Assignment – 8

Name: Pranavya Chanamolu
ID :700739974
Github Link: https://github.com/pxc99740/Neural-network-Assignment-8.git
Video Link: https://drive.google.com/drive/folders/1ECMgDmn4io0Igj-XaDGi16YIzdzD5a-K

1. Add one more hidden layer to 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 data before reconstructions using Matplotlib.

```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')
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))
```

Python

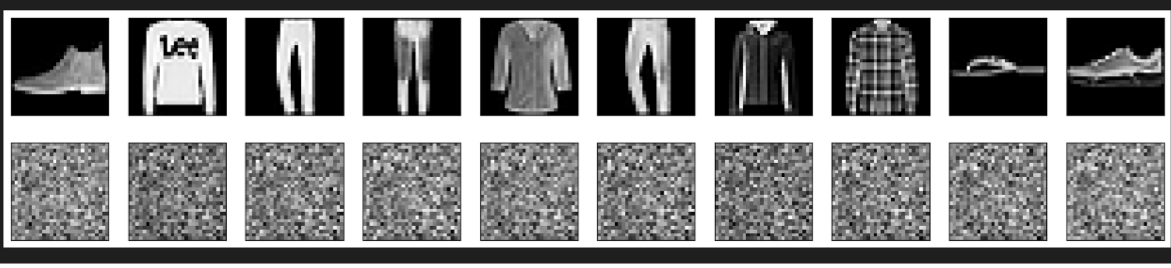
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 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 [==============================] - 14s 42ms/step - loss: 0.6940 - val_loss: 0.6939
Epoch 2/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6938 - val_loss: 0.6936
Epoch 3/5
235/235 [==============================] - 5s 22ms/step - loss: 0.6935 - val_loss: 0.6933
Epoch 4/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6932 - val_loss: 0.6931
Epoch 5/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6929 - val_loss: 0.6928

<keras.src.callbacks.History at 0x79827b8d7910>
```

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

# Define the encoder dimension
encoding_dim = 32

# Define the input placeholder
input_img = Input(shape=(784,))

# Define the first hidden layer
hidden_1 = Dense(256, activation='relu')(input_img)

# Define the second hidden layer
encoded = Dense(encoding_dim, activation='relu')(hidden_1)

# Define the first hidden layer of the decoder
hidden_2 = Dense(256, activation='relu')(encoded)

# Define the output layer
decoded = Dense(784, activation='sigmoid')(hidden_2)

# Define the autoencoder model
autoencoder = Model(input_img, decoded)

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

# Load the fashion MNIST dataset
(x_train, _), (x_test, _) = fashion_mnist.load_data()
```

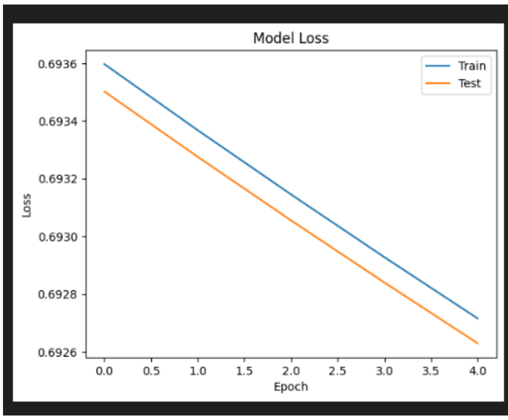
```python
# Normalize the data and flatten the images
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Train the autoencoder
history = autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# Make predictions on the test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize one of the reconstructed images
n = 10  # number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original test image
    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 reconstructed test image
    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()
```

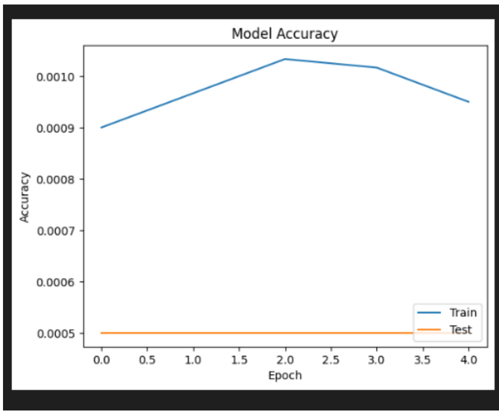
```python
# Plot the loss and accuracy over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

```
Epoch 1/5
235/235 [==============================] - 10s 38ms/step - loss: 0.6936 - accuracy: 9.0000e-04 - val_loss: 0.6935 - val_accuracy: 5.0000e-04
Epoch 2/5
235/235 [==============================] - 6s 25ms/step - loss: 0.6934 - accuracy: 9.6667e-04 - val_loss: 0.6933 - val_accuracy: 5.0000e-04
Epoch 3/5
235/235 [==============================] - 5s 21ms/step - loss: 0.6931 - accuracy: 0.0010 - val_loss: 0.6931 - val_accuracy: 5.0000e-04
Epoch 4/5
235/235 [==============================] - 7s 30ms/step - loss: 0.6929 - accuracy: 0.0010 - val_loss: 0.6928 - val_accuracy: 5.0000e-04
Epoch 5/5
235/235 [==============================] - 6s 26ms/step - loss: 0.6927 - accuracy: 9.5000e-04 - val_loss: 0.6926 - val_accuracy: 5.0000e-04
313/313 [==============================] - 1s 3ms/step
```



Output:





3. Repeat the question 2 on the denoising autoencoder.
4. Plot loss and accuracy using the history object.

```
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')
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))
```

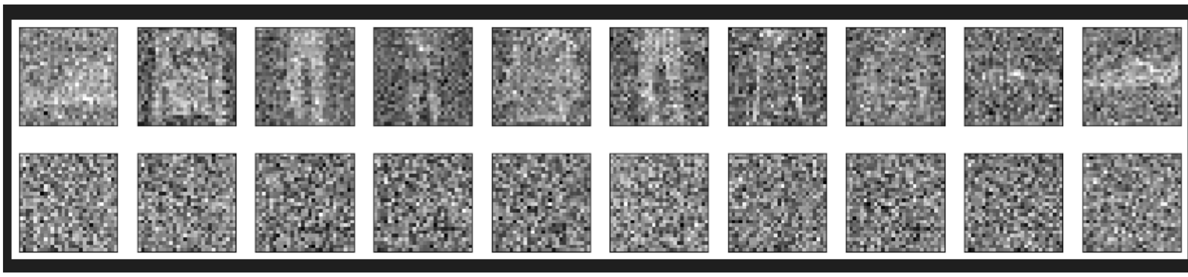
```
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6951 - val_loss: 0.6950
Epoch 2/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6949 - val_loss: 0.6947
Epoch 3/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6947 - val_loss: 0.6945
Epoch 4/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6945 - val_loss: 0.6943
Epoch 5/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6943 - val_loss: 0.6941
Epoch 6/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6941 - val_loss: 0.6939
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6939 - val_loss: 0.6938
Epoch 8/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6937 - val_loss: 0.6936
Epoch 9/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6935 - val_loss: 0.6934
Epoch 10/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6933 - val_loss: 0.6932

<keras.src.callbacks.History at 0x798278541e10>
```

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

# Define the encoder dimension
encoding_dim = 32

# Define the input placeholder
input_img = Input(shape=(784,))

# Define the encoder layer
encoded = Dense(encoding_dim, activation='relu')(input_img)

# Define the decoder layer
decoded = Dense(784, activation='sigmoid')(encoded)

# Define the autoencoder model
autoencoder = Model(input_img, decoded)

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

# Load the fashion MNIST dataset
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize the data and flatten the images
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:])))

# Add noise to the test data
noise_factor = 0.5
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
```

```python
# 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))

# Generate reconstructed images from the noisy test data
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualize one of the noisy test images
plt.figure(figsize=(20, 4))
n = 10
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

# Visualize one of the reconstructed test images
for i in range(n):
    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()

# Plot the loss and accuracy over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.7015 - accuracy: 9.8333e-04 - val_loss: 0.7013 - val_accuracy: 0.0015
Epoch 2/10
235/235 [==============================] - 2s 10ms/step - loss: 0.7011 - accuracy: 9.6667e-04 - val_loss: 0.7009 - val_accuracy: 0.0015
Epoch 3/10
235/235 [==============================] - 2s 10ms/step - loss: 0.7008 - accuracy: 9.8333e-04 - val_loss: 0.7006 - val_accuracy: 0.0015
Epoch 4/10
235/235 [==============================] - 7s 28ms/step - loss: 0.7004 - accuracy: 0.0010 - val_loss: 0.7002 - val_accuracy: 0.0015
Epoch 5/10
235/235 [==============================] - 2s 10ms/step - loss: 0.7000 - accuracy: 0.0011 - val_loss: 0.6999 - val_accuracy: 0.0015
Epoch 6/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6997 - accuracy: 0.0011 - val_loss: 0.6995 - val_accuracy: 0.0016
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6994 - accuracy: 0.0011 - val_loss: 0.6992 - val_accuracy: 0.0015
Epoch 8/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6991 - accuracy: 0.0011 - val_loss: 0.6989 - val_accuracy: 0.0015
Epoch 9/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6988 - accuracy: 0.0012 - val_loss: 0.6986 - val_accuracy: 0.0015
Epoch 10/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6985 - accuracy: 0.0012 - val_loss: 0.6984 - val_accuracy: 0.0015
313/313 [==============================] - 1s 2ms/step
```



<u>Output:</u>