**Neural Networks & Deep Learning - ICP-5**

**Name: Sampath Kumar Medam**

**Student Id: 700760222**

**Github link: <https://github.com/Sampath119/icp_5/blob/main/ICP5.ipynb>**

The autoencoder helps to compress the input images into a lower-dimensional encoded representation and then reconstruct the original images from this representation.

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=105,

batch\_size=256,

shuffle=True,

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

from keras.layers import Input, Dense

from keras.models import Model

from keras.datasets import mnist, fashion\_mnist

import numpy as np

import matplotlib.pyplot as plt

# Define the encoder dimension

encoding\_dim = 32

# Define the input placeholder

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

# Define the first hidden layer

hidden\_1 = Dense(256, activation='relu')(input\_img)

# Define the second hidden layer

encoded = Dense(encoding\_dim, activation='relu')(hidden\_1)

# Define the first hidden layer of the decoder

hidden\_2 = Dense(256, activation='relu')(encoded)

# Define the output layer

decoded = Dense(784, activation='sigmoid')(hidden\_2)

# Define the autoencoder model

autoencoder = Model(input\_img, decoded)

# Compile the model

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

# Load the fashion MNIST dataset

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

# Normalize the data and flatten the images

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 the autoencoder

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

epochs=105,

batch\_size=256,

shuffle=True,

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

# Make predictions on the test data

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

# Visualize one of the reconstructed images

n = 10 # number of images to display

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

for i in range(n):

# Display original test image

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 reconstructed test image

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()

# Plot the loss and accuracy over time

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper right')

plt.show()

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

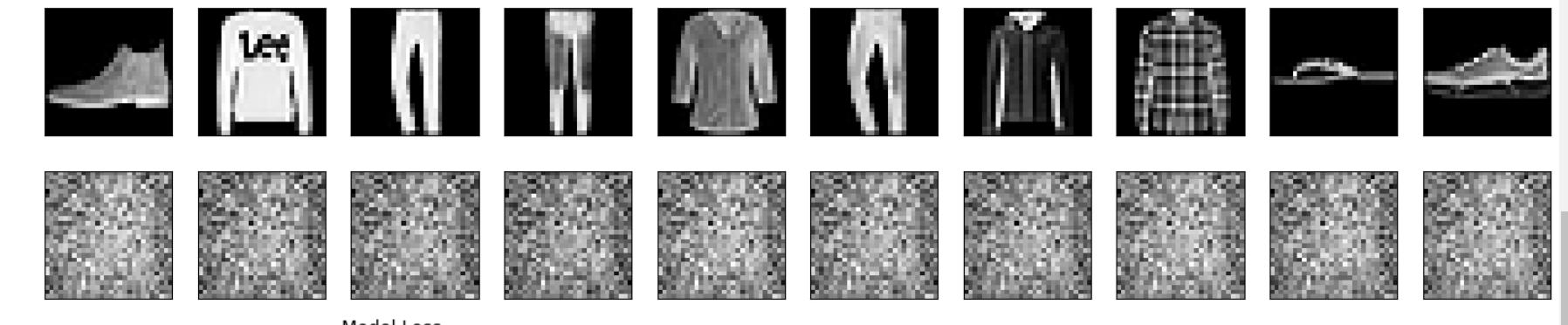
plt.xlabel('Epoch')

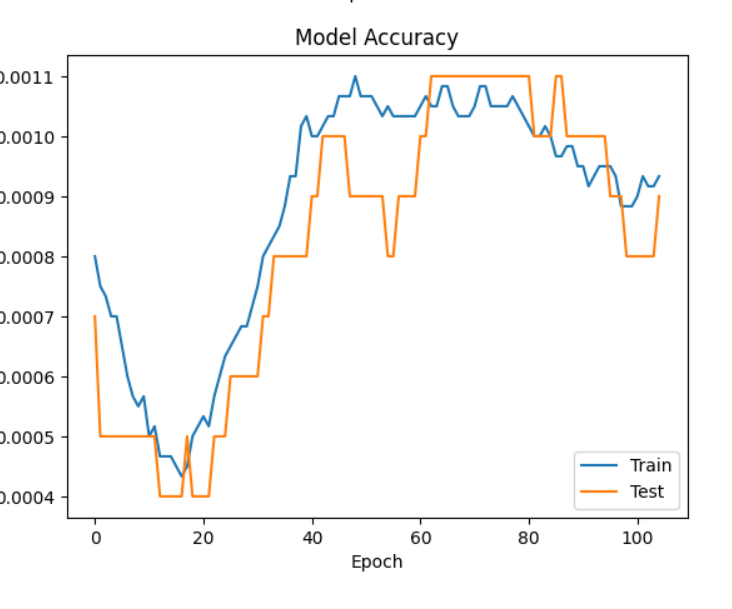
plt.legend(['Train', 'Test'], loc='lower right')

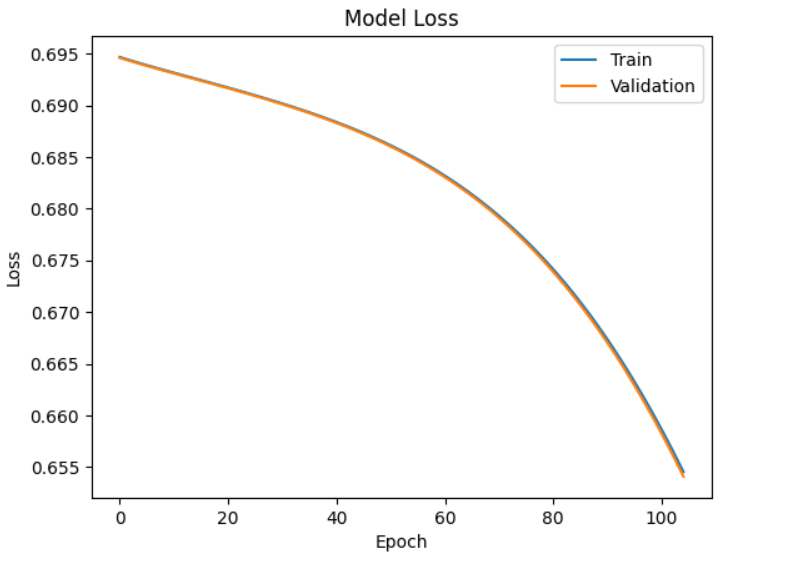
plt.show()

When we add more layers and increase the epoch the image will be more clear. The autoencoder learns to compress the input images into a lower-dimensional representation and then reconstruct the original images from this representation. The plots of loss and accuracy provide insights into the training process, and the visualizations show how well the autoencoder can reconstruct the original images.

**Output:**







When we add noise autoencoder can reconstruct the original images from noisy inputs, and the plots of loss and accuracy provide in the training process.

from keras.layers import Input, Dense

from keras.models import Model

from keras.datasets import fashion\_mnist

import numpy as np

import matplotlib.pyplot as plt

# Define the encoder dimension

encoding\_dim = 32

# Define the input placeholder

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

# Define the encoder layer

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

# Define the decoder layer

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

# Define the autoencoder model

autoencoder = Model(input\_img, decoded)

# Compile the model

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

# Load the fashion MNIST dataset

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

# Normalize the data and flatten the images

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 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)

# Ensure the noisy data is still within the valid range

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

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

# Train the autoencoder

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

epochs=105,

batch\_size=256,

shuffle=True,

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

# Generate reconstructed images from the noisy test data

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

# Visualize noisy and reconstructed test images

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

n = 10

for i in range(n):

# Display original + noise

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

plt.imshow(x\_test\_noisy[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()

# Plot the loss over time

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()

# Plot the accuracy over time

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='lower right')

plt.show()

Output:

