Spring 2024: CS5720 Neural Networks & Deep Learning - ICP-8

Assignment- 8

Name: Afroz Mohammad Student Id : 700758012

GitHub link: https://github.com/Afrozmohammad19/Assignment8

Video Link: https://drive.google.com/file/d/1V3J0YkXjzYn3nEhQHDYBGjfMzmyjtneh/view?usp=sharing

Programming elements:

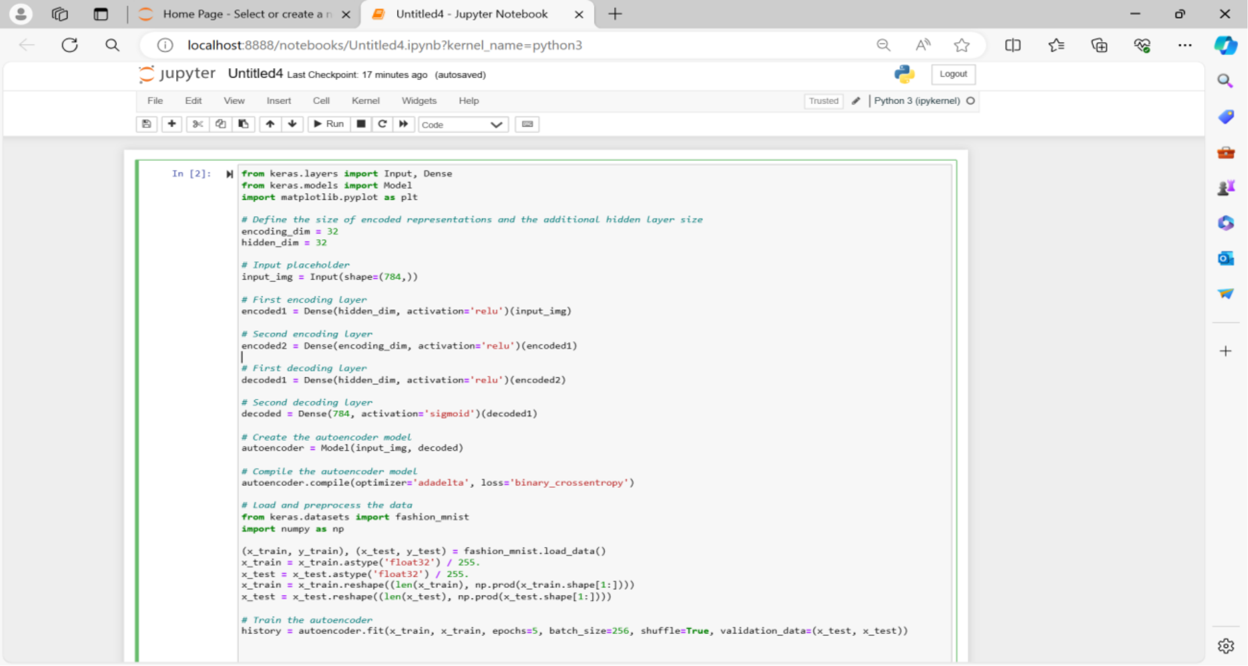
1. Basics of Autoencoders

2. Role of Autoencoders in unsupervised learning

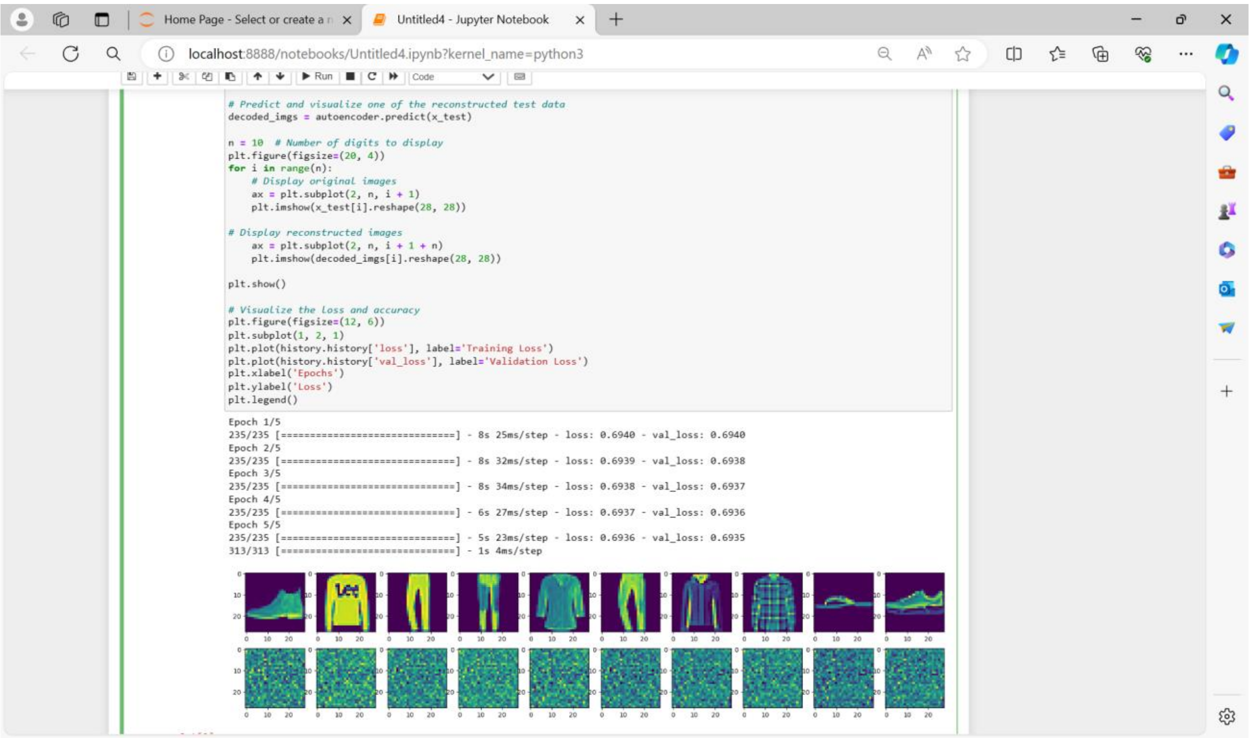
3. Types of Autoencoders

