Laboratory 5

Variant 4

Class Group 105

Group 24

By Dimitrios Gkolias and Vasileios Ntalas

Introduction

In this lab project we are aiming to build a Multilayer Perceptron (MLP) for image classification on the data of the FashionMNIST dataset. We will evaluate how various components and hyperparameters of a neural network and the training process affect the network's performance in terms of its ability to converge, the speed of convergence, and final accuracy on the training and validation sets. We will experiment with these parameters and values of them:

- Learning rate: 0.001, 0.01, 0.1.
- Mini-batch size: 1, 64, 256.
- Number of hidden layers: 0, 1, 2.
- Width (number of neurons in hidden layers): 64, 128, 256.
- Loss functions: Mean Squared Error, Mean Absolute Error, Cross Entropy.

The way we are going to test the changes of each parameter is by starting with a basis of parameters to start, and specifically we start with:

→ *batch_size=64, hidden_layers=1, width=128, loss_name="CrossEntropy"* .

The number of epochs for every of our tests will be *num_epochs=25*.

We will start with these parameters which is the best learning rate for our model. Once we find it, we will keep that value for the learning rate and we will change the next parameter once we find the best value of it also. And we will proceed with the same tactic until we evaluate 3 different numbers/values/types of our parameters and end up to the best combination of values for the parameters. We will proceed each time with the value of the parameter that gives the best accuracy, even though it may have greater training time.

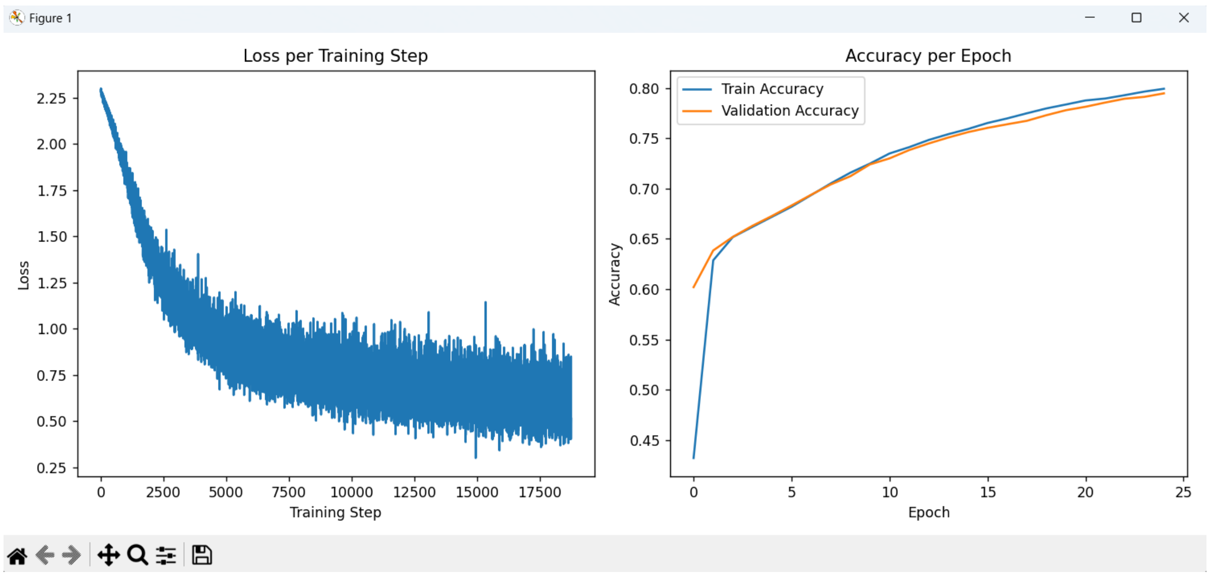
Implementation

Learning rate:

To determine the optimal learning rate for training our Multilayer Perceptron (MLP) model on the FashionMNIST dataset, we evaluated three different values: **0.001**, **0.01**, and **0.1**, while keeping the other hyperparameters fixed at:

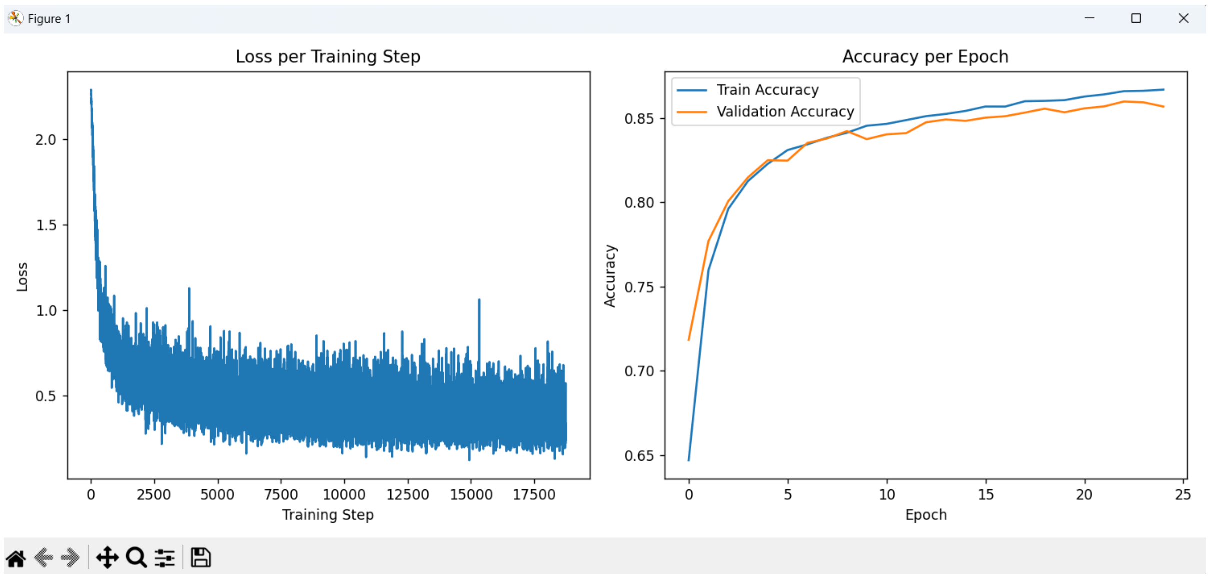
- Batch size: 64
- Hidden layers: 1
- Width: 128 neurons
- Loss function: CrossEntropy
- Number of epochs: 25

