress to 64 features

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

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)

# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Create the autoencoder model
autoencoder = Model(input_layer, decoded)

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

Fit code using early stopping callback and run

-Stopped at epoch 91

```
from tensorflow.keras.callbacks import EarlyStopping

# Define EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,  # Number of epochs with no improvement after which training will be stopped
                               restore_best_weights=True)  # Restores model to best weights with the lowest validation loss

# Assuming x_train and x_test are your training and test datasets
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


#Base code below


# Define the encoder model to get the compressed representation
encoder = Model(input_layer, encoded)

# Encode and decode some images from the test set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

# Visualizing original and reconstructed images
n = 10  # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstructed images
```

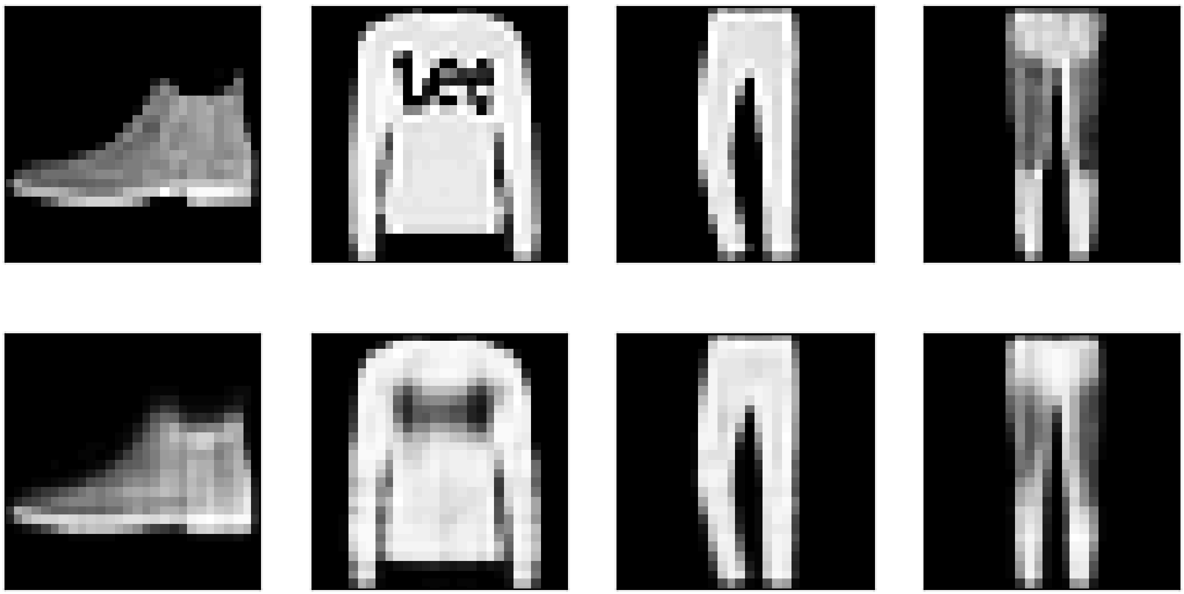
```
Epoch 90/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2687 - val_loss: 0.2718
Epoch 91/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2693 - val_loss: 0.2718
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```



TerminateOnNaN

```
[ ]  from tensorflow.keras.callbacks import TerminateOnNaN

     # Define the TerminateOnNaN callback
     terminate_on_nan = TerminateOnNaN()

     # 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=100,  # Set the number of epochs
                     batch_size=256,
                     shuffle=True,
                     validation_data=(x_test, x_test),
                     callbacks=[terminate_on_nan])  # Add the TerminateOnNaN callback


     #Base Code Below

     # Define the encoder model to get the compressed representation
     encoder = Model(input_layer, encoded)

     # Encode and decode some images from the test set
     encoded_imgs = encoder.predict(x_test)
     decoded_imgs = autoencoder.predict(x_test)

     # Visualizing original and reconstructed images
     n = 10  # Number of images to display
     plt.figure(figsize=(20, 4))
     for i in range(n):
         # Display original images
         ax = plt.subplot(2, n, i + 1)
         plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)

         # Display reconstructed images
         ax = plt.subplot(2, n, i + 1 + n)
         plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)
     plt.show()
```

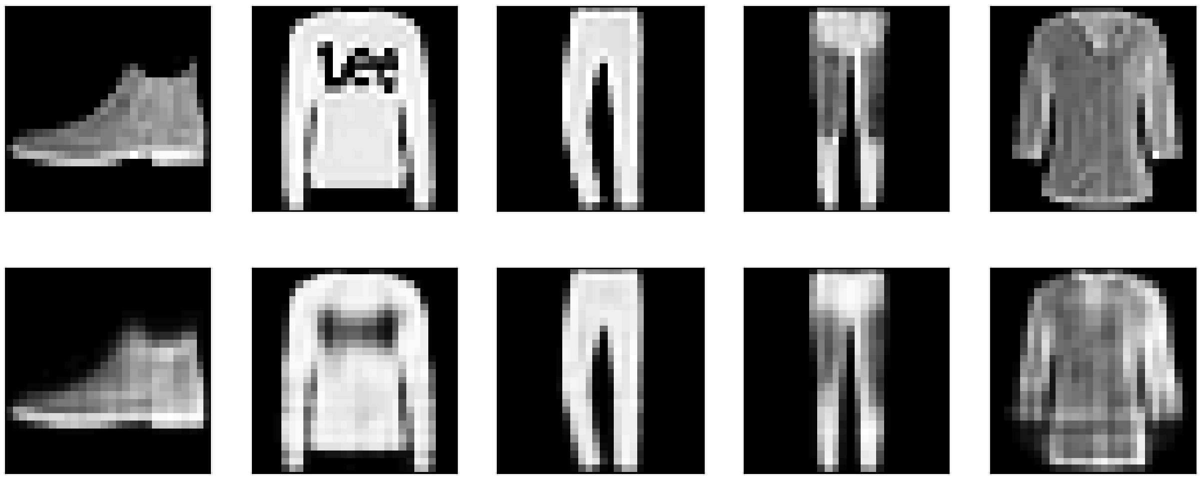
```
Epoch 98/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2686 - val_loss: 0.2717
Epoch 99/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2694 - val_loss: 0.2716
Epoch 100/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2690 - val_loss: 0.2716
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



ModelCheckpoint

```
[ ] 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 monitored metric)
                                 mode='min',  # Minimize the monitored metric (e.g., validation loss)
                                 save_weights_only=False,  # Save the entire model (set to True to save only weights)
                                 verbose=1)  # Print a message when saving the model


    # 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=50,  # Number of epochs
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, x_test),  # Validation data
                    callbacks=[checkpoint])  # Add the ModelCheckpoint callback


    #Base code below


    # Define the encoder model to get the compressed representation
    encoder = Model(input_layer, encoded)

    # Encode and decode some images from the test set
    encoded_imgs = encoder.predict(x_test)
    decoded_imgs = autoencoder.predict(x_test)

    # Visualizing original and reconstructed images
    n = 10  # Number of images to display
    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Display original images
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
```

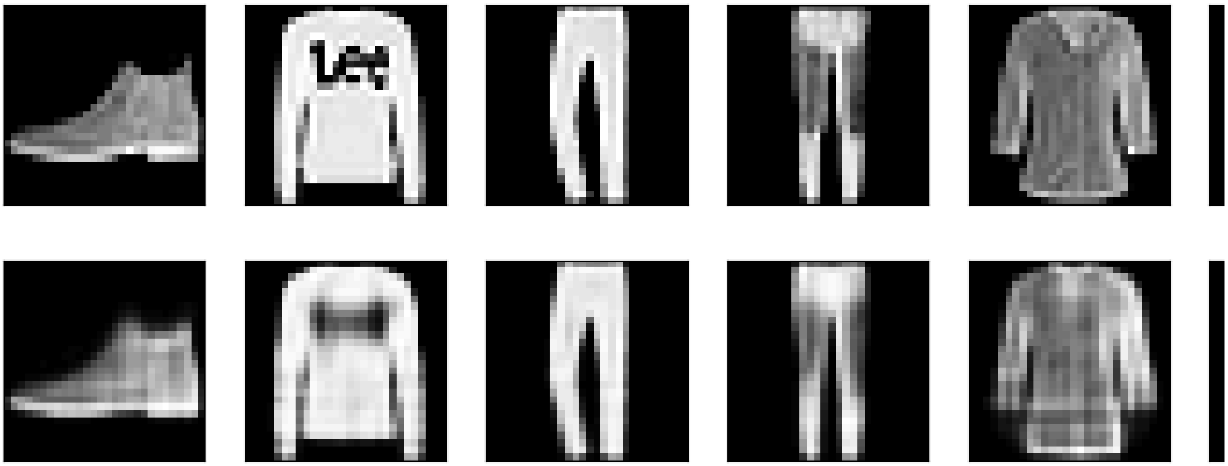
```
219/235 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2681
Epoch 49: val_loss did not improve from 0.27145
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2681 - val_loss: 0.2715
Epoch 50/50
227/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2681
Epoch 50: val_loss improved from 0.27145 to 0.27145, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2681 - val_loss: 0.2714
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



ReduceLROnPlateau

```
[ ]  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 (new_lr = lr * factor)
                                   patience=3,  # Number of epochs with no improvement after which learning rate will be reduced
                                   min_lr=1e-6,  # Lower bound for the learning rate
                                   verbose=1)  # Print message when the learning rate is reduced

     # 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=50,  # Number of epochs
                     batch_size=256,
                     shuffle=True,
                     validation_data=(x_test, x_test),  # Validation data
                     callbacks=[reduce_lr])  # Add the ReduceLROnPlateau callback


