Neural Network & Deep Learning

(CRN-31174), **ASSIGNMENT-ICP4**

Student: ***Sridhar Seepana*** ID:***700742973***

Question1.

```
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', metrics ='accuracy')
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=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/5
235/235 [==============================] - 4s 12ms/step - loss: 0.6937 - accuracy: 0.0036 - val_loss: 0.6936 - val_accuracy: 0.0045
Epoch 2/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 0.0037 - val_loss: 0.6935 - val_accuracy: 0.0046
Epoch 3/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6934 - accuracy: 0.0037 - val_loss: 0.6933 - val_accuracy: 0.0045
Epoch 4/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6933 - accuracy: 0.0037 - val_loss: 0.6932 - val_accuracy: 0.0044
Epoch 5/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6931 - accuracy: 0.0037 - val_loss: 0.6931 - val_accuracy: 0.0044
<keras.callbacks.History at 0x2b9d0410e50>
```

Description:

For the purposes of compressing and reconstructing images from the Fashion MNIST dataset, this code configures an autoencoder neural network using Keras. The autoencoder contains three layers: a hidden layer with 32 neurons (for encoding), an output layer with 784 neurons (for decoding), and an input layer with 784 neurons (matching to the flattened picture pixels). Using the "adadelta" optimizer and "binary_crossentropy" loss function for 5 epochs with a batch size of 256, the autoencoder is trained on the Fashion MNIST pictures.

Question2.

700742973_SRIDHAR_SEEPANA_NN&DL_ICP4.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:51 PM

+ Code + Text

```python
from keras.models import Model

# This is the size of our encoded representation
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
encoded1 = Dense(128, activation='relu')(input_img)
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

# "decoded" is the lossy reconstruction of the input
decoded1 = Dense(128, activation='relu')(encoded2)
decoded2 = Dense(784, activation='sigmoid')(decoded1)

# This model maps an input to its reconstruction
autoencoder = Model(input_img, decoded2)

# This model maps an input to its encoded representation
encoder = Model(input_img, encoded2)

# This is our decoder model
encoded_input = Input(shape=(encoding_dim,))
decoder_layer1 = autoencoder.layers[-2]
decoder_layer2 = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer2(decoder_layer1(encoded_input)))

# Compile the model
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics ='accuracy')

# Load the MNIST dataset
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize and flatten the 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
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/5
235/235 [==============================] - 7s 22ms/step - loss: 0.6939 - accuracy: 0.0027 - val_loss: 0.6938 - val_accuracy: 0.0025
Epoch 2/5
235/235 [==============================] - 4s 19ms/step - loss: 0.6938 - accuracy: 0.0028 - val_loss: 0.6937 - val_accuracy: 0.0025
Epoch 3/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6937 - accuracy: 0.0027 - val_loss: 0.6936 - val_accuracy: 0.0026
Epoch 4/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6936 - accuracy: 0.0027 - val_loss: 0.6935 - val_accuracy: 0.0026
Epoch 5/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6935 - accuracy: 0.0027 - val_loss: 0.6934 - val_accuracy: 0.0028
<keras.callbacks.History at 0x2b9d7038350>
```

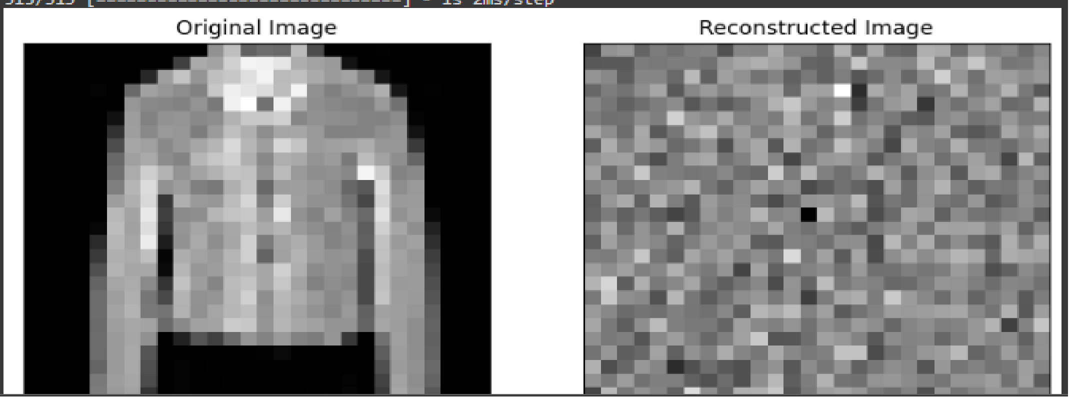
700742973_SRIDHAR_SEEPANA_NN&DL_ICP4.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:51 PM

+ Code + Text

```python
import matplotlib.pyplot as plt

# Get the reconstructed images for the test set
reconstructed_imgs = autoencoder.predict(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(reconstructed_imgs[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
```



Description:

The Fashion MNIST dataset's images are compressed and reconstructed using a deep autoencoder neural network implemented in this code using Keras. The autoencoder contains two layers of encoding: the first layer uses the ReLU activation function to compress the input to 128 neurons, and the second layer uses ReLU activation to further compress it to 32 neurons. The decoder also consists of two layers; the first layer expands the compressed representation back to 128 neurons with ReLU activation, while the second layer recreates the original image with 784 neurons and the sigmoid activation function. Using the "adadelta" optimizer and "binary_crossentropy" loss function for 5 epochs with a batch size of 256, the model is trained on the Fashion MNIST images. Furthermore, the code develops distinct models for the encoder and decoder using Keras.

