COMP 551 - Assignment 3 Report

Elizabeth Kourbatski

Galia Oliel-Sabbag

Colin Song

March 30, 2025

Abstract

In this assignment, we implemented a multilayer perceptron (MLP) from scratch using NumPy to classify handwritten characters from the Kuzushiji-MNIST dataset. We evaluated the performance of three MLP architectures—without hidden layers, with a single hidden layer, and with two hidden layers—each using ReLU activations. We further compared these with variants using Sigmoid and Leaky-ReLU, and explored L2 regularization. Finally, we constructed a convolutional neural network (CNN) using PyTorch to benchmark against our MLPs. Our results demonstrate that non-linear activation functions improve classification accuracy and that CNNs consistently outperform MLPs on image classification tasks, reaffirming the importance of spatial feature extraction and architectures that preserve the structure of image data.

1 Introduction

In this study, we evaluated the performance of multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) on the Kuzushiji-MNIST (KMNIST) dataset. Our results show that increasing model depth and using non-linear activations generally improves performance, with our best MLP (two hidden layer, ReLU activation) achieving 80.46% test accuracy. We further benchmarked a CNN trained with PyTorch, which outperformed all MLP configurations, reaching 95.87% test accuracy after 10 epochs.

These results are broadly consistent with previous findings in research conducted by Cao et al. in 2021. They evaluated deep neural networks trained from similarity-confidence data on the KMNIST dataset and reported that risk-corrected Sconf models achieved test accuracies of 89.25% and 90.00%, respectively. While our models used supervised training, their results demonstrate the effectiveness of learning from weak supervision. Notably, our CNN surpassed all Sconf variants in classification performance, highlighting the advantages of supervised CNN training in capturing fine-grained visual features. Our MLP models, however, underperformed compared to Sconf-NN and Sconf-ABS, suggesting that deeper architectures or risk correction schemes may help mitigate overfitting in shallow networks.

2 Methods

2.1 Multilayer Perceptron Architecture

We implemented a Multilayer Perceptron (MLP) from scratch using `NumPy`. An MLP is a feedforward neural network consisting of an input layer, one or more hidden layers, and an output layer. Each layer transforms its inputs through a linear transformation followed by a non-linear activation function. The pre-activation for layer ℓ is defined as:

$$\mathbf{a}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{z}^{(\ell-1)} + \mathbf{b}^{(\ell)}$$

and the post-activation (i.e., the output of each hidden unit) is given by:

$$\mathbf{z}^{(\ell)} = \varphi(\mathbf{a}^{(\ell)})$$

where φ is an activation function such as ReLU, Sigmoid, or Leaky ReLU. The final output layer uses the softmax function to convert raw scores into class probabilities. We tested hidden layer sizes from the set {32, 64, 128, 256} to observe their effect on performance.

To ensure the reproducibility of our experiments, we fixed all sources of randomness by setting a global random seed (`np.random.seed(1234)`) and using a consistent `random_state=1234` for all data splits. This allowed us to fairly compare configurations and analyze performance trends without noise from inconsistent initialization or training order.

Weight initialization is determined by the activation function: ReLU and Leaky ReLU use He initialization, Sigmoid uses Xavier initialization, and Softmax layers are initialized with small random values.

To prevent overfitting and long training times, we implemented early stopping with a patience of 5 epochs, monitoring validation accuracy.

We evaluated the performance of multiple MLPs on the KMNIST dataset using a grid search over key hyperparameters: learning rates (1e-3, 1e-4), L2 regularization strengths (0.0, 0.01), and batch sizes (32, 64, 128, 256). Each MLP was trained for 20 epochs, a choice made to reduce runtime while still allowing meaningful comparison.

2.2 Convolutional Neural Network Architecture

We implemented a Convolutional Neural Network (CNN) using PyTorch to compare against the Multilayer Perceptron (MLP). CNNs are particularly well-suited to image classification tasks because they can leverage spatial structure and local connectivity within the input. Unlike MLPs, which require flattening the input image into a vector, CNNs operate directly on the 2D grid of pixels, preserving the spatial relationships between neighboring pixels.

The core component of a CNN is the convolutional layer, where small filters (also called kernels) are slid across the input image to extract local features. Formally, a 2D convolution is defined as:

$$[\mathbf{W} * \mathbf{X}](i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{uv} x_{i+u,j+v}$$

where $\mathbf{W}$ is the filter and $\mathbf{X}$ is the input patch. This operation acts as a form of template matching, allowing the network to detect local patterns regardless of their absolute position in the image, which introduces translation invariance.

We designed a Convolutional Neural Network (CNN) to classify images from the Kuzushiji-MNIST dataset. The architecture consisted of three convolutional layers followed by a fully connected layer and an output classification layer. Each convolutional layer was followed by a ReLU activation function and 2×2 max pooling.

We implemented a comprehensive grid to determine the optimal number of filters, filter size, stride, dropout, padding, and hidden units. The number of filters in the convolutional layers determines how many feature maps are learned at each layer. A higher number of filters allows the network to capture more complex and diverse patterns, but it also increases the number of learnable parameters, which may cause overfitting.

We tested filter sizes of 3×3 and 5×5. Smaller kernels capture fine details, while larger ones are better suited for broader features. We also varied the stride, which controls how far the filter moves across the input. A stride of 1 maintains high spatial resolution, while a stride of 2 reduces the size of feature maps and can make the model more efficient, at the cost of detail. Padding was varied to control the spatial dimensions of the output; using no padding reduces feature map size, while padding preserves it. Dropout was used to prevent overfitting, with rates of 0.3 or 0.7. Dropout randomly disables a fraction of the neurons during training, encouraging the network to learn more generalizable features. Finally, we explored different numbers of hidden units in the fully connected layer (32, 64, 128, 256), which affect the model's capacity to combine features before final classification.

We performed randomized hyperparameter search to optimize the performance of our convolutional neural network (CNN). We defined a discrete search space consisting of multiple values for key hyperparameters, including learning rate, number of convolutional filters, kernel size, stride, padding, dropout rate, and the number of hidden units in the fully connected layer. Rather than exhaustively evaluating every possible combination, we randomly sampled 25 configurations from the full Cartesian product of this space to balance exploration with computational efficiency. Each sampled configuration was used to instantiate and train a CNN model for 2 epochs using the Adam optimizer and cross-entropy loss. Validation accuracy was recorded for each run, and the top-performing configurations were identified by sorting the results based on their final validation accuracy. This approach allowed us to efficiently explore the hyperparameter space and select the most promising model configurations for further training and evaluation.

3 Datasets

We used the Kuzushiji-MNIST dataset for all experiments. This dataset consists of 60,000 training and 10,000 test images, each of size 28×28 pixels, representing 10 classes of cursive Japanese characters. Labels are integer-encoded from 0 to 9. All of the images were flattened into 784-dimensional vectors before being input into the MLP.

4 Results

4.1 MLP

We evaluated the performance of multiple multilayer perceptrons (MLPs) on the Kuzushiji-MNIST dataset using a grid search over hyperparameters, which included learning rates (1×10^{-3} and 1×10^{-4}), L2 regularization strengths (0.0 and 0.01), and batch sizes (32, 64, 128, 256). Each MLP was trained for 20 epochs to reduce training time. We experimented with five different MLP models: one with no hidden layers, a single hidden layer with ReLU activation, and three two-hidden-layer models using ReLU, Sigmoid, or Leaky ReLU activations. The number of hidden units in each layer varied across 32, 64, 128, and 256.

We evaluated the performance of multiple multilayer perceptrons (MLPs) on the Kuzushiji-MNIST dataset using a grid search over hyperparameters, which included learning rates (1×10^{-3} and 1×10^{-4}), L2 regularization strengths (0.0 and 0.01), and batch sizes (32, 64, 128, 256). Each MLP was trained for 20 epochs to reduce training time. We experimented with five different MLP models: one with no hidden layers, a single hidden layer with ReLU activation, and three two-hidden-layer models using ReLU, Sigmoid, or Leaky ReLU activations. The number of hidden units in each layer varied across 32, 64, 128, and 256.

The best-performing model was the 2-hidden-layer ReLU MLP, achieving a validation accuracy of **90.00%** and a test accuracy of **80.46%**. This model benefited from increased capacity due to the second hidden layer, while still generalizing well thanks to proper regularization and batch size selection. It outperformed the 1-hidden-layer ReLU model (which achieved 89.85% val, 80.19% test), suggesting that additional depth, can improve performance without necessarily leading to overfitting.

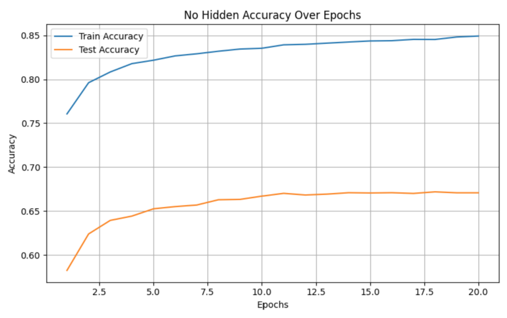
The Leaky ReLU model achieved nearly identical validation accuracy (90.05%) but slightly lower test performance (80.09%). While Leaky ReLU is effective at preventing dead neurons, it may introduce noisier gradients, which could explain its slightly lower generalization. The Sigmoid model suffered from vanishing gradients, yielding a significantly lower test accuracy (70.05%). As expected, the no hidden layer model performed worst (80.80% val, 67.08% test), lacking the capacity to model non-linear decision boundaries.

Model	**Hidden Units**	**LR**	**Reg**	**Batch**	**Val (%)**	**Test (%)**
No Hidden	N/A	0.0001	0.00	64	80.80	67.08
1 Hidden (ReLU)	[256]	0.001	0.00	256	89.85	80.19
2 Hidden (ReLU)	[256,256]	0.001	0.01	256	90.00	80.46
2 Hidden (Sigmoid)	[256,256]	0.001	0.00	32	81.90	70.05
2 Hidden (LeakyReLU)	[128,128]	0.001	0.01	64	90.05	80.09

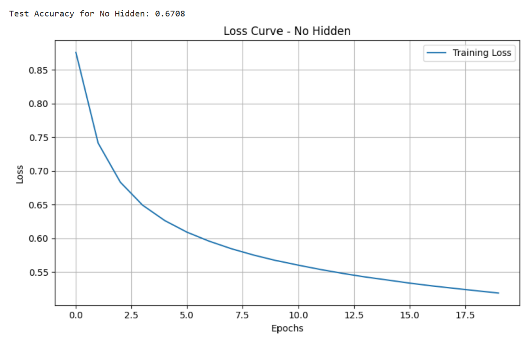
Table 1: Best hyperparameter settings and performance for each MLP model.

To ensure the correctness of our backpropagation implementation, we performed gradient checking on simplified MLPs with two hidden units per layer. Each model used a different activation function—ReLU, Sigmoid, and Leaky ReLU. In all cases, the relative difference between numerical and analytical gradients was below 1×10^{-4}, confirming the correctness of our gradient computations.

All training loss curves showed a smooth decrease over time, indicating that the model was learning properly, the learning rate was well-chosen, and the gradients were well-behaved.

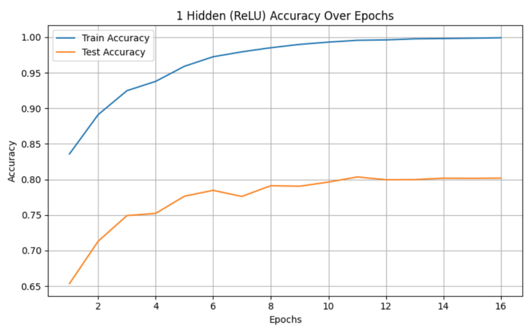


Accuracy - 0 Hidden Layers

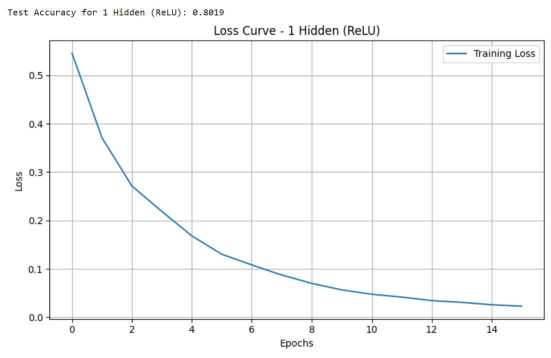


Loss Curve - 0 Hidden Layers

Figure 1: 0 Hidden Layers

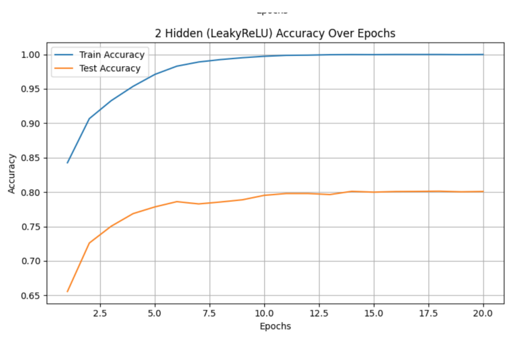


Accuracy - 1 Hidden Layer (ReLU)

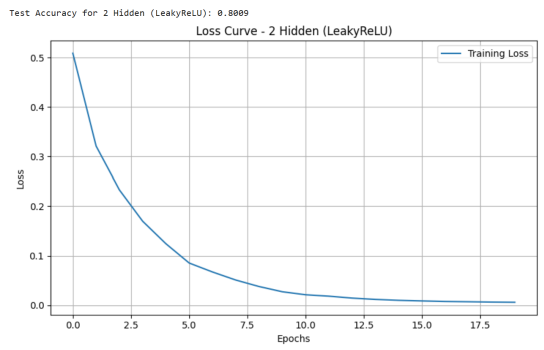


Loss Curve - 1 Hidden Layer (ReLU)

Figure 2: 1 Hidden Layer with ReLU Activation



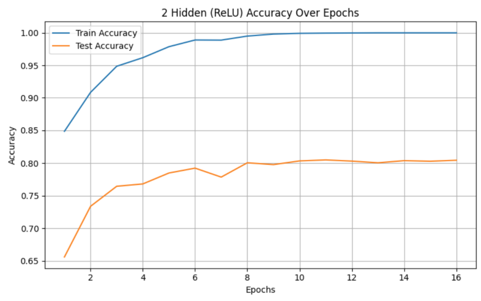
Accuracy - 2 Hidden Layers (Leaky ReLU)



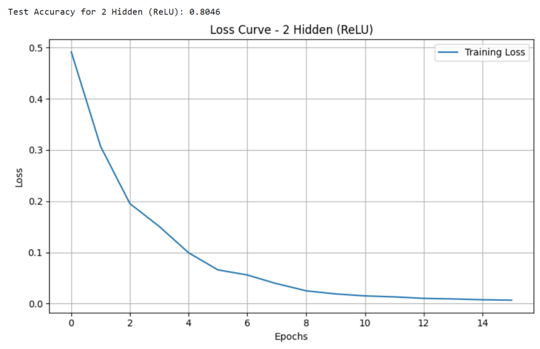
```
Test Accuracy for 2 Hidden (LeakyReLU): 0.8009
```

Loss Curve - 2 Hidden Layers (Leaky ReLU)

Figure 3: 2 Hidden Layers with Leaky ReLU Activation



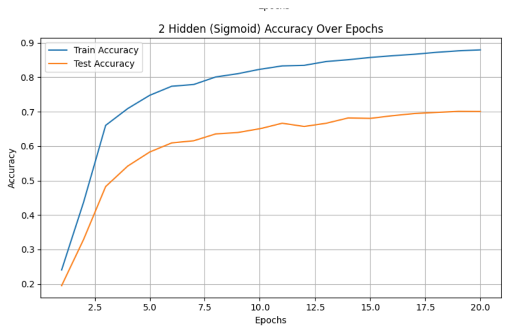
Accuracy - 2 Hidden Layers with Relu



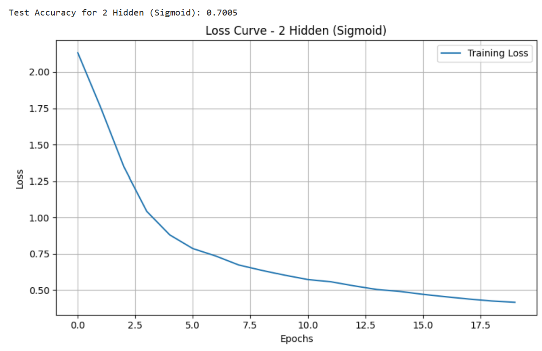
```
Test Accuracy for 2 Hidden (ReLU): 0.8046
```

Loss Curve - 2 Hidden Layers with Relu

Figure 4: 2 Hidden Layers with ReLU



Accuracy - 2 Hidden Layers (Sigmoid)



```
Test Accuracy for 2 Hidden (Sigmoid): 0.7005
```

Loss Curve - 2 Hidden Layers (Sigmoid)

Figure 5: 2 Hidden Layers with Sigmoid Activation

4.2 CNN

To identify the most effective configuration for our convolutional neural network (CNN), we conducted a randomized hyperparameter search over a defined grid of parameter values. Each sampled configuration was evaluated based on validation accuracy after two training epochs. Among the 25 randomly selected combinations, the best-performing model achieved a validation accuracy of **97.49%**. This configuration used a learning rate of `0.001`, `64` convolutional filters, a kernel size of `3`, a stride of `1`, padding `(1)`, a dropout rate of `0.3`, and `32` hidden units in the fully connected layer.

The best model, using a learning rate of `0.001`, 32 convolutional filters, a kernel size of `3`, stride `1`, padding `1`, dropout of `0.3`, and `32` hidden units, was retrained from scratch on the

Table 2: Best CNN Model Configuration

Parameter	Value
Learning Rate	0.001
Filters	32
Kernel Size	3
Stride	1
Padding	1
Dropout	0.3
Hidden Units	32
Validation Accuracy	0.974917
Test Accuracy	0.9587

combined training and validation datasets. Training was conducted for `10` epochs using the Adam optimizer and cross-entropy loss, with early stopping disabled for this final run. The training loss decreased from 0.3594 in the first epoch to 0.0187 by epoch 10. Correspondingly, training accuracy improved from 88.62% to 99.40%. When evaluated on the held-out test set, the model achieved a test accuracy of 95.99%. These results indicate excellent convergence, strong generalization, and minimal signs of overfitting, likely due to the regularizing effect of dropout and the overall effectiveness of the selected architecture.

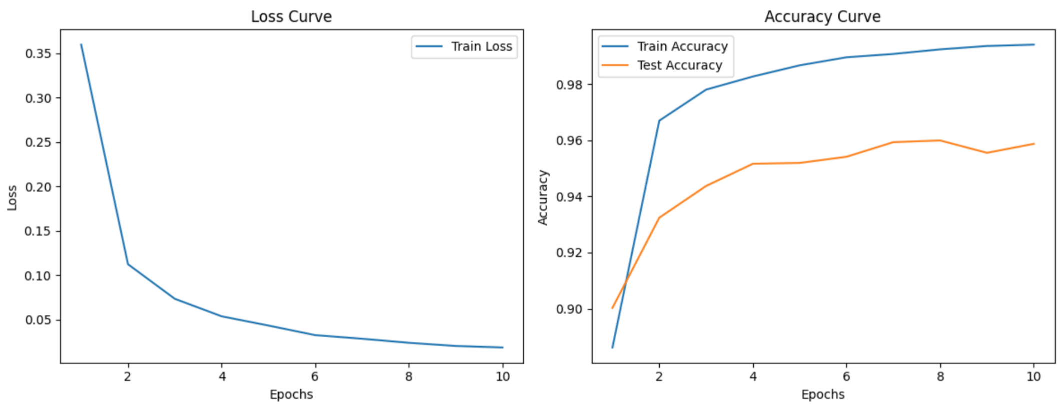


Figure 6: CNN Model Visualization

5 Discussion and Conclusion

Our experiments demonstrate the advantages of convolutional neural networks (CNNs) over multilayer perceptrons (MLPs) for image classification tasks on the Kuzushiji-MNIST dataset. MLPs treat input images as flat vectors, disregarding the spatial relationships between neighboring pixels. This flattening process loses important information about the local patterns in the image, like edges or shapes. As a result, MLPs struggle to recognize patterns that appear in different positions, making them less effective for this kind of task.

In contrast, convolutional neural networks (CNNs) are specifically designed to work with image data in its natural 2D structure. Instead of flattening the input, CNNs apply convolutional filters that slide over the image, allowing the network to detect local patterns such as edges, corners, and textures. These filters are spatially shared across the image, meaning the same set of weights is applied at every location. This reduces the total number of parameters, making CNNs more efficient and less prone to overfitting compared to fully connected networks.By using several convolutional layers, the model first learns basic patterns like edges or shapes, and then builds on them to understand more complex features. This ability to preserve spatial relationships and build up meaningful features makes CNNs especially well-suited for image classification tasks like those in the Kuzushiji-MNIST dataset.

Our randomized search showed that even with only two epochs of training, the CNN outperformed all MLP variants in terms of validation accuracy. After retraining the best CNN model on the combined training and validation data, it achieved a training accuracy of 96.16% and a test accuracy of 95.87%, showing a smooth convergence, indicating strong learning behavior and minimal overfitting. In contrast, the best MLP model—a two-hidden-layer network with ReLU activation—achieved a test accuracy of 80.46%. This result demonstrates that increased depth can improve performance when paired with appropriate regularization and batch size. Despite being deeper, this model avoided overfitting, likely due to its use of L2 regularization and an optimal learning rate. The 1-hidden-layer ReLU model performed slightly worse, with a test accuracy of 80.19%, suggesting that while shallow networks train faster and are easier to optimize, deeper networks have more power when properly tuned.

These results highlight the importance of using models that take advantage of the structure of image data. CNNs are better suited for image classification because they use spatial information and shared filters to learn meaningful patterns efficiently. While MLPs can work in some cases, they are less effective on image tasks compared to CNNs due to their inability to preserve local spatial relationships.

In summary, CNNs outperformed MLPs on the Kuzushiji-MNIST dataset in terms of accuracy, generalization, and training efficiency. These findings suggest that future work could focus on enhancing CNN performance further through techniques such as data augmentation to increase dataset diversity, or by exploring deeper architectures like Residual Networks (ResNets), which are capable of learning more complex features without the degradation typically seen in very deep models. Residual Networks address the degradation problem by introducing skip connections that allow inputs to bypass one or more layers. Instead of learning a direct mapping from input to output, each residual block learns only the residual—the difference between the input and the desired output—which is then added back to the input. Skip connections help preserve gradient flow during backpropagation, making it easier to train deep models and allowing the network to benefit from increased depth without suffering from vanishing gradients or degraded performance.

6 Statement of Contribution

All team members contributed evenly to the coding and write-up portions of this project!

References

1. Kuzushiji-MNIST Dataset https://www.kaggle.com/datasets/anokas/kuzushiji
2. Cao et al. (2021): https://proceedings.mlr.press/v139/cao21b/cao21b.pdf
3. Professor Yue Li's GitHub: https://www.cs.mcgill.ca/~yueli/teaching/COMP551_Winter2025/comp551_winter2025.html
4. Murphy, Kevin P. *Probabilistic Machine Learning: An Introduction*: https://probml.github.io/pml-book/book1.html