Stephen J. R. Smith Faculty of Engineering & Applied Science – Mechatronics & Robotics

ELEC 475: Computer Vision with Deep Learning

Lab 1 – MLP Autoencoder

Aria Maz | 20278460

September 24, 2024

Model Details

The **autoencoderMLP4Layer** is a 4-layer Multi-Layer Perceptron (MLP) autoencoder that takes in 28x28 pixel images (from the MNIST dataset), compresses them into a smaller "bottleneck" representation, and then reconstructs the image back to its original size. Below is a complete textual description of the model that provides enough information to fully recreate it.

Layer-by-Layer Model Architecture Overview

1. **Input Layer**
   - **Input Size**: 28 × 28 = 784 pixels
     - The input is a flattened vector of size 784, representing the 28x28 pixel grayscale image.
2. **Encoder**
   - **First Fully Connected Layer (fc1)**
     - **Input**: A vector of size 784 (the flattened image).
     - **Output**: A vector of size 392.
       - This reduces the size by half.
     - **Activation Function**: ReLU (Rectified Linear Unit).
   - **Second Fully Connected Layer (fc2)**
     - **Input**: The output from the first layer (size 392).
     - **Output**: A bottleneck vector of size 8.
       - The most compressed form of the input data.
     - **Activation Function**: ReLU.
3. **Bottleneck Layer**
   - **Size**: 8
     - This layer holds the compressed latent representation of the image.
4. **Decoder**
   - **Third Fully Connected Layer (fc3)**
     - **Input**: The bottleneck vector (size 8).
     - **Output**: A vector of size 392.
       - This expands the data back toward the original size.
     - **Activation Function**: ReLU.
   - **Fourth Fully Connected Layer (fc4)**
     - **Input**: The output from the third layer (size 392).
     - **Output**: A vector of size 784 (matches the original image size)
     - **Activation Function**: Sigmoid.
5. **Output Layer**
   - **Output Size**: 784 (28x28 pixels)
     - The output is a flattened vector representing the reconstructed image.
   - **Pixel Values**: Between 0 and 1 (as a result of the sigmoid activation).

Forward Pass

The **forward()** method defines how data flows through the model:

1. The input image is passed through the **encoder** (fc1, fc2) to generate the bottleneck representation.
2. The bottleneck representation is then passed through the **decoder** (fc3, fc4) to reconstruct the image.
3. The final output is the reconstructed image with pixel values between 0 and 1.

Summary From Torchummary:

```
==========================================================================
Layer (type:depth-idx)        Output Shape              Param #
==========================================================================
├─Linear: 1-1                 [-1, 1, 392]              307,720
├─Linear: 1-2                 [-1, 1, 8]                3,144
├─Linear: 1-3                 [-1, 1, 392]              3,528
├─Linear: 1-4                 [-1, 1, 784]              308,112
==========================================================================
Total params: 622,504
Trainable params: 622,504
Non-trainable params: 0
Total mult-adds (M): 0.62
==========================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 2.37
Estimated Total Size (MB): 2.39
==========================================================================
```

Training Details

Below is a complete textual description of the training that provides enough information exactly reproduce the training process of the autoencoder.

Optimization Method: Adam Optimizer

- **Learning Rate (lr)**: 1e-3 (0.001).
- **Weight Decay (weight_decay)**: **1e-5** (0.00001).
- **Momentum**: No explicit momentum parameter.
    - Adam automatically adjusts learning rates based on first and second moments of the gradients.

Loss Function: MSELoss (Mean Squared Error).

- **Reduction**: Mean
    - The loss will be averaged over the entire batch.

Learning Rate Scheduler: ReduceLROnPlateau

- **Monitoring**: The scheduler monitors the **training loss**.
    - 'min' indicates it reduces the learning rate when the minimum loss doesn't decrease.

Number of Epochs (n_epochs): The model is trained for **50 epochs**.
Batch Size (batch_size): The batch size is set to **2048**.

Dataset: The model is trained on the **MNIST dataset** (contains 60,000 training images).
DataLoader: The data is **Shuffled** at the start of each epoch.

Model: The model used is an **autoencoder** with a 4-layer Multi-Layer Perceptron (MLP) architecture.
Weight Initialization: The weights are initialized using **Xavier uniform initialization**
    - The biases are initialized to **0.01**.

Device: GPU (if available), otherwise CPU

Results

The system worked as expected, with the MLP Autoencoder successfully trained on the MNIST dataset. Below are some observations on the training and performance of the model, including the loss curve plot and results from testing the model.

Loss Curve Behavior

The loss curve plot shown in **Figure 1** portrays the training loss of the autoencoder model over 50 epochs. The curve follows a typical pattern of a well-trained model. Initially, the loss starts high and drops steeply within the first few epochs, indicating that the model is learning rapidly and has an effective gradient-based optimization. As the epochs progress, the rate of decrease slows and plateaus, reflecting that the model is fine-tuning its weights and converging towards an optimal solution.

In this case, the final loss at the 50th epoch reaches a value at about 0.02134, which shows that the model has successfully minimized the reconstruction error between the input and output images. The steady decline without significant spikes or fluctuations suggests stable training and a learning rate scheduler (ReduceLROnPlateau) that is effectively helping the model avoid overfitting or stagnation.

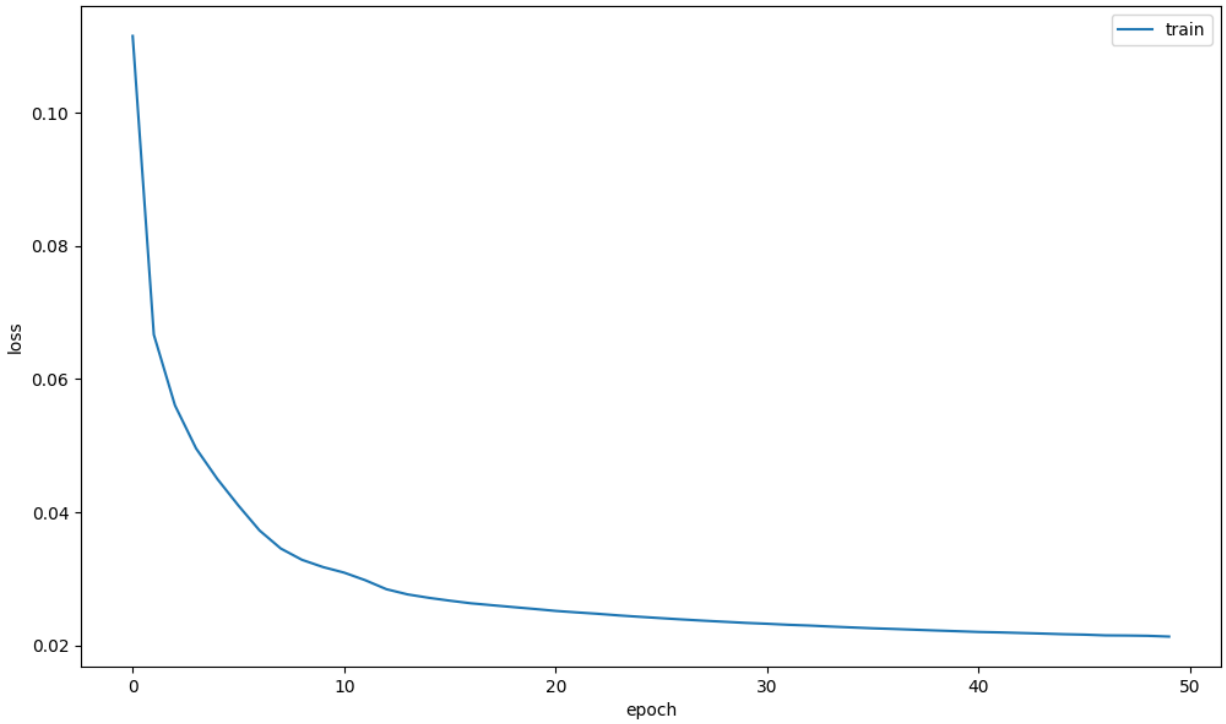

Figure 1: Loss curve plot over 50 epochs during training.

Reconstruction Results

The model performed well in reconstructing images from the MNIST dataset. When an image was passed through the autoencoder, the output was visually similar to the original input, showing that the model was able to retain most details in the reconstructed images. Some minor blurring or loss of fine details were observed in certain reconstructions, but generally the model had successfully learned to compress and reconstruct the data through its bottleneck layer. An example of a reconstruction can be seen in **Figure 2** and **3** below.

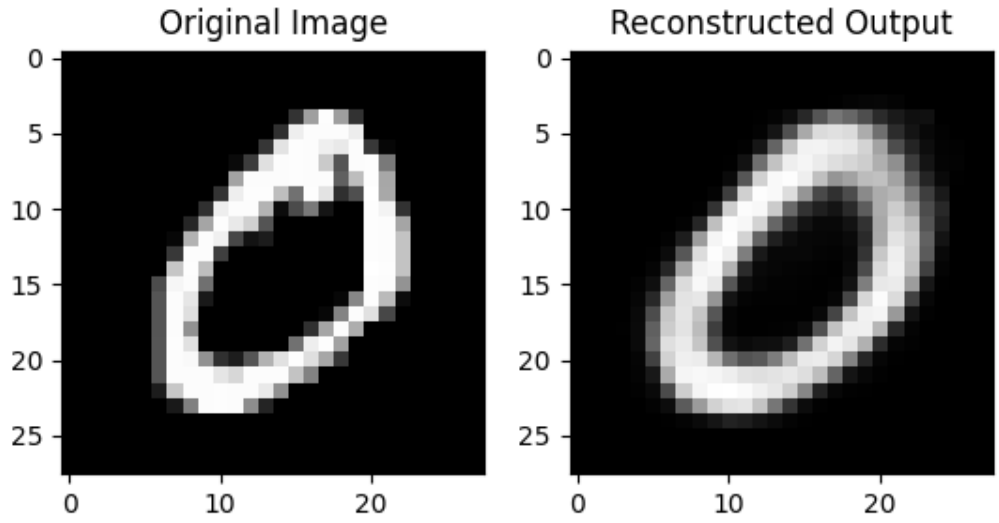

Figure 2: Reconstruction results for index 1.

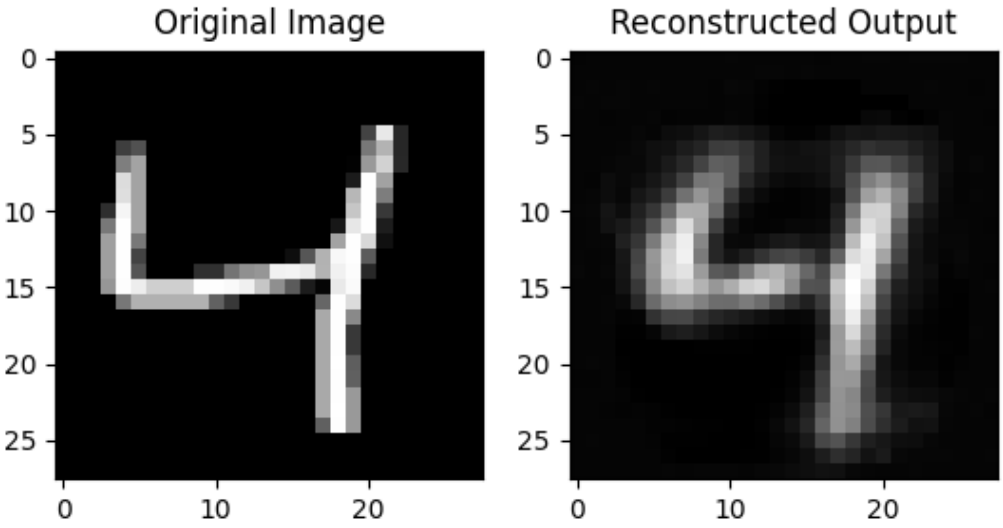

Figure 3: Reconstruction results for index 2.

