In [3]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Set default figure size and use an available Matplotlib style
plt.rcParams['figure.figsize'] = (20, 15)
plt.style.use('ggplot')

# Import required Keras modules
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
```

In [9]:
```
def preprocess(array):
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))  # Change to 28x28
    return array

def noise(array):
    noise_factor = 0.4
    noisy_array = array + noise_factor * np.random.normal(loc=0.0, scale=1.0, siz
    return np.clip(noisy_array, 0.0, 1.0)

def display(array1, array2):
    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)):
        # Display the first set of images
        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)

        # Display the second set of images
        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()
```

Data Preprocessing

In [10]:
```
(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 [12]:
```
encoder_input = layers. Input (shape=(28, 28, 1))
# Encoder
x = layers.Conv2D(32,(3,3),activation="relu", padding="same")(encoder_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)

encoder = Model(encoder_input, x, name='encoder')
encoder.summary()

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

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

Model: "encoder"

Layer (type)	Output Shape	
input_layer (InputLayer)	(None, 28, 28, 1)	
conv2d (Conv2D)	(None, 28, 28, 32)	
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	
conv2d_1 (Conv2D)	(None, 14, 14, 32)	
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	

Total params: 9,568 (37.38 KB)

Trainable params: 9,568 (37.38 KB)

Non-trainable params: 0 (0.00 B)

Model: "autoencoder"

Layer (type)	Output Shape	
input_layer (InputLayer)	(None, 28, 28, 1)	
conv2d (Conv2D)	(None, 28, 28, 32)	
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	
conv2d_1 (Conv2D)	(None, 14, 14, 32)	
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 32)	
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 32)	
conv2d_2 (Conv2D)	(None, 28, 28, 1)	

Total params: 28,353 (110.75 KB)

Trainable params: 28,353 (110.75 KB)

Non-trainable params: 0 (0.00 B)

In [15]:
```
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 ━━━━━━━━━━━━━━━━━━━━ 22s 43ms/step - loss: 0.2500 - val_loss: 0.0729
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0724 - val_loss: 0.0695
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0696 - val_loss: 0.0680
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0683 - val_loss: 0.0672
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0676 - val_loss: 0.0666
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 44ms/step - loss: 0.0668 - val_loss: 0.0661
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0664 - val_loss: 0.0657
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 21s 44ms/step - loss: 0.0661 - val_loss: 0.0654
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 21s 44ms/step - loss: 0.0656 - val_loss: 0.0651
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0654 - val_loss: 0.0649
```

Out[15]: <keras.src.callbacks.history.History at 0x1aa47398210>

In [16]:
```
predictions = autoencoder.predict(test_data)
display(test_data, predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```

In [17]:
```
test_data[0].reshape(-1,28,28,1).shape
```

Out[17]:
```
(1, 28, 28, 1)
```

Encoding Image

In [19]:
```
encoded = encoder.predict(test_data[0].reshape(-1,28,28,1))
encoded.shape
```

