Spring 2024: CS5720 Neural Networks & Deep Learning

Assignment-6

NAME: Lakkireddy Sriram Reddy

STUDENT ID:700758340

Github link: https://github.com/sriram7040/Neural-network-and-deep-learning/tree/main/Week8

Video link:

https://drive.google.com/file/d/1h4GTMU3MpE5Y52Vo34GWKdVQdtorUqU7/view?usp=sharing

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

```
[1] 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
```

Load data

```
[2] (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:])))
```

1. Autoencoder with extra hidden layers

```python
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(32, activation='relu')(encoded)  # bottleneck

decoded = Dense(128, activation='relu')(encoded)
decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

Train autoencoder

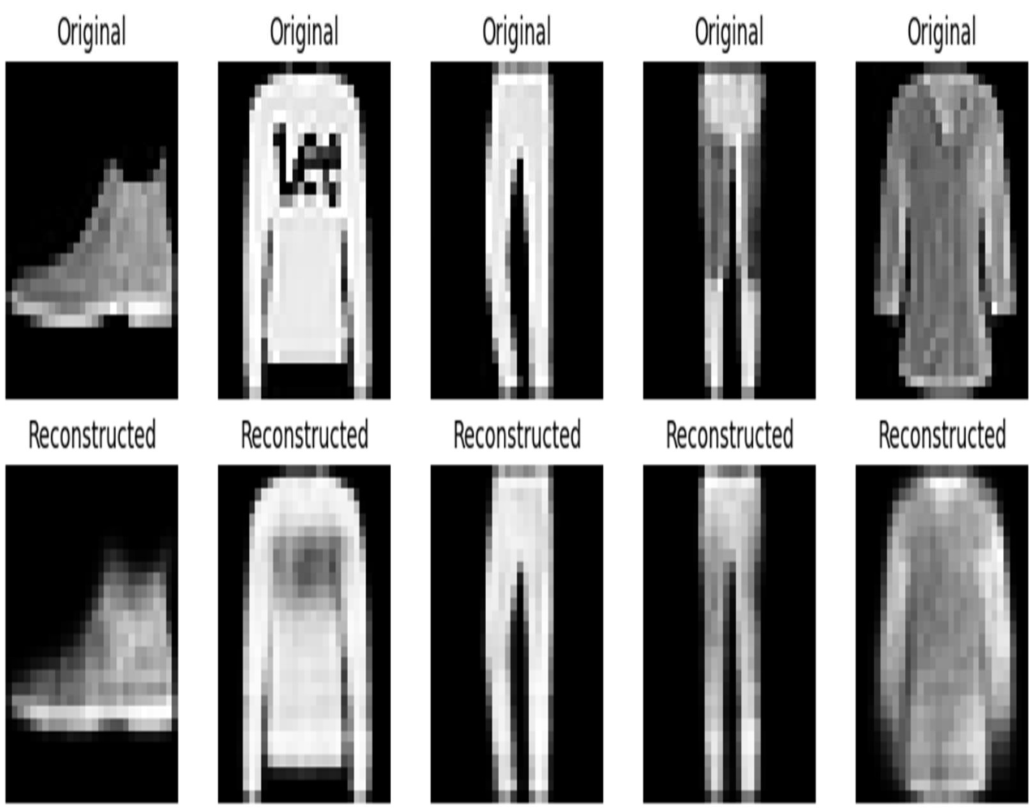
```python
[4] history = autoencoder.fit(x_train, x_train,
                              epochs=10,
                              batch_size=256,
                              shuffle=True,
                              validation_data=(x_test, x_test))
```

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step - loss: 0.4564 - val_loss: 0.3146
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - loss: 0.3088 - val_loss: 0.3022
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 22ms/step - loss: 0.2990 - val_loss: 0.2963
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - loss: 0.2934 - val_loss: 0.2941
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.2903 - val_loss: 0.2903
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.2877 - val_loss: 0.2884
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.2854 - val_loss: 0.2867
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - loss: 0.2831 - val_loss: 0.2852
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.2828 - val_loss: 0.2843
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 22ms/step - loss: 0.2821 - val_loss: 0.2832
```

2. Predict & visualize

```python
decoded_imgs = autoencoder.predict(x_test)

n = 5
plt.figure(figsize=(10, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

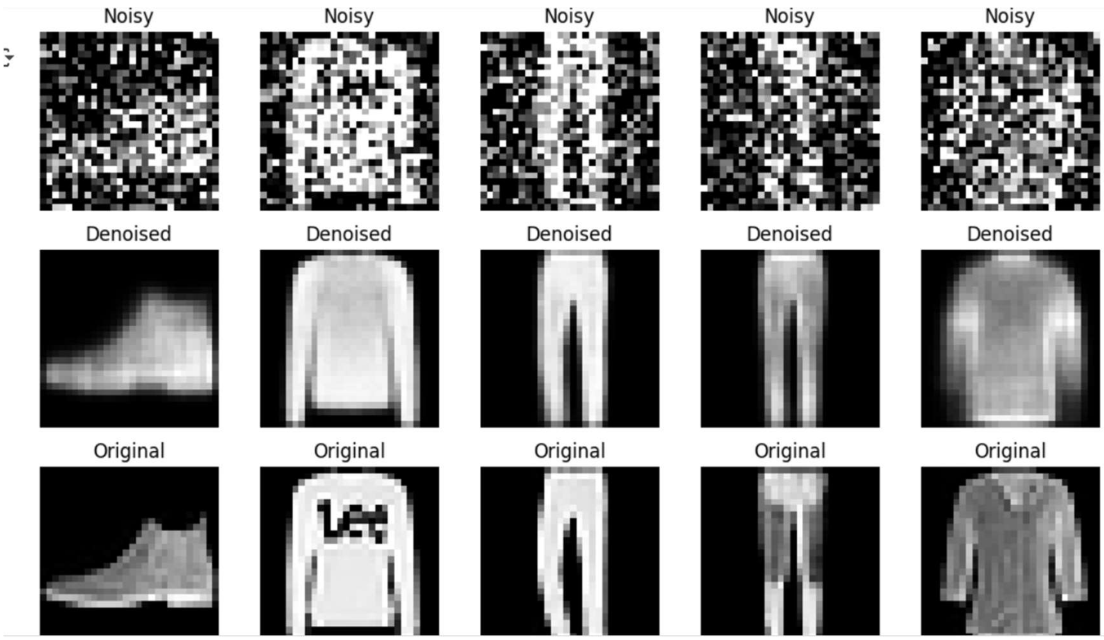
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



3. Denoising Autoencoder

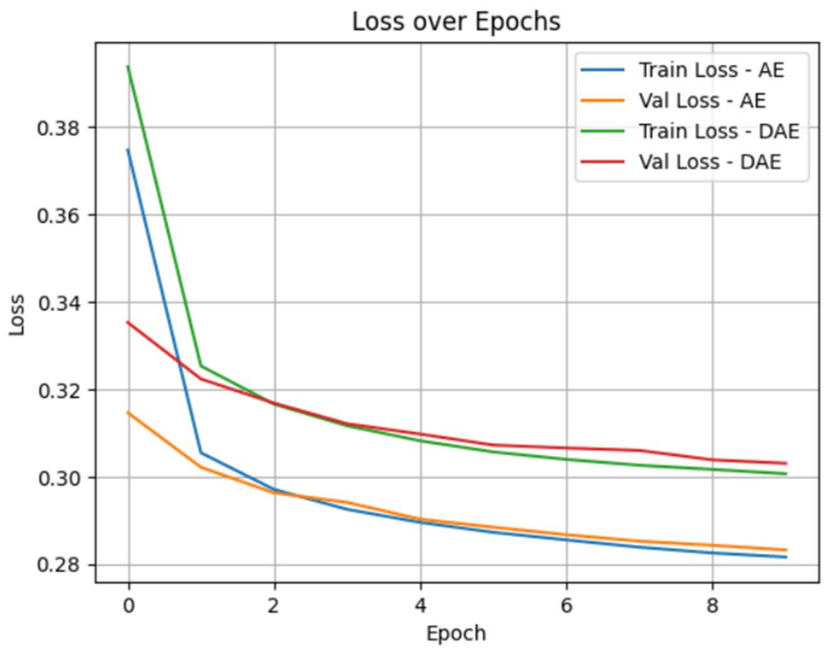
```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.sh
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Create same architecture for denoising autoencoder
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(32, activation='relu')(encoded)

decoded = Dense(128, activation='relu')(encoded)
decoded = Dense(784, activation='sigmoid')(decoded)

denoise_autoencoder = Model(input_img, decoded)
denoise_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train denoising autoencoder
denoise_history = denoise_autoencoder.fit(x_train_noisy, x_train,
                                          epochs=10,
                                          batch_size=256,
                                          shuffle=True,
                                          validation_data=(x_test_noisy, x_test))

# Predict and visualize denoising output
denoised_imgs = denoise_autoencoder.predict(x_test_noisy)
```

```
# Predict and visualize denoising output
denoised_imgs = denoise_autoencoder.predict(x_test_noisy)

plt.figure(figsize=(10, 6))
for i in range(n):
    # Noisy input
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28), cmap='gray')
    plt.title("Noisy")
    plt.axis('off')

    # Denoised
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(denoised_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Denoised")
    plt.axis('off')

    # Ground truth
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')
plt.tight_layout()
plt.show()
```



4. Plot training/validation los

```
plt.plot(history.history['loss'], label='Train Loss - AE')
plt.plot(history.history['val_loss'], label='Val Loss - AE')
plt.plot(denoise_history.history['loss'], label='Train Loss - DAE')
plt.plot(denoise_history.history['val_loss'], label='Val Loss - DAE')
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```



Loss over Epochs
Train Loss - AE
Val Loss - AE
Train Loss - DAE
Val Loss - DAE
0.38
0.36
0.34
0.32
0.30
0.28
Loss
0
2
4
6
8
Epoch