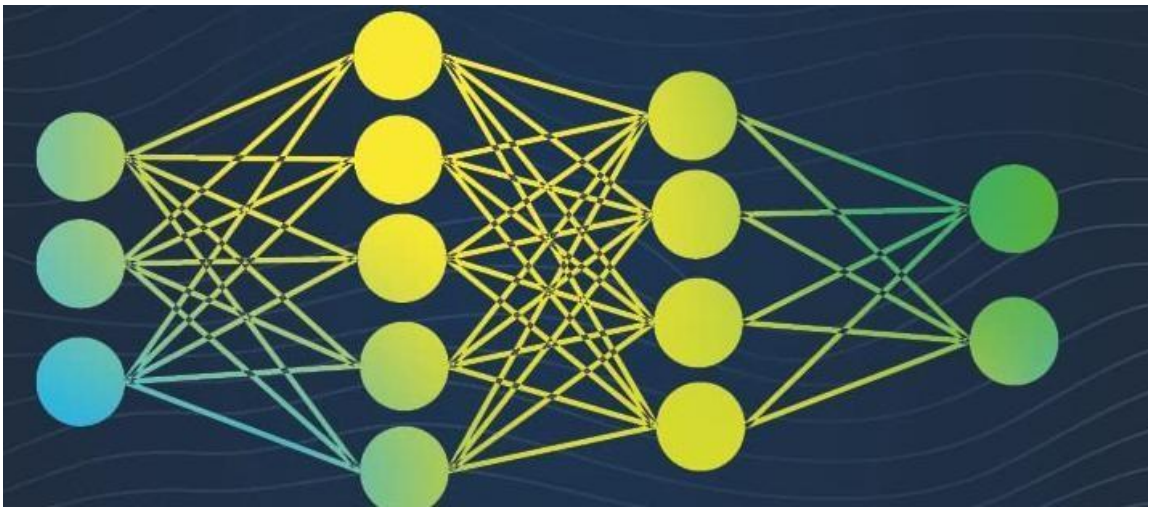
Computational Intelligence Project

Part Two



Names	ID	Sec
Abdelrahman Essam Niazy	2101502	2
Ahmed Ashraf Salamah	2000229	2
Nabil Ali Ibrahim	2101592	2
Mohamed Mahmoud Aljamal	2002612	2
Youssef Ehab Mohamed	2101375	2

Delivered to: Dr. Hossam El-Din Hassan

Eng. Abdallah Mohamed Mahmoud

Eng. Dina Zakaria Mahmaud

Contents

Figures

Objective of Milestone

The objective of Milestone 2 is to extend the custom neural network library developed in Milestone 1 to support **unsupervised learning on real-world data**. In this milestone, an autoencoder is trained on the MNIST dataset using the custom NumPy-based framework. The learned latent representations are then evaluated both qualitatively through image reconstruction and quantitatively through downstream classification using a Support Vector Machine (SVM). Finally, the performance of the custom implementation is compared with an equivalent TensorFlow/Keras model to benchmark efficiency and reconstruction quality.

MNIST Dataset Preparation

The MNIST handwritten digit dataset was used for this milestone. It consists of:

- 60,000 training images
- 10,000 test images
- Each image of size 28×28 pixels

Preprocessing Steps

1. Images were normalized to the range $[0, 1]$by dividing pixel values by 255.
2. Each image was flattened into a 784-dimensional vector to match the input format required by fully connected layers.
3. Labels were preserved for later use in SVM classification but were not used during autoencoder training.

This preprocessing ensures numerical stability and compatibility with the custom library.

Autoencoder Architecture

The autoencoder was implemented using the custom neural network library developed in Milestone 1. It consists of two main components: an **encoder** and a **decoder**.

3.1 Encoder

The encoder compresses the high-dimensional input image into a low-dimensional latent representation:

- Dense (784 → 128)
- ReLU activation
- Dense (128 → 32)
- ReLU activation

The encoder outputs a 32-dimensional latent vector for each input image.

