Git-hub Link: [Sabitha-C/Neural-networks (github.com)]

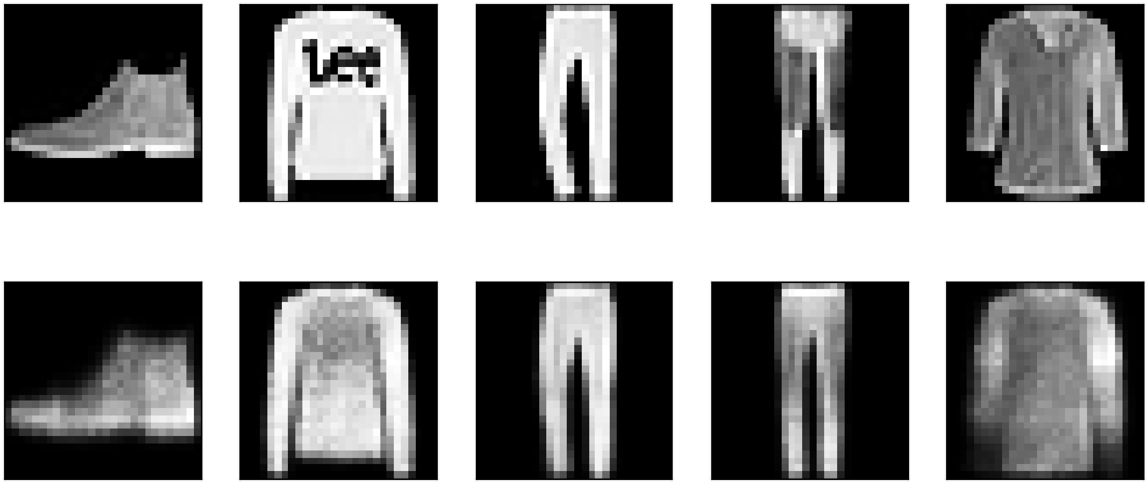
Code:

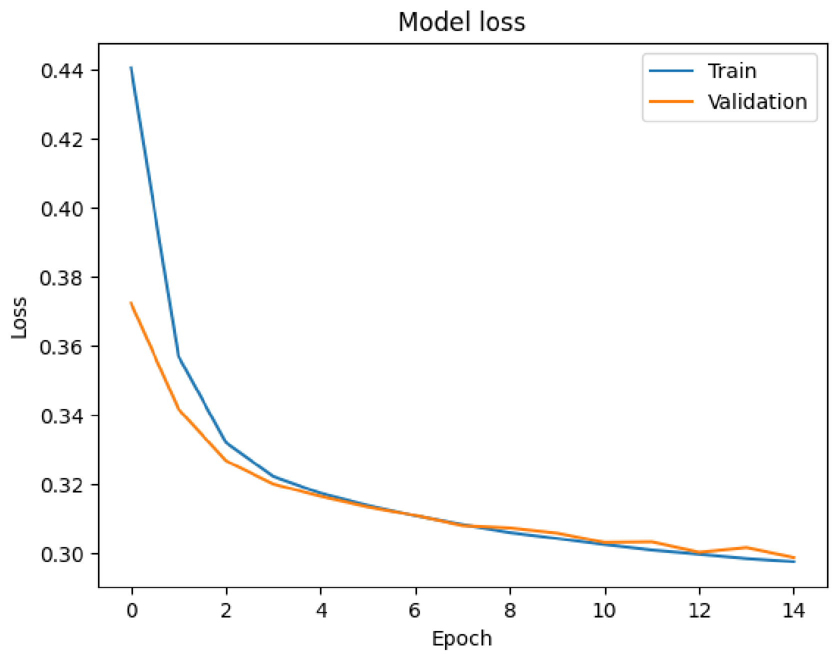
```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt
from keras.optimizers import Adadelta

# Load Fashion MNIST data
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize and reshape 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:])))

# Define hyperparameters
encoding_dim = 32
input_dim = x_train.shape[1]
learning_rate = 1.0
batch_size = 128
epochs = 15

# Define the autoencoder model
input_img = Input(shape=(input_dim,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(encoding_dim, activation='relu')(encoded)

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

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

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

# Predict on test data
decoded_imgs = autoencoder.predict(x_test)

# Plotting function for original and reconstructed images
def plot_images(original, reconstructed, num_images=5):
    n = num_images
    plt.figure(figsize=(10, 4.5))
    for i in range(n):
        # Original images
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(original[i].reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        # Reconstructed images
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(reconstructed[i].reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()

# Visualize original and reconstructed images
plot_images(x_test, decoded_imgs)

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

Output:

```
Epoch 1/15
469/469 [==============================] - 3s 4ms/step - loss: 0.4404 - val_loss: 0.3723
Epoch 2/15
469/469 [==============================] - 3s 7ms/step - loss: 0.3570 - val_loss: 0.3417
Epoch 3/15
469/469 [==============================] - 3s 6ms/step - loss: 0.3320 - val_loss: 0.3267
Epoch 4/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3222 - val_loss: 0.3200
Epoch 5/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3173 - val_loss: 0.3164
Epoch 6/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3138 - val_loss: 0.3134
Epoch 7/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3108 - val_loss: 0.3109
Epoch 8/15
469/469 [==============================] - 3s 5ms/step - loss: 0.3082 - val_loss: 0.3079
Epoch 9/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3059 - val_loss: 0.3072
Epoch 10/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3042 - val_loss: 0.3058
Epoch 11/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3024 - val_loss: 0.3031
Epoch 12/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3009 - val_loss: 0.3032
Epoch 13/15
469/469 [==============================] - 2s 4ms/step - loss: 0.2997 - val_loss: 0.3002
Epoch 14/15
469/469 [==============================] - 2s 5ms/step - loss: 0.2984 - val_loss: 0.3016
Epoch 15/15
469/469 [==============================] - 2s 4ms/step - loss: 0.2975 - val_loss: 0.2987
313/313 [==============================] - 1s 2ms/step
```





Description: I have added an additional hidden layer. After predicting on the test data, I visualized both the original images and their reconstructed versions using Matplotlib. Then, I plotted the loss and accuracy.

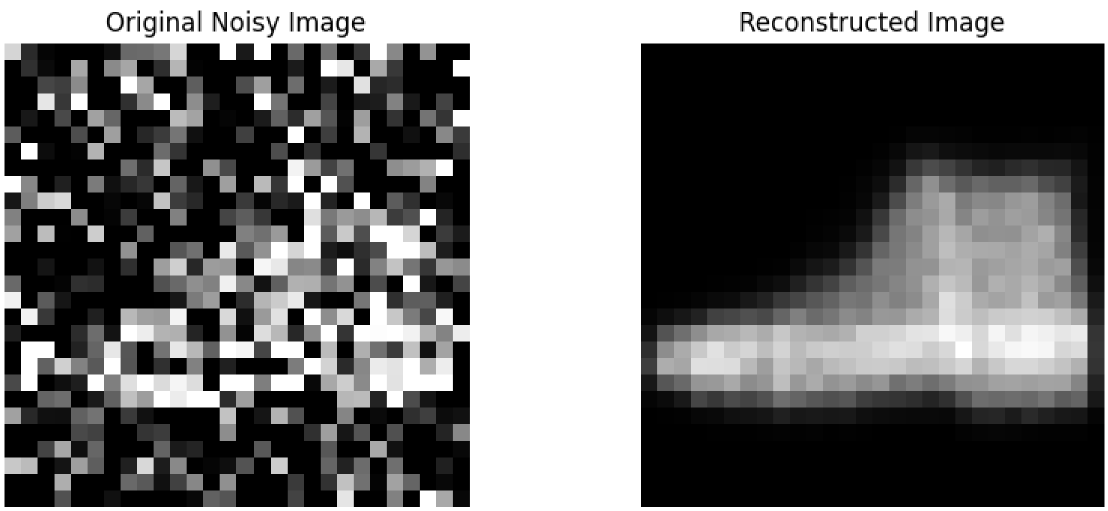
Code:

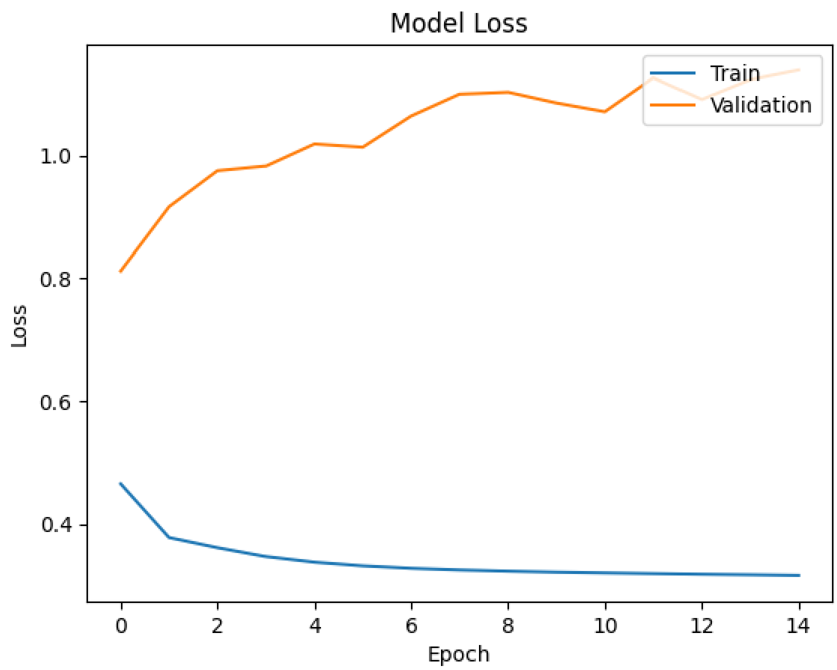
```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt
from keras.optimizers import Adadelta

# Load Fashion MNIST data
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize and reshape 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:])))

# Define hyperparameters
encoding_dim = 32
input_dim = x_train.shape[1]
noise_factor = 0.5
learning_rate = 1.0
batch_size = 128
epochs = 15

# Define the denoising autoencoder model
input_img = Input(shape=(input_dim,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(encoding_dim, activation='relu')(encoded)

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

denoising_autoencoder = Model(input_img, decoded)
optimizer = Adadelta(learning_rate=learning_rate)
denoising_autoencoder.compile(optimizer=optimizer, loss='binary_crossentropy')

# Introducing noise to the input data
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)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

```python
# Train the denoising autoencoder
history = denoising_autoencoder.fit(x_train_noisy, x_train,
                                    epochs=epochs,
                                    batch_size=batch_size,
                                    shuffle=True,
                                    validation_data=(x_test_noisy, x_test_noisy))

# Predict on test data
decoded_imgs = denoising_autoencoder.predict(x_test_noisy)

# Visualize one original and reconstructed image
plt.figure(figsize=(10, 4))
# Original image
plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[0].reshape(28, 28), cmap='gray')
plt.title('Original Noisy Image')
plt.axis('off')
# Reconstructed image
plt.subplot(1, 2, 2)
plt.imshow(decoded_imgs[0].reshape(28, 28), cmap='gray')
plt.title('Reconstructed Image')
plt.axis('off')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

Output:

```
Epoch 1/15
469/469 [==============================] - 3s 4ms/step - loss: 0.4655 - val_loss: 0.8121
Epoch 2/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3778 - val_loss: 0.9166
Epoch 3/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3613 - val_loss: 0.9753
Epoch 4/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3468 - val_loss: 0.9830
Epoch 5/15
469/469 [==============================] - 2s 5ms/step - loss: 0.3376 - val_loss: 1.0190
Epoch 6/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3316 - val_loss: 1.0138
Epoch 7/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3276 - val_loss: 1.0646
Epoch 8/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3249 - val_loss: 1.1002
Epoch 9/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3229 - val_loss: 1.1033
Epoch 10/15
469/469 [==============================] - 3s 7ms/step - loss: 0.3213 - val_loss: 1.0858
Epoch 11/15
469/469 [==============================] - 3s 6ms/step - loss: 0.3201 - val_loss: 1.0716
Epoch 12/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3190 - val_loss: 1.1265
Epoch 13/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3179 - val_loss: 1.0916
Epoch 14/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3171 - val_loss: 1.1237
Epoch 15/15
469/469 [==============================] - 2s 4ms/step - loss: 0.3161 - val_loss: 1.1392
313/313 [==============================] - 1s 2ms/step
```





Description: I have repeated the same for the denoisening autoencoder as well, predicting on the test data, visualizing both the original images and their reconstructed versions using Matplotlib. Then, I plotted the loss and accuracy.