Programming Assignment 3

Name: Mayank Kumar
UCF-ID: 5529148
Email: mayank_kumar@ucf.edu

I. Part-1 Autoencoders

1. Introduction:

Autoencoders are a type of artificial neural network used in unsupervised machine learning and deep learning. They are designed to learn efficient representations or encodings of input data by training the network to reconstruct its own inputs. The architecture typically consists of an encoder, which maps input data to a lower-dimensional representation (encoding), and a decoder, which reconstructs the original data from the encoding. Autoencoders are useful for various tasks, such as data compression, denoising, and feature learning. They can also be employed for anomaly detection and dimensionality reduction in applications like image and text processing. Autoencoders are a fundamental building block in deep learning and can be adapted for various specific use cases, including image generation and semantic similarity measurement.

2. Dataset:

For this assignment, we will use the MNIST dataset which is a widely used dataset in the field of machine learning and computer vision. It stands for the "Modified National Institute of Standards and Technology" dataset and is a collection of handwritten digits.

The MNIST dataset consists of 70,000 grayscale images of handwritten digits, each measuring 28x28 pixels. These images are divided into a training set of 60,000 samples and a test set of 10,000 samples. Each image corresponds to a single digit (0-9) and is labeled with its respective digit. MNIST is a benchmark dataset for digit classification tasks and serves as a fundamental dataset for testing and benchmarking machine learning algorithms.

3. Model Architectures used:

a. Fully Connected Autoencoder:

i. Encoder:

- Flatten Layer: Reshapes the input image tensor into a 1D tensor.
- Linear Layer 1: Takes the flattened input and projects it to a lower-dimensional space (256 units).
- Linear Layer 2: Further reduces dimensionality to 128 units.

ii. Decoder:

- Linear Layer 3: Expands the encoded representation back to 256 units.
- Linear Layer 4: Reconstructs the flattened image.

b. CNN based Autoencoder:

i. Encoder:

- Convolutional Layer 1: Accepts an input tensor with 1 channel and applies a convolution operation with 16 filters, each of size 3x3, and uses padding to maintain spatial dimensions. This layer enhances feature representation through local receptive fields.

- ReLU Activation 1: Introduces non-linearity to the model after the first convolutional layer.

- Max-Pooling Layer 1: Performs max-pooling with a 2x2 kernel, reducing spatial dimensions and emphasizing important features.

- Convolutional Layer 2: Accepts the 16-channel output from the previous layer and applies another convolution operation with 32 filters. This layer captures more complex hierarchical features.

- ReLU Activation 2: Introduces non-linearity after the second convolutional layer.

- Max-Pooling Layer 2: Further reduces spatial dimensions through max-pooling.

ii. Decoder:

- Convolutional Layer 3: Takes the 32-channel encoded representation and applies a convolution operation with 16 filters to reconstruct features.

- ReLU Activation 3: Introduces non-linearity to the reconstructed features.

- Upsampling Layer 1: Upsamples the spatial dimensions using bilinear interpolation, allowing the network to learn finer details during reconstruction.

- Convolutional Layer 4: Processes the upsampled features with another convolution operation, reducing the channel dimension to 8.

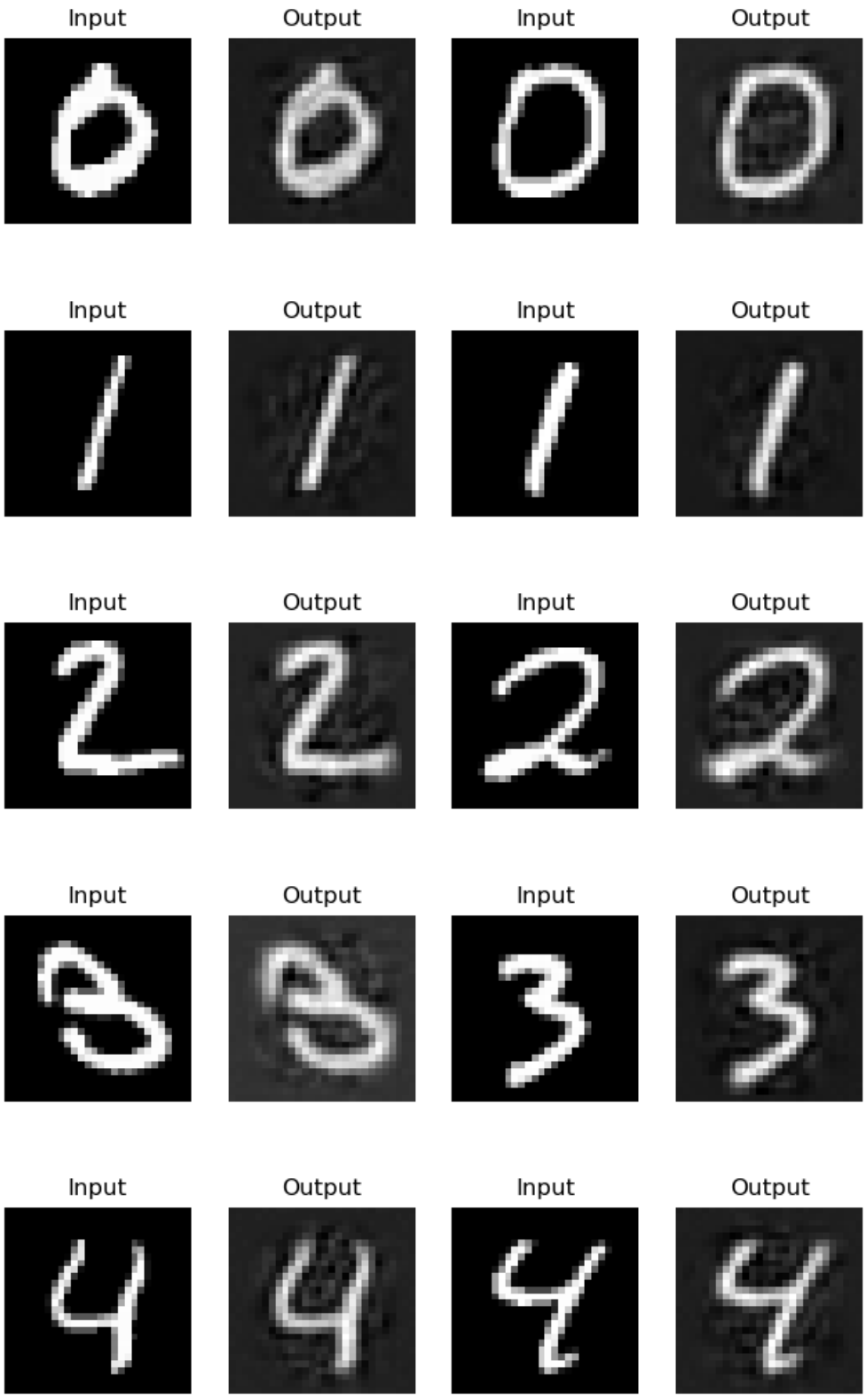
- ReLU Activation 4: Introduces non-linearity to the features after the fourth convolutional layer.

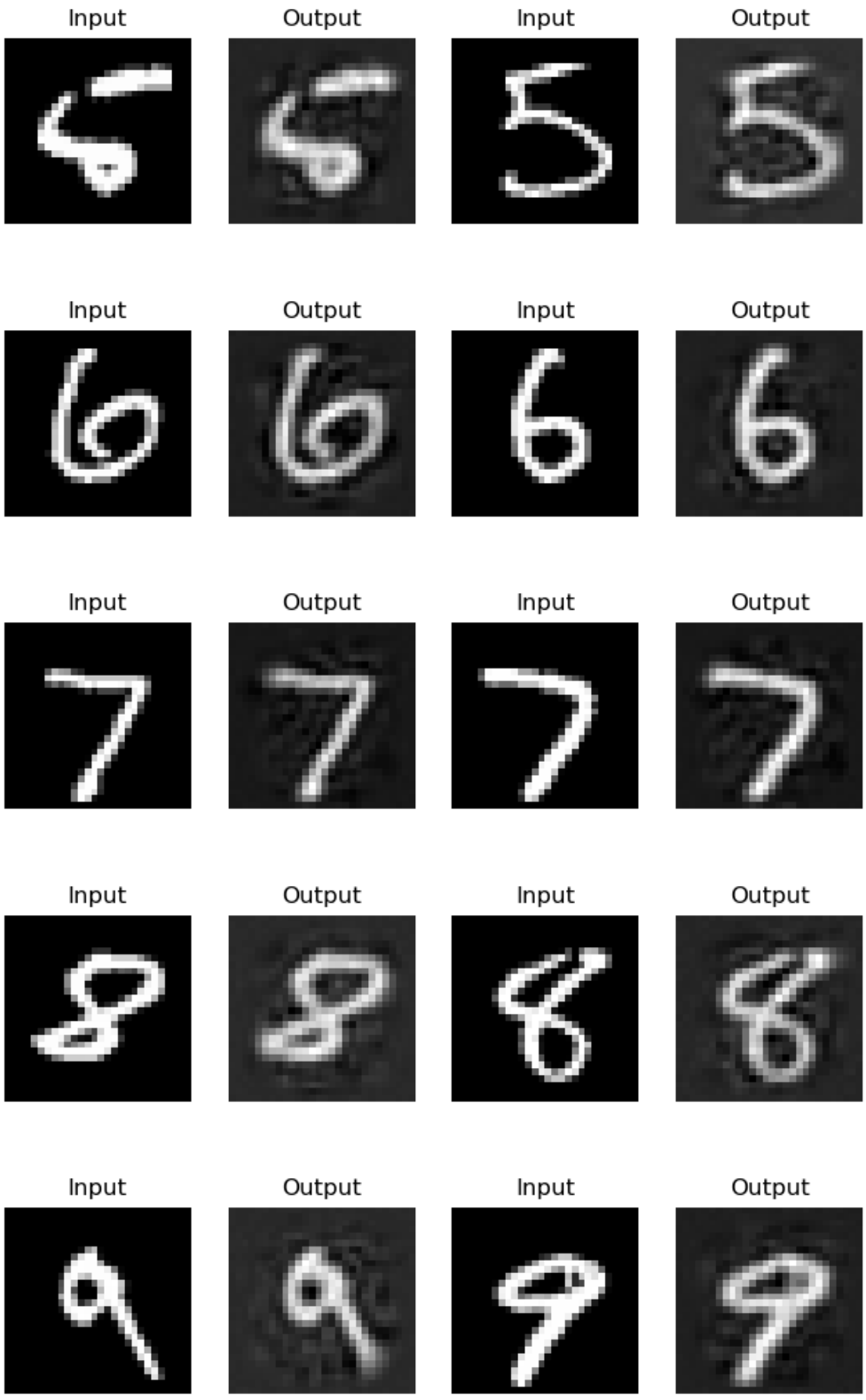
- Upsampling Layer 2: Similar to the first upsampling layer, this further increases the spatial dimensions of the features.

- Convolutional Layer 5: The final convolutional layer takes the 8-channel representation and performs a convolution operation to reconstruct the original 1-channel image.

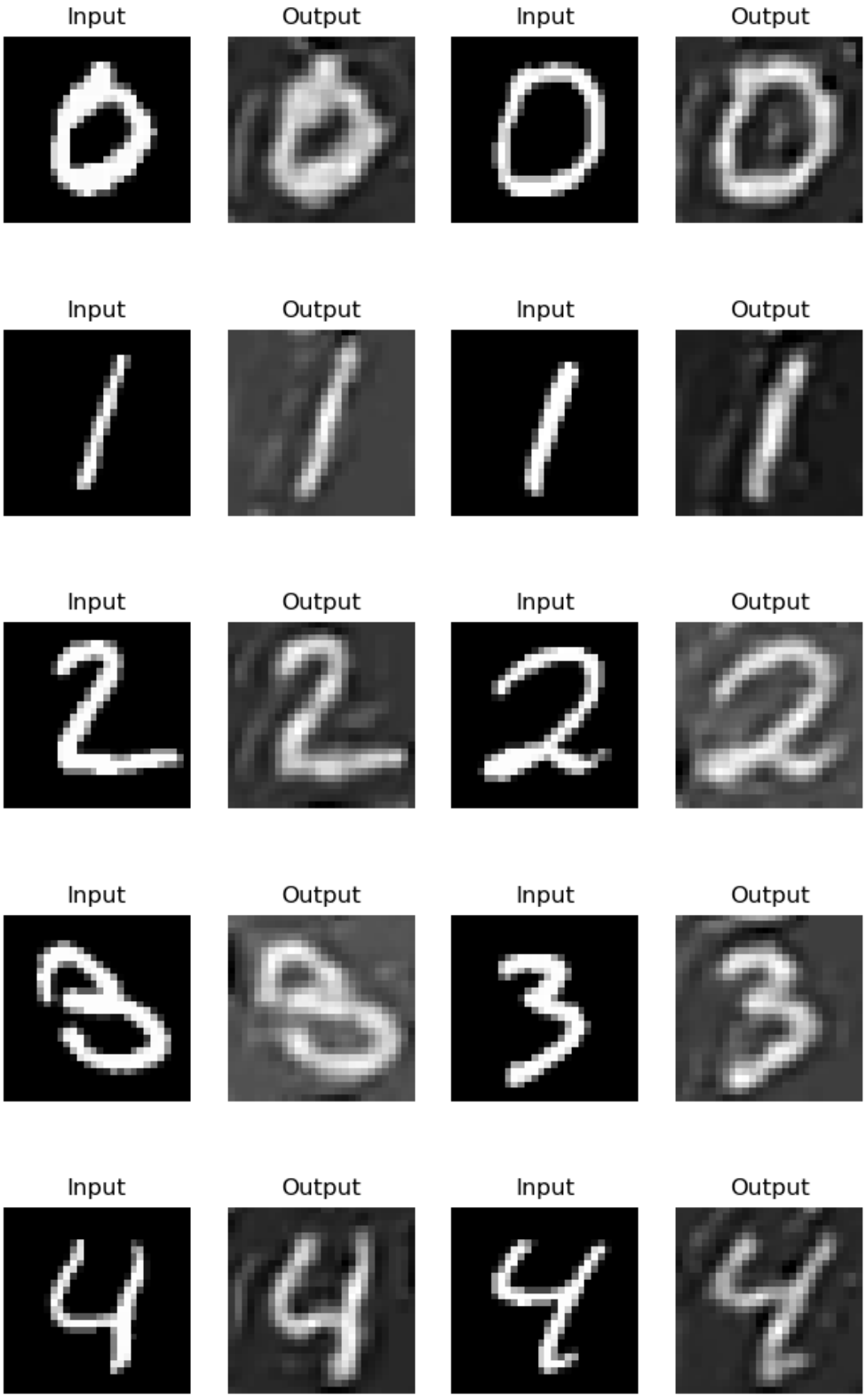
4. **Comparison of Number of parameters:**
    a. **Fully Connected Autoencoder:**
        i. Encoder: 233856
        ii. Decoder: 234512
        iii. Full model: 468368
    b. **CNN based Autoencoder:**
        i. Encoder: 4800
        ii. Decoder: 5857
        iii. Full model: 10657