- Learning rate = 0.001:



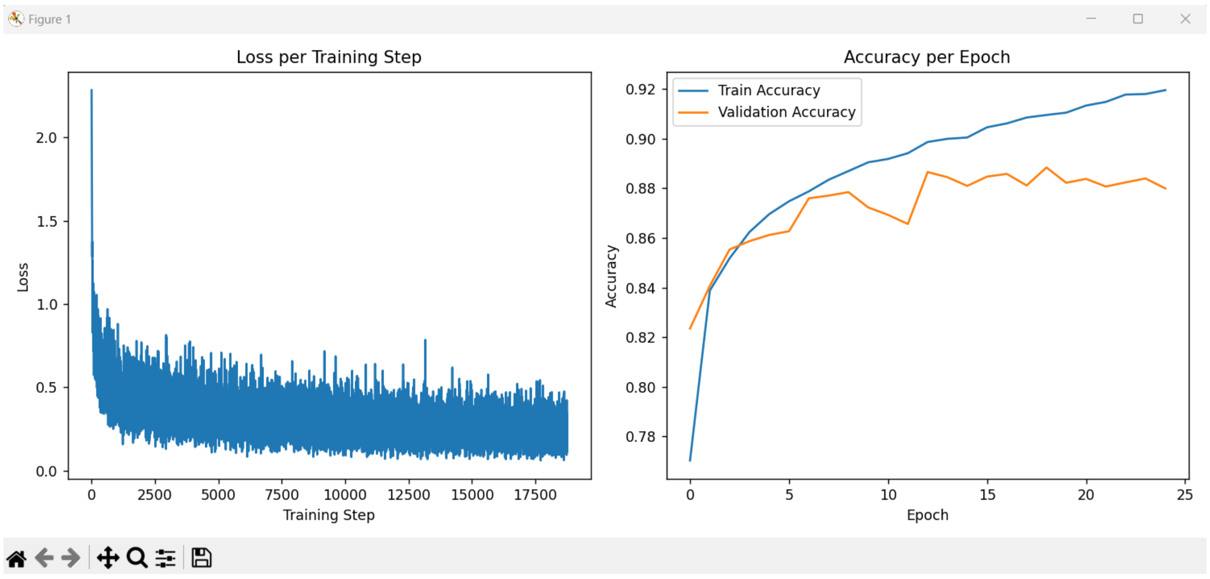
★ Training Accuracy: **<u>0.7992</u>**
★ Validation Accuracy: **<u>0.7947</u>**
★ Test Accuracy: **<u>0.7861</u>**
★ Training time: **<u>197.40 seconds</u>**

- Learning rate = 0.01:



- ★ Training Accuracy: **0.8667**
- ★ Validation Accuracy: **0.8567**
- ★ Test Accuracy: **0.8473**
- ★ Training time: **187.12 seconds**

- Learning rate = 0.1:



- ★ Training Accuracy: **0.9195**
- ★ Validation Accuracy: **0.8799**
- ★ Test Accuracy: **0.8718**
- ★ Training time: **262.33 seconds**

Learning Rate	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Training Time (s)**
0.001	0.7992	0.7947	0.7861	197.40
0.01	0.8667	0.8567	0.8473	187.12
0.1	0.9195	0.8799	**0.8718**	262.33

As shown in the results above, increasing the learning rate generally led to improved accuracy across training, validation, and test sets. A learning rate of **0.001** was too low, leading to slower learning and lower performance across all metrics. The model struggled to converge efficiently, likely due to very small updates to the weights during training.

A learning rate of **0.01** showed significantly better results, achieving higher accuracy with faster convergence compared to 0.001, and also required less training time.

The highest performance was observed at a learning rate of **0.1**, which produced the best accuracy scores on all datasets. However, this came at the cost of **higher training time**, suggesting the model required more epochs to stabilize due to larger step sizes in gradient descent, which might have caused initial overshooting before convergence.

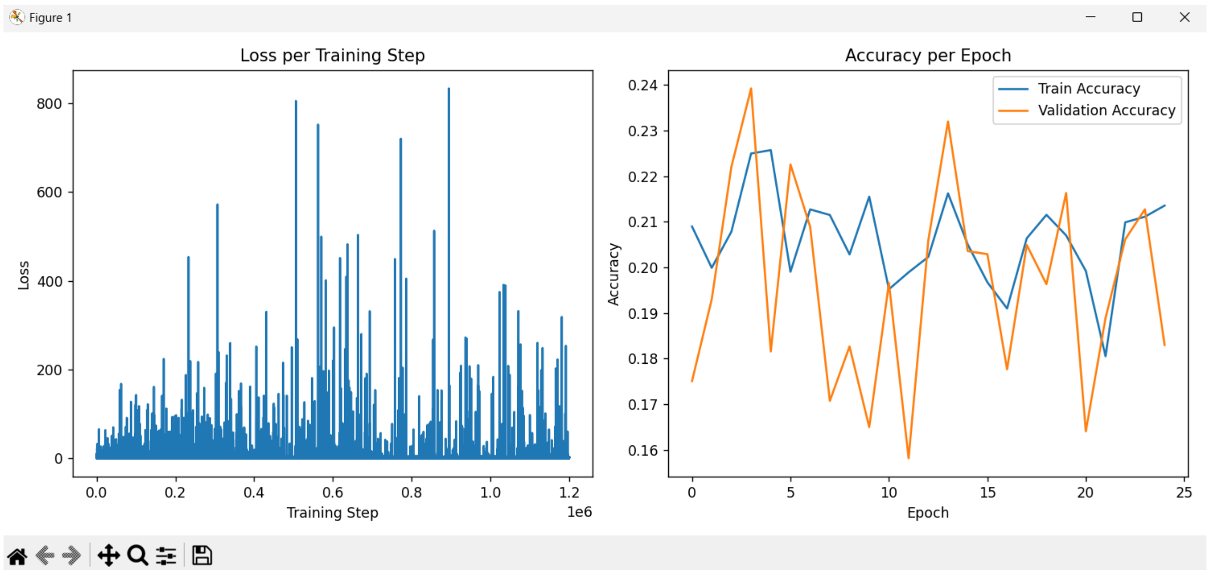
Based on the results, we selected **learning rate = 0.1** as the optimal choice for the subsequent experiments. It provided the best overall accuracy, despite the slightly increased training time, which is acceptable given the performance improvement.

Mini-batch size:

After determining that the best learning rate is **0.1**, we proceeded to evaluate the effect of different **mini-batch sizes** on training performance, keeping all other hyperparameters constant:

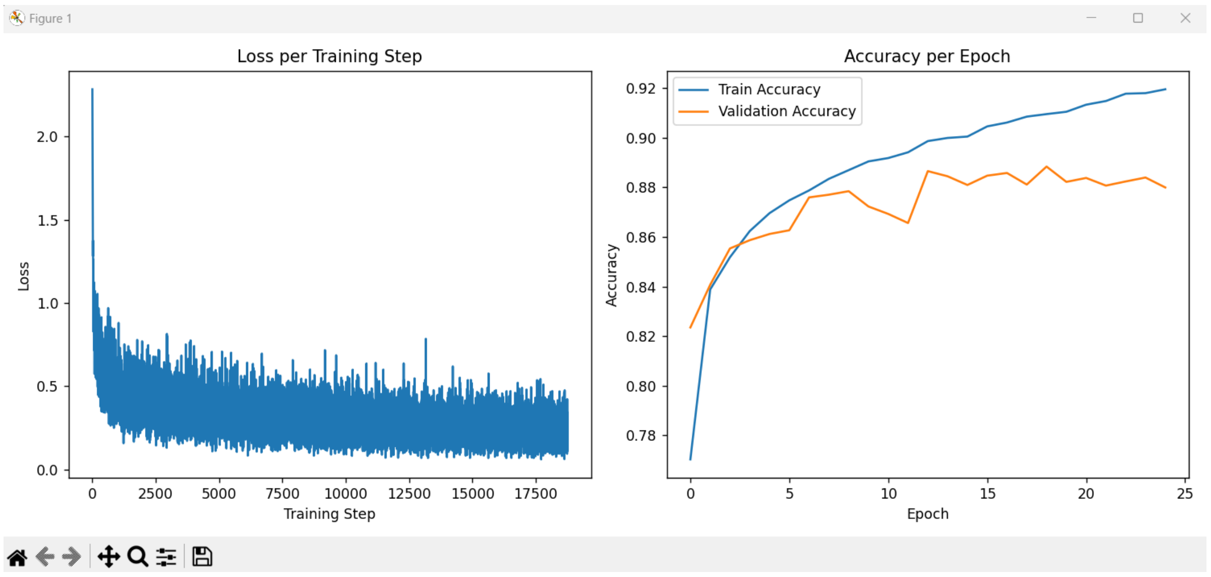
- Learning rate: 0.1
- Hidden layers: 1
- Width: 128 neurons
- Loss function: CrossEntropy
- Number of epochs: 25

- Mini-batch size = 1:



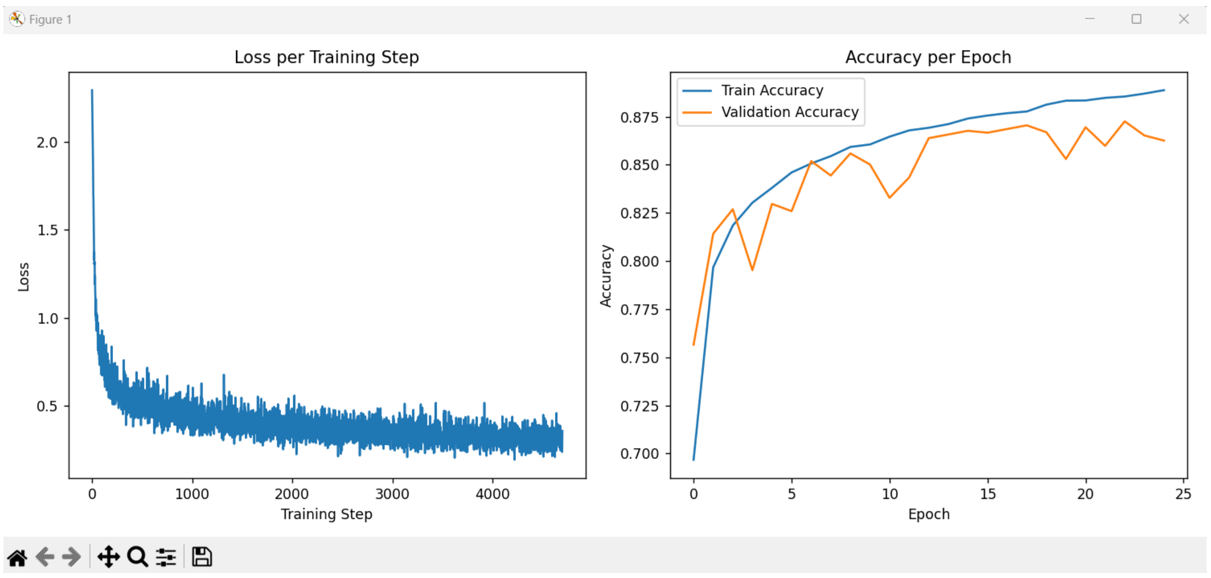
- ★ Training Accuracy: **0.2136**
- ★ Validation Accuracy: **0.1830**
- ★ Test Accuracy: **0.1830**
- ★ Training time: **1378.13 seconds**

- Mini-batch size = 64:



- ★ Training Accuracy: **0.9195**
- ★ Validation Accuracy: **0.8799**
- ★ Test Accuracy: **0.8718**
- ★ Training time: **343.92 seconds**

- Mini-batch size = 256:



★ Training Accuracy: **0.8888**
★ Validation Accuracy: **0.8626**
★ Test Accuracy: **0.8490**
★ Training time: **332.88seconds**

Mini-Batch Size	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Training Time (s)**
1	0.2136	0.1830	0.1830	1378.13
64	0.9195	0.8799	**0.8718**	343.92
256	0.8888	0.8626	0.8490	332.88

- **Mini-batch size of 1** (i.e., stochastic gradient descent) performed **very poorly** in terms of accuracy. The model did not converge properly, and all accuracy metrics remained around random guess levels (~18%). Additionally, the training time was extremely long (**1378 seconds**), due to the inefficiency of updating weights after every individual sample.
- **Mini-batch size of 64** provided the **best overall performance**, with the highest accuracy on training, validation, and test sets. It also achieved a good balance between performance and training time. The mini-batch strategy allowed for stable gradient estimation and efficient parallel computation.
- **Mini-batch size of 256** resulted in slightly lower accuracy compared to batch size 64, particularly on the validation and test sets. This suggests a possible trade-off: while larger batches may lead to more stable gradients, they might also generalize slightly worse due to less frequent weight updates per epoch. On the positive side, the training time was slightly shorter than that of batch size 64, due to fewer iterations per epoch.

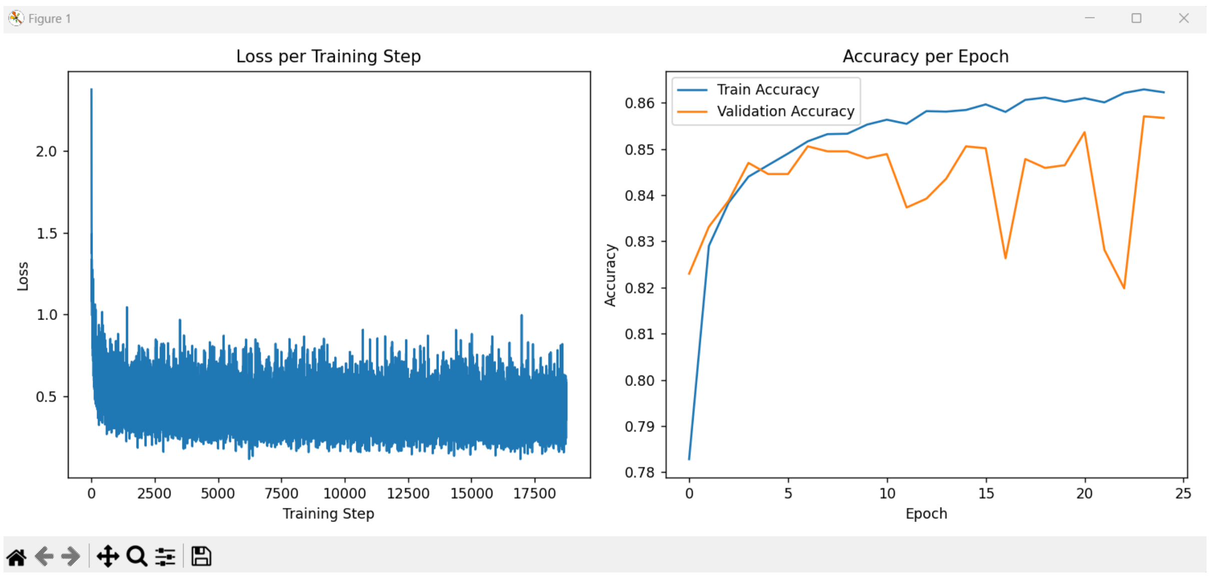
The **mini-batch size of 64** achieved the best combination of accuracy and efficiency, and was selected for use in the remaining experiments. It allowed the model to learn effectively and generalize well, while keeping training time reasonable.

Number of hidden layers:

With the best-performing values for learning rate (**0.1**) and mini-batch size (**64**) already determined, we proceeded to investigate how the **number of hidden layers** impacts the MLP's performance. We tested models with **0, 1, and 2 hidden layers**, while keeping all other parameters constant:

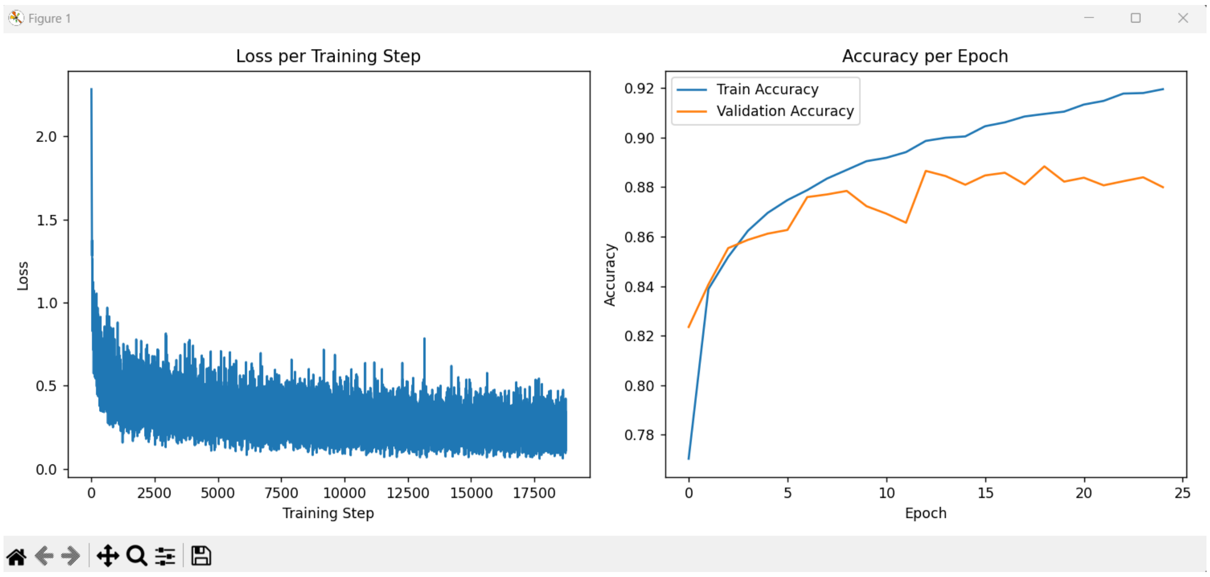
- Learning rate: 0.1
- Batch size: 64
- Width: 128 neurons per hidden layer
- Loss function: CrossEntropy
- Number of epochs: 25

- Number of hidden layers = 0:



