Summer 2024: CS5720: Neural Network Deep Learning: In Class Programming Assignment-5

Name: Veda Siva Vasishta Yakkala ID: 700758715

Github link: <https://github.com/VasishtaYakkala/Neural-Network-and-Deep-Learning-icp5/blob/main/Neural_Network_and_Deep_Learning_icp5.ipynb>

1. Add one more hidden layer to autoencoder

Code:

1. from keras.layers import Input, Dense
2. from keras.models import Model
3. # Define the size of our encoded representation
4. encoding\_dim = 32  # 32 floats -> compression factor of 24.5 assuming the input is 784 floats
5. # Input placeholder
6. input\_img = Input(shape=(784,))
7. # Encoded representation
8. encoded = Dense(128, activation='relu')(input\_img)  # Increased size of hidden layer
9. # Additional hidden layer
10. hidden\_layer = Dense(64, activation='relu')(encoded)  # Adding a hidden layer with 64 neurons
11. # Decoded representation
12. decoded = Dense(784, activation='sigmoid')(hidden\_layer)
13. # Autoencoder model
14. autoencoder = Model(input\_img, decoded)
15. # Compile the autoencoder (defaults to adam optimizer with default learning rate)
16. autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')
17. # Loading and preprocessing the Fashion MNIST data
18. from keras.datasets import fashion\_mnist
19. import numpy as np
20. (x\_train, \_), (x\_test, \_) = fashion\_mnist.load\_data()
21. x\_train = x\_train.astype('float32') / 255.
22. x\_test = x\_test.astype('float32') / 255.
23. x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))
24. x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))
25. # Training the autoencoder
26. autoencoder.fit(x\_train, x\_train,
27. epochs=8,  # Increased epochs
28. batch\_size=128,  # Smaller batch size
29. shuffle=True,
30. validation\_data=(x\_test, x\_test))

Output:

Epoch 1/8

469/469 [==============================] - 8s 15ms/step - loss: 0.3476 - val\_loss: 0.3039

Epoch 2/8

469/469 [==============================] - 5s 10ms/step - loss: 0.2948 - val\_loss: 0.2919

Epoch 3/8

469/469 [==============================] - 5s 10ms/step - loss: 0.2863 - val\_loss: 0.2860

Epoch 4/8

469/469 [==============================] - 6s 13ms/step - loss: 0.2819 - val\_loss: 0.2825

Epoch 5/8

469/469 [==============================] - 9s 20ms/step - loss: 0.2792 - val\_loss: 0.2803

Epoch 6/8

469/469 [==============================] - 9s 19ms/step - loss: 0.2772 - val\_loss: 0.2785

Epoch 7/8

469/469 [==============================] - 11s 24ms/step - loss: 0.2757 - val\_loss: 0.2771

Epoch 8/8

469/469 [==============================] - 4s 9ms/step - loss: 0.2744 - val\_loss: 0.2760

<keras.src.callbacks.History at 0x7c47f6a611b0>

2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data.

Also, visualize the same test data before reconstruction using Matplotlib

Code:

import matplotlib.pyplot as plt

# Predict on test data

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

# Function to plot images

def plot\_images(original\_images, decoded\_images, num\_images=10):

    plt.figure(figsize=(20, 4))

    for i in range(num\_images):

        # Original images

        ax = plt.subplot(2, num\_images, i + 1)

        plt.imshow(original\_images[i].reshape(28, 28), cmap='gray')

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

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

        # Reconstructed images

        ax = plt.subplot(2, num\_images, i + 1 + num\_images)

