In [1]:
```
import tensorflow
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import keras
from keras import layers
```

In [2]:
```
#Load data set
data = pd.read_csv('mnist_784_csv.csv')
data
```

Out[2]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel776	pixel777	pixel778	pixel77
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

70000 rows × 785 columns

In [3]:
```
y = data["class"]
x = data.drop(labels = ["class"], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.14, random_state=42)
```

In [4]:
```
x_train
```

Out[4]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel77
55078	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
57840	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
36470	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
10579	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
7683	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
37194	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
6265	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
54886	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
860	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
15795	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

60199 rows × 784 columns

In [5]:
```
x_train=x_train.to_numpy()
x_test=x_test.to_numpy()
```

In [6]:
```
x_train.shape,x_test.shape
```

Out[6]:
```
((60199, 784), (9801, 784))
```

In [7]:
```
x_train =x_train.reshape((60199, 28, 28))
x_test = x_test.reshape((9801, 28, 28))
```

In [8]:
```
x_train.shape
```

Out[8]:
```
(60199, 28, 28)
```

In [12]:
```
from keras import regularizers
encoding_dim = 32
input_img = keras.Input(shape=(784,))

# "encoded"
encoded = layers.Dense(encoding_dim, activation='relu', activity_regularizer=regularizers.l1(10e-5
# "decoded"
decoded = layers.Dense(784, activation='sigmoid')(encoded)
# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
```

In [13]:
```
# This model maps an input to its encoded representation
encoder = keras.Model(input_img, encoded)
```

In [14]:
```
# 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))
```

In [15]:
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

In [16]:
```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), 784))
x_test = x_test.reshape((len(x_test), 784))
print(x_train.shape)
print(x_test.shape)
```

```
(60199, 784)
(9801, 784)
```

In [17]:
```
x_train
```

Out[17]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [18]:
```
autoencoder.fit(x_train, x_train,
                epochs=120,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
236/236 [==============================] - 1s 4ms/step - loss: 0.0978 - val_loss: 0.0981
Epoch 67/120
236/236 [==============================] - 1s 4ms/step - loss: 0.0978 - val_loss: 0.0981
Epoch 68/120
236/236 [==============================] - 1s 3ms/step - loss: 0.0977 - val_loss: 0.0980
Epoch 69/120
236/236 [==============================] - 1s 3ms/step - loss: 0.0977 - val_loss: 0.0980
Epoch 70/120
236/236 [==============================] - 1s 3ms/step - loss: 0.0977 - val_loss: 0.0980
Epoch 71/120
236/236 [==============================] - 1s 4ms/step - loss: 0.0976 - val_loss: 0.0980
Epoch 72/120
236/236 [==============================] - 1s 3ms/step - loss: 0.0976 - val_loss: 0.0978
Epoch 73/120
236/236 [==============================] - 1s 4ms/step - loss: 0.0975 - val_loss: 0.0979
Epoch 74/120
236/236 [==============================] - 1s 4ms/step - loss: 0.0975 - val_loss: 0.0979
Epoch 75/120
236/236 [==============================] - 1s 4ms/step - loss: 0.0975 - val_loss: 0.0979
Epoch 76/120
```

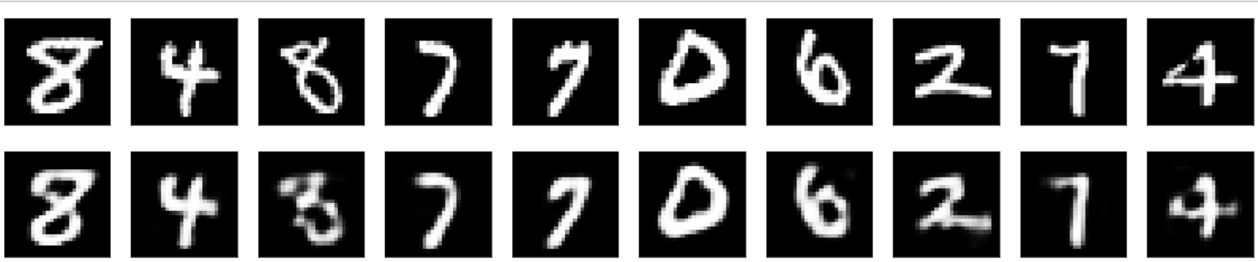
In [25]:
```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
307/307 [==============================] - 0s 518us/step
307/307 [==============================] - 0s 621us/step
```

In [26]:
```
# 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()
```



In [24]:
```
reconstr_error =autoencoder.evaluate(x_test, x_test, verbose=2)
reconstr_error
```

```
307/307 - 0s - loss: 0.0967 - 234ms/epoch - 763us/step
```

Out[24]:
```
0.09673042595386505
```

In [ ]: