Name: Subrato Tapaswi	**Class/Roll No.: D16AD/60**	**Grade:**

Title of Experiment: Autoencoder for image denoising.

Objective of Experiment: The objective is to design and implement an autoencoder based image denoising system. The aim is to assess the effectiveness of autoencoders in reducing noise from images while preserving essential visual information, aiming to enhance image quality through denoising techniques.

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

Problem Statement: Image noise, which can be caused by various factors, can really mess up digital pictures and make it difficult for computers to understand them. The presence of noise in digital images can deteriorate image quality and hinder the accuracy of various computer vision tasks. So, the challenge is to create a system that uses autoencoders to clean up noisy images, making them look better and more useful for computer tasks like recognizing objects or faces.

Description / Theory:
Autoencoders are a type of artificial neural network used in machine learning and deep learning for various tasks, including image denoising. The primary objective of using autoencoders for image denoising is to remove noise from input images while preserving their essential features.

1. Architecture: An autoencoder for image denoising also consists of two main parts: an encoder and a decoder.
   Encoder: The encoder takes a noisy 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, capturing the underlying structure of the image while filtering out noise.

Decoder: The decoder takes the encoded representation and attempts to reconstruct a denoised version of the original image from it.

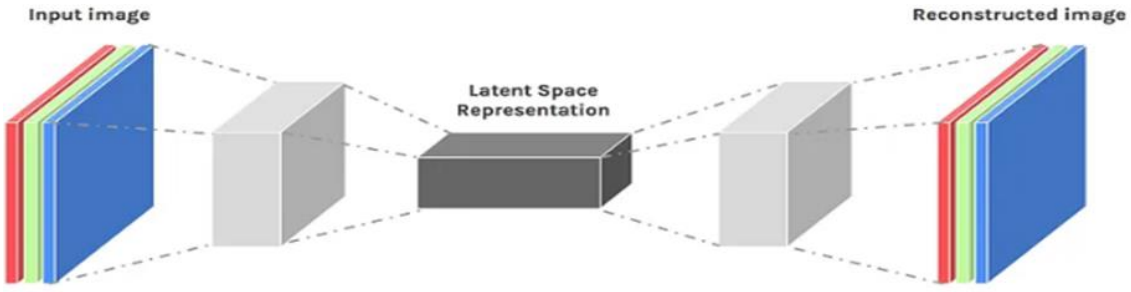
2. Training: Autoencoders for image denoising are trained using unsupervised learning. The training objective is to minimize the reconstruction error, which measures how well the decoder can reconstruct the clean (noise-free) image from its encoding. The training dataset consists of pairs of noisy images and their corresponding clean counterparts.

3. Denoising: The key to image denoising with autoencoders is the reduction of noise in the encoded representation compared to the noisy input image. By filtering out noise in the encoding, the decoder can generate a denoised version of the original image.

Applications -

Autoencoders for image denoising find applications in various domains, including medical imaging, where removing noise is crucial for accurate diagnosis. They are also used in enhancing the quality of images in low-light conditions, improving the efficiency of image processing pipelines, and aiding computer vision tasks that require clean input data.

Algorithm/ Pseudo Code:



Autoencoder architecture

Deep Learning/Odd Sem 2023-23/Experiment 3b

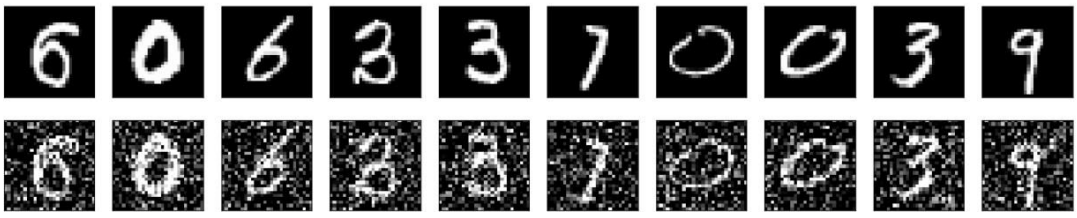
Program:

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)
```

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)
```



Artificial Intelligence and Data Science Department

Deep Learning/Odd Sem 2023-23/Experiment 3b

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)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
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()
```

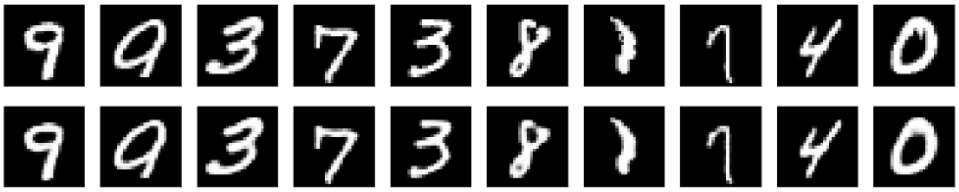
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 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
```

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

In [7]:
```
predictions = autoencoder.predict(test_data)
display(test_data, predictions)
```

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



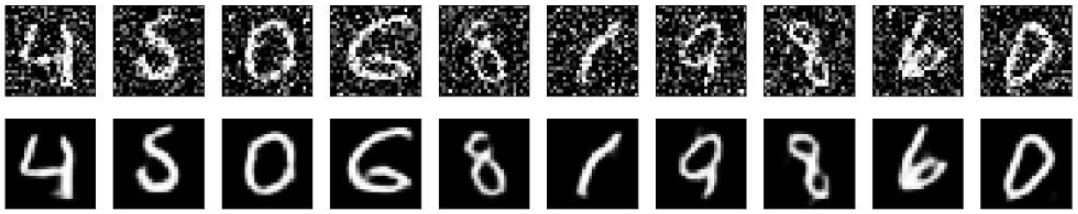
```
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(noisy_test_data, test_data),
)
```

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

Out[8]: <keras.callbacks.History at 0x1e51b54cfd0>

In [9]:
```
predictions = autoencoder.predict(noisy_test_data)
display(noisy_test_data, predictions)
```

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



Results and Discussions: We managed to reduce digital image noise using autoencoders. The encoded versions captured the important image details, and when we decoded them, the images looked much better without noise. However, we did lose some fine details in the process, but it was a trade-off for getting rid of the noise.