ASSIGNMENT-4

NIKHIL VEGGALAM

700747677

[NikhilVeggalam76770/NeuralNetworkDeepLearning-Assognment4 (github.com)](https://github.com/NikhilVeggalam76770/NeuralNetworkDeepLearning-Assognment4)

1.

from keras.layers import Input, Dense

from keras.models import Model

# this is the size of our encoded representations

encoding\_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder

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

# "encoded" is the encoded representation of the input

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

# "decoded" is the lossy reconstruction of the input

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

# this model maps an input to its reconstruction

autoencoder = Model(input\_img, decoded)

# this model maps an input to its encoded representation

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

from keras.datasets import mnist, fashion\_mnist

import numpy as np

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

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:])))

autoencoder.fit(x\_train, x\_train,

                epochs=5,

                batch\_size=256,

                shuffle=True,

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

OUTPUT:

A screenshot of a computer program

Description automatically generated

2.

from keras.layers import Input, Dense

from keras.models import Model

# Define input shape

input\_shape = (784,)

# Define encoding dimensions

encoding\_dim1 = 64

encoding\_dim2 = 32

# Define input layer

input\_img = Input(shape=input\_shape)

encoded1 = Dense(encoding\_dim1, activation='relu')(input\_img)

encoded2 = Dense(encoding\_dim2, activation='relu')(encoded1)

decoded1 = Dense(encoding\_dim1, activation='relu')(encoded2)

decoded2 = Dense(input\_shape[0], activation='sigmoid')(decoded1)

autoencoder = Model(input\_img, decoded2)

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

from keras.datasets import mnist, fashion\_mnist

import numpy as np

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

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:])))

# Train model

history = autoencoder.fit(x\_train, x\_train,

                          epochs=20,

                          batch\_size=256,

                          shuffle=True,

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

# Predict on test data

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

# Visualize reconstructed image and original image

import matplotlib.pyplot as plt

# Choose an index of a test image to visualize

idx = 10

# Reshape the test image

test\_img = x\_test[idx].reshape(28, 28)

# Reshape the reconstructed image

reconstructed\_img = decoded\_imgs[idx].reshape(28, 28)

# Plot the original and reconstructed images side by side

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(test\_img, cmap='gray')

plt.title('Original Image')

plt.subplot(1, 2, 2)

plt.imshow(reconstructed\_img, cmap='gray')

plt.title('Reconstructed Image')

plt.show()

OUTPUT:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

3.

from keras.layers import Input, Dense

from keras.models import Model

# this is the size of our encoded representations

encoding\_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder

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

# "encoded" is the encoded representation of the input

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

# "decoded" is the lossy reconstruction of the input

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

# this model maps an input to its reconstruction

autoencoder = Model(input\_img, decoded)

# this model maps an input to its encoded representation

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy', metrics=['accuracy'])

from keras.datasets import fashion\_mnist

import numpy as np

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

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:])))

#introducing noise

noise\_factor = 0.5

x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)

x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)

history=autoencoder.fit(x\_train\_noisy, x\_train,

                epochs=10,

                batch\_size=256,

                shuffle=True,

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

OUTPUT:

A white background with black text

Description automatically generated

import matplotlib.pyplot as plt

# Get the reconstructed images

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

# Select one image to display

img\_to\_display = 0

# Display the original, noisy, and reconstructed images side by side

plt.subplot(1, 3, 1)

plt.imshow(x\_test[img\_to\_display].reshape(28, 28))

plt.title('Original')

plt.subplot(1, 3, 2)

plt.imshow(x\_test\_noisy[img\_to\_display].reshape(28, 28))

plt.title('Noisy')

plt.subplot(1, 3, 3)

plt.imshow(reconstructed\_imgs[img\_to\_display].reshape(28, 28))

plt.title('Reconstructed')

plt.show()

OUTPUT:

A screenshot of a computer generated image

Description automatically generated

4.

# Plot the loss and accuracy over epochs

plt.subplot(2, 1, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.legend()

plt.subplot(2, 1, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.legend()

plt.show()

OUTPUT:

A graph of training and validation

Description automatically generated