ASSIGNMENT-8

Prem Kumar Kamma – 700756204

GITHUB LINK - https://github.com/PremKumarKamma/Assignment8

```
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))
```

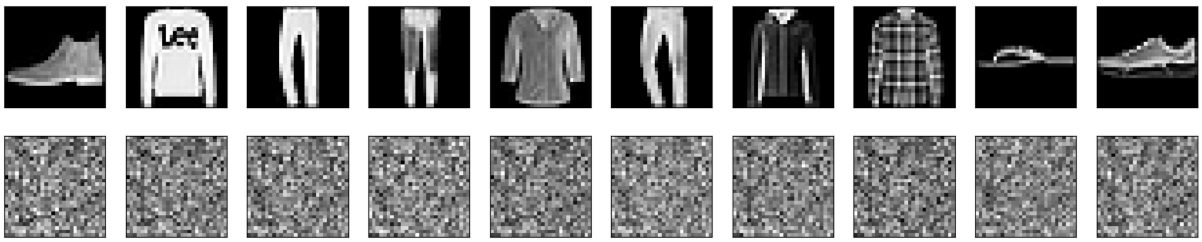
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 1s 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 [==============================] - 8s 26ms/step - loss: 0.6937 - val_loss: 0.6936
Epoch 2/5
235/235 [==============================] - 5s 20ms/step - loss: 0.6935 - val_loss: 0.6934
Epoch 3/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6933 - val_loss: 0.6932
Epoch 4/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6931 - val_loss: 0.6930
Epoch 5/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6929 - val_loss: 0.6928
<keras.src.callbacks.History at 0x7e9ed58673a0>
```

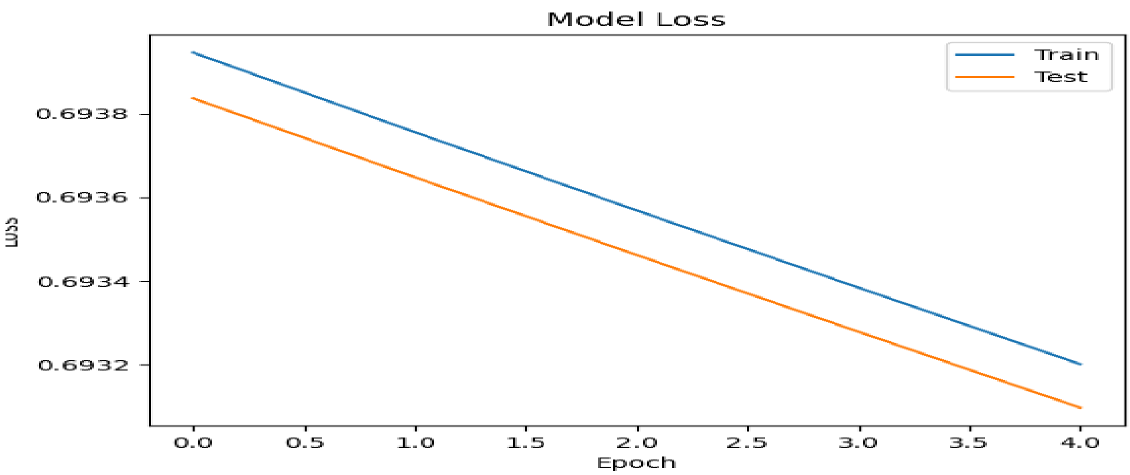
```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# 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()

# 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:])))
```

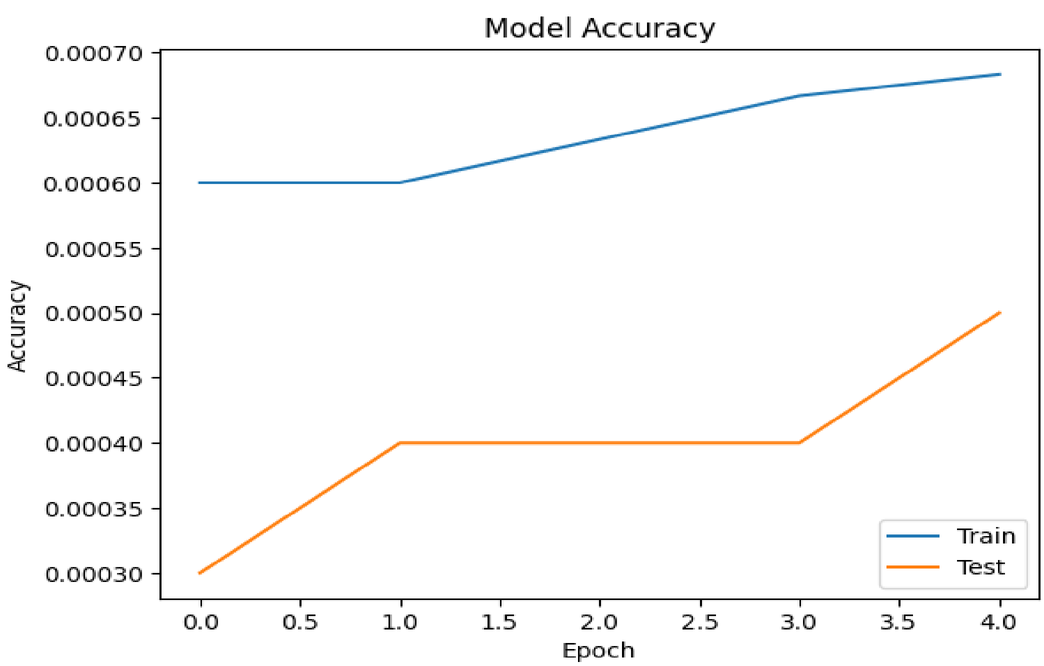
```python
# Train the autoencoder
history = autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# 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()

