| | | |
|---|---|---|
| **Name:** Prasad Jawale | **Class/Roll No:** D16AD 20 | **Grade:** |

**Title of Experiment:** Autoencoders for Image Compression.

**Objective of Experiment:** The objective of this project is to design and implement an autoencoder-based image compression system that can effectively reduce the size of input images while preserving their essential visual information. This system aims to explore the potential of autoencoders in the field of image compression, leveraging their ability to learn compact representations of images.

**Outcome of Experiment:** Thus, we implemented an autoencoder model trained on a dataset of images, which has learned to encode and decode images efficiently**.**

**Problem Statement:** Traditional image compression techniques, such as JPEG, often result in loss of image quality due to the use of lossy compression algorithms. The problem is to develop an autoencoder-based image compression system that addresses this issue by compressing images into a lower-dimensional representation while minimizing the loss of critical visual details. This system should strike a balance between compression ratio and image quality, making it suitable for various applications such as efficient storage, transmission, and sharing of images while maintaining their perceptual fidelity.

**Description / Theory:**

***Architecture:***

An autoencoder consists of two main parts:

- Encoder: The encoder takes an input image and maps it to a lower-dimensional representation called the "latent space" or "encoding." This encoding typically has fewer dimensions than the original image, which leads to compression.

- Decoder: The decoder takes the encoded representation and attempts to reconstruct the original image from it.

***Training:***

Autoencoders are trained using unsupervised learning. The training objective is to minimize the reconstruction error, which measures how well the decoder can reconstruct the input image from its encoding. Common loss functions for image compression tasks include mean squared error (MSE) or binary cross-entropy, depending on whether the images are continuous or binary (e.g., grayscale or black-and-white).

***Compression:***

The key to image compression with autoencoders is the reduction in the dimensionality of the encoding compared to the original image. By encoding the image in a lower-dimensional space, it's possible to represent the image using fewer bits, which results in compression.

***Benefits:***

Autoencoders offer lossy compression, meaning that some information is lost during compression, but the goal is to preserve the essential features for human perception. They can be used for various image compression applications, including reducing storage space and speeding up image transmission over networks.

***Trade – offs:***

The trade-off in using autoencoders for compression is the balance between compression ratio and image quality. More aggressive compression may lead to greater loss of image quality. The choice of architecture, hyperparameters, and the size of the latent space can impact compression performance.

***Applications:***

Autoencoders for image compression are used in various fields, including medical imaging, video streaming, and image transmission in low-bandwidth environments. They are also used as a pre-processing step for other computer vision tasks, where reducing the dimensionality of images can improve efficiency

# Program:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
```

In [2]:
```
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
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:])))
```

In [3]:
```
input_img = Input(shape = (784,))
encoded = Dense(128, activation = 'relu')(input_img)
decoded = Dense(784, activation = 'sigmoid')(encoded)
```

In [4]:
```
autoencoder = Model(input_img, decoded)
```

In [5]:
```
autoencoder.compile(optimizer = 'adam', loss = 'binary_crossentropy')
```

In [6]:
```
autoencoder.fit(x_train, x_train, epochs = 5, batch_size = 28, shuffle = True, validation_data = (x_test, x_test))
```

```
Epoch 1/5
2143/2143 [==============================] - 9s 4ms/step - loss: 0.1128 - val_loss: 0.0769
Epoch 2/5
2143/2143 [==============================] - 11s 5ms/step - loss: 0.0735 - val_loss: 0.0704
Epoch 3/5
2143/2143 [==============================] - 9s 4ms/step - loss: 0.0698 - val_loss: 0.0684
Epoch 4/5
2143/2143 [==============================] - 11s 5ms/step - loss: 0.0684 - val_loss: 0.0675
Epoch 5/5
2143/2143 [==============================] - 12s 5ms/step - loss: 0.0676 - val_loss: 0.0670
```

Out[6]:
```
<keras.callbacks.History at 0x2278e748e20>
```

In [7]:
```
encoded_imgs = autoencoder.predict(x_test)
decoded_imgs = encoded_imgs
```

```
313/313 [==============================] - 1s 2ms/step
```

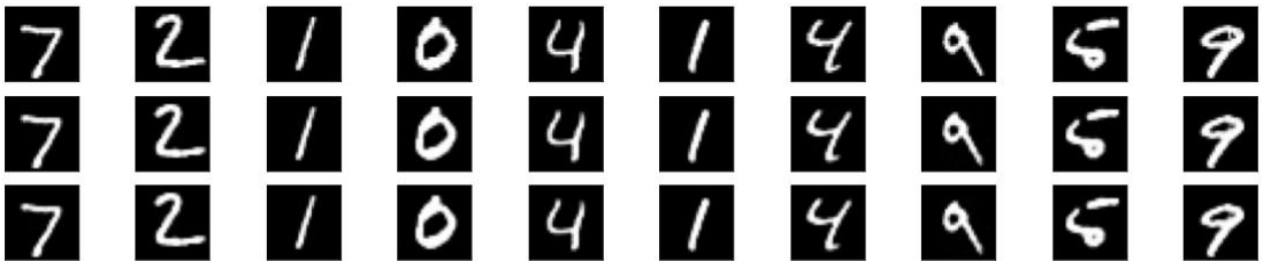
In [8]:
```
n = 10
plt.figure(figsize = (20, 4))

for i in range(n):
    # Original Images
    ax = plt.subplot(3, 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)

    # Encoded Images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(encoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Decoded Images
    ax = plt.subplot(3, n, i + 1 + 2 * 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()
```



**Results and Discussions:**

- Dataset Loading and Preprocessing: The MNIST dataset, consisting of grayscale hand-written digits, is loaded. The pixel values of the images are scaled between 0 and 1 by dividing by 255.0. The data is reshaped into flat vectors of size 784 (28×28).

- Autoencoder Architecture: The autoencoder architecture is defined with an input layer of 784 neurons (the flattened image). A dense layer with 128 neurons and ReLU activation serves as the encoder. Another dense layer with 784 neurons and sigmoid activation serves as the decoder. The autoencoder model is created, which maps the input to its reconstruction.

- Training: The model is compiled with the Adam optimizer and binary cross-entropy loss function. The training process involves 5 epochs with a batch size of 28, and the data is shuffled during training. Both the training and validation datasets are the same (x train and x test). Encoding and Decoding The trained autoencoder is used to encode and decode the images from the test

- Encoding & Decoding: The trained autoencoder is used to encode and decode the images from the test dataset. The encoded images represent a compressed form of the input data. The decoded images are the model's attempt to reconstruct the original images from their encoded representations.

- Visualization: The program generates a visual comparison of original, encoded, and decoded images for 10 samples from the test dataset. Three rows of images are displayed: the top row for original images, the middle row for encoded images, and the bottom row for decoded images.

The autoencoder successfully learns a compressed representation of the input images in its encoded layer. The encoded images capture the essential features of the original images, albeit with a lower dimensionality. The decoded images show that the model can reconstruct the input data with reasonable accuracy, although there may be some loss of fine details. This type of autoencoder can be used for various applications, including image denoising, dimensionality reduction, and feature extraction.