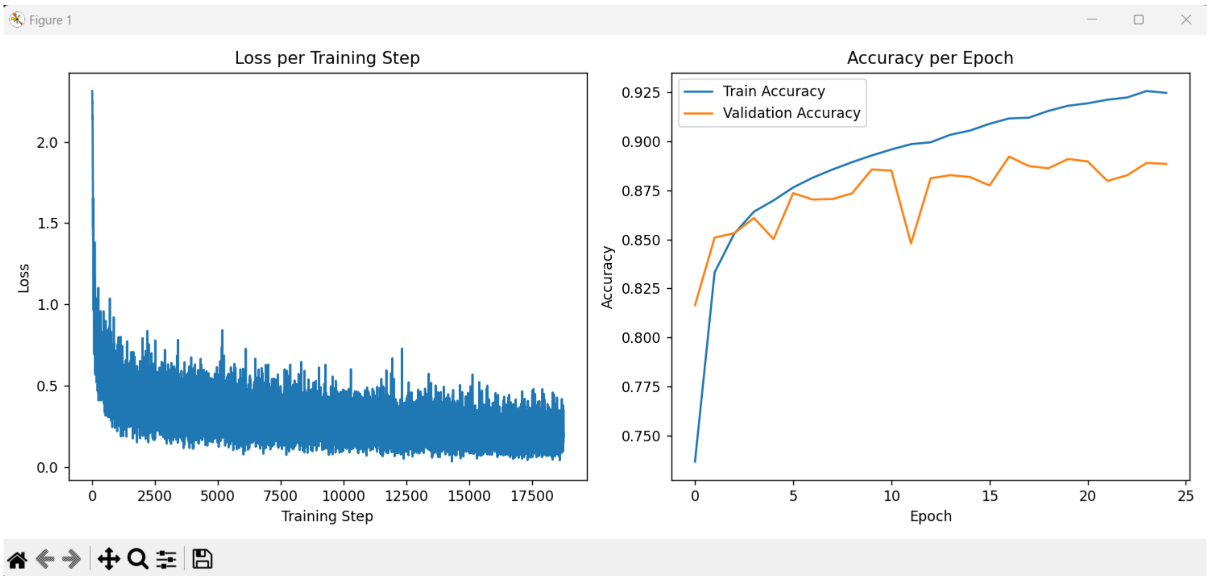
★ Training Accuracy: **0.8623**
★ Validation Accuracy: **0.8568**
★ Test Accuracy: **0.8439**
★ Training time: **168.67 seconds**

- Number of hidden layers = 1:



- ★ Training Accuracy: **<u>0.9195</u>**
- ★ Validation Accuracy: **<u>0.8799</u>**
- ★ Test Accuracy: **<u>0.8718</u>**
- ★ Training time: **<u>167.30 seconds</u>**

- Number of hidden layers = 2:



- ★ Training Accuracy: **<u>0.9247</u>**
- ★ Validation Accuracy: **<u>0.8885</u>**
- ★ Test Accuracy: **<u>0.8836</u>**
- ★ Training time: **<u>187.46 seconds</u>**

Number of Hidden Layers	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Training Time (s)**

0	0.8623	0.8568	0.8439	168.67
1	0.9195	0.8799	0.8718	167.30
2	0.9247	0.8885	**0.8836**	187.46

- The model with **0 hidden layers** (essentially a logistic regression classifier) already achieved a relatively strong performance, with validation and test accuracies in the mid-80% range. This shows that even linear models can capture some meaningful patterns in the FashionMNIST dataset.
- Adding **1 hidden layer** significantly improved accuracy across the board. The model was able to learn more complex features, resulting in a **validation accuracy increase of over 2%** and a **test accuracy gain of nearly 3%** compared to the shallow model. Notably, this improvement came **without increasing training time**, which remained almost identical.
- Adding a **2nd hidden layer** led to a further slight improvement in performance, particularly on the validation and test sets. The model achieved the **highest accuracies overall**, indicating that deeper networks can better capture the complexity of the data. However, the **training time also increased** (about 12% longer than the 1-hidden-layer model), suggesting a trade-off between accuracy and computational efficiency.

The model with **2 hidden layers** delivered the best overall performance in terms of accuracy, and the increase in training time was relatively modest. Therefore, we selected **2 hidden layers** as the optimal configuration for our MLP architecture. This depth provides a good balance between learning capacity and training efficiency for this classification task.

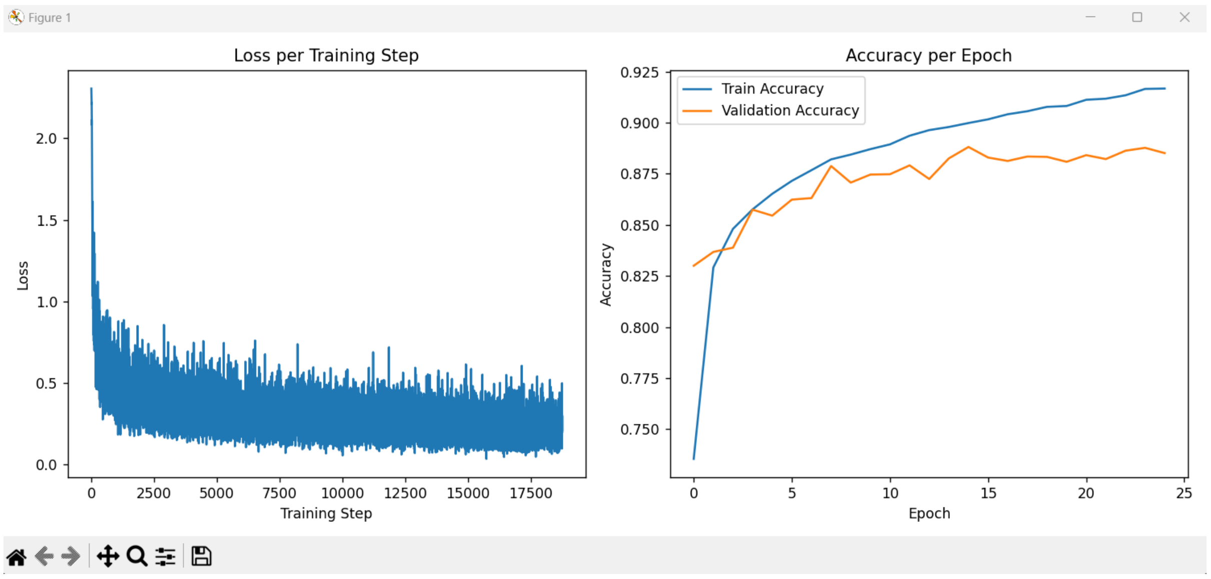
Width (number of neurons in hidden layers):

Having determined the optimal number of hidden layers (**2**), we now explore how the **width** of each hidden layer—i.e., the number of neurons—affects the performance of our Multilayer Perceptron (MLP) on the FashionMNIST dataset. We tested three different widths: **64**, **128**, and **256** neurons per hidden layer.