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

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

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

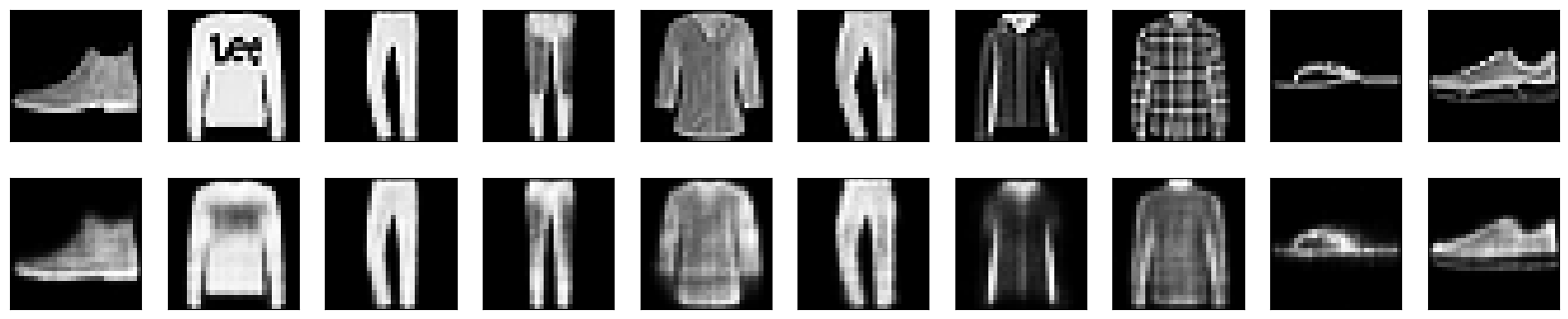
    plt.show()

# Visualize original and reconstructed images

plot\_images(x\_test, decoded\_imgs)

Output:

13/313 [==============================] - 1s 3ms/step



3. Repeat the question 2 on the denoisening autoencoder

Code:

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Input, Dense, GaussianNoise

from keras.models import Model

# Define the size of our encoded representation

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

# Input placeholder with added noise

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

noisy\_img = GaussianNoise(0.5)(input\_img)  # Adding Gaussian noise with stddev=0.5

# Encoded representation

encoded = Dense(128, activation='relu')(noisy\_img)  # Increased size of hidden layer

# Additional hidden layer

hidden\_layer = Dense(64, activation='relu')(encoded)  # Adding a hidden layer with 64 neurons

# Decoded representation

decoded = Dense(784, activation='sigmoid')(hidden\_layer)

# Denoising Autoencoder model

autoencoder = Model(input\_img, decoded)

# Compile the autoencoder (defaults to adam optimizer with default learning rate)

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

# Loading and preprocessing the Fashion MNIST data

from keras.datasets import fashion\_mnist

(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:])))

# Add Gaussian noise to the training and test data

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)

# Clip the values to be between 0 and 1

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

# Training the denoising autoencoder

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

                epochs=25,  # Increased epochs for better training

                batch\_size=100,

                shuffle=True,

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

# Predict on test data

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

# Function to plot images

def plot\_images(original\_images, noisy\_images, decoded\_images, num\_images=10):

    plt.figure(figsize=(20, 6))

    for i in range(num\_images):

        # Original images

        ax = plt.subplot(3, num\_images, i + 1)

        plt.imshow(original\_images[i].reshape(28, 28), cmap='gray')

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

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

        if i == 0:

            ax.set\_title('Original Images')

        # Noisy images

        ax = plt.subplot(3, num\_images, i + 1 + num\_images)

        plt.imshow(noisy\_images[i].reshape(28, 28), cmap='gray')

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

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

        if i == 0:

            ax.set\_title('Noisy Input')

        # Reconstructed images

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

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

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

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

        if i == 0:

            ax.set\_title('Denoised Output')

    plt.tight\_layout()

    plt.show()

# Visualize original, noisy, and denoised images

plot\_images(x\_test, x\_test\_noisy, decoded\_imgs)

Output:

Epoch 1/25

600/600 [==============================] - 8s 12ms/step - loss: 0.3721 - val\_loss: 0.3522

Epoch 2/25

600/600 [==============================] - 9s 15ms/step - loss: 0.3300 - val\_loss: 0.3334

Epoch 3/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3239 - val\_loss: 0.3248

Epoch 4/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3207 - val\_loss: 0.3215

Epoch 5/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3183 - val\_loss: 0.3193

Epoch 6/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3164 - val\_loss: 0.3101

Epoch 7/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3149 - val\_loss: 0.3068

Epoch 8/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3138 - val\_loss: 0.3067

Epoch 9/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3129 - val\_loss: 0.3035

Epoch 10/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3122 - val\_loss: 0.3056

Epoch 11/25

600/600 [==============================] - 12s 20ms/step - loss: 0.3116 - val\_loss: 0.3033

Epoch 12/25

600/600 [==============================] - 9s 16ms/step - loss: 0.3112 - val\_loss: 0.3014

Epoch 13/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3106 - val\_loss: 0.3005

Epoch 14/25

600/600 [==============================] - 8s 13ms/step - loss: 0.3103 - val\_loss: 0.3004

Epoch 15/25

600/600 [==============================] - 6s 11ms/step - loss: 0.3099 - val\_loss: 0.2995

Epoch 16/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3096 - val\_loss: 0.2993

Epoch 17/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3093 - val\_loss: 0.2987

Epoch 18/25

600/600 [==============================] - 8s 13ms/step - loss: 0.3091 - val\_loss: 0.2994

Epoch 19/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3089 - val\_loss: 0.2997

Epoch 20/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3086 - val\_loss: 0.2985

Epoch 21/25

600/600 [==============================] - 6s 11ms/step - loss: 0.3086 - val\_loss: 0.2985

Epoch 22/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3084 - val\_loss: 0.2984

Epoch 23/25

600/600 [==============================] - 6s 10ms/step - loss: 0.3083 - val\_loss: 0.2978

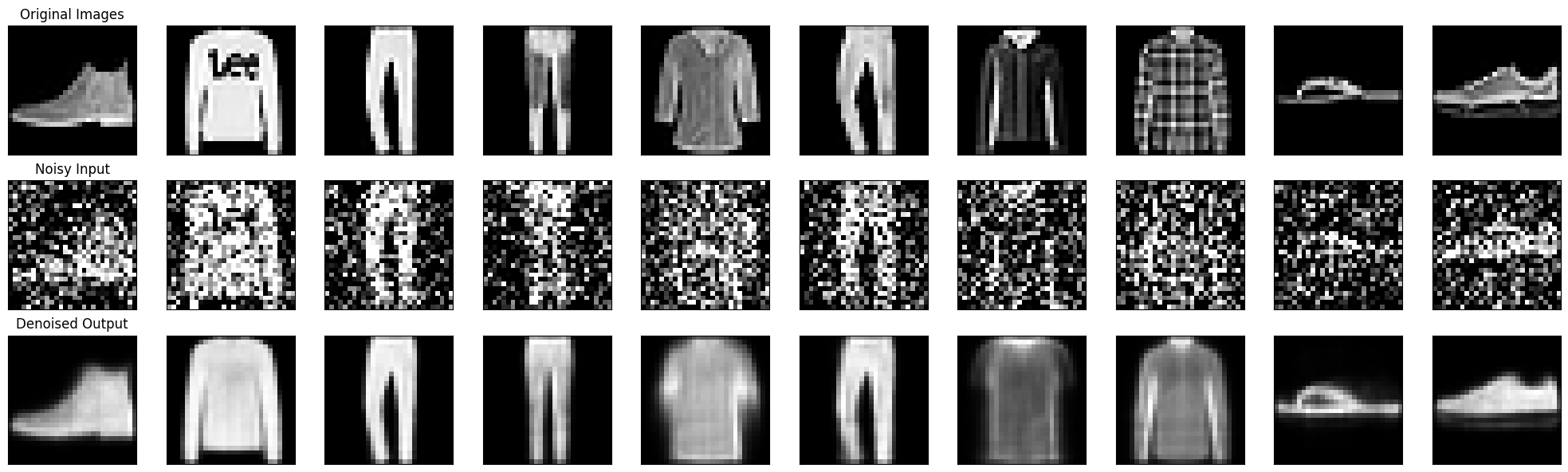
Epoch 24/25

600/600 [==============================] - 7s 12ms/step - loss: 0.3081 - val\_loss: 0.2977

