In [1]:

```
#


#Implementation of a simple autoencoder

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 flo

# 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)
```

In [2]:

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

In [3]:

```
# 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 [4]:

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

In [5]:

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

In [6]:

```
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)
```

In [7]:

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

```
Epoch 1/50
235/235 [==============================] - 19s 51ms/step - loss: 0.3852 -
val_loss: 0.1896
Epoch 2/50
235/235 [==============================] - 3s 15ms/step - loss: 0.1801 - v
al_loss: 0.1538
Epoch 3/50
235/235 [==============================] - 4s 15ms/step - loss: 0.1491 - v
al_loss: 0.1335
Epoch 4/50
235/235 [==============================] - 3s 13ms/step - loss: 0.1313 - v
al_loss: 0.1213
Epoch 5/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1203 - v
al_loss: 0.1129
Epoch 6/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1124 - v
al_loss: 0.1072
Epoch 7/50
235/235 [==============================] - 3s 13ms/step - loss: 0.1075 - v
al_loss: 0.1031
Epoch 8/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1032 - v
al_loss: 0.0999
Epoch 9/50
235/235 [==============================] - 3s 15ms/step - loss: 0.1002 - v
al_loss: 0.0972
Epoch 10/50
235/235 [==============================] - 4s 15ms/step - loss: 0.0981 - v
al_loss: 0.0956
Epoch 11/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0968 - v
al_loss: 0.0944
Epoch 12/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0953 - v
al_loss: 0.0937
Epoch 13/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0950 - v
al_loss: 0.0933
Epoch 14/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0945 - v
al_loss: 0.0929
Epoch 15/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0943 - v
al_loss: 0.0929
Epoch 16/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0940 - v
al_loss: 0.0925
Epoch 17/50
235/235 [==============================] - 4s 15ms/step - loss: 0.0936 - v
al_loss: 0.0924
Epoch 18/50
```

```
235/235 [==============================] - 3s 14ms/step - loss: 0.0937 - v
al_loss: 0.0923
Epoch 19/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0934 - v
al_loss: 0.0923
Epoch 20/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0934 - v
al_loss: 0.0922
Epoch 21/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0936 - v
al_loss: 0.0921
Epoch 22/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0933 - v
al_loss: 0.0920
Epoch 23/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0931 - v
al_loss: 0.0919
Epoch 24/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0931 - v
al_loss: 0.0919
Epoch 25/50
235/235 [==============================] - 4s 16ms/step - loss: 0.0933 - v
al_loss: 0.0919
Epoch 26/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0930 - v
al_loss: 0.0919
Epoch 27/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0933 - v
al_loss: 0.0918
Epoch 28/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0929 - v
al_loss: 0.0918
Epoch 29/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0930 - v
al_loss: 0.0917
Epoch 30/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0928 - v
al_loss: 0.0917
Epoch 31/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0927 - v
al_loss: 0.0917
Epoch 32/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - v
al_loss: 0.0917
Epoch 33/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - v
al_loss: 0.0917
Epoch 34/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - v
al_loss: 0.0917
Epoch 35/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - v
al_loss: 0.0916
Epoch 36/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0927 - v
al_loss: 0.0916
Epoch 37/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0926 - v
al_loss: 0.0916
Epoch 38/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0926 - v
```