4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image

5. Use case: Stacked autoencoder


Home Page - Select or create a n
Untitled4 - Jupyter Notebook
localhost:8888/notebooks/Untitled4.ipynb?kernel_name=python3
jupyter Untitled4 Last Checkpoint: 17 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [2]:

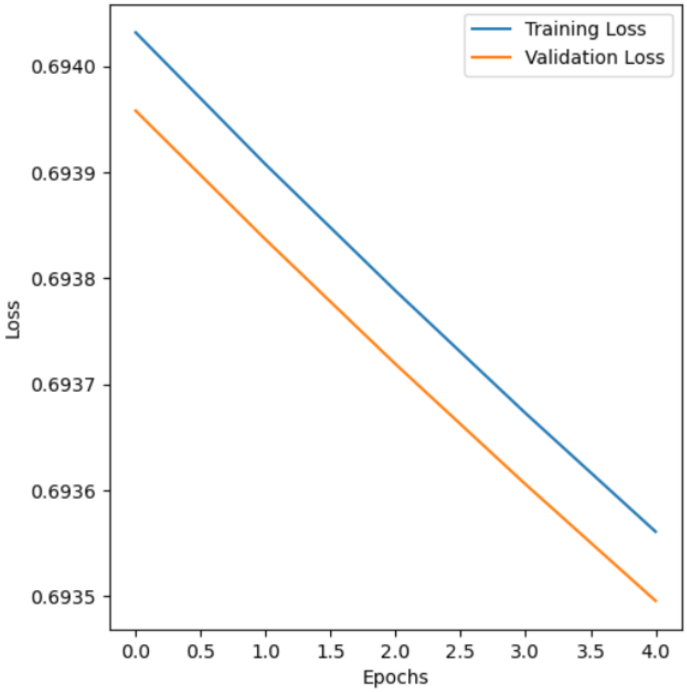
```
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt

# 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))
```


Home Page - Select or create a n
Untitled4 - Jupyter Notebook
localhost:8888/notebooks/Untitled4.ipynb?kernel_name=python3
Run
Code
0
10
20

```
# 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()
```

```
Epoch 1/5
235/235 [==============================] - 8s 25ms/step - loss: 0.6940 - val_loss: 0.6940
Epoch 2/5
235/235 [==============================] - 8s 32ms/step - loss: 0.6939 - val_loss: 0.6938
Epoch 3/5
235/235 [==============================] - 8s 34ms/step - loss: 0.6938 - val_loss: 0.6937
Epoch 4/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6937 - val_loss: 0.6936
Epoch 5/5
235/235 [==============================] - 5s 23ms/step - loss: 0.6936 - val_loss: 0.6935
313/313 [==============================] - 1s 4ms/step
```

```
Out[2]: <matplotlib.legend.Legend at 0x28092191450>
```



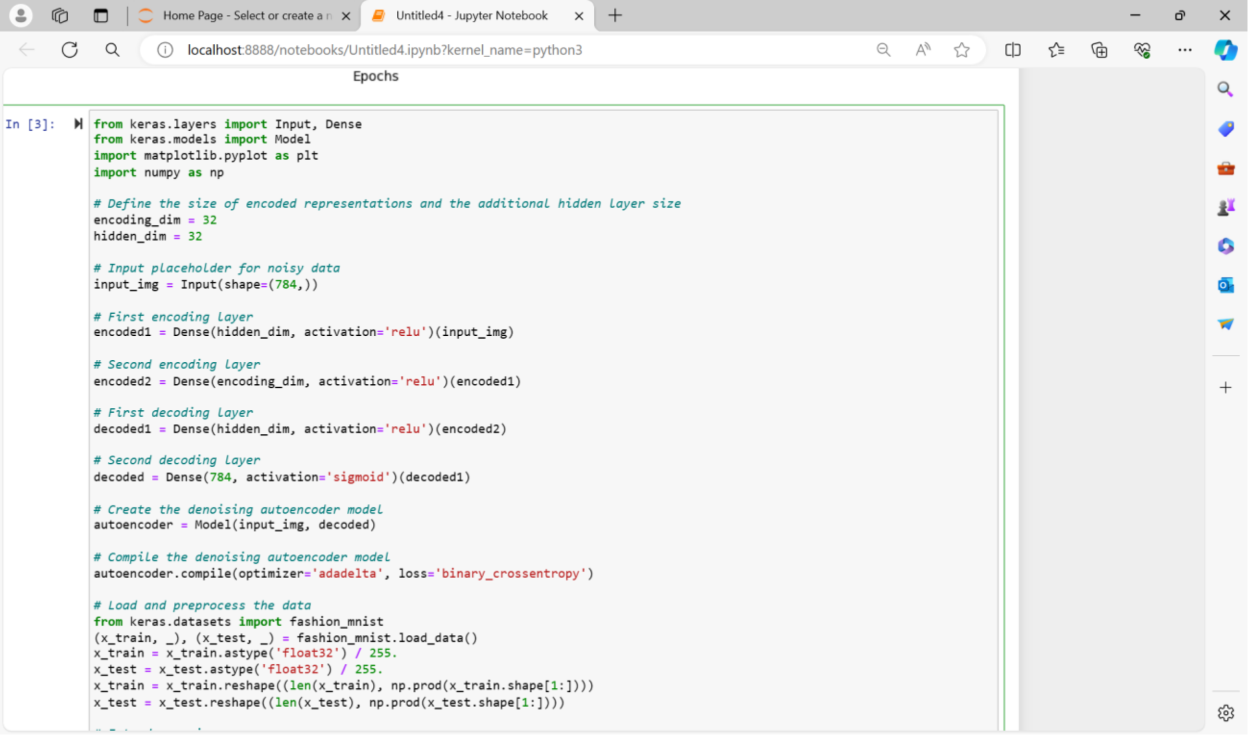
In class programming:

1. Add one more hidden layer to autoencoder

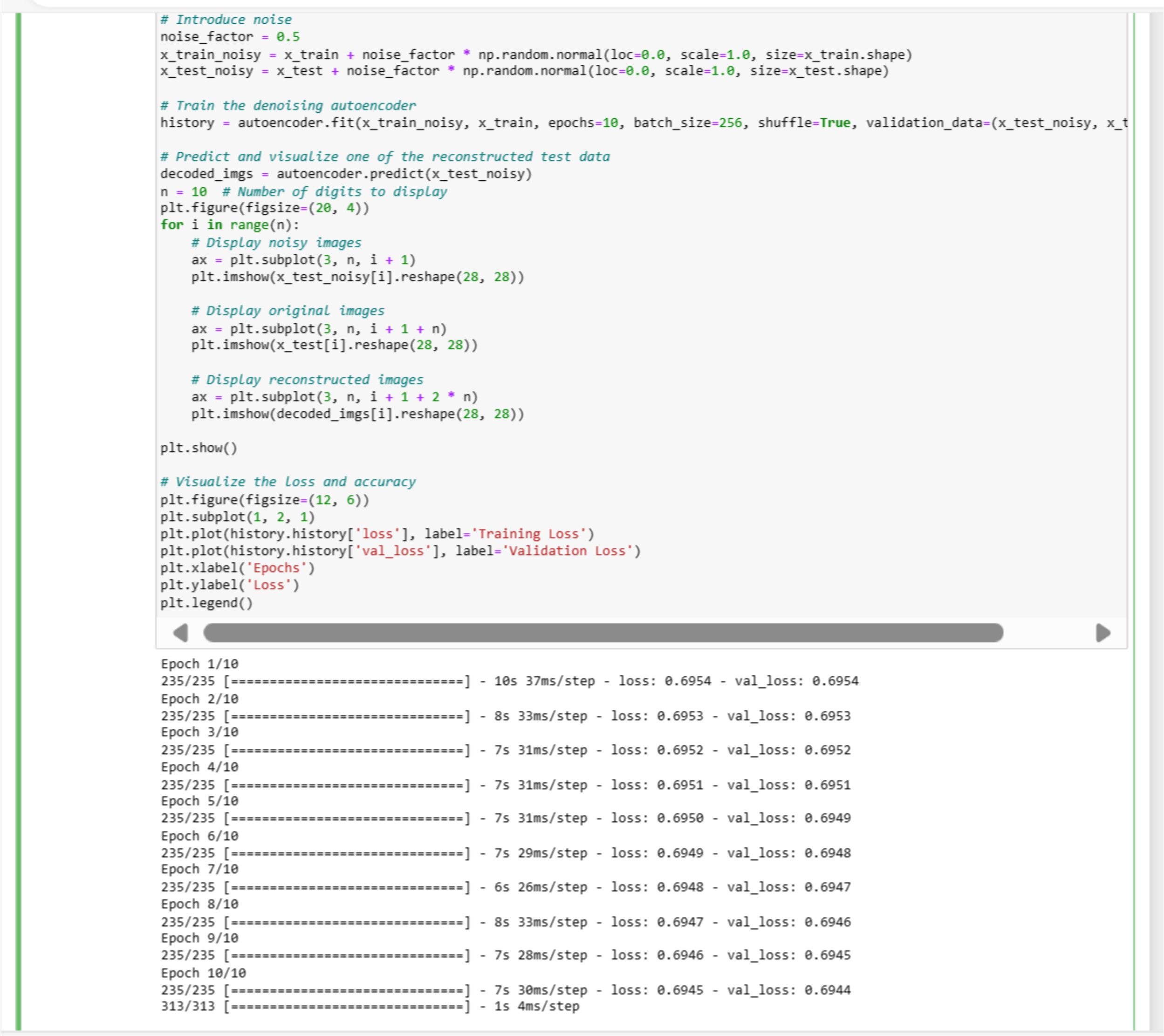
2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

3. Repeat the question 2 on the denoisening autoencoder

4. plot loss and accuracy using the history object

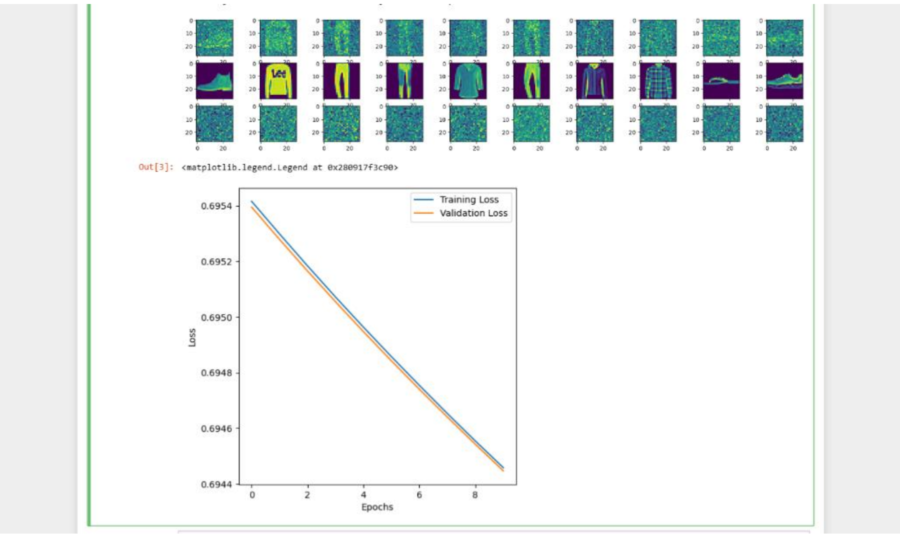

Home Page - Select or create a n
Untitled4 - Jupyter Notebook
localhost:8888/notebooks/Untitled4.ipynb?kernel_name=python3
Epochs
In [3]:

```
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')

# 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_t

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

```
Epoch 1/10
235/235 [==============================] - 10s 37ms/step - loss: 0.6954 - val_loss: 0.6954
Epoch 2/10
235/235 [==============================] - 8s 33ms/step - loss: 0.6953 - val_loss: 0.6953
Epoch 3/10
235/235 [==============================] - 7s 31ms/step - loss: 0.6952 - val_loss: 0.6952
Epoch 4/10
235/235 [==============================] - 7s 31ms/step - loss: 0.6951 - val_loss: 0.6951
Epoch 5/10
235/235 [==============================] - 7s 31ms/step - loss: 0.6950 - val_loss: 0.6949
Epoch 6/10
235/235 [==============================] - 7s 29ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 7/10
235/235 [==============================] - 6s 26ms/step - loss: 0.6948 - val_loss: 0.6947
Epoch 8/10
235/235 [==============================] - 8s 33ms/step - loss: 0.6947 - val_loss: 0.6946
Epoch 9/10
235/235 [==============================] - 7s 28ms/step - loss: 0.6946 - val_loss: 0.6945
Epoch 10/10
235/235 [==============================] - 7s 30ms/step - loss: 0.6945 - val_loss: 0.6944
313/313 [==============================] - 1s 4ms/step
```



```
Out[3]: <matplotlib.legend.Legend at 0x280917f3c90>
```