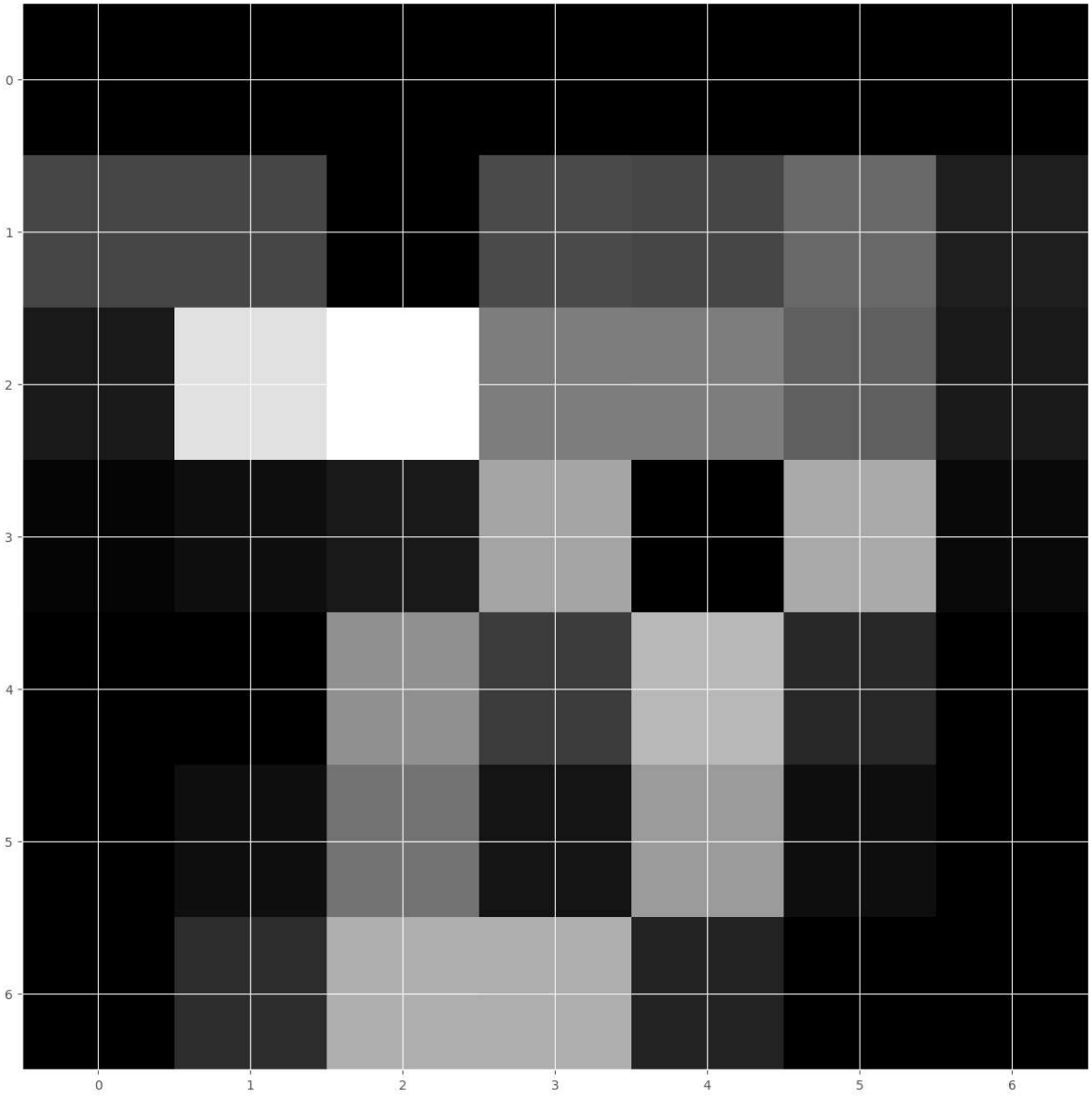
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
```

Out[19]:
```
(1, 7, 7, 32)
```

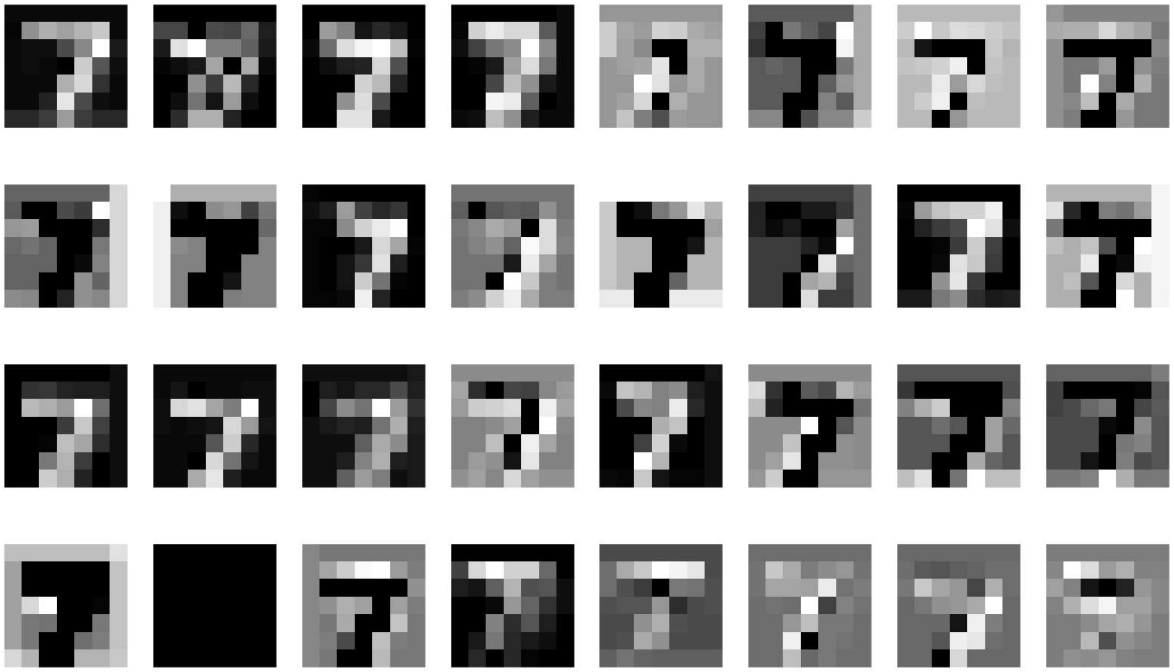
In [20]:
```
plt.imshow(encoder.predict(test_data[0].reshape(-1,28,28,1)).reshape(7,7,-1)[:,:,
