Mayank Jadhav // CSE-20 // DL // exp5

```
import keras
from keras import layers

# 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 image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
```

```
# This model maps an input to its encoded representation
encoder = keras.Model(input_img, encoded)
```

```
# This is our encoded (32-dimensional) input
encoded_input = keras.Input(shape=(encoding_dim,))
# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
```

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

```
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
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:])))
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

Double-click (or enter) to edit

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/50
235/235 [==============================] - 3s 10ms/step - loss: 0.2777 - val_loss: 0.1911
Epoch 2/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1713 - val_loss: 0.1535
Epoch 3/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1437 - val_loss: 0.1331
Epoch 4/50
235/235 [==============================] - 2s 10ms/step - loss: 0.1282 - val_loss: 0.1212
Epoch 5/50
235/235 [==============================] - 2s 10ms/step - loss: 0.1182 - val_loss: 0.1129
Epoch 6/50
235/235 [==============================] - 2s 10ms/step - loss: 0.1109 - val_loss: 0.1067
Epoch 7/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1056 - val_loss: 0.1022
Epoch 8/50
235/235 [==============================] - 3s 13ms/step - loss: 0.1018 - val_loss: 0.0990
```

```
Epoch 9/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0991 - val_loss: 0.0968
Epoch 10/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0973 - val_loss: 0.0953
Epoch 11/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0961 - val_loss: 0.0943
Epoch 12/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0953 - val_loss: 0.0937
Epoch 13/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0948 - val_loss: 0.0932
Epoch 14/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0944 - val_loss: 0.0929
Epoch 15/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0941 - val_loss: 0.0927
Epoch 16/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0940 - val_loss: 0.0925
Epoch 17/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0938 - val_loss: 0.0924
Epoch 18/50
235/235 [==============================] - 3s 15ms/step - loss: 0.0937 - val_loss: 0.0923
Epoch 19/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0936 - val_loss: 0.0923
Epoch 20/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0934 - val_loss: 0.0921
Epoch 21/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0934 - val_loss: 0.0921
Epoch 22/50
235/235 [==============================] - 4s 15ms/step - loss: 0.0933 - val_loss: 0.0920
Epoch 23/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0933 - val_loss: 0.0920
Epoch 24/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0932 - val_loss: 0.0919
Epoch 25/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 26/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 27/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 28/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 29/50
```

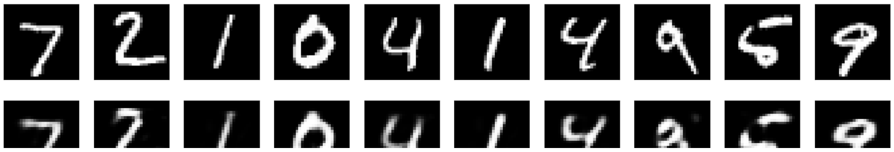
```
# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
```

```
# Use Matplotlib (don't ask)
import matplotlib.pyplot as plt

n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    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 reconstruction
    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()
```



2s completed at 3:02 PM