NN&DL-ICP4

Name: Akshay Chandre

Id:700741715

Video Link: https://drive.google.com/file/d/1Q5q6739jq78ueMJc3LsSaZVeS1r5Rnwd/view?usp=sharing

GitLink: https://github.com/akshaychandre47/AkshayChandre_NN-DL_Summer/blob/main/NN%26DL_700741715_ICP4.ipynb

1. Add one more hidden layer to autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt
from keras.datasets import fashion_mnist
import numpy as np

# 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 first hidden layer of the autoencoder
encoded1 = Dense(128, activation='relu')(input_img)
# "encoded" is the second hidden layer of the autoencoder
encoded2 = Dense(64, activation='relu')(encoded1)
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(encoded2)

# "decoded" is the first hidden layer in the decoder
decoded1 = Dense(64, activation='relu')(encoded)
# "decoded" is the second hidden layer in the decoder
decoded2 = Dense(128, activation='relu')(decoded1)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(decoded2)
```

```
# 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')

# Print the summary of the autoencoder architecture
autoencoder.summary()

# Load the Fashion MNIST dataset and preprocess the data
(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 the autoencoder on x_train and x_train
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

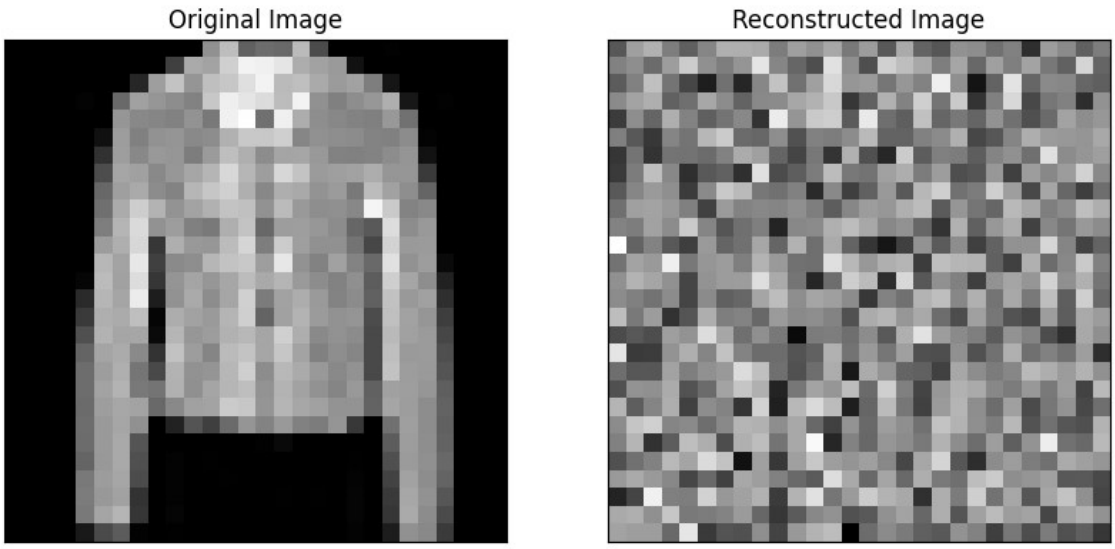
```
=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Epoch 1/5
235/235 [==============================] - 20s 52ms/step - loss: 0.6931 - accuracy: 0.0011 - val_loss: 0.6931 - val_accuracy: 7.0000e-04
Epoch 2/5
235/235 [==============================] - 7s 30ms/step - loss: 0.6931 - accuracy: 0.0011 - val_loss: 0.6930 - val_accuracy: 6.0000e-04
Epoch 3/5
235/235 [==============================] - 11s 47ms/step - loss: 0.6930 - accuracy: 0.0012 - val_loss: 0.6930 - val_accuracy: 6.0000e-04
Epoch 4/5
235/235 [==============================] - 7s 31ms/step - loss: 0.6930 - accuracy: 0.0012 - val_loss: 0.6929 - val_accuracy: 6.0000e-04
Epoch 5/5
235/235 [==============================] - 6s 26ms/step - loss: 0.6929 - accuracy: 0.0012 - val_loss: 0.6929 - val_accuracy: 6.0000e-04
<keras.callbacks.History at 0x7bcf8c2a5de0>
```

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**

```
# Predict the test data
reconstructed_images = autoencoder.predict(x_test)
denoised_images = autoencoder.predict(x_test)
# Choose one random image index from the test data for visualization
image_index = np.random.randint(0, len(x_test))
# Choose a random image from the test set
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))
# Plot the original image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Original Image")

# Plot the reconstructed image
ax = plt.subplot(1, 2, 2)
plt.imshow(denoised_images[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")

plt.show()
```



3. Repeat the question 2 on the denoisening autoencoder

```
# "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')

# Load the Fashion MNIST dataset and preprocess the 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:])))

# Introduce 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)
```

```
# Train the denoising autoencoder on x_train_noisy and x_train
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

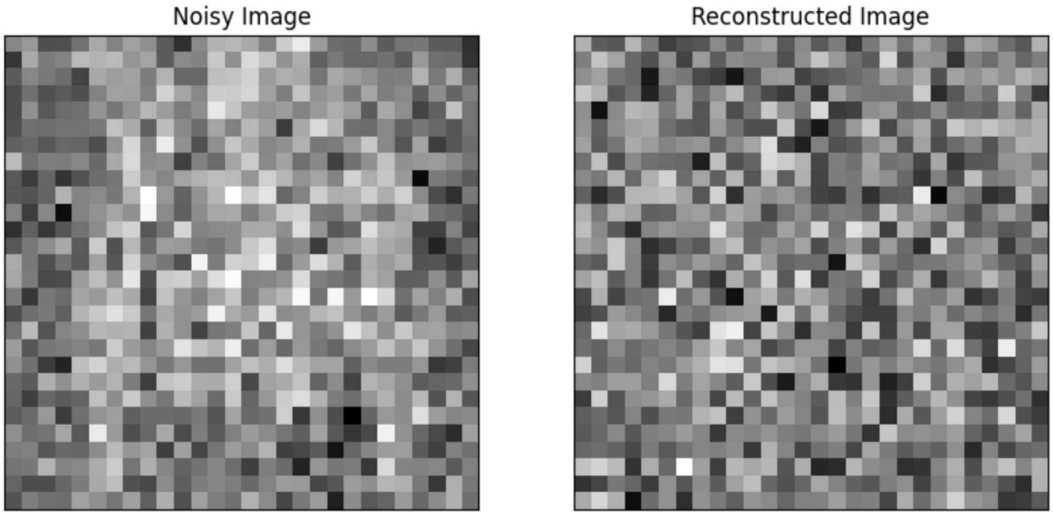
```
# Predict the test data after denoising
denoised_images = autoencoder.predict(x_test_noisy)
# Choose one random image index from the test data for visualization
image_index = np.random.randint(0, len(x_test))

# Choose a random image from the test set
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))

# Plot the original noisy image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Noisy Image")

# Plot the reconstructed image
ax = plt.subplot(1, 2, 2)
plt.imshow(denoised_images[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")
plt.show()
```

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

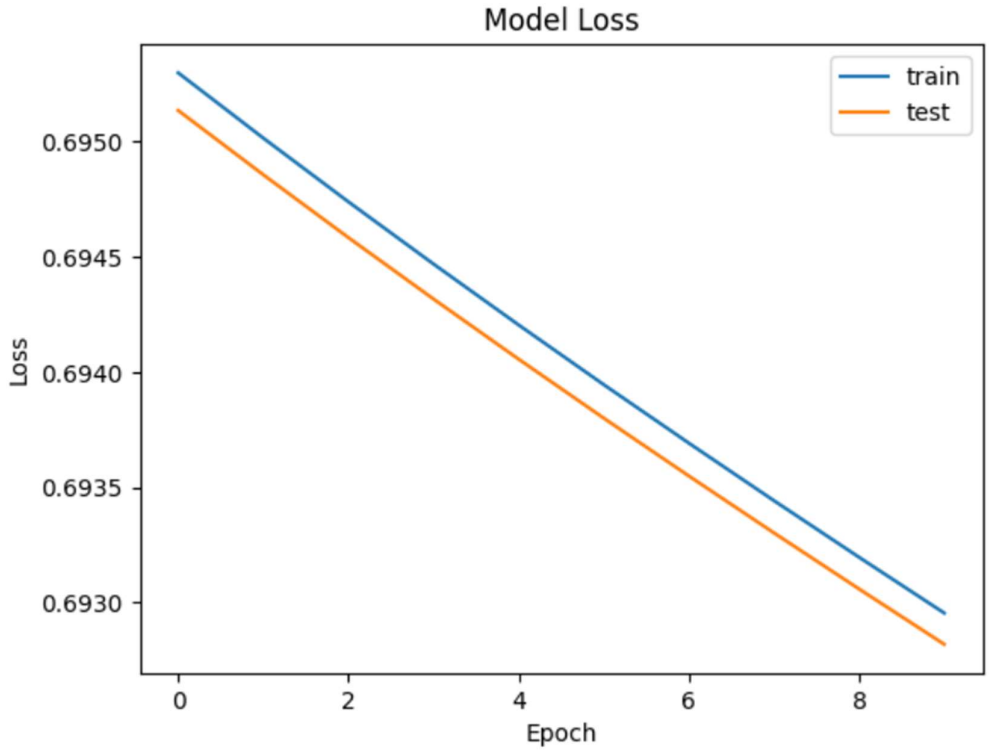


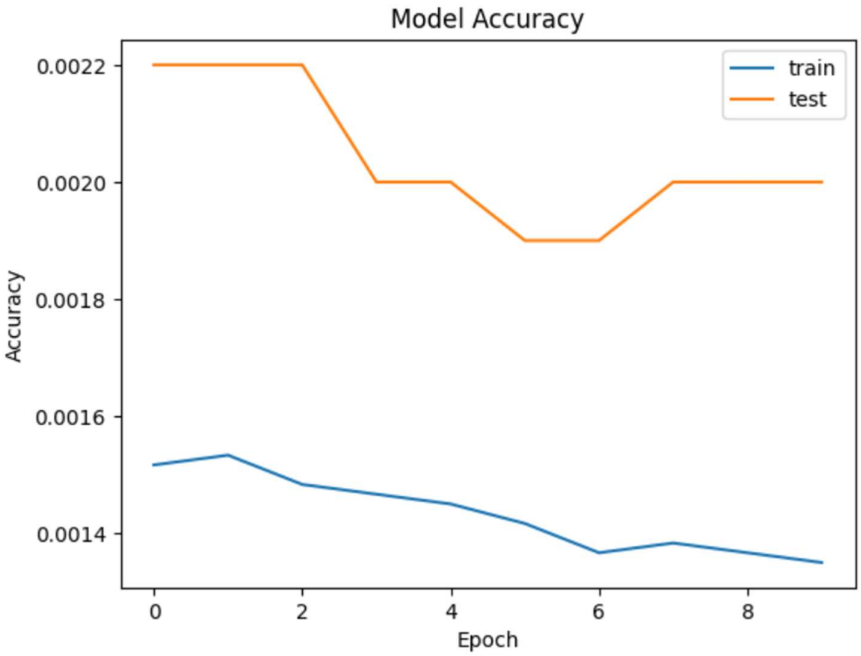
4. plot loss and accuracy using the history object

```
# Train the autoencoder
history = autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))

# Plot the loss
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Plot the accuracy
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```




Model Accuracy
train
test
0.0022
0.0020
0.0018
0.0016
0.0014
Accuracy
0
2
4
6
8
Epoch