In [1]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
```

In [2]:

```
def preprocess(array) :
    #Normalizes the supplied array and reshapes it into the appropriate format.
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))
    return array

def noise(array) :
    #Adds random noise to each image in the supplied array.
    noise_factor = 0.4
    noisy_array = array + noise_factor * np.random.normal(
        loc=0.0, scale=1.0, size=array.shape
    )
    return np.clip(noisy_array, 0.0, 1.0)
```

In [3]:

```
def display(array1, array2) :
    #Displays ten random images from each one of the supplied arrays.
    n = 10
    indices = np.random.randint(len(array1), size=n)
    images1 = array1[indices, :]
    images2 = array2[indices, :]

    plt.figure(figsize=(20, 4))
    for i, (image1, image2) in enumerate(zip(images1, images2)) :
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(image1.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(image2.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

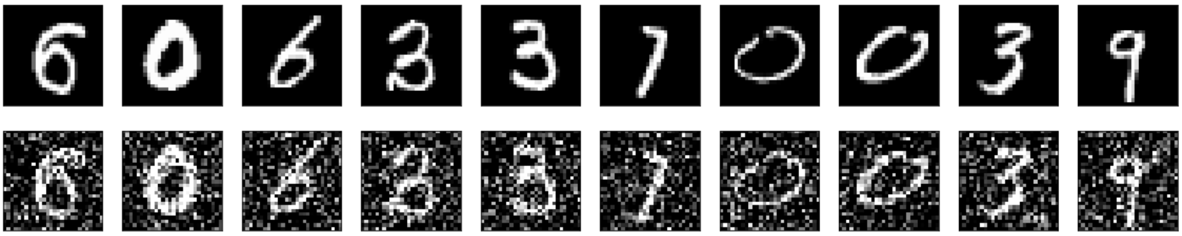
In [4]:

```
(train_data, _), (test_data, _) = mnist.load_data()

# Normalize and reshape the data
train_data = preprocess(train_data)
test_data = preprocess(test_data)

# Create a copy of the data with added noise
noisy_train_data = noise(train_data)
noisy_test_data = noise(test_data)

# Display the train data and a version of it with added noise
display(train_data, noisy_train_data)
```



In [5]:

```
input = layers.Input(shape=(28, 28, 1))

# Encoder
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decoder
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same"
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same"
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

# Autoencoder
autoencoder = Model(input, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0         
                                                                 
 conv2d (Conv2D)             (None, 28, 28, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0         
 2D)                                                             
                                                                 
 conv2d_transpose (Conv2DTra  (None, 14, 14, 32)       9248      
 nspose)                                                         
                                                                 
 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 32)       9248      
 ranspose)                                                       
                                                                 
 conv2d_2 (Conv2D)           (None, 28, 28, 1)         289       
                                                                 
=================================================================
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
_________________________________________________________________
```

In [6]:

```
autoencoder.fit(
    x=train_data,
    y=train_data,
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(test_data, test_data),
)
```

```
Epoch 1/10
469/469 [==============================] - 63s 133ms/step - loss: 0.1303 - val_los
s: 0.0735
Epoch 2/10
469/469 [==============================] - 67s 142ms/step - loss: 0.0716 - val_los
s: 0.0693
Epoch 3/10
469/469 [==============================] - 65s 139ms/step - loss: 0.0691 - val_los
s: 0.0680
Epoch 4/10
469/469 [==============================] - 65s 138ms/step - loss: 0.0680 - val_los
s: 0.0671
Epoch 5/10
469/469 [==============================] - 63s 135ms/step - loss: 0.0673 - val_los
s: 0.0668
Epoch 6/10
469/469 [==============================] - 64s 137ms/step - loss: 0.0668 - val_los
s: 0.0660
Epoch 7/10
469/469 [==============================] - 65s 139ms/step - loss: 0.0663 - val_los
s: 0.0657
Epoch 8/10
469/469 [==============================] - 105s 224ms/step - loss: 0.0660 - val_lo
ss: 0.0654
Epoch 9/10
469/469 [==============================] - 111s 236ms/step - loss: 0.0656 - val_lo
ss: 0.0651
Epoch 10/10
469/469 [==============================] - 113s 241ms/step - loss: 0.0654 - val_lo
ss: 0.0648
```

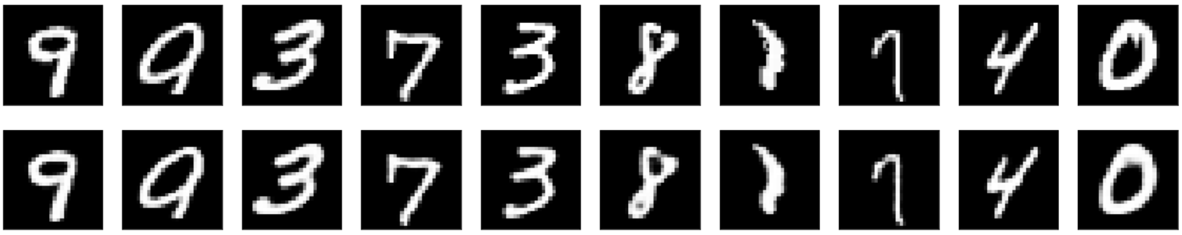
Out[6]:

```
<keras.callbacks.History at 0x1e5148416d0>
```

In [7]:

```python
predictions = autoencoder.predict(test_data)
display(test_data, predictions)
```

```
313/313 [==============================] - 10s 30ms/step
```



In [8]:

```python
autoencoder.fit(
    x=noisy_train_data,
    y=train_data,
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(noisy_test_data, test_data),
)
```

```
Epoch 1/10
469/469 [==============================] - 112s 240ms/step - loss: 0.1022 - val_lo
ss: 0.0940
Epoch 2/10
469/469 [==============================] - 110s 234ms/step - loss: 0.0935 - val_lo
ss: 0.0919
Epoch 3/10
469/469 [==============================] - 108s 230ms/step - loss: 0.0918 - val_lo
ss: 0.0905
Epoch 4/10
469/469 [==============================] - 108s 230ms/step - loss: 0.0907 - val_lo
ss: 0.0900
Epoch 5/10
469/469 [==============================] - 109s 231ms/step - loss: 0.0899 - val_lo
ss: 0.0890
Epoch 6/10
469/469 [==============================] - 108s 229ms/step - loss: 0.0894 - val_lo
ss: 0.0886
Epoch 7/10
469/469 [==============================] - 112s 238ms/step - loss: 0.0888 - val_lo
ss: 0.0880
Epoch 8/10
469/469 [==============================] - 111s 237ms/step - loss: 0.0884 - val_lo
ss: 0.0875
Epoch 9/10
469/469 [==============================] - 112s 238ms/step - loss: 0.0880 - val_lo
ss: 0.0873
Epoch 10/10
469/469 [==============================] - 110s 234ms/step - loss: 0.0876 - val_lo
ss: 0.0871
```

Out[8]:

```
<keras.callbacks.History at 0x1e51b54cfd0>
```

In [9]:

```python
predictions = autoencoder.predict(noisy_test_data)
display(noisy_test_data, predictions)
```

```
313/313 [==============================] - 9s 29ms/step
```