```
al_loss: 0.0916
Epoch 39/50
235/235 [==============================] - 3s 14ms/step - loss: 0.0927 - v
al_loss: 0.0918
Epoch 40/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - v
al_loss: 0.0916
Epoch 41/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0926 - v
al_loss: 0.0915
Epoch 42/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0926 - v
al_loss: 0.0915
Epoch 43/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0926 - v
al_loss: 0.0915
Epoch 44/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0926 - v
al_loss: 0.0916
Epoch 45/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0926 - v
al_loss: 0.0916
Epoch 46/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0925 - v
al_loss: 0.0915
Epoch 47/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0925 - v
al_loss: 0.0915
Epoch 48/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0926 - v
al_loss: 0.0915
Epoch 49/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0926 - v
al_loss: 0.0915
Epoch 50/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0926 - v
al_loss: 0.0915
```

Out[7]:

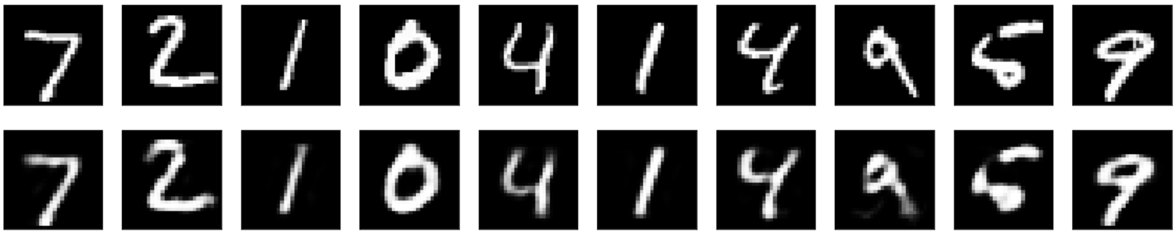
```
<tensorflow.python.keras.callbacks.History at 0x28f1c9048b0>
```

In [8]:

```
# 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)
```

In [9]:

```
# 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 [10]:

```
#Implementation of deep autoencoder

input_img = keras.Input(shape=(784,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(32, activation='relu')(encoded)

decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)


autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 3/100
235/235 [==============================] - 5s 19ms/step - loss: 0.1352 - v
al_loss: 0.1231
Epoch 4/100
235/235 [==============================] - 5s 23ms/step - loss: 0.1221 - v
al_loss: 0.1156
Epoch 5/100
235/235 [==============================] - 4s 15ms/step - loss: 0.1161 - v
al_loss: 0.1121
Epoch 6/100
235/235 [==============================] - 5s 20ms/step - loss: 0.1122 - v
al_loss: 0.1082
Epoch 7/100
235/235 [==============================] - 5s 21ms/step - loss: 0.1091 - v
al_loss: 0.1053
Epoch 8/100
235/235 [==============================] - 5s 20ms/step - loss: 0.1059 - v
al_loss: 0.1030
Epoch 9/100
235/235 [==============================] - 4s 17ms/step - loss: 0.1037 - v
```

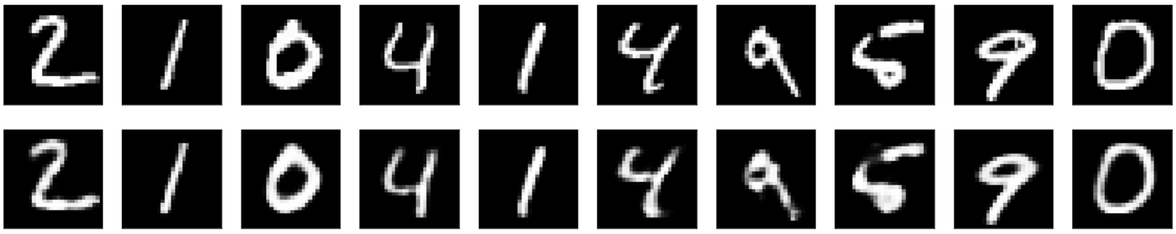
In [11]:

```
#Implementation of convolution autoencoder

import keras
from keras import layers

input_img = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')


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 = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))



tensorboard --logdir=/tmp/autoencoder


from keras.callbacks import TensorBoard

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
```

```
  File "<ipython-input-11-ac8a375c6f6c>", line 41
    tensorboard --logdir=/tmp/autoencoder
                        ^
SyntaxError: invalid syntax
```

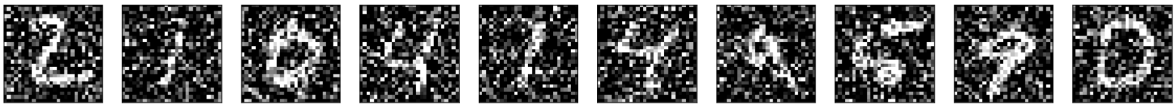
In [12]:

```
decoded_imgs = autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    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 + 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 [14]:

```
#Application to image denoising

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 = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.sha

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)


n = 10
plt.figure(figsize=(20, 2))
for i in range(1, n + 1):
    ax = plt.subplot(1, n, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



In [ ]: