

Neural Networks and Deep Learning – ICP 5

N. Manesh
700756918

GitHub link: <https://github.com/Manesh1712/ICP5>

Video Link: <https://drive.google.com/file/d/1xA5ds2QWC-KQ0nSB1U1UfpGGFMxLLogp/view?usp=sharing>

1. Add one more hidden layer to autoencoder

```
In [10]: 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))

         shuffle=True,
         validation_data=(x_test, x_test))

Epoch 1/5
235/235 [=====] - 5s 17ms/step - loss: 0.6953 - accuracy: 9.0000e-04 - val_loss: 0.6952 - val_accuracy: 8.0000e-04
Epoch 2/5
235/235 [=====] - 3s 14ms/step - loss: 0.6951 - accuracy: 9.0000e-04 - val_loss: 0.6949 - val_accuracy: 8.0000e-04
Epoch 3/5
235/235 [=====] - 3s 14ms/step - loss: 0.6948 - accuracy: 9.1667e-04 - val_loss: 0.6947 - val_accuracy: 8.0000e-04
Epoch 4/5
235/235 [=====] - 3s 13ms/step - loss: 0.6946 - accuracy: 9.3333e-04 - val_loss: 0.6945 - val_accuracy: 9.0000e-04
Epoch 5/5
235/235 [=====] - 3s 13ms/step - loss: 0.6944 - accuracy: 9.5000e-04 - val_loss: 0.6943 - val_accuracy: 9.0000e-04

:[10]: <keras.src.callbacks.History at 0x23310f8b310>
```

```
In [11]: from keras.layers import Input, Dense
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 21ms/step - loss: 0.6944 - accuracy: 0.0011 - val_loss: 0.6943 - val_accuracy: 0.0016
Epoch 2/5
235/235 [=====] - 4s 17ms/step - loss: 0.6943 - accuracy: 0.0011 - val_loss: 0.6942 - val_accuracy: 0.0016
Epoch 3/5
235/235 [=====] - 4s 17ms/step - loss: 0.6941 - accuracy: 0.0011 - val_loss: 0.6940 - val_accuracy: 0.0016
Epoch 4/5
235/235 [=====] - 4s 17ms/step - loss: 0.6940 - accuracy: 0.0011 - val_loss: 0.6939 - val_accuracy: 0.0015
Epoch 5/5
235/235 [=====] - 4s 17ms/step - loss: 0.6939 - accuracy: 0.0011 - val_loss: 0.6938 - val_accuracy: 0.0015
```

```
it[11]: <keras.src.callbacks.History at 0x23310fb19d0>
```

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

```
[16]: ► 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")

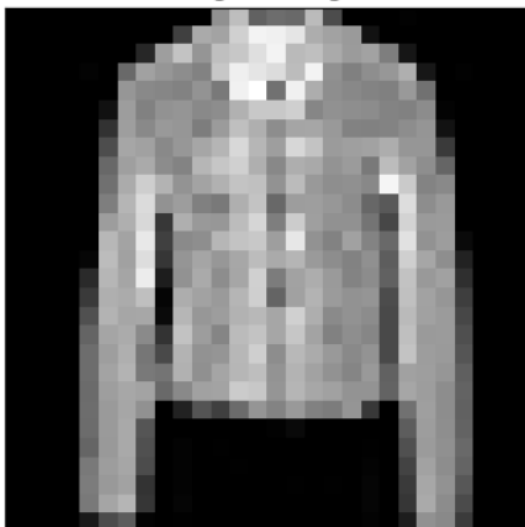
# Plot the reconstructed image
ax = plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")

plt.show()
```

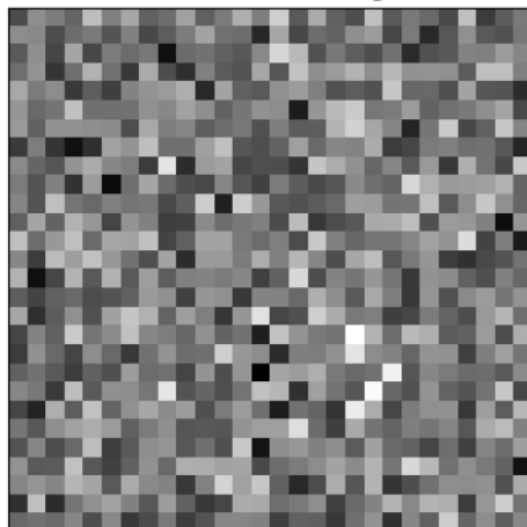
313/313 [=====] - 1s 3ms/step

313/313 [=====] - 1s 3ms/step

Original Image



Reconstructed Image



3. Repeat the question 2 on the denoising autoencoder

```
In [17]: 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 [=====] - 5s 16ms/step - loss: 0.6982 - accuracy: 5.3333e-04 - val_loss: 0.6979 - val_accuracy: 8.0000e-04
Epoch 2/10
235/235 [=====] - 3s 13ms/step - loss: 0.6978 - accuracy: 5.3333e-04 - val_loss: 0.6976 - val_accuracy: 8.0000e-04
Epoch 3/10
235/235 [=====] - 3s 11ms/step - loss: 0.6975 - accuracy: 5.3333e-04 - val_loss: 0.6972 - val_accuracy: 8.0000e-04
Epoch 4/10
235/235 [=====] - 3s 14ms/step - loss: 0.6971 - accuracy: 5.3333e-04 - val_loss: 0.6969 - val_accuracy: 9.0000e-04
Epoch 5/10
235/235 [=====] - 3s 13ms/step - loss: 0.6968 - accuracy: 5.5000e-04 - val_loss: 0.6966 - val_accuracy: 9.0000e-04
Epoch 6/10
235/235 [=====] - 3s 13ms/step - loss: 0.6965 - accuracy: 5.6667e-04 - val_loss: 0.6963 - val_accuracy: 9.0000e-04
Epoch 7/10
235/235 [=====] - 3s 13ms/step - loss: 0.6962 - accuracy: 6.0000e-04 - val_loss: 0.6960 - val_accuracy: 9.0000e-04
Epoch 8/10
235/235 [=====] - 3s 13ms/step - loss: 0.6960 - accuracy: 6.1667e-04 - val_loss: 0.6957 - val_accuracy: 9.0000e-04
Epoch 9/10
235/235 [=====] - 4s 15ms/step - loss: 0.6957 - accuracy: 6.0000e-04 - val_loss: 0.6955 - val_accuracy: 9.0000e-04
Epoch 10/10
235/235 [=====] - 3s 15ms/step - loss: 0.6954 - accuracy: 6.0000e-04 - val_loss: 0.6952 - val_accuracy: 9.0000e-04
```



```
In [20]: 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))

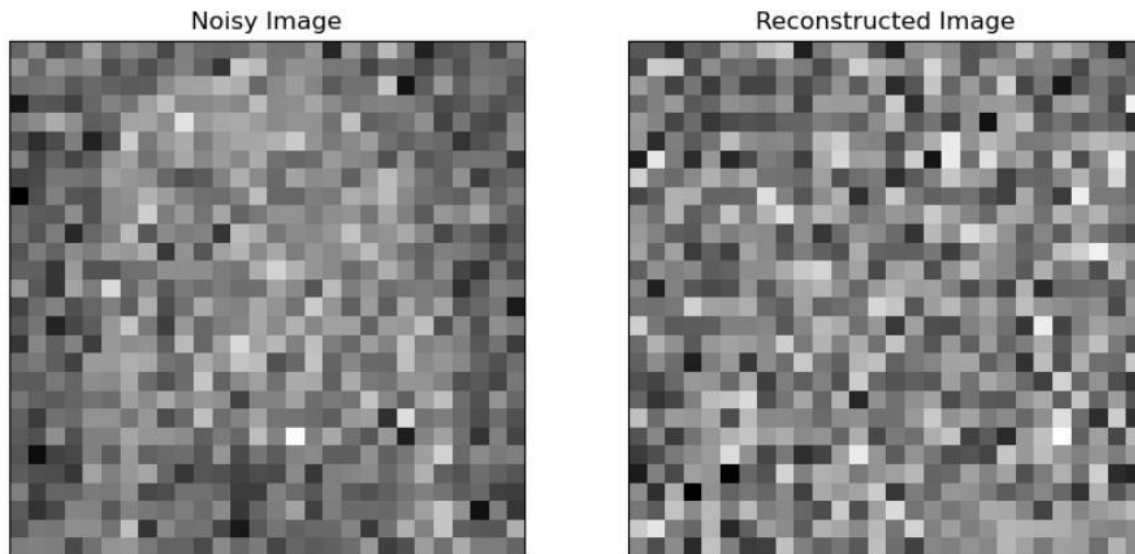
# Plot the original noisy image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Noisy Image")

# 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
```

313/313 [=====] - 1s 3ms/step



4. plot loss and accuracy using the history object

```
In [25]: 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()
```

```
plt.show()
```

```
Epoch 1/10
235/235 [=====] - 3s 14ms/step - loss: 0.6944 - accuracy: 0.0028 - val_loss: 0.6942 - val_accuracy: 0.0018
Epoch 2/10
235/235 [=====] - 3s 14ms/step - loss: 0.6942 - accuracy: 0.0027 - val_loss: 0.6940 - val_accuracy: 0.0018
Epoch 3/10
235/235 [=====] - 3s 13ms/step - loss: 0.6940 - accuracy: 0.0027 - val_loss: 0.6938 - val_accuracy: 0.0018
Epoch 4/10
235/235 [=====] - 3s 13ms/step - loss: 0.6938 - accuracy: 0.0027 - val_loss: 0.6936 - val_accuracy: 0.0018
Epoch 5/10
235/235 [=====] - 3s 13ms/step - loss: 0.6936 - accuracy: 0.0027 - val_loss: 0.6934 - val_accuracy: 0.0019
Epoch 6/10
235/235 [=====] - 3s 14ms/step - loss: 0.6935 - accuracy: 0.0027 - val_loss: 0.6933 - val_accuracy: 0.0019
Epoch 7/10
235/235 [=====] - 3s 13ms/step - loss: 0.6933 - accuracy: 0.0027 - val_loss: 0.6931 - val_accuracy: 0.0020
Epoch 8/10
235/235 [=====] - 3s 13ms/step - loss: 0.6931 - accuracy: 0.0027 - val_loss: 0.6929 - val_accuracy: 0.0021
Epoch 9/10
235/235 [=====] - 3s 14ms/step - loss: 0.6930 - accuracy: 0.0028 - val_loss: 0.6928 - val_accuracy: 0.0021
Epoch 10/10
235/235 [=====] - 3s 13ms/step - loss: 0.6928 - accuracy: 0.0028 - val_loss: 0.6926 - val_accuracy: 0.0021
```

