import keras

from keras import layers

# Size of encoded representations

encoding\_dim = 32

# input image

input\_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input

encoded = layers.Dense(encoding\_dim, activation='relu')(input\_img)

# "decoded" is the lossy reconstruction of the input

decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction

autoencoder = keras.Model(input\_img, decoded)

# This model maps an input to its encoded representation

encoder = keras.Model(input\_img, encoded)

# This is the encoded (32-dimensional) input

encoded\_input = keras.Input(shape=(encoding\_dim,))

# Retrieve the last layer of the autoencoder model

decoder\_layer = autoencoder.layers[-1]

# Create the decoder model

decoder = keras.Model(encoded\_input, decoder\_layer(encoded\_input))

#optimizer

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

#import MNISTf

from keras.datasets import mnist

import numpy as np

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

#normalization

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

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

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

#autoencoder

autoencoder.fit(x\_train, x\_train,

                epochs=10,

                batch\_size=256,

                shuffle=True,

                validation\_data=(x\_test, x\_test))

# Encode and decode some digits

# Note that we take them from the \*test\* set

encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs = decoder.predict(encoded\_imgs)

# Use Matplotlib (don't ask)

import matplotlib.pyplot as plt

n = 10  # How many digits we will display

plt.figure(figsize=(20, 4))

for i in range(n):

    # Display original

    ax = plt.subplot(2, n, i + 1)

    plt.imshow(x\_test[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

    # Display reconstruction

    ax = plt.subplot(2, n, i + 1 + n)

    plt.imshow(decoded\_imgs[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

plt.show()