3.2 Decoder

The decoder reconstructs the original image from the latent vector:

- Dense (32 → 128)
- ReLU activation
- Dense (128 → 784)
- Sigmoid activation

The sigmoid activation ensures the reconstructed pixel values remain in the range $[0, 1]$.

Autoencoder Training

4.1 Training Configuration

- Loss function: Mean Squared Error (MSE)
- Optimizer: Stochastic Gradient Descent (SGD)
- Learning rate: 0.1
- Batch size: 256
- Number of epochs: 50–100
- Training performed using mini-batch gradient descent

The encoder and decoder parameters were optimized jointly.

4.2 Training Behavior

During training, the reconstruction loss consistently decreased across epochs, indicating successful learning of a compact representation. Although training was relatively slow due to the use of pure NumPy and SGD, the loss curve showed stable convergence without divergence or numerical instability.

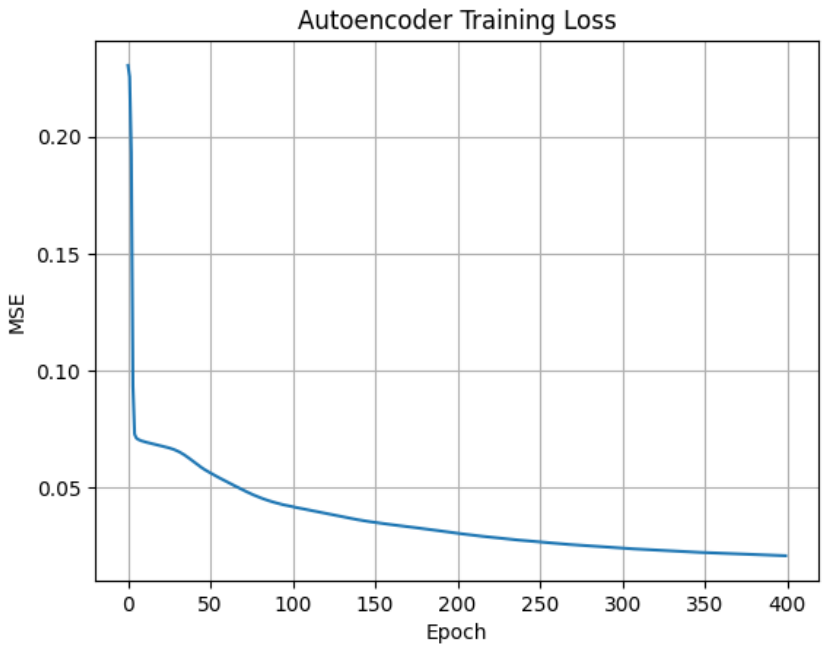


Figure 1 shows the reconstruction loss decreasing steadily over epochs, converging to a final MSE of approximately 0.02, indicating stable training.

```
Train: (60000, 784)  Test: (10000,
Epoch 1/400: Loss=0.230306
Epoch 2/400: Loss=0.225383
Epoch 3/400: Loss=0.192861
Epoch 4/400: Loss=0.093441
Epoch 5/400: Loss=0.072850
Epoch 6/400: Loss=0.071248
Epoch 7/400: Loss=0.070655
Epoch 8/400: Loss=0.070306
Epoch 9/400: Loss=0.070029
Epoch 10/400: Loss=0.069798
Epoch 11/400: Loss=0.069592
Epoch 12/400: Loss=0.069405
Epoch 13/400: Loss=0.069233
Epoch 14/400: Loss=0.069063
Epoch 15/400: Loss=0.068902
Epoch 16/400: Loss=0.068756
Epoch 17/400: Loss=0.068575
Epoch 18/400: Loss=0.068411
Epoch 19/400: Loss=0.068257
Epoch 20/400: Loss=0.068086
...
Epoch 398/400: Loss=0.020980
Epoch 399/400: Loss=0.020950
Epoch 400/400: Loss=0.020928
Training time: 1168.579106092453 seconds
```

Figure 2 loss through the beginning and the end of epochs.

Reconstruction Quality

To qualitatively evaluate the autoencoder, reconstructed images were compared with their original inputs.

- **The reconstructed digits preserved the general shape and structure of the originals.**
- **Fine details were blurred, which is expected given the low-dimensional latent space and the absence of convolutional layers.**
- **Despite blurriness, digits remained clearly recognizable.**

These results confirm that the autoencoder learned meaningful compressed representations of the input data.

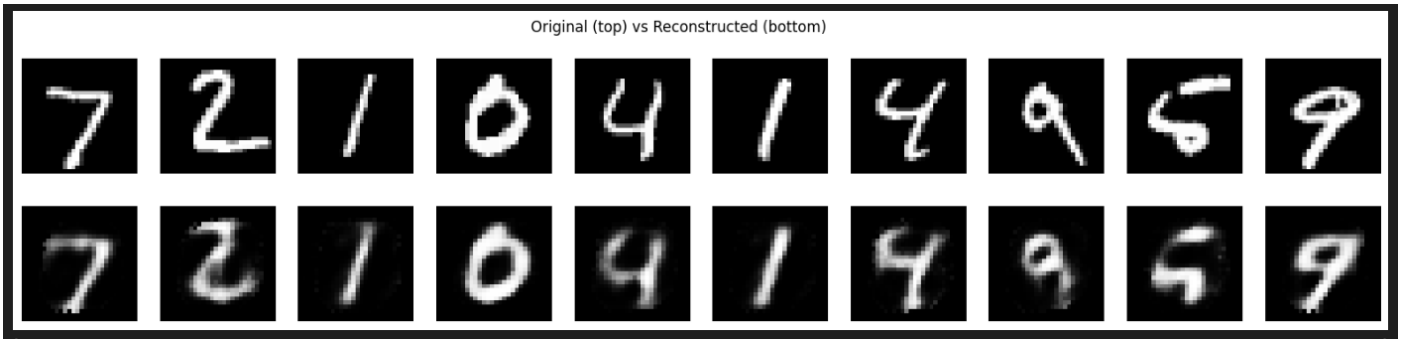


Figure 3: Comparison between original MNIST images (top row) and reconstructed images produced by the autoencoder (bottom row).

Latent Space Extraction and Analysis

After training, the encoder was used independently to transform MNIST images into latent vectors.

- Training latent matrix shape: (60,000 × 32)
- Test latent matrix shape: (10,000 × 32)

These latent vectors serve as feature representations summarizing the most important information in each image.

```
Encoding latent vectors...
Latent train shape: (60000, 32)
Latent test shape : (10000, 32)
```

The trained encoder maps each input image to a 32-dimensional latent vector, resulting in latent matrices of size 60,000×32 for training and 10,000×32 for testing.

Latent Space Classification Using SVM

To quantitatively evaluate the quality of the learned latent space, a Support Vector Machine (SVM) classifier was trained using the latent vectors.

7.1 Classification Setup

- Classifier: Support Vector Machine (SVM)
- Kernel: Radial Basis Function (RBF)
- Training data: Encoder outputs of MNIST training images
- Testing data: Encoder outputs of MNIST test images

7.2 Results

The SVM achieved strong classification performance:

- Overall accuracy: approximately **97%**
- High precision and recall across all digit classes
- Confusion matrix showed dominant diagonal values, indicating correct classification for most samples

