Open in Colab

Dikshant Buwa RollNo-04 DeepLearning BE-CSE(DS)

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
```

```
(x_train, _), (x_test, _) = 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:])))
print(x_train.shape)
print(x_test.shape)
```

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

```
encoding_dim = 32
input_img= keras.Input(shape=(784,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder=keras.Model(input_img,decoded)
```

```
encoder = keras.Model(input_img, encoded)
```

```
encoded_input= keras.Input(shape=(encoding_dim,))
decoder_layer=autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
```

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

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

```
Epoch 1/50
235/235 [==============================] - 2s 5ms/step - loss: 0.2780 - val_loss: 0.1915
Epoch 2/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1723 - val_loss: 0.1557
Epoch 3/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1453 - val_loss: 0.1335
Epoch 4/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1276 - val_loss: 0.1201
Epoch 5/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1169 - val_loss: 0.1117
Epoch 6/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1098 - val_loss: 0.1056
Epoch 7/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1047 - val_loss: 0.1014
Epoch 8/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1011 - val_loss: 0.0984
Epoch 9/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0987 - val_loss: 0.0965
Epoch 10/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0971 - val_loss: 0.0952
Epoch 11/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0961 - val_loss: 0.0944
Epoch 12/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0954 - val_loss: 0.0939
Epoch 13/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0949 - val_loss: 0.0935
Epoch 14/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0945 - val_loss: 0.0931
Epoch 15/50
```

```
235/235 [==============================] - 1s 4ms/step - loss: 0.0942 - val_loss: 0.0930
Epoch 16/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0940 - val_loss: 0.0928
Epoch 17/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0938 - val_loss: 0.0926
Epoch 18/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0937 - val_loss: 0.0924
Epoch 19/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0936 - val_loss: 0.0924
Epoch 20/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0935 - val_loss: 0.0922
Epoch 21/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0934 - val_loss: 0.0921
Epoch 22/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0933 - val_loss: 0.0920
Epoch 23/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0932 - val_loss: 0.0920
Epoch 24/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0932 - val_loss: 0.0919
Epoch 25/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 26/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0931 - val_loss: 0.0918
Epoch 27/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 28/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 29/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0930 - val_loss: 0.0917
```

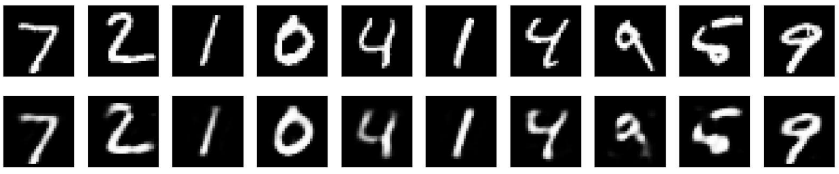
```
# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
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()
```



✓ 0s completed at 10:07 AM