Epoch 25/25

600/600 [==============================] - 6s 9ms/step - loss: 0.3079 - val\_loss: 0.2979

313/313 [==============================] - 1s 2ms/step



4. plot loss and accuracy using the history object

Code:

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Input, Dense, GaussianNoise

from keras.models import Model

from keras.datasets import fashion\_mnist

# Load and preprocess Fashion MNIST data

(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:])))

# Add Gaussian noise to the data

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)

# Clip values to be between 0 and 1

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

# Define the denoising autoencoder model

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

noisy\_img = GaussianNoise(0.5)(input\_img)

encoded = Dense(128, activation='relu')(noisy\_img)

hidden\_layer = Dense(64, activation='relu')(encoded)

decoded = Dense(784, activation='sigmoid')(hidden\_layer)

autoencoder = Model(input\_img, decoded)

# Compile the model

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

# Train the denoising autoencoder

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

                          epochs=20,

                          batch\_size=128,

                          shuffle=True,

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

# Plot training history (loss)

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

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

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

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

Output:

Epoch 1/20

469/469 [==============================] - 8s 15ms/step - loss: 0.3783 - val\_loss: 0.3550

Epoch 2/20

469/469 [==============================] - 5s 11ms/step - loss: 0.3313 - val\_loss: 0.3413

Epoch 3/20

469/469 [==============================] - 7s 15ms/step - loss: 0.3249 - val\_loss: 0.3341

Epoch 4/20

469/469 [==============================] - 5s 11ms/step - loss: 0.3218 - val\_loss: 0.3237

Epoch 5/20

469/469 [==============================] - 7s 15ms/step - loss: 0.3193 - val\_loss: 0.3246

Epoch 6/20

469/469 [==============================] - 7s 15ms/step - loss: 0.3174 - val\_loss: 0.3117

Epoch 7/20

469/469 [==============================] - 7s 14ms/step - loss: 0.3159 - val\_loss: 0.3101

Epoch 8/20

469/469 [==============================] - 6s 12ms/step - loss: 0.3147 - val\_loss: 0.3075

Epoch 9/20

469/469 [==============================] - 7s 14ms/step - loss: 0.3137 - val\_loss: 0.3065

Epoch 10/20

469/469 [==============================] - 5s 11ms/step - loss: 0.3129 - val\_loss: 0.3046

Epoch 11/20

469/469 [==============================] - 5s 12ms/step - loss: 0.3123 - val\_loss: 0.3048

Epoch 12/20

469/469 [==============================] - 7s 15ms/step - loss: 0.3119 - val\_loss: 0.3034

Epoch 13/20

469/469 [==============================] - 6s 13ms/step - loss: 0.3114 - val\_loss: 0.3031

Epoch 14/20

469/469 [==============================] - 7s 15ms/step - loss: 0.3109 - val\_loss: 0.3019

Epoch 15/20

469/469 [==============================] - 5s 12ms/step - loss: 0.3106 - val\_loss: 0.3018

Epoch 16/20

469/469 [==============================] - 6s 14ms/step - loss: 0.3102 - val\_loss: 0.3015

Epoch 17/20

469/469 [==============================] - 5s 11ms/step - loss: 0.3100 - val\_loss: 0.3004

Epoch 18/20

469/469 [==============================] - 6s 13ms/step - loss: 0.3098 - val\_loss: 0.3003

Epoch 19/20

469/469 [==============================] - 6s 12ms/step - loss: 0.3095 - val\_loss: 0.2994

Epoch 20/20

469/469 [==============================] - 6s 12ms/step - loss: 0.3093 - val\_loss: 0.2988

A graph with a line and numbers

Description automatically generated