Then in the second code, method reconstructs images from the Fashion MNIST test set using the trained autoencoder, and then uses matplotlib to provide a comparison between an original image and its associated rebuilt image.

Question3.

700742973_SRIDHAR_SEEPANA_NN&DL_ICP4.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:51 PM

+ Code + Text

```python
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',metrics ='accuracy')
from keras.datasets import fashion_mnist
import numpy as np
(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:])))

#introducing noise
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)

autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

```
Epoch 1/10
235/235 [==============================] - 4s 14ms/step - loss: 0.6964 - accuracy: 7.8333e-04 - val_loss: 0.6963 - val_accuracy: 8.0000e-04
Epoch 2/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6962 - accuracy: 8.0000e-04 - val_loss: 0.6961 - val_accuracy: 8.0000e-04
Epoch 3/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6959 - accuracy: 8.1667e-04 - val_loss: 0.6959 - val_accuracy: 8.0000e-04
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6957 - accuracy: 8.6667e-04 - val_loss: 0.6956 - val_accuracy: 7.0000e-04
Epoch 5/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6955 - accuracy: 8.6667e-04 - val_loss: 0.6954 - val_accuracy: 7.0000e-04
Epoch 6/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6952 - accuracy: 8.6667e-04 - val_loss: 0.6952 - val_accuracy: 8.0000e-04
```

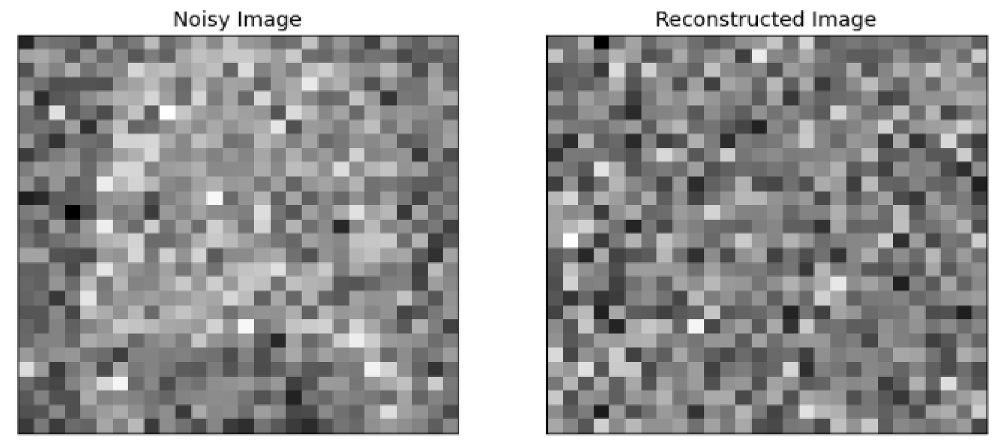
700742973_SRIDHAR_SEEPANA_NN&DL_ICP4.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:51 PM

+ Code + Text