# 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.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 [==============================] - 9s 25ms/step - loss: 0.6939 - accuracy: 6.0000e-04 - val_loss: 0.6938 - val_accuracy: 3.0000e-04
Epoch 2/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6938 - accuracy: 6.0000e-04 - val_loss: 0.6936 - val_accuracy: 4.0000e-04
Epoch 3/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6936 - accuracy: 6.3333e-04 - val_loss: 0.6935 - val_accuracy: 4.0000e-04
Epoch 4/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6934 - accuracy: 6.6667e-04 - val_loss: 0.6933 - val_accuracy: 4.0000e-04
Epoch 5/5
235/235 [==============================] - 6s 25ms/step - loss: 0.6932 - accuracy: 6.8333e-04 - val_loss: 0.6931 - val_accuracy: 5.0000e-04
313/313 [==============================] - 1s 3ms/step
```







```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 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 12ms/step - loss: 0.6957 - val_loss: 0.6957
Epoch 2/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6955 - val_loss: 0.6955
Epoch 3/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6954 - val_loss: 0.6953
Epoch 4/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6952 - val_loss: 0.6952
Epoch 5/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6950 - val_loss: 0.6950
Epoch 6/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6947 - val_loss: 0.6947
Epoch 8/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6945 - val_loss: 0.6945
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6944 - val_loss: 0.6944
Epoch 10/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6942 - val_loss: 0.6942
<keras.src.callbacks.History at 0x7e9ed36bc9d0>
```

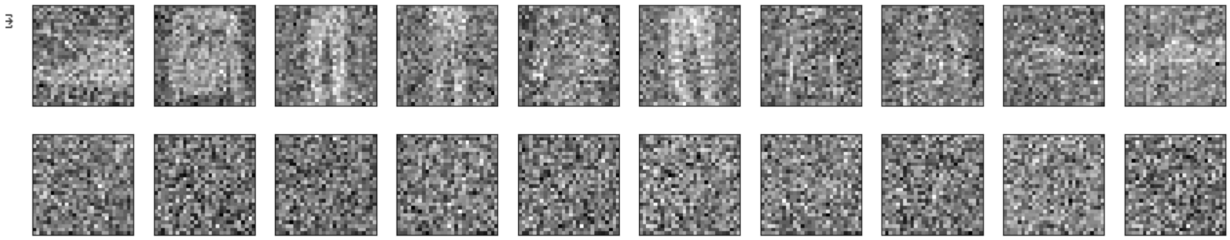
```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# 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)
```

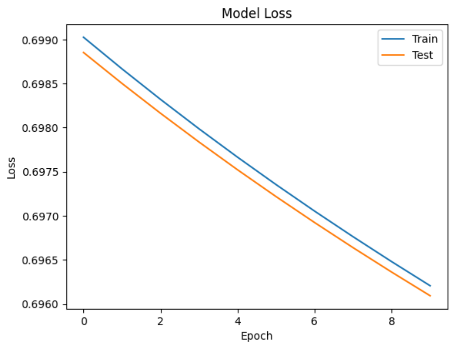
```python
# Train the autoencoder
history = autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))

# 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')
```

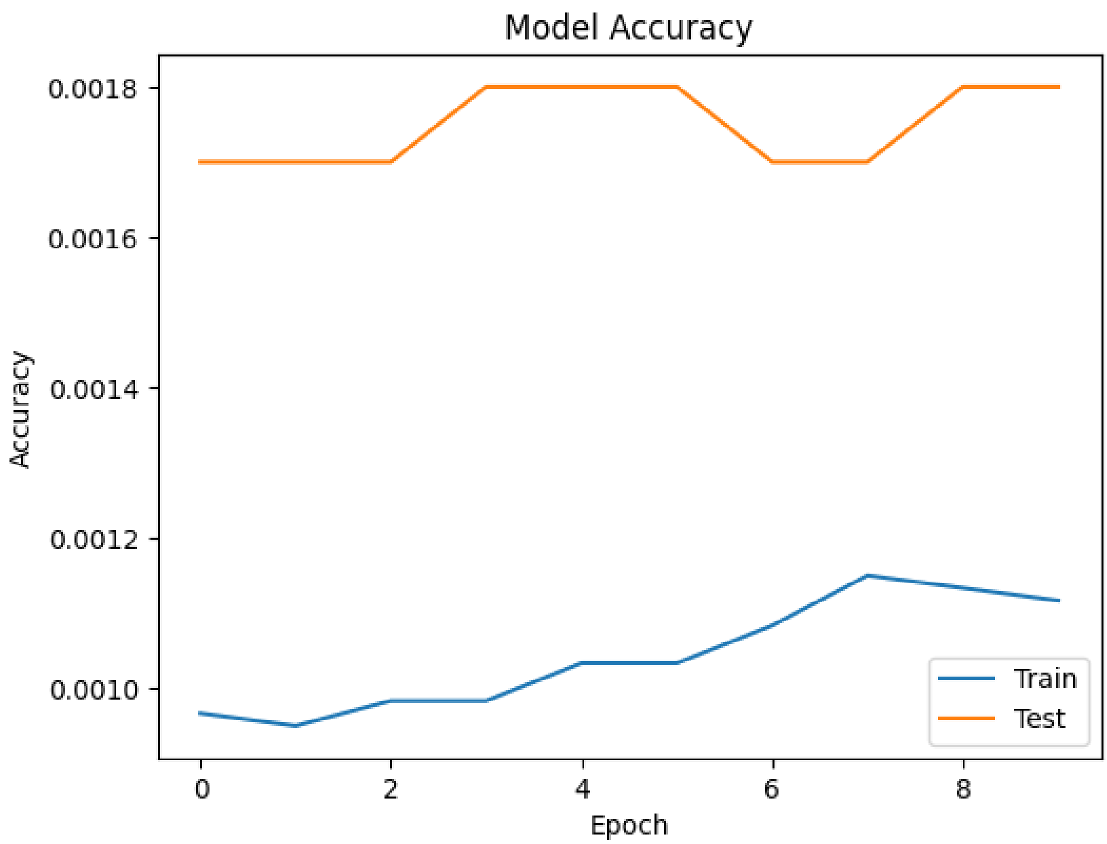

```python
# Plot the loss and accuracy over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```


```python
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6990 - accuracy: 9.6667e-04 - val_loss: 0.6989 - val_accuracy: 0.0017
Epoch 2/10
235/235 [==============================] - 2s 11ms/step - loss: 0.6987 - accuracy: 9.5000e-04 - val_loss: 0.6985 - val_accuracy: 0.0017
Epoch 3/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6983 - accuracy: 9.8333e-04 - val_loss: 0.6982 - val_accuracy: 0.0017
Epoch 4/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6980 - accuracy: 9.8333e-04 - val_loss: 0.6978 - val_accuracy: 0.0018
Epoch 5/10
235/235 [==============================] - 2s 11ms/step - loss: 0.6977 - accuracy: 0.0010 - val_loss: 0.6975 - val_accuracy: 0.0018
Epoch 6/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6974 - accuracy: 0.0010 - val_loss: 0.6972 - val_accuracy: 0.0018
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6971 - accuracy: 0.0011 - val_loss: 0.6969 - val_accuracy: 0.0017
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6968 - accuracy: 0.0012 - val_loss: 0.6966 - val_accuracy: 0.0017
Epoch 9/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6965 - accuracy: 0.0011 - val_loss: 0.6964 - val_accuracy: 0.0018
Epoch 10/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6962 - accuracy: 0.0011 - val_loss: 0.6961 - val_accuracy: 0.0018
313/313 [==============================] - 1s 2ms/step
```




Model Loss
Train
Test
0.6990
0.6985
0.6980
0.6975
0.6970
0.6965
0.6960
Loss
0
2
4
6
8
Epoch


Model Accuracy
0.0018
0.0016
0.0014
0.0012
0.0010
Accuracy
Train
Test
0
2
4
6
8
Epoch