All other parameters were held constant:

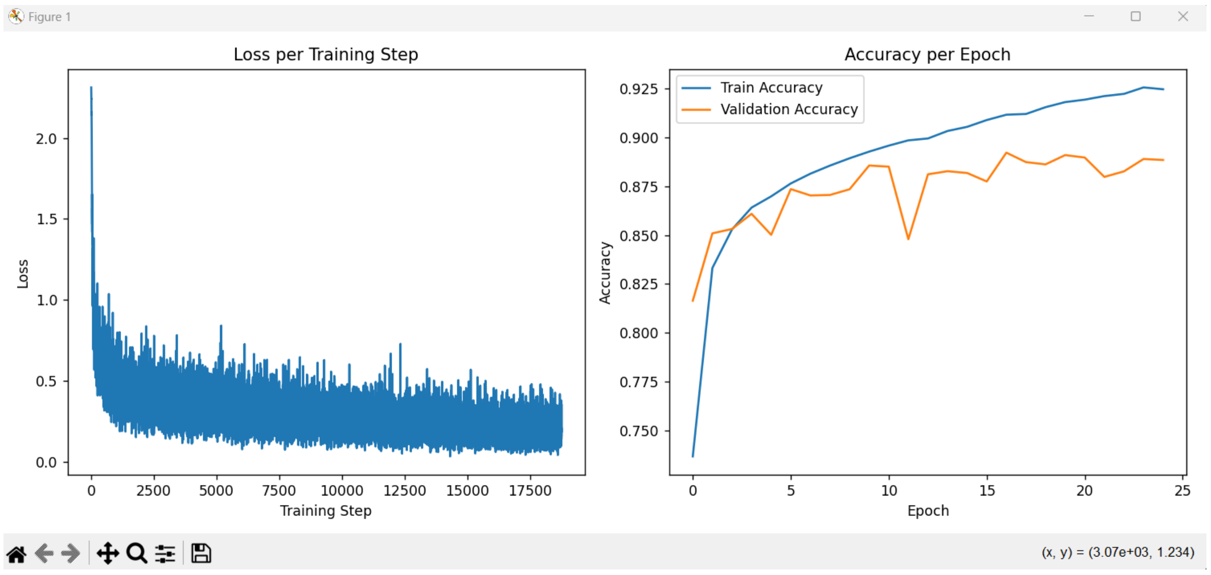
- Learning rate: 0.1
- Batch size: 64
- Hidden layers: 2
- Loss function: CrossEntropy
- Number of epochs: 25

- Width = 64:



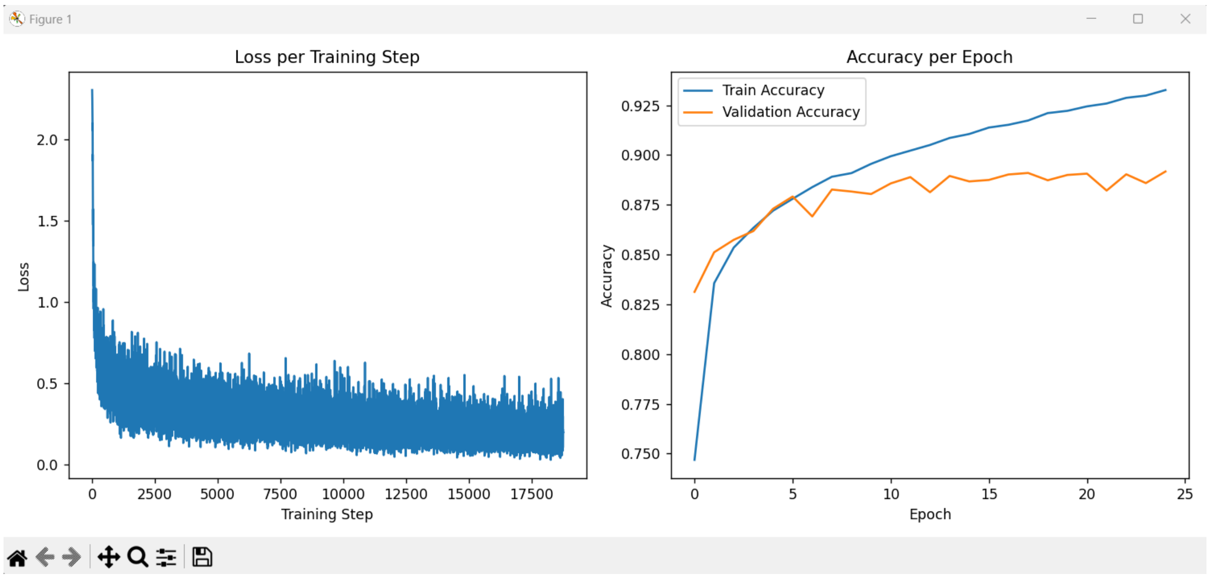
- ★ Training Accuracy: **<u>0.9167</u>**
- ★ Validation Accuracy: **<u>0.8851</u>**
- ★ Test Accuracy: **<u>0.8782</u>**
- ★ Training time: **<u>184.34 seconds</u>**

- Width = 128:



- ★ Training Accuracy: **<u>0.9247</u>**
- ★ Validation Accuracy: **<u>0.8885</u>**
- ★ Test Accuracy: **<u>0.8836</u>**
- ★ Training time: **<u>216.14 seconds</u>**

- Width = 256:



★ Training Accuracy: **0.9327**
★ Validation Accuracy: **0.8918**
★ Test Accuracy: **0.8866**
★ Training time: **195.83 seconds**

Hidden Layer Width	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Training Time (s)**
64	0.9167	0.8851	0.8782	184.34
128	0.9247	0.8885	0.8836	216.14
256	0.9327	0.8918	**0.8866**	195.83

- With **64 neurons per layer**, the model achieved strong results, already reaching close to 88% validation accuracy and nearly 88% test accuracy. The training time was also the lowest among the three options, making it a computationally efficient configuration.
- Increasing the width to **128 neurons** improved performance slightly across all metrics, especially test accuracy, which increased to **88.36%**. However, the training time also increased noticeably—by over **30 seconds** compared to the 64-neuron configuration—indicating higher computational cost due to the increased number of parameters.
- At **256 neurons**, the model reached its **best accuracy** on all datasets. Test accuracy rose to **88.66%**, and training accuracy reached over **93%**. Surprisingly, the training time was **lower than for width = 128**, suggesting potential fluctuations in processing efficiency depending on hardware utilization and batch processing speed.