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x1aa5516cb90>
```



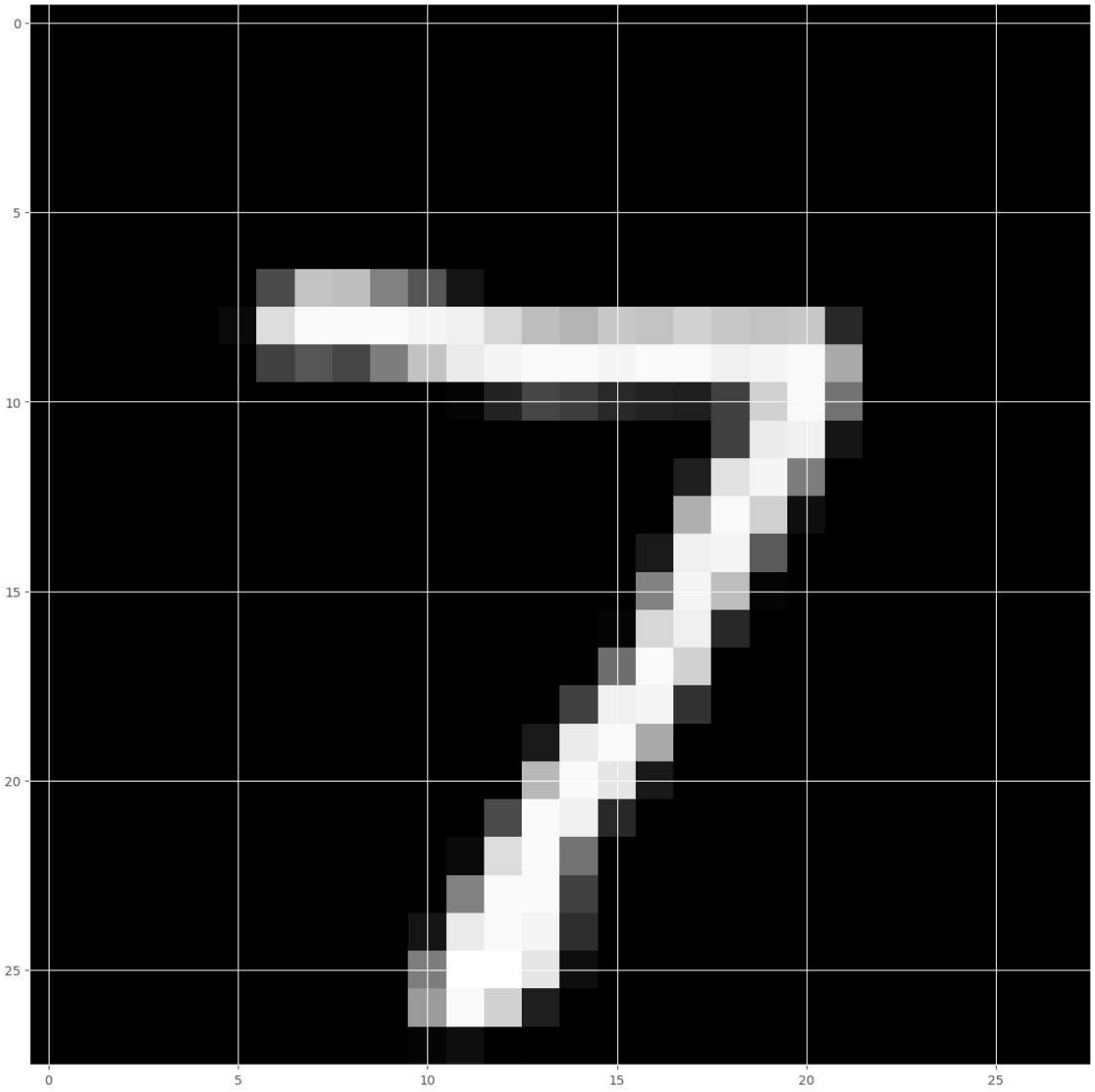
In [22]:
```
for i in range(32):
    ax = plt.subplot(5, 8, i+1)
    plt.imshow(encoded.reshape(7,7,-1)[:,:,i])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```