Denoising Results

The autoencoder was also tested on its ability to denoise noisy images. Random noise was added to the MNIST images, and the autoencoder successfully removed much of the noise, producing clearer, denoised outputs. While some residual noise remained in a few cases, the model generally performed well in restoring the original image from the noisy input. An example of denoising can be seen in **Figure 4** and **5** below.

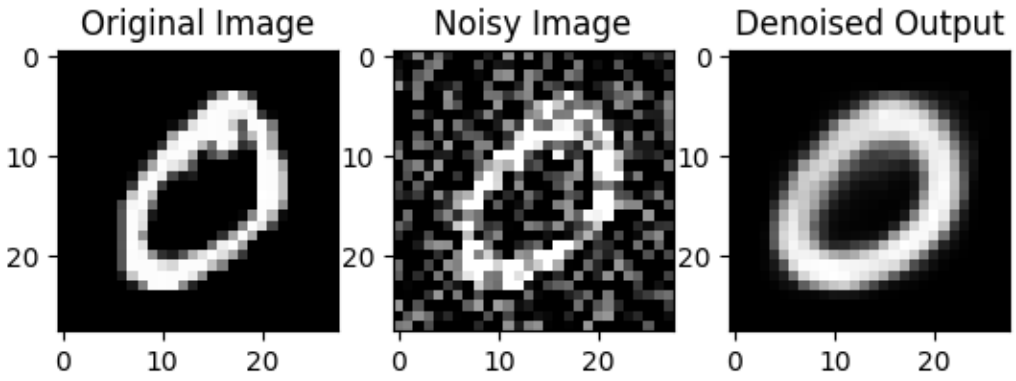

Figure 4: Denoising results for index 1.

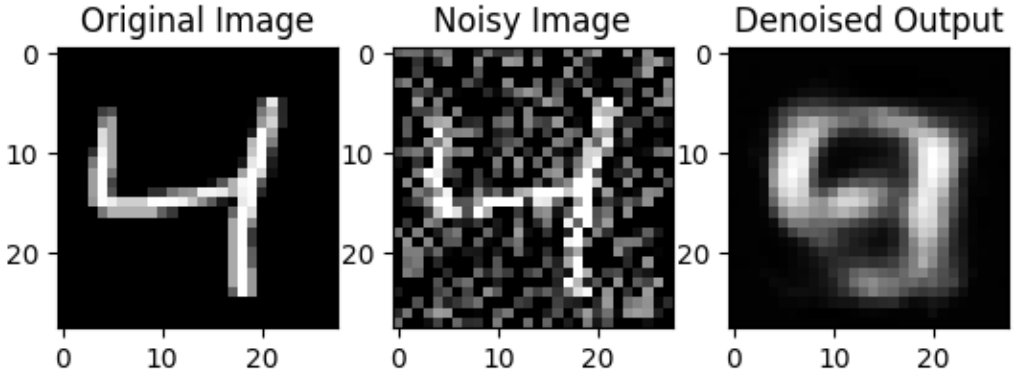

Figure 5: Denoising results for index 1.

Bottleneck Interpolation Results

The transitions between the two images were smooth, and the interpolated images showed gradual shifts from the first image chosen from the dataset to the second image chosen from the dataset. This demonstrated that the autoencoder had learned a compact, continuous representation of the data in its bottleneck layer. An example of bottleneck interpolation can be seen in **Figure 6** below.

Figure 6: Bottleneck Interpolation results from index 1 to index 2.