CS-515 HW3 Report

1.1 Linear Autoencoder

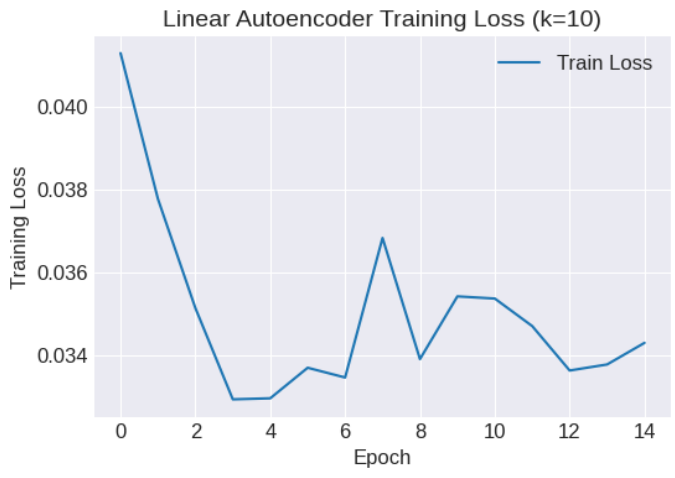
In this report, I present the results of training a linear autoencoder on the MNIST dataset with different values of k. The autoencoder is implemented using PyTorch and trained using the mean squared error (MSE) loss function. The linear autoencoder architecture consists of an encoder with a linear layer of k units and a decoder with a linear layer of input_size (784) units. The autoencoder is trained for 15 epochs with the Adam optimizer (learning rate = 0.001). I experimented with three values of k: 10, 50, and 100.

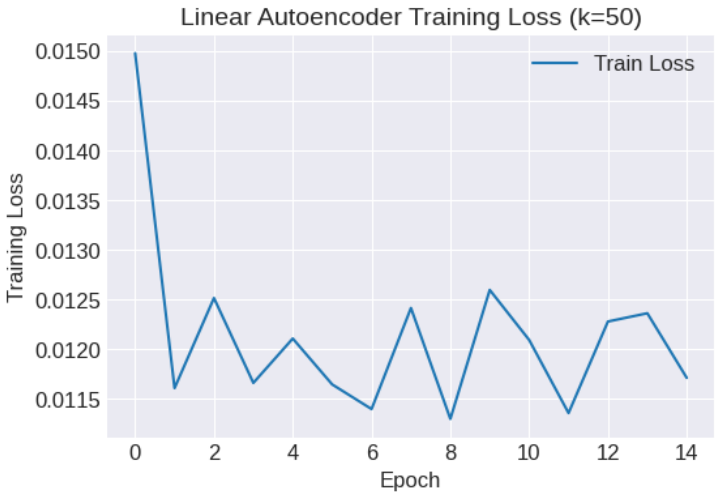
Results:

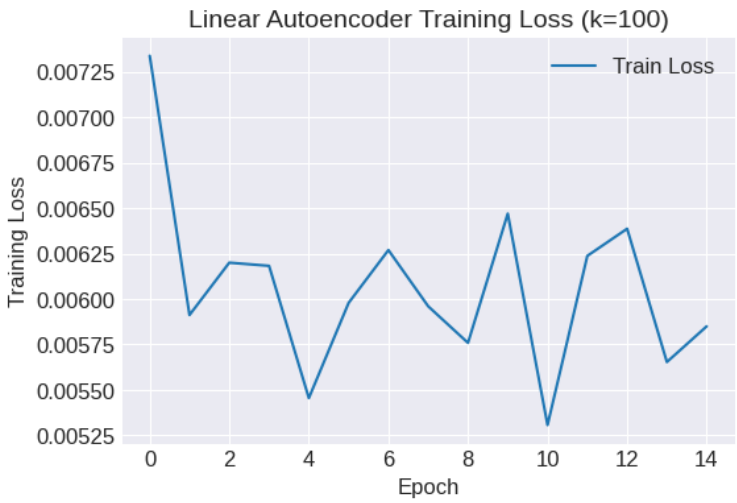
A linear autoencoder model was implemented and trained using various values for the bottleneck layer size (k). The test loss, measured using Mean Squared Error (MSE), decreased as the value of k increased. Specifically, for k=10, the test loss was 0.0341, while for k=50, it was 0.0116, and for k=100, the test loss further reduced to 0.0058. These results indicate that as the size of the bottleneck layer increases, the autoencoder's ability to reconstruct the input data improves, leading to a lower reconstruction error. However, it is essential to consider that a larger bottleneck layer may result in increased computational complexity and potentially overfitting. Therefore, it is crucial to find a balance between the size of the bottleneck layer and the desired reconstruction performance to avoid overfitting and maintain a computationally efficient model. Train and Validation loss in Table 1 and other tables in this report show the final values after traning.

Table 1

Linear-Autoencoder			
k	**Train Loss**	**Validation Loss**	**Test Loss**
10	0.0343	0.0338	0.0341
50	0.0117	0.0121	0.0116
100	0.0058	0.0061	0.0058

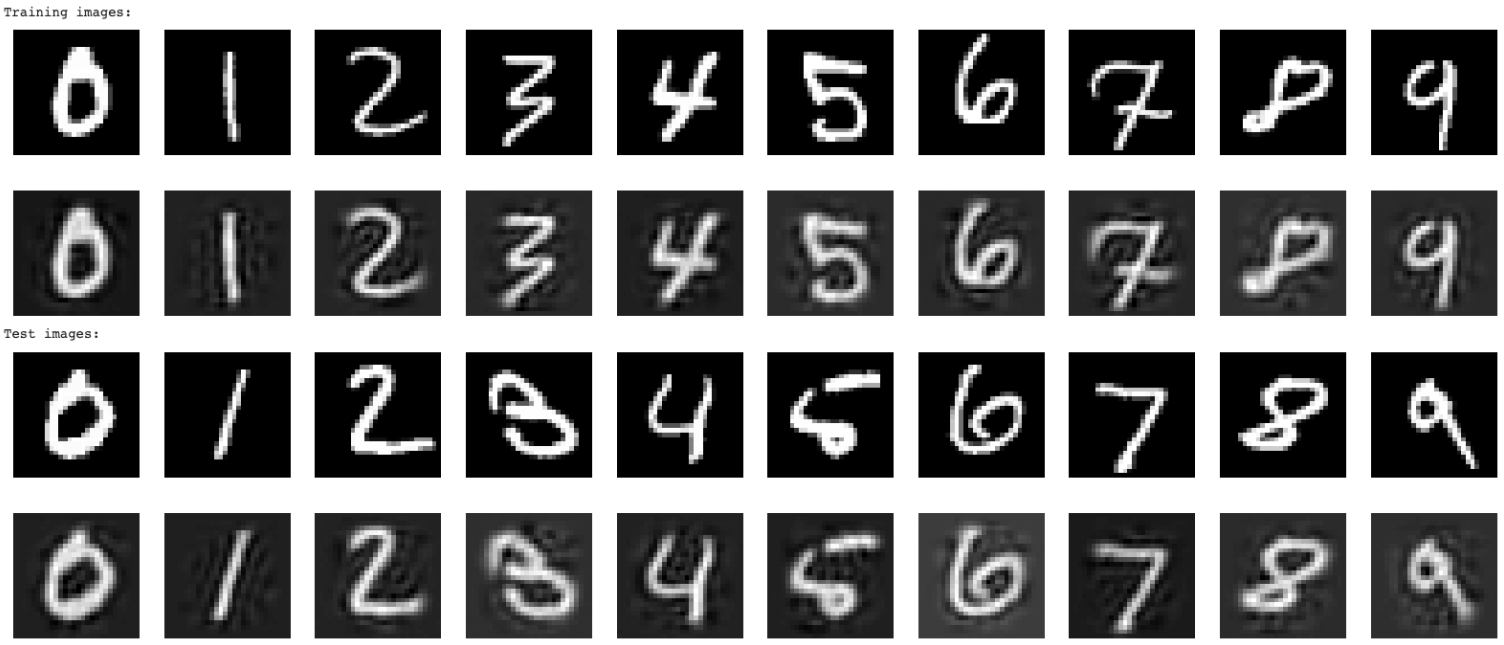






The results indicate that as the value of k increases, the autoencoder achieves better performance, as evidenced by the lower training and validation losses. This suggests that increasing the number of units in the hidden layer of the autoencoder allows the model to learn more complex representations of the input data, leading to better reconstruction performance. In this experiment, I observed that a linear autoencoder with more hidden units (higher k value) performs better in reconstructing the MNIST dataset. However, it is important to consider the trade-off between the complexity of the model and the risk of overfitting, especially when applying the autoencoder to other datasets or applications.

1.2 Traning and Test images



In the experiment, I aimed to evaluate the performance of a linear autoencoder with a chosen latent space dimension of k=100 in reconstructing images of handwritten digits. I selected 10 training images and 10 test images, each representing one unique digit from 0 to 9. After encoding and decoding these images using my trained autoencoder, I visualized the original and reconstructed images side-by-side. As observed from the resulting graphs, the reconstructed images closely resemble the original images, indicating that my autoencoder model effectively captured the critical features of the handwritten digits. This demonstrates that the chosen linear autoencoder with k=100 maintains an adequate balance between dimensionality reduction and information preservation, allowing for efficient compression and reconstruction of the input images.

1.3 Nonlinear Autoencoder

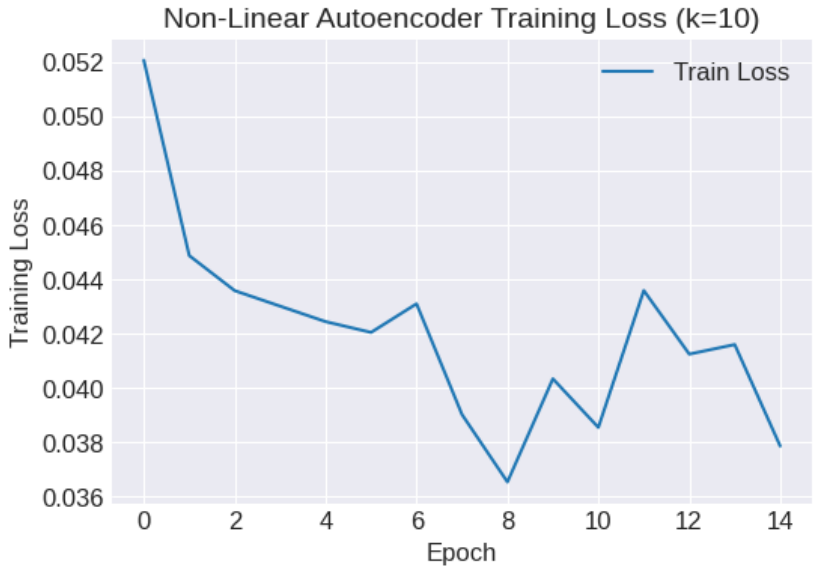
The autoencoder architecture in this part consists of an encoder and a decoder. The encoder has a single linear layer followed by a ReLU activation function, whereas the decoder has a single linear layer followed by a sigmoid activation function. The model was trained using the mean squared error (MSE) loss and Adam optimizer with a learning rate of 0.001. The autoencoder was trained for three different values of k (10, 50, 100) for 15 epochs each.

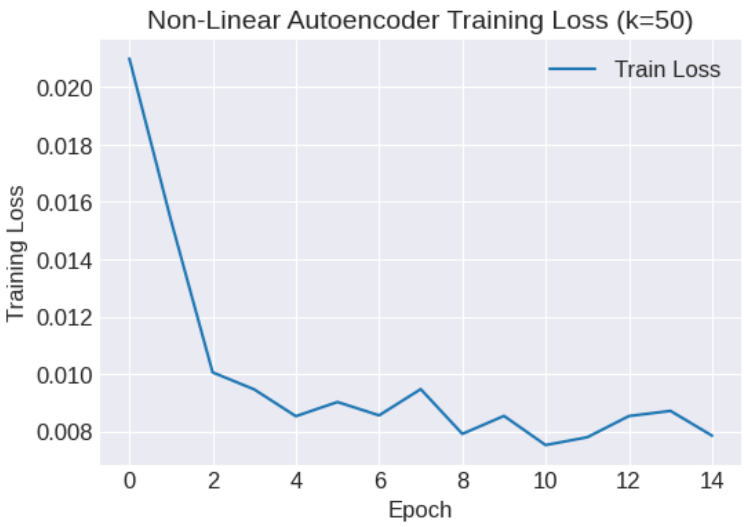
Results:

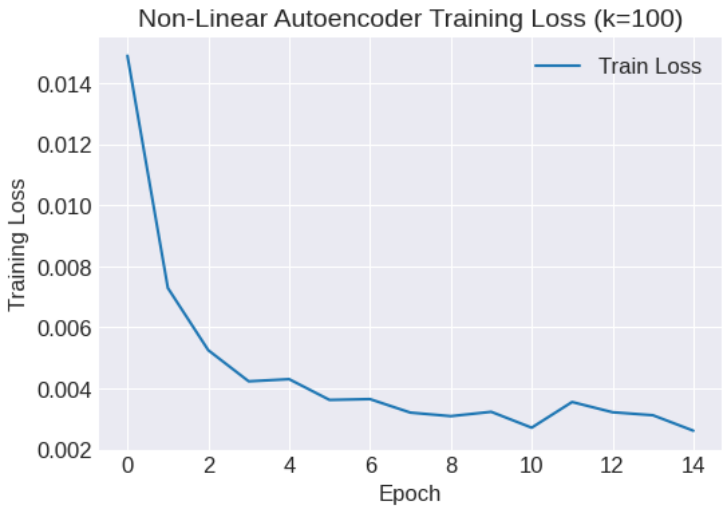
In this new implementation, a nonlinear autoencoder model has been created by introducing non-linear activation functions in both the encoder and decoder sections. The encoder now has a ReLU activation function, while the decoder has a Sigmoid activation function. The test results demonstrate an improvement in the reconstruction performance of the nonlinear autoencoder when compared to the linear autoencoder for each k value as it can be seen by table 2.

Table 2

Nonlinear-Autoencoder			
k	**Train Loss**	**Validation Loss**	**Test Loss**
10	0.0379	0.0385	0.0337
50	0.0079	0.0081	0.0076
100	0.0026	0.0029	0.0028







These results show that introducing nonlinearity into the autoencoder has resulted in a better ability to reconstruct input data, leading to lower reconstruction errors. The nonlinear autoencoder can capture more complex patterns in the data, making it a more effective model for this specific problem than a linear one. The training loss and validation loss for all k values decreased over the epochs. The lowest validation loss was obtained for k = 100, which suggests that a higher-dimensional latent space yields better reconstruction performance. The non-linear autoencoder was able to achieve reasonable reconstruction performance on the MNIST dataset. Increasing the latent space dimension (k) led to lower validation loss, indicating better reconstruction capabilities.

In general, adding non-linear activation functions, like ReLU and sigmoid, can increase the expressive power of the autoencoder model. This allows the model to learn more complex representations and can improve reconstruction performance. In the linear case, the model can only learn linear transformations of the input data. However, real-world data is often non-linear, and thus, a linear autoencoder might not be able to capture the underlying patterns effectively. By adding non-linear activation functions, the autoencoder can learn non-linear mappings of the input data, which might lead to better reconstructions. It's important to note that the improvement in performance is not guaranteed. The actual improvement depends on the dataset, complexity of the underlying patterns, and the chosen architecture of the autoencoder.

1.4 Stacked Autoencoder

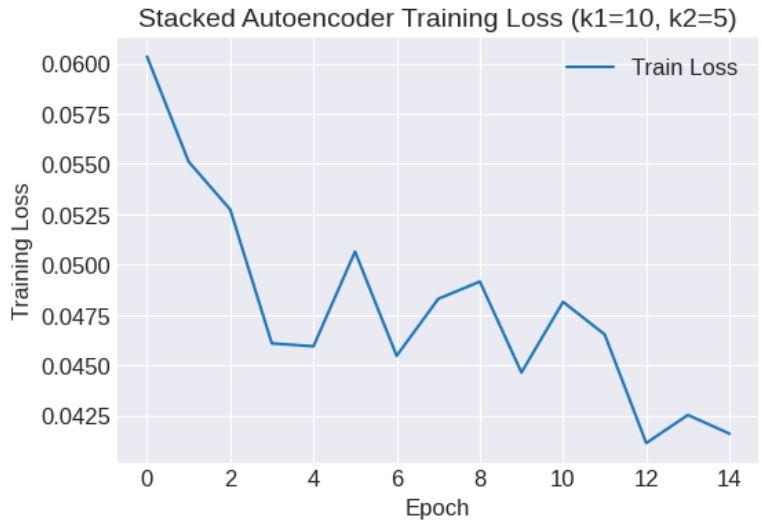
I implemented a stacked autoencoder using PyTorch with an input size of 784, corresponding to the flattened 28x28 pixel images in the MNIST dataset. The autoencoder consisted of an encoder and a decoder with ReLU activations between the linear layers. I experimented with three different hidden layer size configurations: (10, 5), (50, 25), and (100, 50). The autoencoder was trained using the Mean Squared Error (MSE) loss function and the Adam optimizer with a learning rate of 0.001. I trained the model for 15 epochs and recorded the training and validation losses at each epoch.

