Image Compression and Decompression

Downloading the data and preprocessing it

In [1]:

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n
pz
11493376/11490434 [==============================] - 0s 0us/step
(60000, 28, 28) (10000, 28, 28)
```

In [2]:

```
x_train.shape,x_test.shape
```

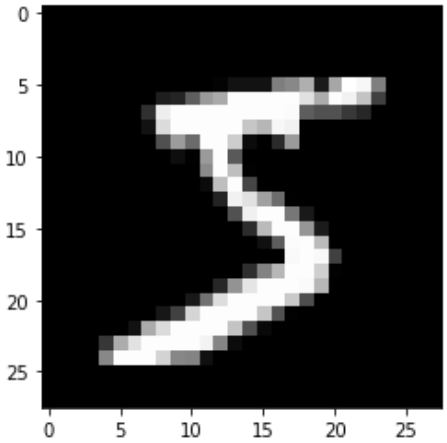
Out[2]:

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

Visualising training data image

In [5]:

```
from matplotlib import pyplot as plt
import numpy as np

first_image = x_train[0]
first_image = np.array(first_image, dtype='float')
pixels = first_image.reshape((28, 28))
plt.imshow(pixels, cmap='gray')
plt.show()
```



Creating the Autoencoder

In [10]:

```
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')
```

In [11]:

```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
from keras.utils import plot_model
plot_model(autoencoder, to_file='model_plot.png', show_shapes=True, show_layer_names=Tru
e)
```

Training the autoencoder

In [ ]:

```
history = autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test, x_test))
```

In [25]:

```
autoencoder.save("autoencoder.h5")
```

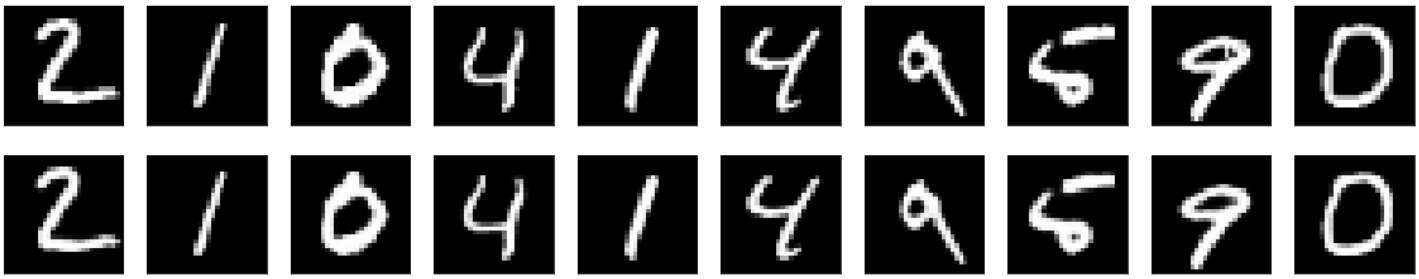
In [17]:

```
from keras.models import load_model
autoencoder=load_model("autoencoder.h5")
```

Testing the trained model and comparing it with the original data

In [18]:

```
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()
```



Visualising the states of a image through the autoencoder

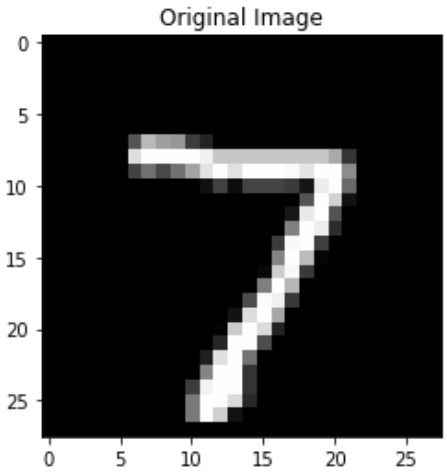
In [8]:

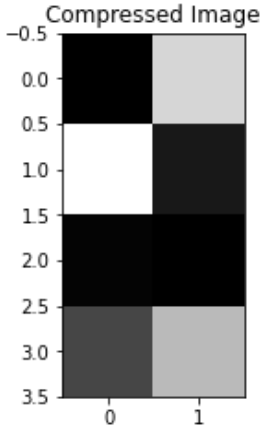
```
from tensorflow.keras import Sequential
import tensorflow as tf

#encoder model
model=tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(16,(3,3),activation ='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(2,2),
```

```
    tf.keras.layers.Conv2D(8,(3,3),activation ='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(8,(3,3),activation ='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
 ])
```

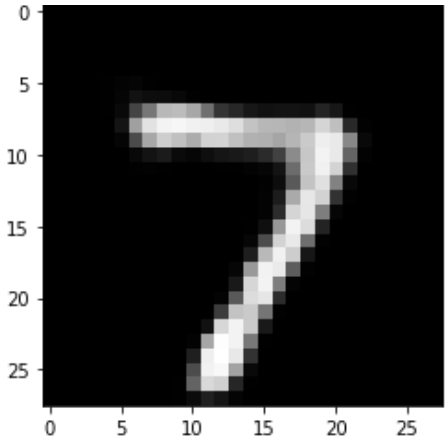
In [ ]:

```
def visualize(img,encoder):
    code = encoder.predict(img[None])[0]

    # Display original
    plt.title("Original Image")
    plt.imshow(x_test[0].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.show()

    #Display compressed
    plt.subplot(1,3,2)
    plt.title("Compressed Image")
    plt.imshow(code.reshape([code.shape[-1]//2,-1]))
    plt.show()

    # Display reconstruction
    plt.title("Decompressed Image")
    plt.imshow(decoded_imgs[0].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.show()

visualize(x_test[0],model)
```





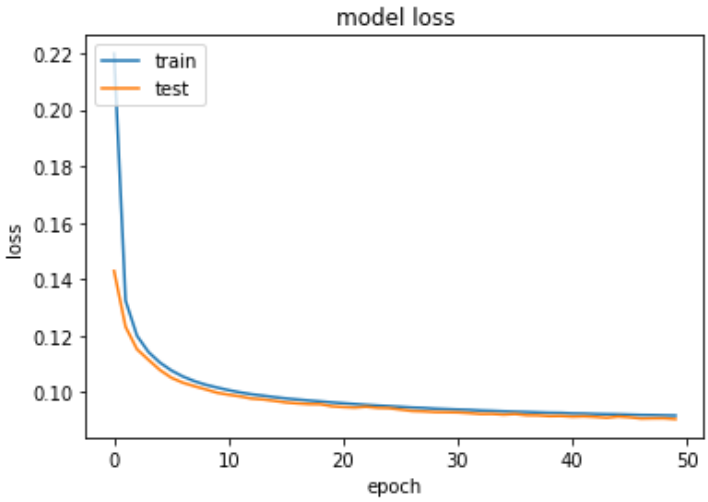
Decompressed Image



Analysing the loss wrt epoch

In [ ]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Denoising model for the Decompressed Image

Adding noise to the train and test data

In [3]:

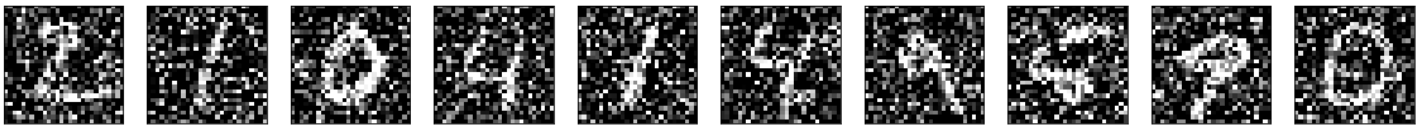
```
# Adding random noise to the images
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_tra
in.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.
shape)

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

Visualising the training data

In [6]:

```
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)
print("Training Data:")
plt.show()
```

```
Training Data:
```



Creating the encoder model

In [12]:

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

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# At this point the representation is (7, 7, 32)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(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')
```

In [13]:

```
autoencoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 28, 28, 1)         289
=================================================================
Total params: 28,353
Trainable params: 28,353
```

```
Non-trainable params: 0
_________________________________________________________________
```

Training the model

In [ ]:

```
history2 = autoencoder.fit(x_train_noisy, x_train,
                epochs=50,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

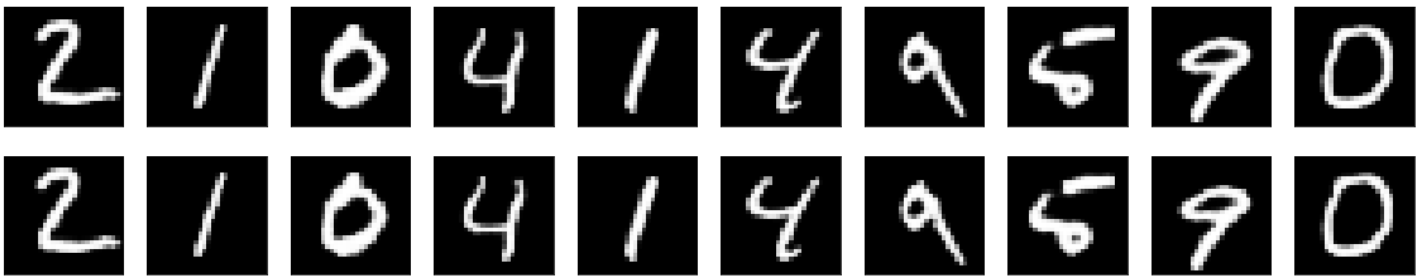
In [ ]:

```
from keras import models
autoencoder = models.load_model('denoising_model.h5')
```

Visualising the results of denoising the decompressed data

In [19]:

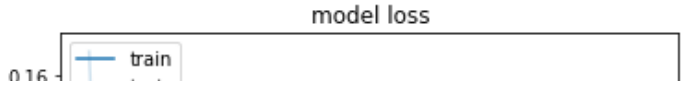
```
denoised_imgs = autoencoder.predict(decoded_imgs)

n = 10
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    plt.imshow(decoded_imgs[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(denoised_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

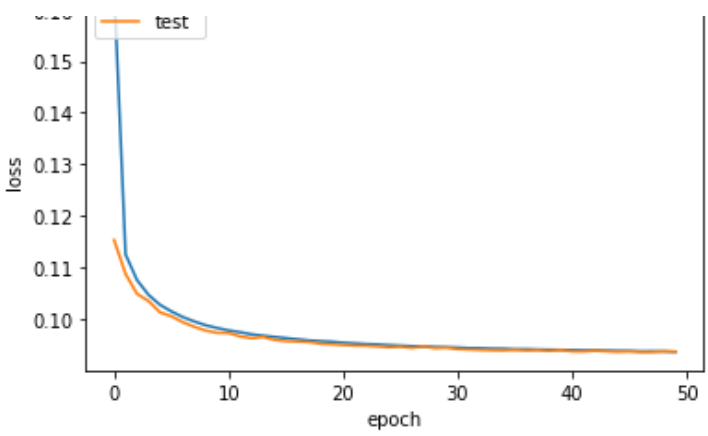


Analysing the loss wrt epoch

In [15]:

```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





Quality Metrics - PSNR

In [23]:

```
from math import log10, sqrt
import cv2
import numpy as np

def PSNR(original, compressed):
    mse = np.mean((original - decompressed) ** 2)
    if(mse == 0):  # MSE is zero means no noise is present in the signal .
                   # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr

psnr=0
for i in range(0,50):
  original = x_test[i].reshape(28, 28)
  decompressed =denoised_imgs[i].reshape(28,28)
  value = PSNR(original, decompressed)
  psnr+=value
psnr=psnr/50
print(f"PSNR value is {psnr} dB with denoising")
```

```
PSNR value is 75.57137665800288 dB with denoising
```