Spring 2024: CS5720

Neural Networks & Deep Learning – Assignment -8

NAME:NANDIMANDALAM VENKATA VINAY VARMA

STUDENT ID:700745193

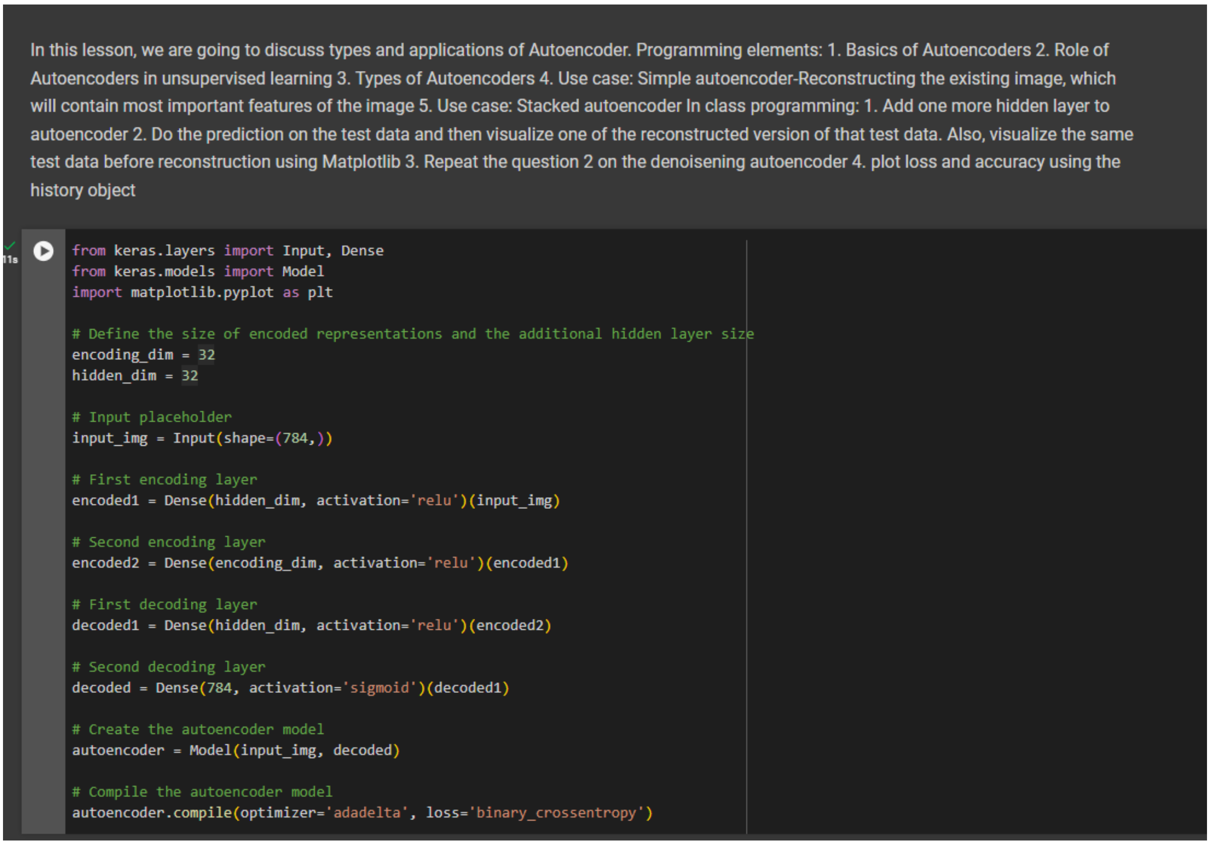
GitHub Link:

https://github.com/venkatavinayvarma/NeuralNetworks_Assignment8.git

GitHub Link:

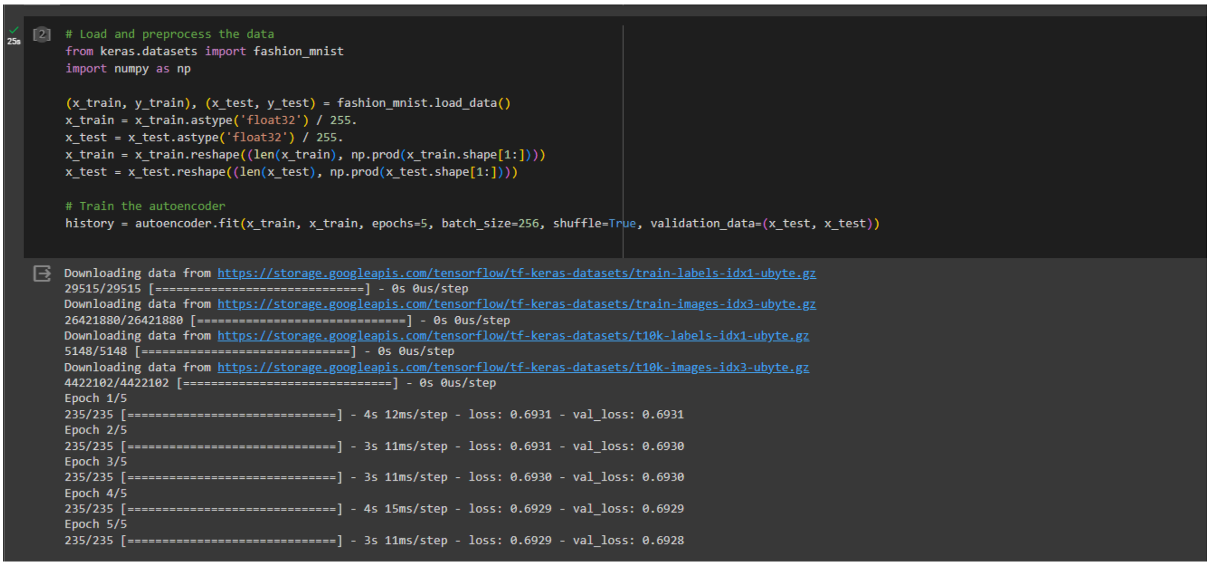
Video Link: https://drive.google.com/drive/folders/1B0X1eq38WGeVXGh2-kyPpdM1e71SFWM5?usp=sharing

Lesson Overview: In this lesson, we are going to discuss types and applications of Autoencoder. Programming elements: 1. Basics of Autoencoders 2. Role of Autoencoders in unsupervised learning 3. Types of Autoencoders 4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image 5. Use case: Stacked autoencoder In class programming: 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 test data before reconstruction using Matplotlib 3. Repeat the question 2 on the denoisening autoencoder 4. plot loss and accuracy using the history object.



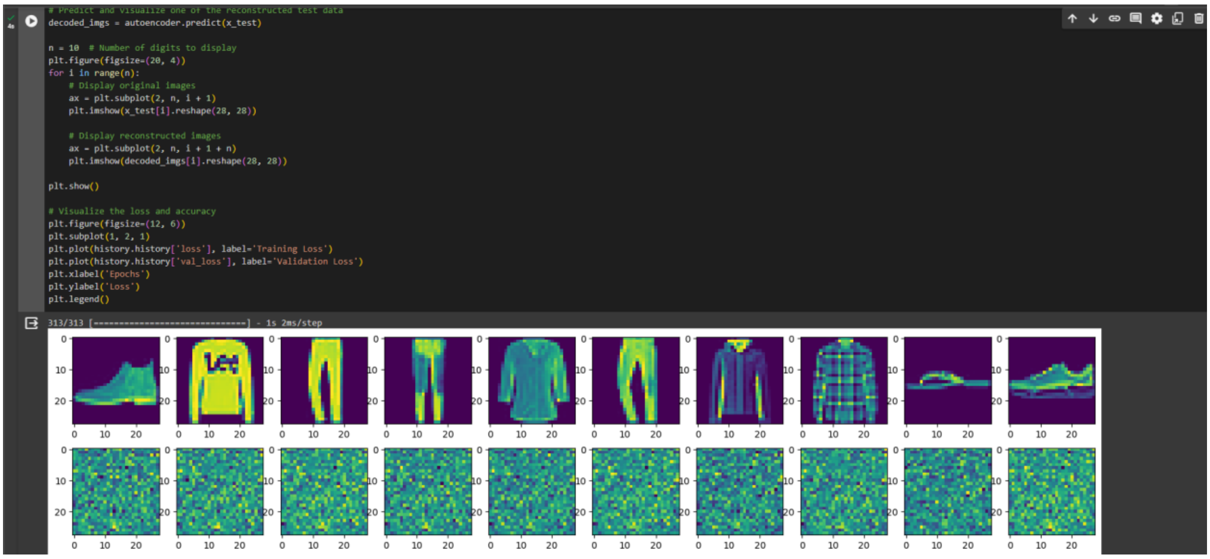
In this lesson, we are going to discuss types and applications of Autoencoder. Programming elements: 1. Basics of Autoencoders 2. Role of Autoencoders in unsupervised learning 3. Types of Autoencoders 4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image 5. Use case: Stacked autoencoder In class programming: 1. Add one more hidden layer to autoencoder 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 3. Repeat the question 2 on the denoisening autoencoder 4. plot loss and accuracy using the history object

```
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt

# Define the size of encoded representations and the additional hidden layer size
encoding_dim = 32
hidden_dim = 32

# Input placeholder
input_img = Input(shape=(784,))

# First encoding layer
encoded1 = Dense(hidden_dim, activation='relu')(input_img)

# Second encoding layer
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

# First decoding layer
decoded1 = Dense(hidden_dim, activation='relu')(encoded2)

# Second decoding layer
decoded = Dense(784, activation='sigmoid')(decoded1)

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

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



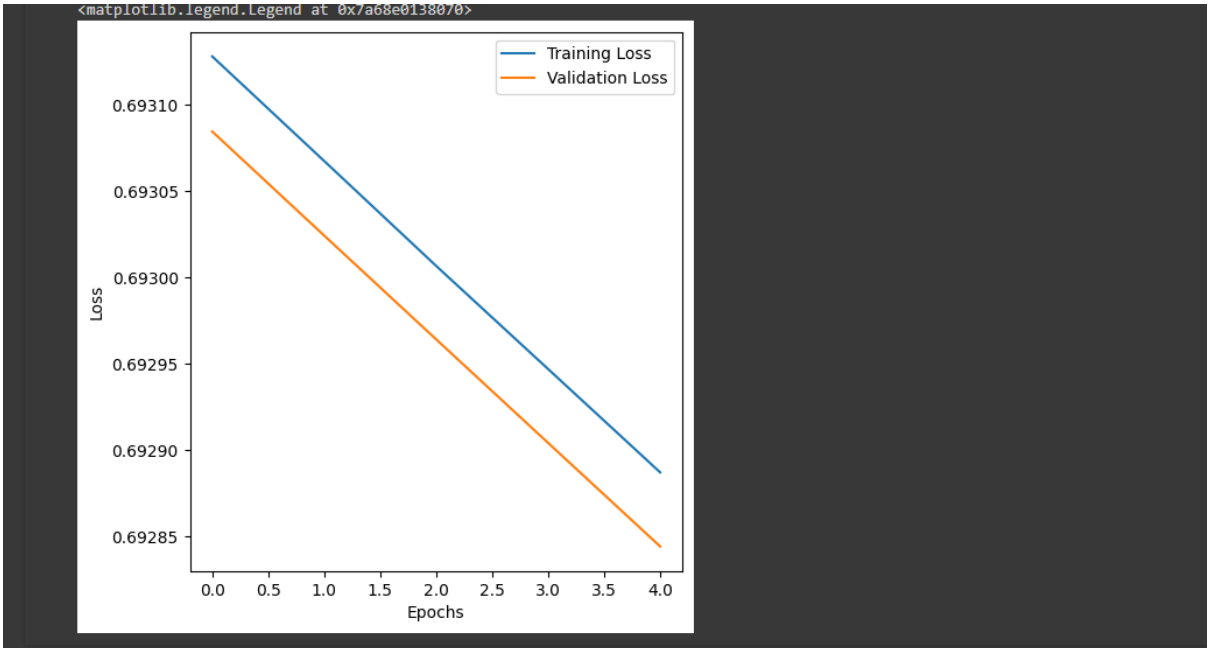
```
# Load and preprocess the data
from keras.datasets import fashion_mnist
import numpy as np

(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:])))

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

```
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 [==============================] - 4s 12ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 2/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6931 - val_loss: 0.6930
Epoch 3/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6930 - val_loss: 0.6930
Epoch 4/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6929 - val_loss: 0.6929
Epoch 5/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6929 - val_loss: 0.6928
```

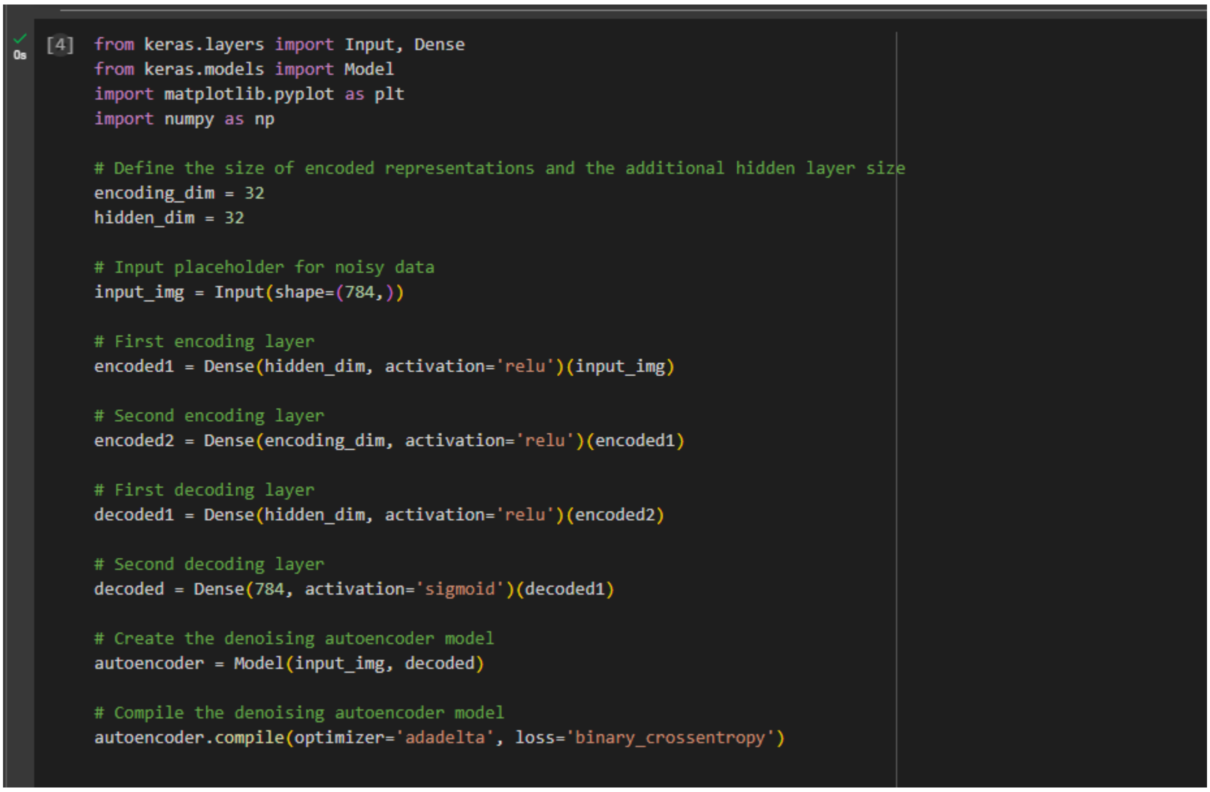


```
# Predict and visualize one of the reconstructed test data
decoded_imgs = autoencoder.predict(x_test)

n = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))

    # Display reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))

plt.show()

# Visualize the loss and accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

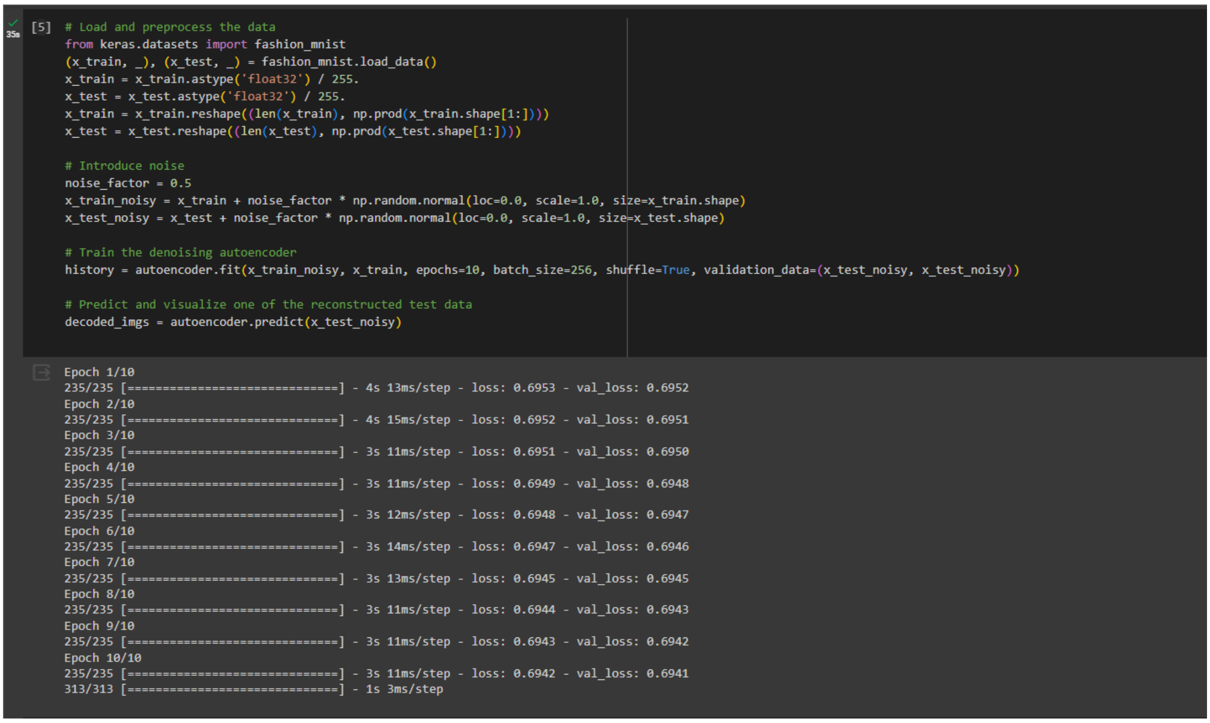
```
313/313 [==============================] - 1s 2ms/step
```



```
<matplotlib.legend.Legend at 0x7a68e0138070>
```

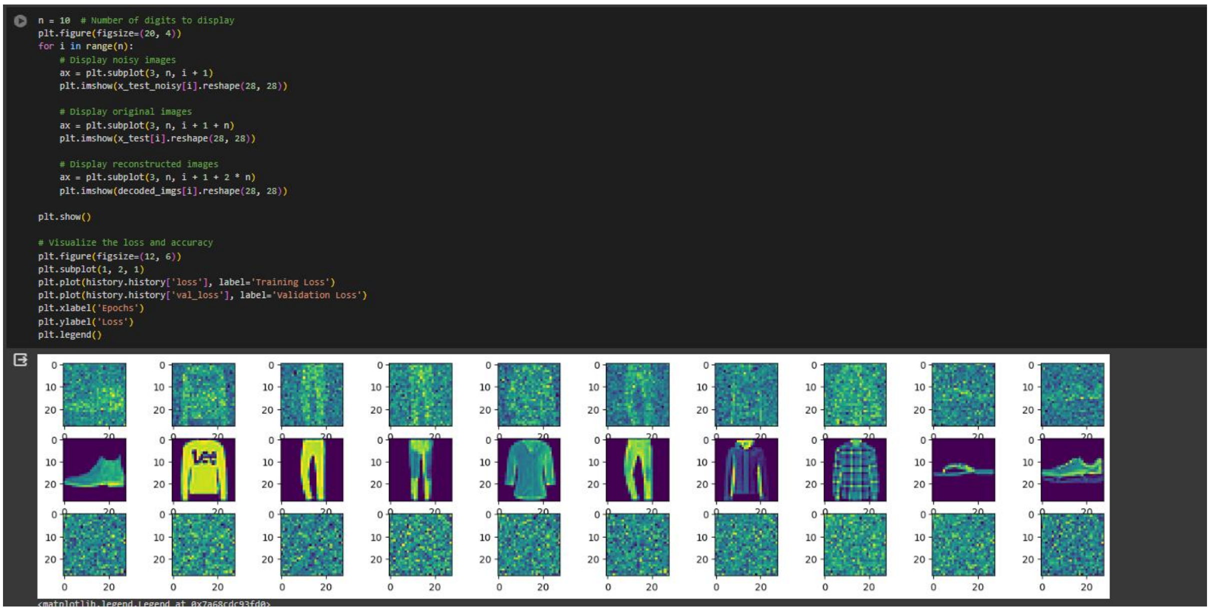


```python
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt
import numpy as np

# Define the size of encoded representations and the additional hidden layer size
encoding_dim = 32
hidden_dim = 32

# Input placeholder for noisy data
input_img = Input(shape=(784,))

# First encoding layer
encoded1 = Dense(hidden_dim, activation='relu')(input_img)

# Second encoding layer
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

# First decoding layer
decoded1 = Dense(hidden_dim, activation='relu')(encoded2)

# Second decoding layer
decoded = Dense(784, activation='sigmoid')(decoded1)

# Create the denoising autoencoder model
autoencoder = Model(input_img, decoded)

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

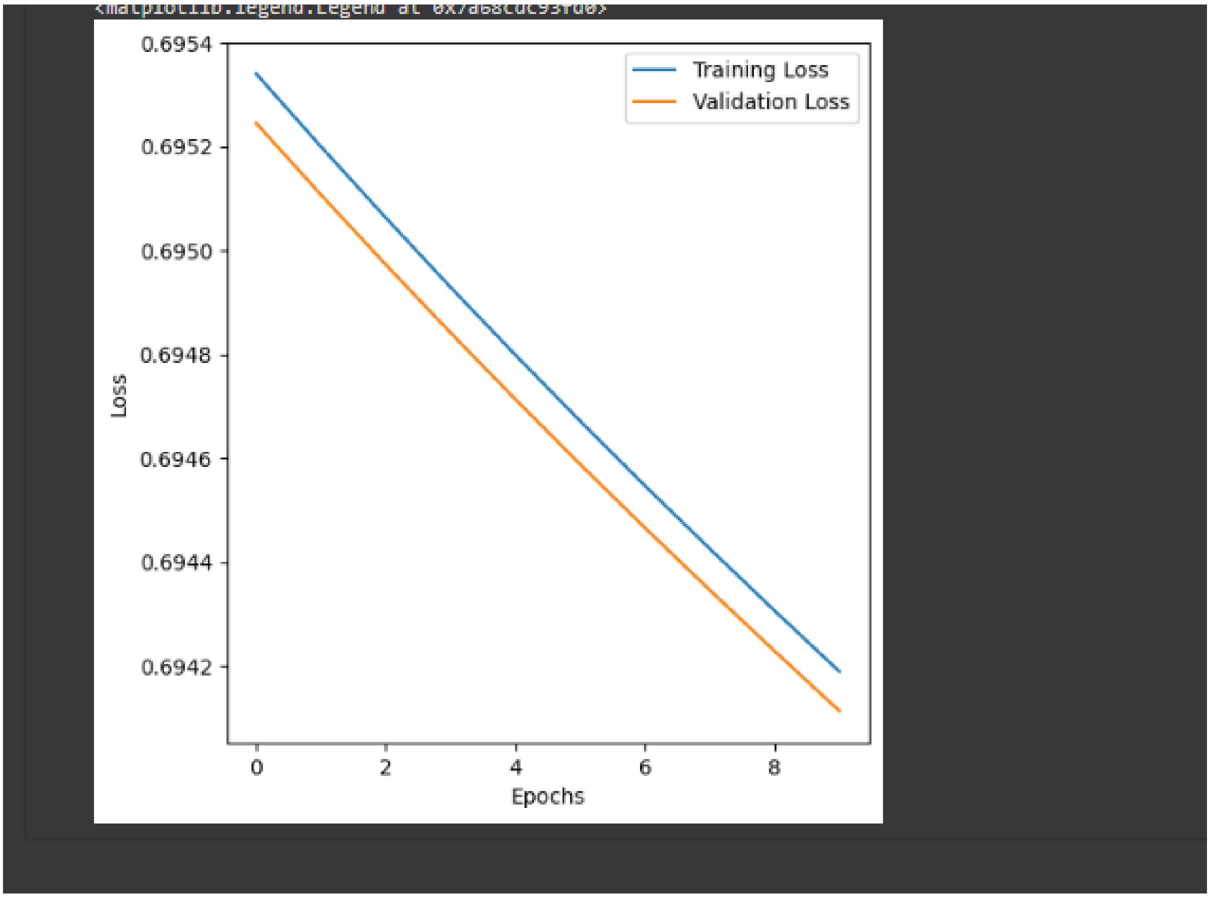


```python
# Load and preprocess the data
from keras.datasets import fashion_mnist
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Introduce noise
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
history = autoencoder.fit(x_train_noisy, x_train, epochs=10, batch_size=256, shuffle=True, validation_data=(x_test_noisy, x_test_noisy))

# Predict and visualize one of the reconstructed test data
decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
Epoch 1/10
235/235 [==============================] - 4s 13ms/step - loss: 0.6953 - val_loss: 0.6952
Epoch 2/10
235/235 [==============================] - 4s 15ms/step - loss: 0.6952 - val_loss: 0.6951
Epoch 3/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6951 - val_loss: 0.6950
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 5/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6948 - val_loss: 0.6947
Epoch 6/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6947 - val_loss: 0.6946
Epoch 7/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6945 - val_loss: 0.6945
Epoch 8/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6944 - val_loss: 0.6943
Epoch 9/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6943 - val_loss: 0.6942
Epoch 10/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6942 - val_loss: 0.6941
313/313 [==============================] - 1s 3ms/step
```



```
n = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display noisy images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))

    # Display original images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test[i].reshape(28, 28))

    # Display reconstructed images
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))

plt.show()

# Visualize the loss and accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```



```
<matplotlib.legend.Legend at 0x7a68cdc93fd0>
```