Assignment 1 Part BNeural Network with Pytorch

short line

Rusan Vaidya

Student ID: 24886400  
27-03-2024

94691 - Deep Learning

Master of Data Science and Innovation

University of Technology of Sydney

Table of Contents

[1. Problem Understanding 2](#_if9iwijz2pg3)

[a. Problem Statement 2](#_bnjfuh5ewogw)

[b. Key Objectives 2](#_o86v2knav45o)

[2. Data Preparation 3](#_yg03y3w5hig9)

[3. Modeling 4](#_6vwh14nc44ga)

[a. Experiment 1 4](#_n3u1hrinbttc)

[b. Experiment 2 4](#_6a7qysk7v047)

[c. Experiment 3 4](#_r82z7lgx8bpc)

[d. Experiment 4 4](#_pfc1n6wv2pal)

[4. Evaluation 5](#_mngiyx5tet6w)

[a. Experiment 1 5](#_w2nvakzcmjd5)

[b. Experiment 2 5](#_u2dq9bdmdm4n)

[c. Experiment 3 5](#_a5axp9xf0cew)

[5. Conclusion 8](#_9hr11g79asdx)

[6. References 9](#_ma3y8ytjvp1y)

# Problem Understanding

## Problem Statement

This project aims to solve the task by building a neural network model and accurately identifying handwritten Japanese Hiragana characters. The model is evaluated based on its performance on the training and unseen data.

## Key Objectives

The key objective of this research is to find out the following:

1. To develop a model trained on handwritten characters from the MNIST dataset of Hiragana character images.
2. To achieve high accuracy in classifying the characters correctly and minimize loss.

short dash

# Data Preparation

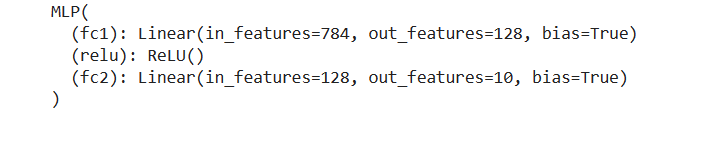
The Japanese MNIST dataset consists of 70000 images of size 28 by 28 images of handwritten Hiragana characters and has ten classes as target variables. The dataset, already split into training and testing sets, consists of a tensor array of images for features and a binary class matrix for targets across ten classes, achieved through one-hot encoding. Initially, the dataset is reshaped to include dimensions for the number of rows, height, width, and channels. Next, the feature data in both the train and test sets are converted to a float type for consistency. It is standardized to further preprocess the data for model training by dividing each tensor by 255, normalizing the image pixel values to a 0-1 range, and ensuring the data is clean and ready for effective model training.

short dash

# Modeling

This research is based on four distinct experiments with variations in model architecture, optimizer and loss functions to obtain the optimal combination that minimizes the error, avoids overfitting, and performs well across training, validation and testing data. The image size was 28 by 28. The image was flattened out to 748 and fed to the neural network as input, giving ten classification outputs.

## Experiment 1

The initial experiment created a three-layer neural model and trained it for 500 epochs using CrossEntropyLoss and SGD optimizer with a learning rate 0.001 to optimize the model and update the weights. This served as the benchmark model for further enhancement.

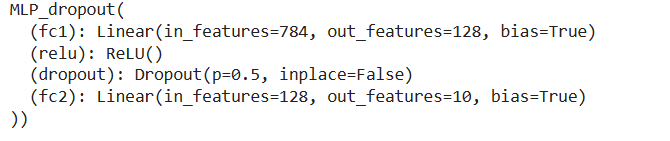
## Experiment 2

Improvising the first experiment, a momentum of 0.9 was added to the SGD optimizer to improve the loss, aiming for better performance in the second experiment.



## Experiment 3

In experiment three, a dropout layer was added to the model for regularization with an early stop with a threshold of five sets to prevent overfitting. Here, dropout omits neurons during training to promote robust feature learning, while the early stopping ceases the training when validation performance is plateaued.



## Experiment 4

For experiment four, the base model setup is similar to experiment 2, with the addition of an early stop to balance efficient optimization with overfitting prevention to enhance model performance.

short dash

# Evaluation

## Experiment 1

The training loss decreased steadily over 500 epochs in the first experiment, and validation loss was concurrent with training loss. The accuracy is also close together and stabilized at a high level, suggesting that the model has learning features without significant overfitting. The model also performs well in the test set, with 81.49% accuracy, and the loss is 0.5965.

## Experiment 2

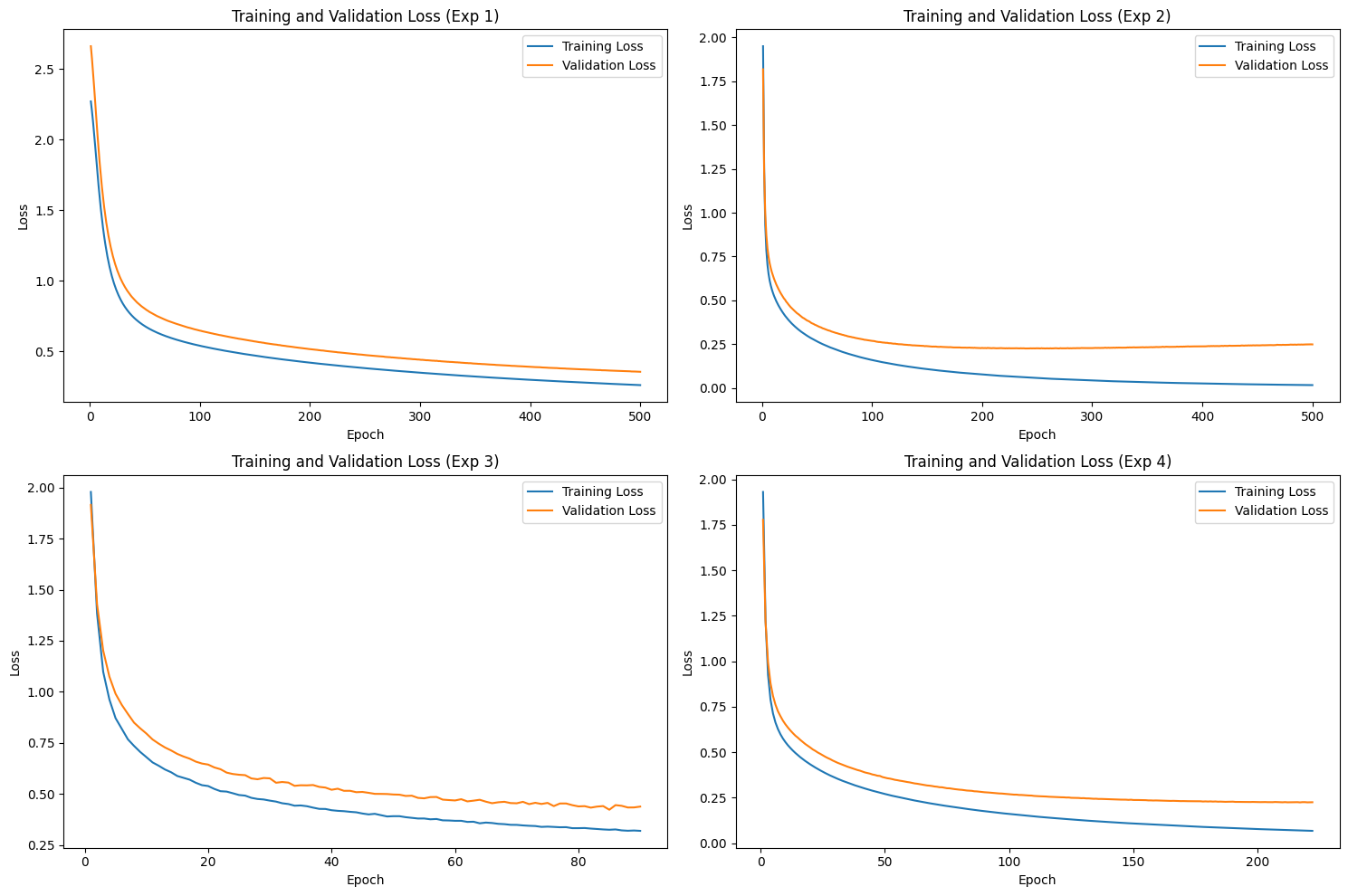
The momentum introduction in the model seems to have a significant impact as the model has decreased its loss and gained accuracy quicker than the initial experiment. The final accuracy for both sets appears to be very high and very close, indicating the model performs well in the training and validation data. The model also performs well in the test set, with 88.11% accuracy, and the loss is 0.5086.

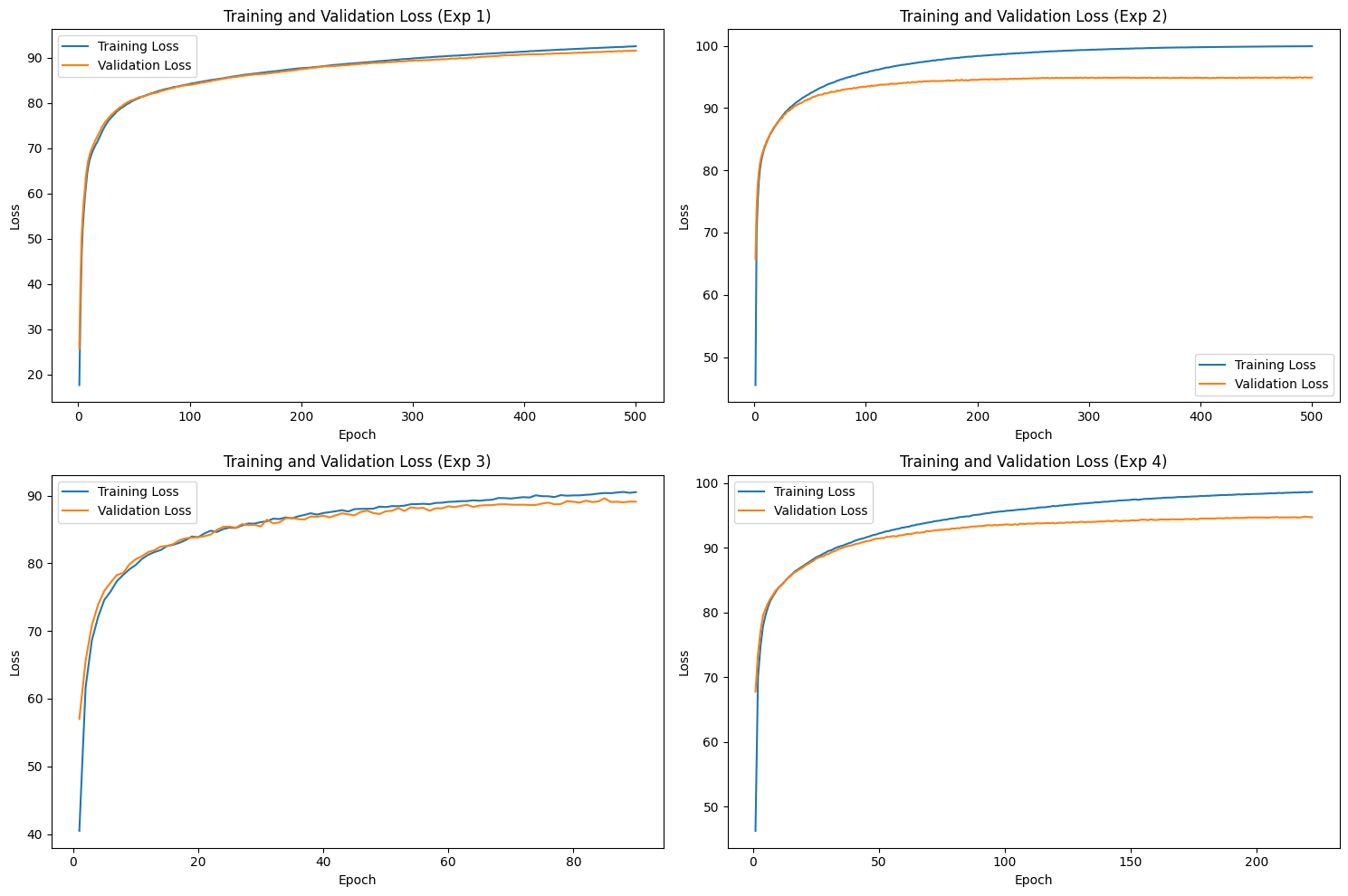
## Experiment 3

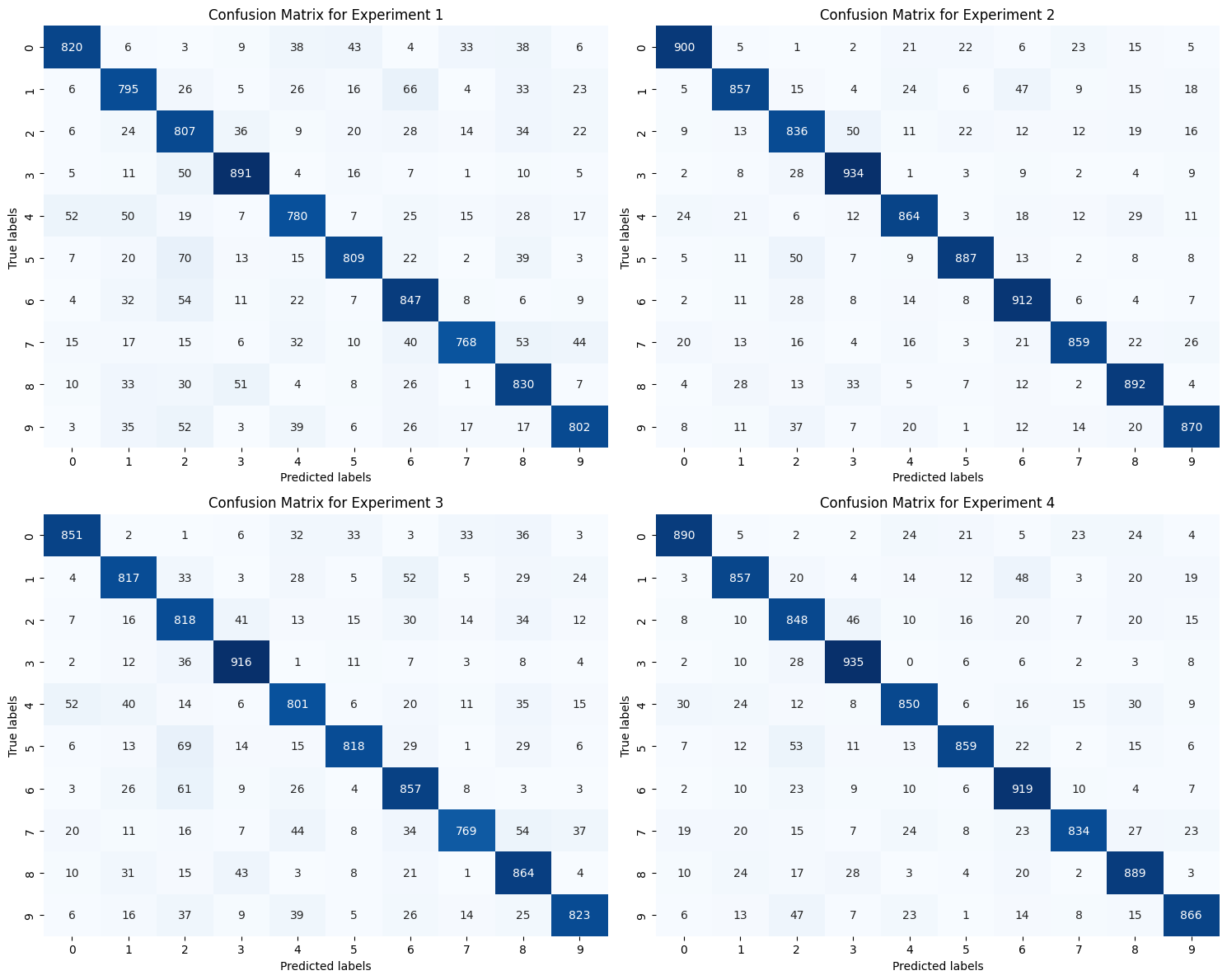
Adding dropout appears to have caused an increase in variance in training loss and reduced the accuracy slightly. However, the validation accuracy remained high, indicating robust generalization and Early stopping was triggered at epoch 90. The model also performs well in the test set, with 83.34% accuracy, and the loss is 0.5388.

## Experiment 4

Combining momentum and early stopping resulted in a quick decline in loss and rapid improvement in accuracy and early plateau. The generalization of training and validation appears to be very effective. The model also performs well in the test set, with 87.47% accuracy, and the loss is 0.4461.







The above confusion matrices provide the corresponding results of the four experiments. Experiment 1 demonstrates a solid baseline with a reasonable number of correct classifications. Experiment 2 exhibits a marked improvement with higher accurate values. Although experiment 3 has significant overall performance, there is a shift in the pattern of misclassifications. Experiment 4 indicates a high number of correct predictions with few misclassifications.

short dash

# Conclusion

In conclusion, after evaluating the performance across various models and evaluating it in multiple metrics, experiment 4 emerges as a superior model, as evidenced by low loss and high accuracy on the test set. Despite its proficiency, it shows room for improvement in class-specific predictions, hinting at class imbalance or feature representation issues. Further tuning of hyperparameters and a broader evaluation spectrum could enhance its performance.

short dash

# References

“Early Stopping in PyTorch.” *Stack Overflow*, <https://stackoverflow.com/questions/71998978/early-stopping-in-pytorch>.

short dash