**DL ASSIGNMENT 8**

**Name: Sushmitha Chelledi**

**ID: 700744167**

**Use Case Description:**

**Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image ]**

**Use case: Stacked autoencoder In-class programming:**

1. **Add one more hidden layer to the autoencoder.**
2. **Do the prediction on the test data and then visualize one of the reconstructed versions of that test data. Also, visualize the same test data before reconstruction using Matplotlib.**
3. **Repeat the question 2 on the denoising autoencoder.**
4. **plot loss and accuracy using the history objec**t.

**Source Code:**

**from keras.layers import Input, Dense**

**from keras.models import Model**

**from keras.datasets import mnist**

**import numpy as np**

**encoding\_dim = 64**

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

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

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

**autoencoder = Model(input\_img, decoded)**

**encoder = Model(input\_img, encoded)**

**encoded\_input = Input(shape=(encoding\_dim,))**

**decoder\_layer = autoencoder.layers[-1]**

**decoder = Model(encoded\_input, decoder\_layer(encoded\_input))**

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

**(x\_train, \_), (x\_test, \_) = 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:])))**

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

**epochs=5,**

**batch\_size=256,**

**shuffle=True,**

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

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

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

**# graph**

**import matplotlib.pyplot as plt**

**plt.plot(history.history['loss'], color="green")**

**plt.plot(history.history['val\_loss'], color="blue")**

**plt.title('model loss')**

**plt.legend(['train', 'validation'], loc='upper right')**

**plt.show()**

**input\_size = 784**

**hidden\_size = 128**

**code\_size = 32**

**input\_img = Input(shape=(input\_size,))**

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

**code = Dense(code\_size, activation='relu')(hidden\_1)**

**hidden\_2 = Dense(hidden\_size, activation='relu')(code)**

**output\_img = Dense(input\_size, activation='sigmoid')(hidden\_2)**

**autoencoder = Model(input\_img, output\_img)**

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

**(x\_train, \_), (x\_test, \_) = 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:])))**

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

**epochs=5,**

**batch\_size=256,**

**shuffle=True,**

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

**ncoded\_imgs = encoder.predict(x\_test)**

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

**import matplotlib.pyplot as plt**

**n = 3**

**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()**

**# graph**

**plt.plot(history.history['loss'], color="green")**

**plt.plot(history.history['val\_loss'], color="blue")**

**plt.title('model loss')**

**plt.legend(['train', 'validation'], loc='upper right')**

**plt.show()**

**from keras.layers import Input, Dense**

**from keras.models import Model, Sequential**

**# Scales the training and test data to range between 0 and 1.**

**max\_value = float(x\_train.max())**

**x\_train = x\_train.astype('float32') / max\_value**

**x\_test = x\_test.astype('float32') / max\_value**

**x\_train.shape, x\_test.shape**

**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:])))**

**(x\_train.shape, x\_test.shape)**

**input\_dim = x\_train.shape[1]**

**encoding\_dim = 64**

**compression\_factor = float(input\_dim) / encoding\_dim**

**print("Compression factor: %s" % compression\_factor)**

**autoencoder = Sequential()**

**autoencoder.add(**

**Dense(encoding\_dim, input\_shape=(input\_dim,), activation='relu')**

**)**

**autoencoder.add(**

**Dense(input\_dim, activation='sigmoid')**

**)**

**autoencoder.summary()**

**input\_img = Input(shape=(input\_dim,))**

**encoder\_layer = autoencoder.layers[0]**

**encoder = Model(input\_img, encoder\_layer(input\_img))**

**encoder.summary()**

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

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

**epochs=5,**

**batch\_size=256,**

**shuffle=True,**

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

**num\_images = 5**

**np.random.seed(42)**

**random\_test\_images = np.random.randint(x\_test.shape[0], size=num\_images)**

**noise = np.random.normal(loc=0.1, scale=0.1, size=x\_test.shape)**

**noised\_images = x\_test + noise**

**encoded\_imgs = encoder.predict(noised\_images)**

**decoded\_imgs = autoencoder.predict(noised\_images)**

**Screenshots of the output:**

Graphical user interface, text

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, text

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedA picture containing table

Description automatically generated

**Description**: In this assignment, I did autoencoder-Reconstructing for the existing image, which will contain most important features of the image 5. And Stacked autoencoder in class programming then Added one more hidden layer to autoencoder. Did the prediction on the test data and then visualized one of the reconstructed versions of that test data. Also, visualized the same test data before reconstruction using Matplotlib. Repeated the process of step on the denoising autoencoder. Retrieved the plot loss and accuracy using the history object.

Video Link: <https://drive.google.com/file/d/1fAHgRebxTgaRkq66ZjYWkYYacRhhzaQL/view?usp=sharing>

Githublink: <https://github.com/ChellediSushmitha/Neural-assignment-8>

Source code: <http://localhost:8888/notebooks/DL_Assignment_8.ipynb>