     #Base code below


     # Define the encoder model to get the compressed representation
     encoder = Model(input_layer, encoded)

     # Encode and decode some images from the test set
     encoded_imgs = encoder.predict(x_test)
     decoded_imgs = autoencoder.predict(x_test)

     # Visualizing original and reconstructed images
     n = 10  # Number of images to display
     plt.figure(figsize=(20, 4))
     for i in range(n):
         # Display original images
         ax = plt.subplot(2, n, i + 1)
         plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)

         # Display reconstructed images
```

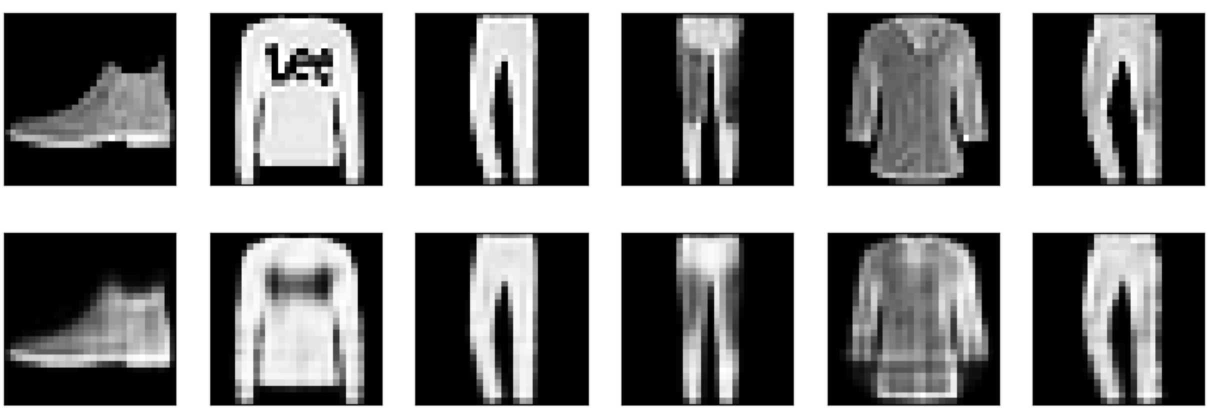
```
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2681 - val_loss: 0.2712 - learning_rate: 1.0000e-06
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2692 - val_loss: 0.2712 - learning_rate: 1.0000e-06
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2688 - val_loss: 0.2712 - learning_rate: 1.0000e-06
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2680 - val_loss: 0.2712 - learning_rate: 1.0000e-06
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



All four callbacks

```
[ ] from tensorflow.keras.callbacks import EarlyStopping
    from tensorflow.keras.callbacks import TerminateOnNaN
    from tensorflow.keras.callbacks import ModelCheckpoint
    from tensorflow.keras.callbacks import ReduceLROnPlateau

    # Define EarlyStopping
    early_stopping = EarlyStopping(monitor='val_loss',
                                   patience=10,  # Number of epochs with no improvement after which training will be stopped
                                   restore_best_weights=True)  # Restores model to best weights with the lowest validation loss

    # Define the TerminateOnNaN callback
    terminate_on_nan = TerminateOnNaN()

    # 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 monitored metric)
                                 mode='min',  # Minimize the monitored metric (e.g., validation loss)
                                 save_weights_only=False,  # Save the entire model (set to True to save only weights)
                                 verbose=1)  # Print a message when saving the model

    # 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 (new_lr = lr * factor)
                                  patience=3,  # Number of epochs with no improvement after which learning rate will be reduced
                                  min_lr=1e-6,  # Lower bound for the learning rate
                                  verbose=1)  # Print message when the learning rate is reduced


    #Put all callbacks in
    autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                    epochs=100,  # Number of epochs
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, x_test),  # Validation data
                    callbacks=[reduce_lr, checkpoint, terminate_on_nan, early_stopping])  # Add the ReduceLROnPlateau callback


    #Base code below
```

```
Epoch 29: val_loss did not improve from 0.27118
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2689 - val_loss: 0.2712 - learning_rate: 1.0000e-06
Epoch 30/100
217/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2688
Epoch 30: val_loss did not improve from 0.27118
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2688 - val_loss: 0.2712 - learning_rate: 1.0000e-06
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



Reload data from autoencoder_best.keras and use on testing data.

I predict the results will be the same from the loaded model as the previous.

```
[ ] from tensorflow.keras.models import load_model

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

    # Define the encoder model to get the compressed representation
    encoder = Model(input_layer, encoded)

    # Encode and decode some images from the test set
    encoded_imgs = encoder.predict(x_test)
    decoded_imgs = best_autoencoder.predict(x_test)

    # Visualizing original and reconstructed images
    n = 10  # Number of images to display
    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Display original images
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        # Display reconstructed images
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

```
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```