DATASET LINK:https://www.kaggle.com/datasets/andhikawb/fashion-mnist-png

1. Undercomplete Autoencoder

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np

# -----------------------------
# 1. Load Dataset from Folder
# -----------------------------
def load_user_dataset(data_dir, img_size=(28, 28), batch_size=32,
color_mode="grayscale"):
    dataset = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        labels=None,              # Autoencoder is unsupervised
        color_mode=color_mode,    # "rgb" or "grayscale"
        image_size=img_size,
        batch_size=batch_size,
        shuffle=True
    )
    dataset = dataset.map(lambda x: x / 255.0)  # normalize [0,1]
    return dataset

# ----------------------------------
# 2. Build Undercomplete Autoencoder
# ----------------------------------
def build_undercomplete_autoencoder(input_shape=(28, 28, 1),
latent_dim=16):
    inp = layers.Input(shape=input_shape)

    # Encoder
    x = layers.Conv2D(32, 3, strides=2, padding="same",
activation="relu")(inp)  # 14x14
    x = layers.Conv2D(64, 3, strides=2, padding="same",
activation="relu")(x)    # 7x7
    x = layers.Conv2D(latent_dim, 3, strides=2, padding="same",
activation="relu")(x)  # 4x4

    encoded = x  # bottleneck

    # Decoder
    x = layers.Conv2DTranspose(64, 3, strides=2, padding="same",
activation="relu")(encoded)   # 7x7
    x = layers.Conv2DTranspose(32, 3, strides=2, padding="same",
activation="relu")(x)         # 14x14
```

```
    x = layers.Conv2DTranspose(input_shape[2], 3, strides=2,
padding="same", activation="sigmoid")(x)  # 28→32

    # Crop to original size (28x28)
    x = layers.Cropping2D(((2, 2), (2, 2)))(x)

    autoencoder = models.Model(inp, x,
name="undercomplete_autoencoder")
    return autoencoder

# ----------------------------
# 3. Train Autoencoder
# ----------------------------
def train_autoencoder(data_dir, img_size=(28, 28),
color_mode="grayscale", epochs=10, batch_size=32):
    dataset = load_user_dataset(data_dir, img_size, batch_size,
color_mode)

    # Convert dataset to numpy arrays for training/validation split
    x_all = tf.concat(list(dataset), axis=0).numpy()
    x_train, x_test = x_all[:int(0.8*len(x_all))],
x_all[int(0.8*len(x_all)):]

    input_shape = x_train.shape[1:]  # (28,28,1) or (28,28,3)
    autoencoder = build_undercomplete_autoencoder(input_shape)

    autoencoder.compile(optimizer="adam", loss="mse")
    history = autoencoder.fit(x_train, x_train,
                              epochs=epochs,
                              batch_size=batch_size,
                              shuffle=True,
                              validation_data=(x_test, x_test))
    return autoencoder, x_test

# ----------------------------
# 4. Display Reconstructions
# ----------------------------
def show_reconstructions(autoencoder, x_test, n=5):
    decoded_imgs = autoencoder.predict(x_test[:n])

    plt.figure(figsize=(10, 4))
    for i in range(n):
        # Original
        ax = plt.subplot(2, n, i+1)
        plt.imshow(x_test[i].squeeze(), cmap="gray")
        plt.title("Original")
        plt.axis("off")
```

```
        # Reconstructed
        ax = plt.subplot(2, n, i+1+n)
        plt.imshow(decoded_imgs[i].squeeze(), cmap="gray")
        plt.title("Reconstructed")
        plt.axis("off")
    plt.show()

# ----------------------------
# 5. Run Example
# ----------------------------
if __name__ == "__main__":
    # ✅ Change this to your dataset path
    data_dir = "/content/drive/MyDrive/0"

    autoencoder, x_test = train_autoencoder(
        data_dir,
        img_size=(28,28),      # Resize all images to 28x28
        color_mode="grayscale", # or "rgb" if colored dataset
        epochs=10,
        batch_size=32
    )

    show_reconstructions(autoencoder, x_test, n=5)


2. #Overcomplete Autoencoders

import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np

# ----------------------------
# 1. Load Dataset from Folder
# ----------------------------
def load_user_dataset(data_dir, img_size=(28, 28), batch_size=32,
color_mode="grayscale"):
    dataset = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        labels=None,
        color_mode=color_mode,    # "grayscale" or "rgb"
        image_size=img_size,
        batch_size=batch_size,
        shuffle=True
    )
    dataset = dataset.map(lambda x: x / 255.0)  # normalize
    return dataset
```

```
# ----------------------------------
# 2. Build Overcomplete Autoencoder
# ----------------------------------
def build_overcomplete_autoencoder(input_shape=(28, 28, 1),
latent_dim=256):
    inp = layers.Input(shape=input_shape)

    # Flatten input → dense latent space with *larger dimension*
(overcomplete)
    x = layers.Flatten()(inp)
    encoded = layers.Dense(latent_dim, activation="relu")(x)    # latent
> input size

    # Decoder
    x = layers.Dense(np.prod(input_shape),
activation="sigmoid")(encoded)
    decoded = layers.Reshape(input_shape)(x)

    autoencoder = models.Model(inp, decoded,
name="overcomplete_autoencoder")
    return autoencoder

# ----------------------------
# 3. Train Autoencoder
# ----------------------------
def train_autoencoder(data_dir, img_size=(28, 28),
color_mode="grayscale", epochs=10, batch_size=32):
    dataset = load_user_dataset(data_dir, img_size, batch_size,
color_mode)

    # Convert dataset to numpy arrays for training/validation split
    x_all = tf.concat(list(dataset), axis=0).numpy()
    x_train, x_test = x_all[:int(0.8*len(x_all))],
x_all[int(0.8*len(x_all)):]

    input_shape = x_train.shape[1:]
    autoencoder = build_overcomplete_autoencoder(input_shape,
latent_dim=256)

    autoencoder.compile(optimizer="adam", loss="mse")
    history = autoencoder.fit(x_train, x_train,
                              epochs=epochs,
                              batch_size=batch_size,
                              shuffle=True,
                              validation_data=(x_test, x_test))
    return autoencoder, x_test

# ----------------------------
```

```python
# 4. Display Reconstructions
# ----------------------------
def show_reconstructions(autoencoder, x_test, n=5):
    decoded_imgs = autoencoder.predict(x_test[:n])

    plt.figure(figsize=(10, 4))
    for i in range(n):
        # Original
        ax = plt.subplot(2, n, i+1)
        plt.imshow(x_test[i].squeeze(), cmap="gray")
        plt.title("Original")
        plt.axis("off")

        # Reconstructed
        ax = plt.subplot(2, n, i+1+n)
        plt.imshow(decoded_imgs[i].squeeze(), cmap="gray")
        plt.title("Reconstructed")
        plt.axis("off")
    plt.show()

# ----------------------------
# 5. Run Example
# ----------------------------
if __name__ == "__main__":
    # ✅ Change this to your dataset path
    data_dir = "/content/drive/MyDrive/0"

    autoencoder, x_test = train_autoencoder(
        data_dir,
        img_size=(28,28),      # Resize all images to 28x28
        color_mode="grayscale", # or "rgb"
        epochs=10,
        batch_size=32
    )

    show_reconstructions(autoencoder, x_test, n=5)
```