import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.layers import Input, Dense, Flatten, Reshape

from tensorflow.keras.models import Model

from tensorflow.keras.datasets import mnist

# Load the MNIST dataset

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

# Normalize images to the range [0, 1]

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

# Reshape images to (28, 28, 1) for consistency with Conv2D inputs

x\_train = np.reshape(x\_train, (x\_train.shape[0], 28, 28, 1))

x\_test = np.reshape(x\_test, (x\_test.shape[0], 28, 28, 1))

def add\_noise(images, noise\_factor=0.5):

noisy\_images = images + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=images.shape)

noisy\_images = np.clip(noisy\_images, 0., 1.) # Clip values to maintain the range [0, 1]

return noisy\_images

# Add noise to the training and test data

x\_train\_noisy = add\_noise(x\_train)

x\_test\_noisy = add\_noise(x\_test)

# Encoder

input\_img = Input(shape=(28, 28, 1)) # Input shape (28x28x1)

x = Flatten()(input\_img)

x = Dense(128, activation='relu')(x)

x = Dense(64, activation='relu')(x)

encoded = Dense(32, activation='relu')(x) # Latent space (32-dimensional)

# Decoder

x = Dense(64, activation='relu')(encoded)

x = Dense(128, activation='relu')(x)

x = Dense(28 \* 28, activation='sigmoid')(x) # Reconstruct image

decoded = Reshape((28, 28, 1))(x) # Reshape to (28, 28, 1)

# Denoising Autoencoder Model

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

autoencoder.fit(x\_train\_noisy, x\_train, epochs=10, batch\_size=256, validation\_data=(x\_test\_noisy, x\_test))

# Predict on test data

decoded\_imgs = autoencoder.predict(x\_test\_noisy)

# Plot some noisy and reconstructed images

n = 10 # Number of images to display

plt.figure(figsize=(20, 4))

for i in range(n):

# Noisy image

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test\_noisy[i].reshape(28, 28), cmap="gray")

ax.axis('off')

# Reconstructed image

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_imgs[i].reshape(28, 28), cmap="gray")

ax.axis('off')

plt.show()

# Basic Autoencoder Model (without noise)

input\_img = Input(shape=(28, 28, 1))

x = Flatten()(input\_img)

x = Dense(128, activation='relu')(x)

x = Dense(64, activation='relu')(x)

encoded = Dense(32, activation='relu')(x)

x = Dense(64, activation='relu')(encoded)

x = Dense(128, activation='relu')(x)

x = Dense(28 \* 28, activation='sigmoid')(x)

decoded = Reshape((28, 28, 1))(x)

basic\_autoencoder = Model(input\_img, decoded)

basic\_autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the basic autoencoder

basic\_autoencoder.fit(x\_train, x\_train, epochs=10, batch\_size=256, validation\_data=(x\_test, x\_test))

# Predict on test data

decoded\_imgs\_basic = basic\_autoencoder.predict(x\_test)

# Plot comparison between the denoising and basic autoencoders

plt.figure(figsize=(20, 4))

for i in range(n):

# Denoising Autoencoder

ax = plt.subplot(3, n, i + 1)

plt.imshow(decoded\_imgs[i].reshape(28, 28), cmap="gray")

ax.axis('off')

# Basic Autoencoder

ax = plt.subplot(3, n, i + 1 + n)

plt.imshow(decoded\_imgs\_basic[i].reshape(28, 28), cmap="gray")

ax.axis('off')

# Noisy Image

ax = plt.subplot(3, n, i + 1 + 2 \* n)

plt.imshow(x\_test\_noisy[i].reshape(28, 28), cmap="gray")

ax.axis('off')

plt.show()