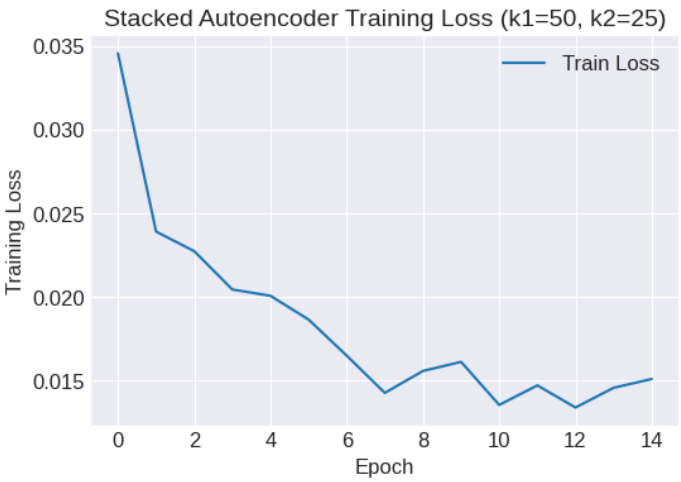
Results

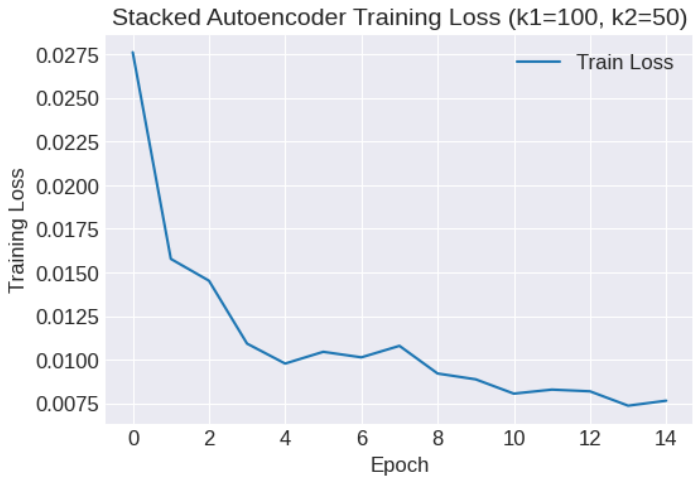
I created a stacked autoencoder model with two hidden layers in both the encoder and decoder sections. The non-linear activation functions, ReLU and Sigmoid, are still used for the encoder and decoder, respectively. The model is trained and tested using different combinations of k1 and k2 values, where k1 represents the size of the first hidden layer and k2 represents the size of the second hidden layer (the bottleneck layer).
The test results for the stacked autoencoder are as follows

Table 3

Stacked-Autoencoder			
k	**Train Loss**	**Validation Loss**	**Test Loss**
10	0.0416	0.0410	0.0412
50	0.0151	0.0133	0.0133
100	0.0076	0.0073	0.0070







Based on the results, as the hidden layer sizes (k1 and k2) increased, both the training and validation losses decreased, indicating that the autoencoder performance improved. This observation suggests that larger hidden layers can learn more complex representations of the data, leading to better reconstruction capabilities. The validation loss curves for all configurations show a consistent downward trend, demonstrating that the autoencoder is generalizing well to unseen data. This is a positive sign, as it indicates the model is not overfitting during training.

Comparing these results with those from the previous nonlinear autoencoder, the stacked autoencoder has slightly higher reconstruction errors for all tested configurations. This could be attributed to the fact that the stacked architecture may require more training epochs or better hyperparameter tuning to achieve better performance. The results also suggest that introducing additional hidden layers may not necessarily lead to better performance, and choosing the right model architecture is crucial to find an optimal balance between complexity and reconstruction performance.