```python
import matplotlib.pyplot as plt

# Get the reconstructed images for the test set
reconstructed_imgs = autoencoder.predict(x_test_noisy)

# 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(reconstructed_imgs[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 3ms/step
```



Description:

In order to reduce noise in the photos in the Fashion MNIST dataset, this code creates a denoising autoencoder using Keras. To learn a representation that can efficiently denoise the input data, the autoencoder is trained on noisy images (obtained by adding random noise to the original images) and their corresponding clean images. The "adadelta" optimizer and "binary_crossentropy" loss function are used during the model's 10-epoch training process, which uses noisy data as input and the original, clear images as the desired output.

This code uses the denoising autoencoder trained on the noisy Fashion MNIST images to reconstruct and compare a specific example of a noisy image with its corresponding denoised version. The left subplot displays the original noisy image, and the right subplot shows the reconstructed denoised image.

Question4.

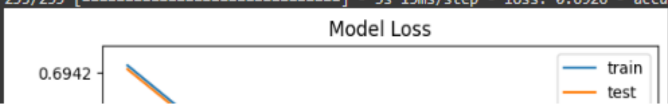
700742973_SRIDHAR_SEEPANA_NN&DL_ICP4.ipynb

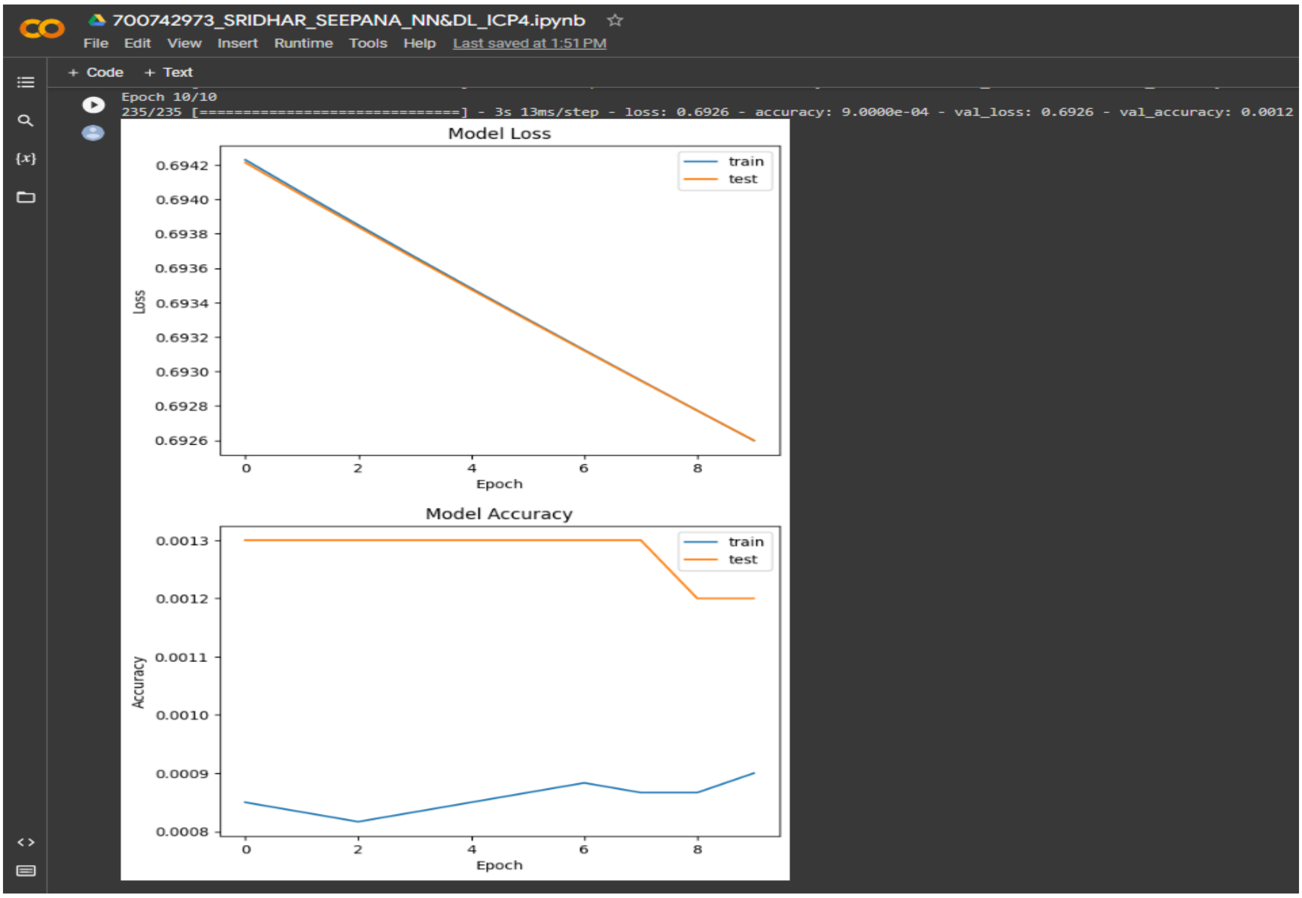
File Edit View Insert Runtime Tools Help Last saved at 1:51 PM

+ Code + Text

```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_noisy))

# 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()
```

```
Epoch 1/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6942 - accuracy: 8.5000e-04 - val_loss: 0.6942 - val_accuracy: 0.0013
Epoch 2/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6940 - accuracy: 8.3333e-04 - val_loss: 0.6940 - val_accuracy: 0.0013
Epoch 3/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6939 - accuracy: 8.1667e-04 - val_loss: 0.6938 - val_accuracy: 0.0013
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6937 - accuracy: 8.3333e-04 - val_loss: 0.6937 - val_accuracy: 0.0013
Epoch 5/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 8.5000e-04 - val_loss: 0.6935 - val_accuracy: 0.0013
Epoch 6/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6933 - accuracy: 8.6667e-04 - val_loss: 0.6933 - val_accuracy: 0.0013
Epoch 7/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6931 - accuracy: 8.8333e-04 - val_loss: 0.6931 - val_accuracy: 0.0013
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6929 - accuracy: 8.6667e-04 - val_loss: 0.6929 - val_accuracy: 0.0013
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6928 - accuracy: 8.6667e-04 - val_loss: 0.6928 - val_accuracy: 0.0012
Epoch 10/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6926 - accuracy: 9.0000e-04 - val_loss: 0.6926 - val_accuracy: 0.0012
```





Description:

On the noisy Fashion MNIST dataset, this code trains the denoising autoencoder and presents the loss and accuracy metrics. To monitor the model's performance during training and possible overfitting, the first set of plots displays the training and validation loss over epochs, while the second set of plots shows the training and validation accuracy over epochs.

My GitHub Link:

https://github.com/700742973-SRIDHAR-SEEPANA/NN-DL_ICP4