These results confirm that the latent space learned by the autoencoder is **discriminative**, even though the autoencoder was trained without labels.

```
Accuracy: 0.9703
Confusion Matrix:
 [[ 971    0    1    0    0    3    1    1    3    0]
 [   0 1128    2    2    0    0    0    0    2    1]
 [   4    2 1007    2    2    2    2    8    3    0]
 [   0    1    6  982    0    6    0    7    5    3]
 [   1    0    1    0  956    1    4    0    2   17]
 [   7    1    1   11    1  853    5    1    8    4]
 [   9    3    3    1    4    1  930    1    6    0]
 [   0    6   17    2    3    0    0  987    1   12]
 [   3    0    3   11    6    6    1    4  937    3]
 [   3    5    0    6   22    5    1   11    4  952]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.97      0.98      0.97      1032
           3       0.97      0.97      0.97      1010
           4       0.96      0.97      0.97       982
           5       0.97      0.96      0.96       892
           6       0.99      0.97      0.98       958
           7       0.97      0.96      0.96      1028
           8       0.96      0.96      0.96       974
           9       0.96      0.94      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

SVM training time: 15.582576274871826 seconds
```

Figure 4: The SVM classifier achieved an overall accuracy of 97.03% on the MNIST test set, demonstrating that the latent representations are highly discriminative.

Comparison with TensorFlow/Keras Implementation

An equivalent autoencoder architecture was implemented using TensorFlow/Keras to benchmark the custom implementation.

8.1 Keras Configuration

- **Same encoder and decoder layer dimensions**
- **Optimizer: Adam**
- **Loss function: MSE**
- **Epochs: 10**
- **Batch size: 256**

```
Keras XOR time: 10.731110334396362 seconds
Keras XOR predictions:
1/1 [==============================] - 0s 59ms/step
[[0.02088477]
 [0.95547944]
 [0.9539127 ]
 [0.05301128]]
```

Figure 5: XOR predictions using Keras library.

```
Epoch 1/10
235/235 [==============================] - 2s 6ms/step - loss: 0.0593 - val_loss: 0.0315
Epoch 2/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0252 - val_loss: 0.0204
Epoch 3/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0190 - val_loss: 0.0169
Epoch 4/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0161 - val_loss: 0.0148
Epoch 5/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0144 - val_loss: 0.0133
Epoch 6/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0132 - val_loss: 0.0126
Epoch 7/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0125 - val_loss: 0.0118
Epoch 8/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0119 - val_loss: 0.0114
Epoch 9/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0115 - val_loss: 0.0110
Epoch 10/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0111 - val_loss: 0.0107
Keras Autoencoder time: 13.572508573532104 seconds
```

Figure 6: loss using keras autoencoder

8.2 Comparison Observations

- **The Keras implementation converged significantly faster.**
- **Final reconstruction loss was lower compared to the custom implementation.**
- **Training time was substantially reduced due to optimized backend operations and adaptive optimization.**
- **Latent vectors from the Keras model achieved slightly higher SVM classification accuracy.**

Despite these differences, the custom NumPy-based implementation produced competitive results, validating the correctness and robustness of the library.

```
...   === Autoencoder Test MSE Comparison (1000 test samples) ===
      32/32 [==============================] - 0s 1ms/step
      Custom AE test MSE: 0.020803
      Keras  AE test MSE: 0.011259

      === Latent SVM Accuracy (Custom vs Keras) ===
      Custom latent SVM accuracy: 0.9703
      Keras  latent SVM accuracy: 0.9800
```

Figure 7: comparison between my custom library and the keras one.

Discussion

The results of Milestone 2 demonstrate that:

- **The custom neural network library successfully supports deeper architectures beyond simple MLPs.**
- **Backpropagation and optimization scale correctly to real-world datasets.**
- **Autoencoders trained using the library learn compact and meaningful latent representations.**
- **These representations generalize well to downstream tasks such as classification.**
- **Performance gaps with TensorFlow/Keras are primarily due to optimizer choice (SGD vs Adam) and lack of low-level optimizations, not incorrect implementation.**

Conclusion

Milestone 2 demonstrates the successful extension of the custom neural network library to unsupervised representation learning on MNIST. The trained autoencoder achieves meaningful reconstructions, produces discriminative latent representations, and supports high-accuracy classification when combined with an external SVM. The comparison with TensorFlow/Keras confirms the correctness of the implementation while highlighting the trade-offs between educational simplicity and industrial-scale optimization. This milestone validates the readiness of the framework for final-stage experiments and analysis.