In [24]:
```
output = autoencoder.predict(test_data[0].reshape(-1,28,28,1))
plt.imshow(output.reshape(28,28))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

Out[24]:
```
<matplotlib.image.AxesImage at 0x1aa4a573650>
```



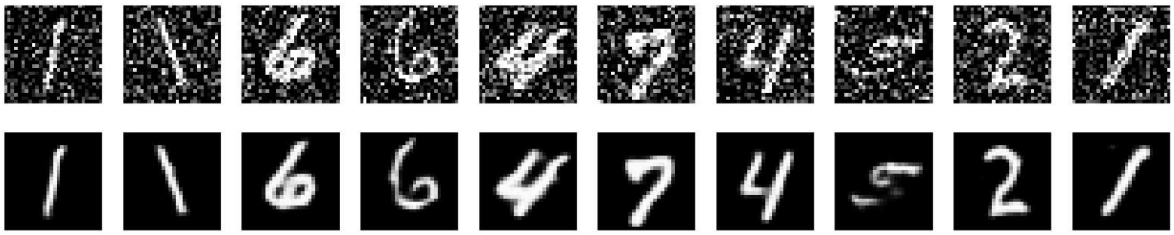
In [26]:
```
autoencoder.fit(
    x=noisy_train_data,
    y=train_data,
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(noisy_test_data, test_data)
)
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.1157 - val_loss: 0.0934
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 42ms/step - loss: 0.0938 - val_loss: 0.0913
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 42ms/step - loss: 0.0918 - val_loss: 0.0902
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0907 - val_loss: 0.0893
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 20s 43ms/step - loss: 0.0899 - val_loss: 0.0888
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 25s 53ms/step - loss: 0.0894 - val_loss: 0.0881
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 37s 78ms/step - loss: 0.0888 - val_loss: 0.0877
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 36s 77ms/step - loss: 0.0883 - val_loss: 0.0873
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 36s 77ms/step - loss: 0.0880 - val_loss: 0.0873
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 36s 77ms/step - loss: 0.0876 - val_loss: 0.0868
```

Out[26]: `<keras.src.callbacks.history.History at 0x1aa4a58dc90>`

In [28]:
```
predictions = autoencoder.predict(noisy_test_data)
display(noisy_test_data,predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step
```



In [ ]: