Neural Networks & Deep Learning: ICP4

Name: Pavani Medavarthi
Student ID: 700741643

Video Link :
https://drive.google.com/file/d/1Nact4tC6Q1iyHIpLhAHCpHtL43pYUnKQ/view?usp=drive_link

GitHub Link: https://github.com/Pavanimedavarthi/NN-DL_Summer-2

IN CLASS PROGRAMMING:

1. Add One more hidden layer to the autoencoder:

In this step, we change the architecture of the autoencoder to include an additional hidden layer. This new layer will be added after the original encoded layer but before the decoder. The ReLU activation function determines the number of nodes in this new layer to be 64.

700741643_PavaniMedavarthi_ICP4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
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 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))
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 784)]             0

 dense_4 (Dense)             (None, 128)               100480

 dense_5 (Dense)             (None, 64)                8256

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (Dense)             (None, 64)                2112

 dense_8 (Dense)             (None, 128)               8320

 dense_9 (Dense)             (None, 784)               101136

=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
235/235 [==============================] - 7s 25ms/step - loss: 0.6930 - accuracy: 0.0011 - val_loss: 0.6929 - val_accuracy: 5.0000e-04
Epoch 2/5
235/235 [==============================] - 8s 36ms/step - loss: 0.6929 - accuracy: 0.0011 - val_loss: 0.6929 - val_accuracy: 5.0000e-04
Epoch 3/5
235/235 [==============================] - 10s 42ms/step - loss: 0.6929 - accuracy: 0.0011 - val_loss: 0.6928 - val_accuracy: 5.0000e-04
Epoch 4/5
235/235 [==============================] - 5s 23ms/step - loss: 0.6928 - accuracy: 0.0012 - val_loss: 0.6928 - val_accuracy: 5.0000e-04
Epoch 5/5
235/235 [==============================] - 5s 22ms/step - loss: 0.6927 - accuracy: 0.0012 - val_loss: 0.6927 - val_accuracy: 5.0000e-04
<keras.callbacks.History at 0x78026b1fdd20>
```

Explanation:

By adding this extra hidden layer, the autoencoder now has an additional layer to capture more complex patterns and features in the data.

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:**

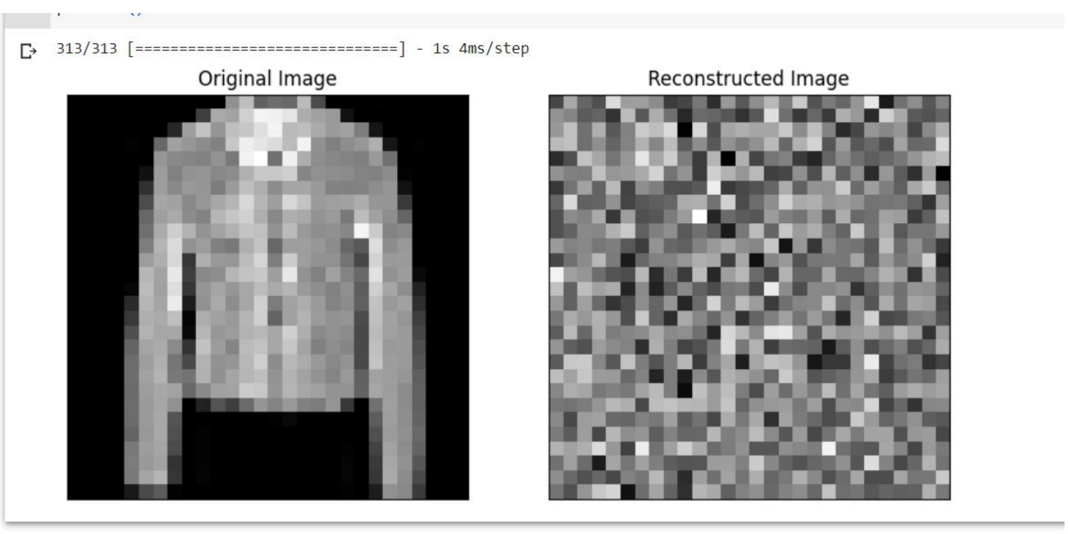
After training the autoencoder, we make predictions on the test data and use Matplotlib to exhibit one randomly picked reconstructed image alongside its original image.

700741643_PavaniMedavarthi_ICP4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
# Predict the test data
reconstructed_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()
```



Explanation:

We use the trained autoencoder to predict the reconstructed images from the test data. Then, we randomly choose one test image, display the original image using plt.imshow(), and show the corresponding reconstructed image using the same function.

3. **Repeat the question 2 on the denoising autoencoder:**

We use the same approach as in question 2 for the denoising autoencoder. After training the denoising autoencoder, we make predictions on the noisy test data and use Matplotlib to visualize one randomly selected denoised image alongside its original noisy image.

```
# "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))
```

700741643_PavaniMedavarthi_ICP4.ipynb

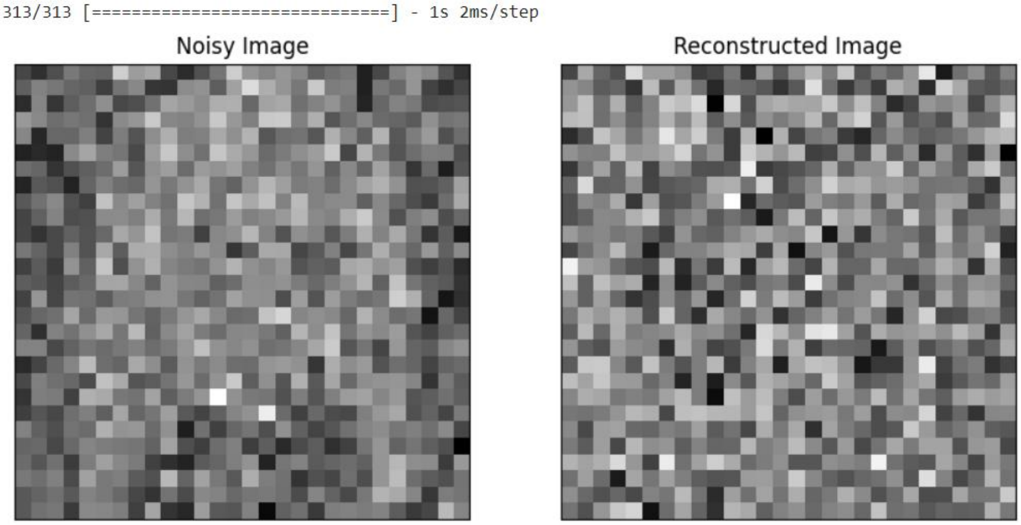
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
# 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()
```

```python
plt.show()
```

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



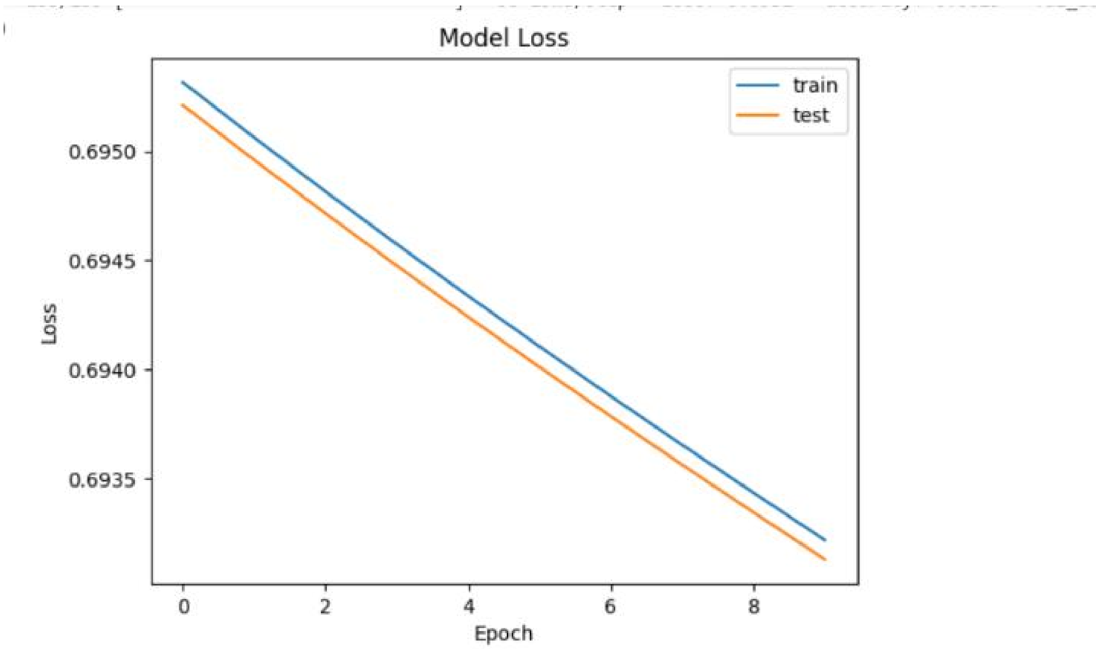
Explanation:

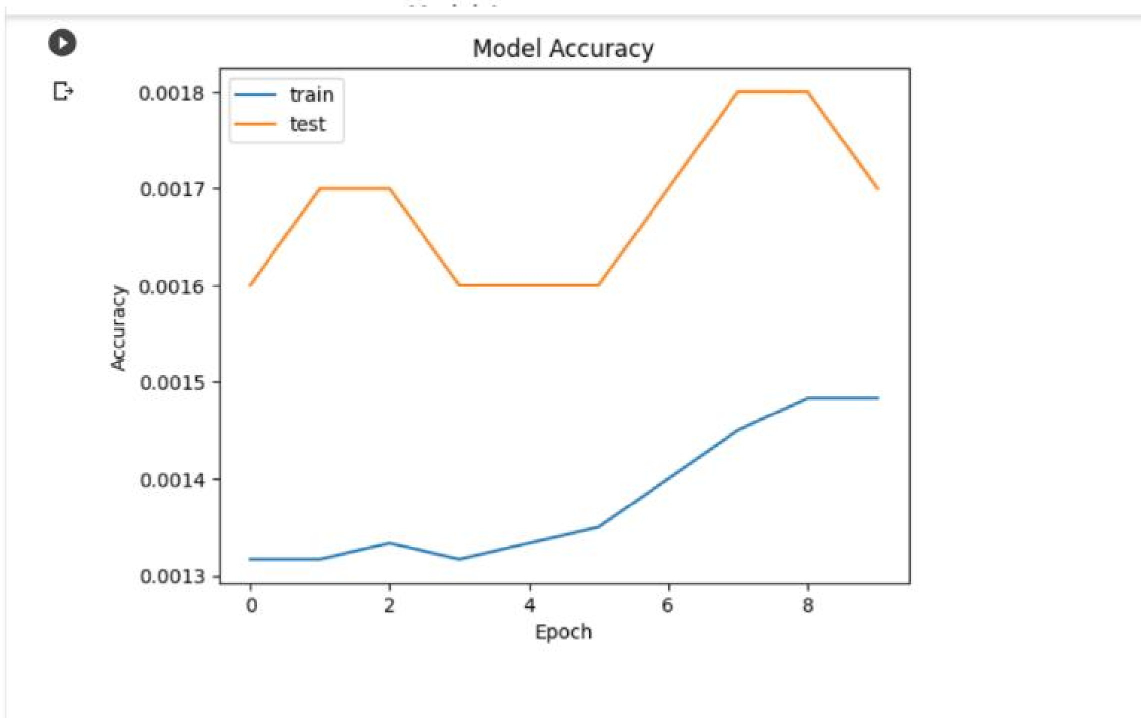
In this step, we used the trained denoising autoencoder to predict the denoised images from the noisy test data x_test_noisy. After predictions, we randomly selected one noisy test image with the index image_index and displayed the original noisy image using plt.imshow(), enabling us to observe the Fashion MNIST image before denoising. We also displayed the corresponding denoised image using the same function, allowing us to visualize the image after the denoising process using the denoising autoencoder.

4. **Plot loss and accuracy using the history object:**

   We track the loss during the training process and plot the training and validation losses to see how the autoencoder's performance varies over time.


```python
import matplotlib.pyplot as plt

# 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()
```





Explanation:

During training, we store the training and validation loss in the history object. Then, we access the loss values for each epoch using history. history['loss'] and history. history['val_loss']. We plot these values using Matplotlib to observe how the autoencoder's loss changes over the training process, which can provide insights into the model's performance and overfitting tendencies.