The width of **256 neurons per hidden layer** provided the best balance of performance and training time. It achieved the highest accuracy across all data splits while avoiding a significant increase in computational cost. Therefore, we selected **256** as the optimal hidden layer width for our final model configuration.

Loss functions:

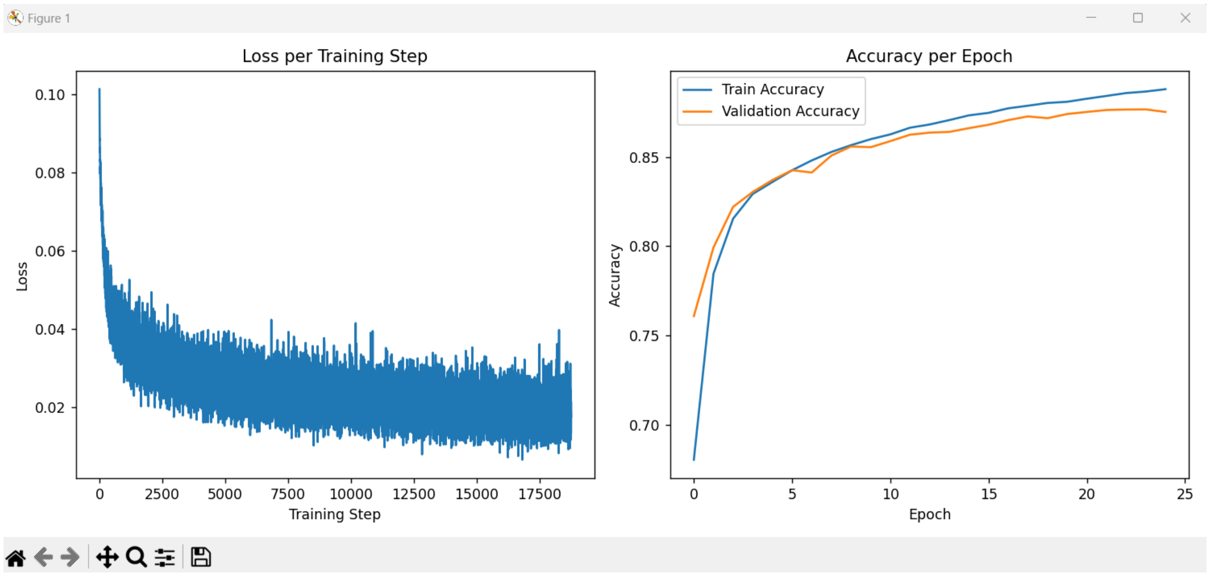
After optimizing the model architecture and key hyperparameters (learning rate, mini-batch size, number of hidden layers, and width), we evaluated the impact of different **loss functions** on training efficiency and classification performance. The loss functions tested were:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **Cross Entropy Loss**

All other parameters remained fixed at their optimal values:

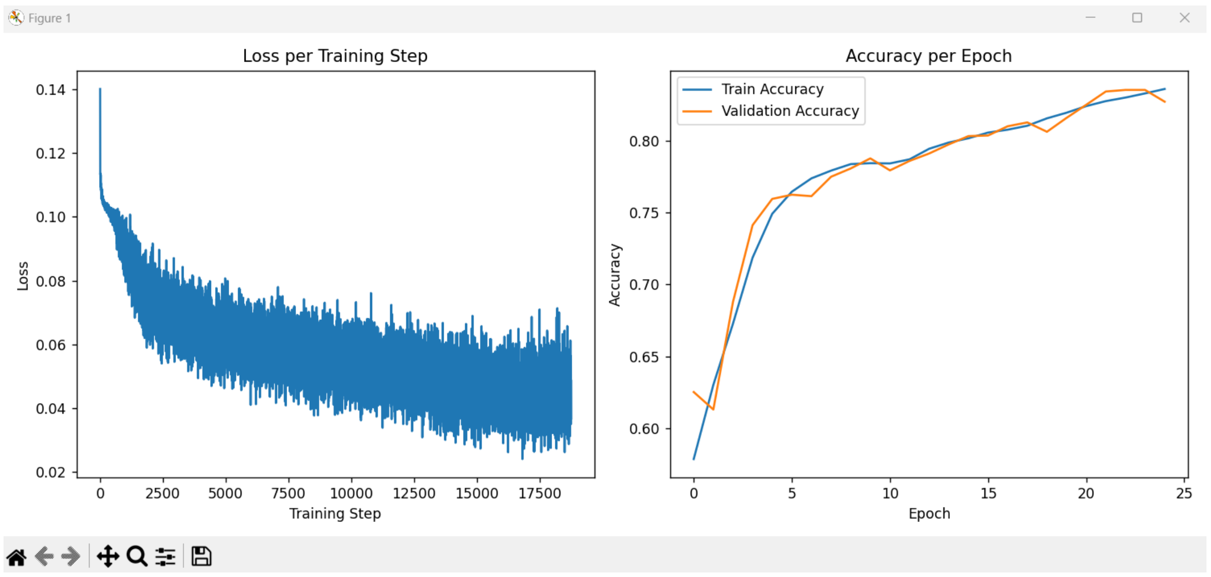
- Learning rate: 0.1
- Batch size: 64
- Hidden layers: 2
- Width: 256 neurons
- Number of epochs: 25

- Mean Squared Error:



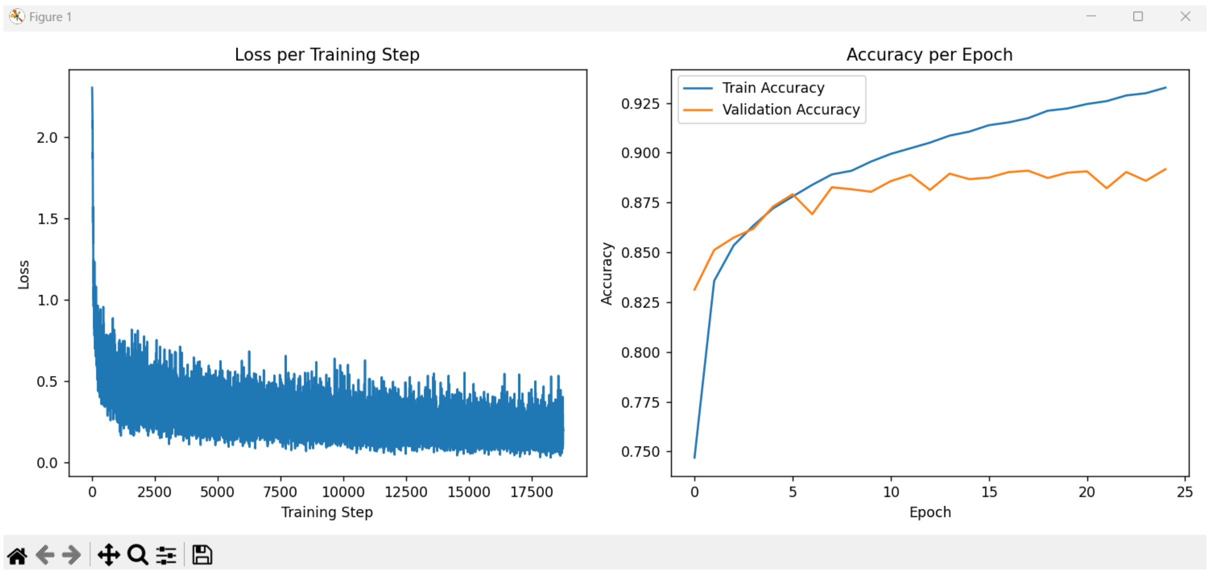
★ Training Accuracy: **<u>0.8879</u>**
★ Validation Accuracy: **<u>0.8752</u>**
★ Test Accuracy: **<u>0.8673</u>**
★ Training time: **<u>198.37 seconds</u>**

- Mean Absolute Error:



★ Training Accuracy: **<u>0.8359</u>**
★ Validation Accuracy: **<u>0.8271</u>**
★ Test Accuracy: **<u>0.8191</u>**
★ Training time: **<u>183.61 seconds</u>**

- Cross Entropy:



★ Training Accuracy: **<u>0.9327</u>**
★ Validation Accuracy: **<u>0.8918</u>**
★ Test Accuracy: **<u>0.8866</u>**
★ Training time: **<u>195.83 seconds</u>**

Loss Function	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Training Time (s)**
Mean Squared Error (MSE)	0.8879	0.8752	0.8673	198.37
Mean Absolute Error (MAE)	0.8359	0.8271	0.8191	183.61
Cross Entropy	0.9327	0.8918	**0.8866**	195.83

- **Cross Entropy Loss** outperformed both MSE and MAE in every metric. This is expected, as Cross Entropy is specifically designed for classification problems and encourages the model to output high confidence in the correct class, leading to better accuracy and convergence.
- **Mean Squared Error (MSE)** performed reasonably well, achieving decent accuracy levels close to Cross Entropy, but slightly lower across training, validation, and test datasets. This is likely due to the fact that MSE penalizes large deviations equally across all output values, which is not ideal for classification tasks where the outputs are probabilities.
- **Mean Absolute Error (MAE)** had the **lowest performance** among the three loss functions. It tends to struggle in classification problems because it provides less informative gradients, making it harder for the model to adjust weights effectively during training.
- In terms of **training time**, all three loss functions were relatively close, with MAE being the fastest but at the cost of significantly reduced accuracy. Cross Entropy had a slightly longer training time than MAE but achieved the best overall performance.

The **Cross Entropy Loss** function provided the highest accuracy across training, validation, and test sets, while maintaining a reasonable training time. Due to its strong theoretical suitability and empirical performance in classification tasks, **Cross Entropy** is selected as the best loss function for our final MLP model.

Discussion

Throughout this lab project, we systematically explored how various hyperparameters and design choices affect the performance of a Multilayer Perceptron (MLP) trained on the FashionMNIST dataset. By isolating each parameter—learning rate, mini-batch size, number of hidden layers, width of layers, and loss function—we were able to observe their individual impact on convergence, accuracy, and training efficiency. Our final observations are the following:

1. **Learning Rate**
   The learning rate had a significant influence on the model's ability to converge. Lower rates (e.g., 0.001) resulted in slower and less effective learning, while a learning rate of 0.1 produced the highest training, validation, and test accuracies. Although it required

a slightly longer training time, the accuracy gains justified this cost, highlighting the importance of step size in efficient optimization.

2. **Mini-Batch Size**
   Training with a mini-batch size of 1 (i.e., stochastic gradient descent) severely hindered the model's learning process and drastically increased training time. A mini-batch size of 64 struck the best balance, enabling efficient and stable training while delivering the highest accuracy. Larger batch sizes like 256 showed reasonable accuracy but began to exhibit signs of reduced generalization.

3. **Number of Hidden Layers**
   As expected, increasing the number of hidden layers improved the model's ability to learn complex patterns. While a shallow model (0 hidden layers) performed decently, the addition of one and then two hidden layers yielded noticeable improvements, especially on the validation and test sets. The model with two hidden layers demonstrated the best performance, validating the effectiveness of deeper architectures in capturing nonlinear patterns.

4. **Width of Hidden Layers**
   Expanding the width of each hidden layer also contributed to performance improvements. The model with 256 neurons per layer outperformed those with 64 and 128 neurons, achieving the highest accuracy across all datasets. Interestingly, this configuration did not incur a significantly longer training time, indicating an efficient utilization of computational resources.

5. **Loss Functions**
   Among the three loss functions evaluated—Mean Squared Error (MSE), Mean Absolute Error (MAE), and Cross Entropy—**Cross Entropy** clearly emerged as the most suitable for this classification task. It consistently delivered the highest accuracy and maintained training efficiency. MSE performed moderately well, while MAE lagged behind significantly, both in accuracy and learning dynamics.

Conclusion

This series of experiments highlights the sensitivity of neural network performance to design and training choices. While certain parameters like learning rate and loss function had a more pronounced impact on convergence and accuracy, others such as width and depth contributed incrementally to overall performance. Our results confirm the common deep learning heuristics: **moderate mini-batch sizes, deeper architectures, and task-appropriate loss functions are critical to optimal performance**.

By the end of our experimentation, we arrived at a highly effective MLP configuration:

- Learning rate: 0.1

- Mini-batch size: 64
- Hidden layers: 2
- Width: 256 neurons per layer
- Loss function: Cross Entropy

This model achieved a **test accuracy of 88.66%**, which demonstrates the strong classification capability of a well-tuned MLP on the FashionMNIST dataset. The project underscores the importance of thoughtful hyperparameter tuning and serves as a practical illustration of how each component contributes to model performance in deep learning.