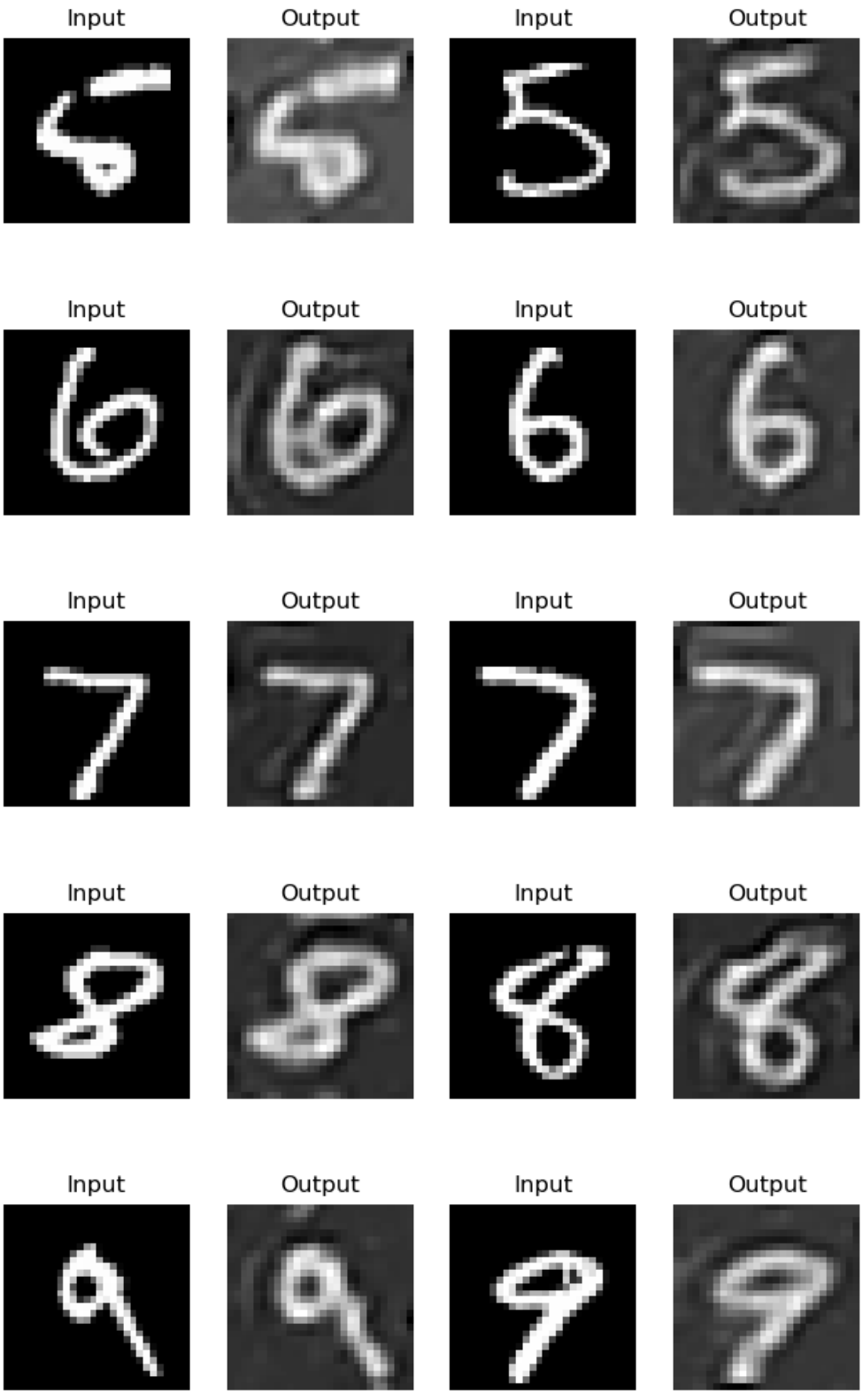
5. **Results:**
    a. **Fully Connected:**


Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output

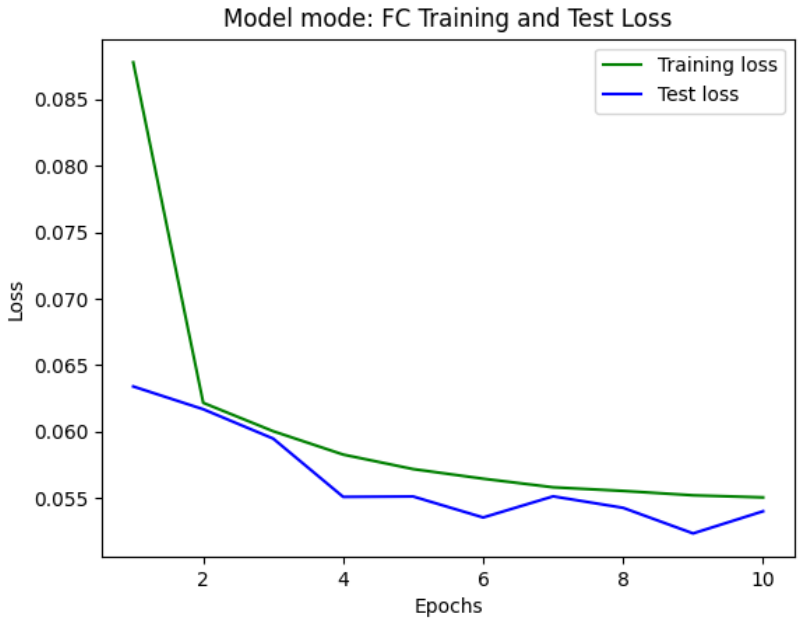

Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output

b. CNN:

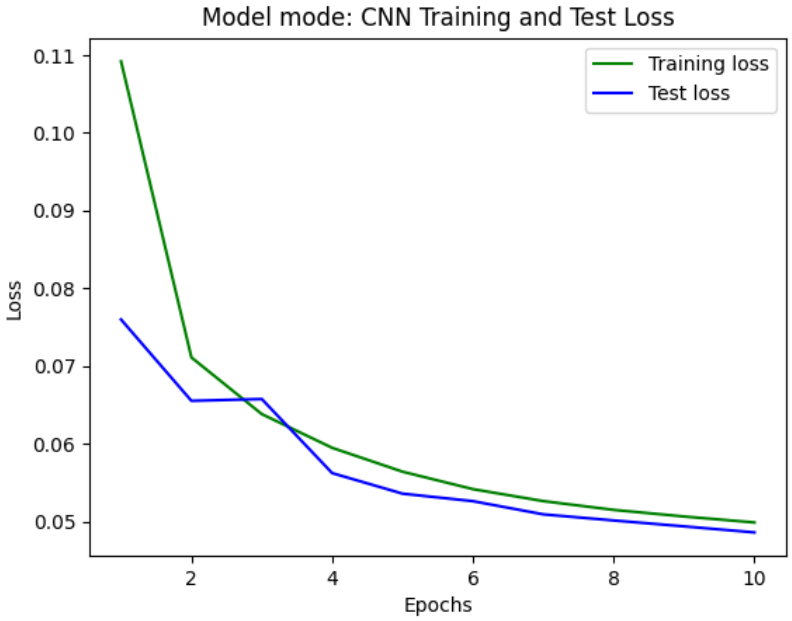

Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output


Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output
Input
Output

6. **Loss plots:**
    a. **Fully Connected:**



    b. **CNN:**



7. **Discussion:** We observe that the CNN based autoencoder is able to achieve similar visual performance as the fully connected autoencoder with only **2.27%** of its parameters.

II. Part-2 KNN

1. Introduction:

The K-Nearest Neighbors (KNN) algorithm is a versatile and straightforward machine learning approach employed for both classification and regression tasks. Its fundamental principle rests on the assumption that similar data points share similar labels or values. When tasked with predicting the label or value for a new, unlabeled data point, KNN identifies the k nearest neighbors from the training set, employing a chosen distance metric, commonly Euclidean distance (L2-norm). For classification, the algorithm assigns the majority class among these neighbors to the new data point, while for regression, it calculates the average (or weighted average) of the values of the k nearest neighbors. The choice of the hyperparameter "K" is pivotal and depends on the specific characteristics of the dataset and the nature of the problem at hand. Despite its simplicity and ease of implementation, KNN does have limitations, such as being computationally expensive for large datasets and sensitivity to irrelevant or redundant features.

In this experiment, we will compare performance for K=3, 5, 7.

2. Dataset:

The scikit-learn "Digits" dataset is a built-in collection of 8x8 pixel grayscale images representing handwritten digits (0 through 9). Each image is represented as an 8x8 matrix, with each pixel capturing the grayscale intensity of the digit at that location. The dataset is commonly used for practicing and testing machine learning classification algorithms. It consists of 1,797 samples, where each sample corresponds to an image, and each image is accompanied by a label indicating the digit it represents. This dataset is particularly valuable for tasks such as digit recognition, making it a widely used resource for individuals learning and experimenting with machine learning techniques, especially in the context of image classification and pattern recognition.

3. Results and Discussion:

K	Accuracy (%)
3	99.6%
5	99.4%
7	99.2%

Table 1: KNN Classifier Performance

We observe that as the value of K increases, there is a slight dip in performance.