```
Dimport keras
from keras import layers
from keras.datasets import mnist
import numpy as np
from keras.optimizers import Adam # import Adam optimizer
from keras.losses import binary_crossentropy # import binary_crossentropy loss function

encoding_dim=32
#this is our input image
input_img=keras.Input(shape=(784,))
#&quot;encoded&quot; is the encoded representation of the input
encoded=layers.Dense(encoding_dim, activation='relu')(input_img) # Use string 'relu'
#&quot;decoded&quot; is the lossy reconstruction of the input
decoded=layers.Dense(784, activation='sigmoid')(encoded) # Use string 'sigmoid'
#creating autoencoder model
autoencoder=keras.Model(input_img,decoded)
#create the encoder model
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,)) #Retrive 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))
# Compile the model with Adam optimizer and binary crossentropy loss
autoencoder.compile(optimizer=Adam(), loss=binary_crossentropy)

#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255. # Use 'float32' as a string
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)
#train autoencoder with training dataset
autoencoder.fit(X_train,X_train, epochs=50, batch_size=256, shuffle=True,
validation_data=(X_test,X_test))
encoded_imgs=encoder.predict(X_test)
decoded_imgs=decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # How many digits we
plt.figure(figsize=(40, 4))
for i in range(10): # display original
    ax = plt.subplot(3, 20, 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 encoded

    ax = plt.subplot(3, 20, i + 1 + 20)
    plt.imshow(encoded_imgs[i].reshape(8,4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False) # display

    # reconstruction
    ax = plt.subplot(3, 20, 2*20 +i+ 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```