```
import matplotlib.pyplot as plt              # for plotting and displaying
import numpy as np                           # for numerical computations
import pandas as pd                          # imported for data handling (
import tensorflow as tf                      # TensorFlow for deep learning

from sklearn.metrics import accuracy_score, precision_score, recall_score  #
from sklearn.model_selection import train_test_split                       #
from tensorflow.keras import layers, losses                                #
from tensorflow.keras.datasets import fashion_mnist                        #
from tensorflow.keras.models import Model
```

```
# === Cell 2: Load and preprocess Fashion MNIST dataset ===
(x_train, _), (x_test, _) = fashion_mnist.load_data()  # load images (ignore
# x_train: shape (60000, 28, 28); x_test: shape (10000, 28, 28)
# normalize pixel values to [0,1] for better training stability
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# print dataset shapes to verify loading
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

```
# === Cell 3: Define Autoencoder architecture ===
latent_dim = 64  # size of the encoded representation (compressed feature di

# Create custom Autoencoder model class
class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()    # initialize base Model
    self.latent_dim = latent_dim           # store latent dimension size

    # define encoder part: flattens image and compresses into latent_dim fea
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),                    # convert 28x28 into 784 vector
      layers.Dense(latent_dim, activation='relu'),  # encode into smaller fe
    ])

    # define decoder part: reconstructs image from latent vector
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),       # map latent vector bac
      layers.Reshape((28, 28))                       # reshape back to 28x28
    ])

  # define forward pass (called automatically during training/inference)
  def call(self, x):
    encoded = self.encoder(x)        # compress input
    decoded = self.decoder(encoded)  # reconstruct image
    return decoded
```

```
# instantiate autoencoder model
autoencoder = Autoencoder(latent_dim)
```

```
# === Cell 4: Compile and train model ===
# compile model with Adam optimizer and mean squared error loss
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

# train model for 10 epochs, using training data as both input and output (a
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))  # validation on test data
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - loss: 0.0395 - val_loss: 0.0133
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - loss: 0.0123 - val_loss: 0.0107
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 0.0104 - val_loss: 0.0099
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0096 - val_loss: 0.0095
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0093 - val_loss: 0.0093
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - loss: 0.0092 - val_loss: 0.0091
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0091 - val_loss: 0.0091
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0089 - val_loss: 0.0092
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 0.0089 - val_loss: 0.0091
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0089 - val_loss: 0.0090
<keras.src.callbacks.history.History at 0x7ef3148240e0>
```

```
# === Cell 5: Encode and decode images ===
# encode test images into latent space
encoded_imgs = autoencoder.encoder(x_test).numpy()

# decode latent representations back into images
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

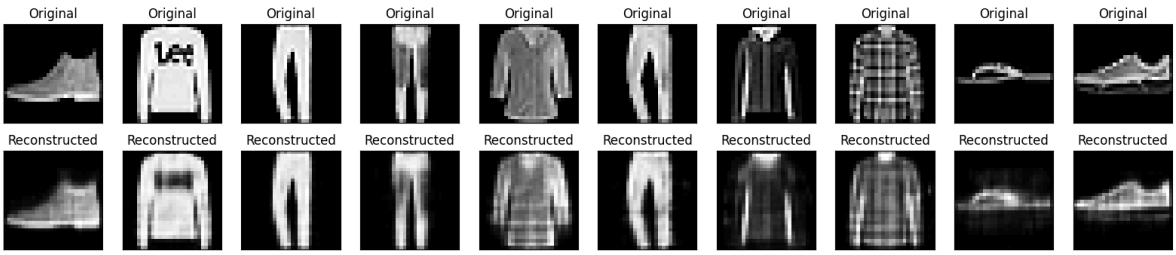
```
# === Cell 6: Display original vs reconstructed images ===
n = 10                                  # number of images to display
plt.figure(figsize=(20, 4))             # define figure size

for i in range(n):
  # display original image
  ax = plt.subplot(2, n, i + 1)         # subplot in first row
  plt.imshow(x_test[i])                 # show original test image
  plt.title("Original")                 # title for clarity
  plt.gray()                            # display in grayscale
```

```
    ax.get_xaxis().set_visible(False)      # hide x-axis
    ax.get_yaxis().set_visible(False)      # hide y-axis

    # display reconstructed image
    ax = plt.subplot(2, n, i + 1 + n)      # subplot in second row
    plt.imshow(decoded_imgs[i])            # show reconstructed image
    plt.title("Reconstructed")             # title for clarity
    plt.gray()                             # grayscale display
    ax.get_xaxis().set_visible(False)      # hide x-axis
    ax.get_yaxis().set_visible(False)      # hide y-axis

plt.show()                                 # show the full plot
```



Start coding or generate with AI.