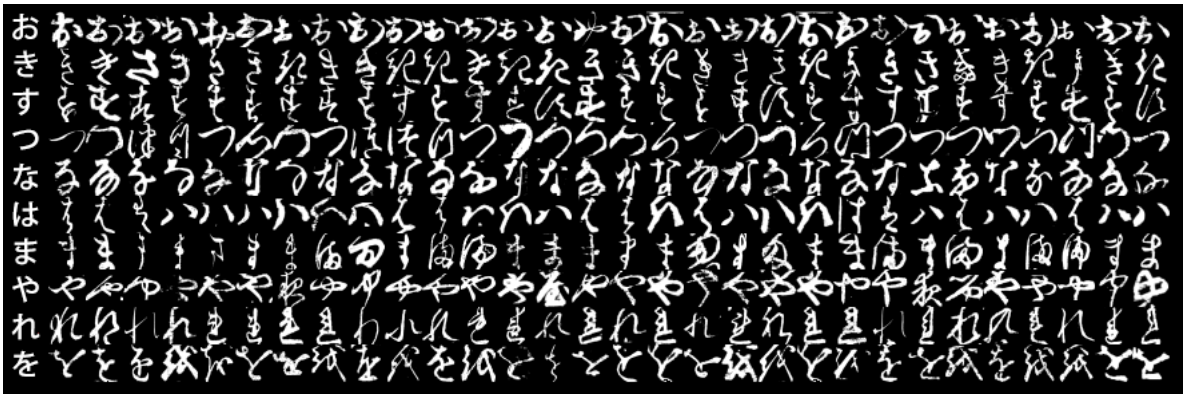
UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Assignment 1 - Neural Networks
part B report:

Japanese MNIST



Thirada Tiamklang
Student id: 14337188

94691 Deep Learning - Autumn 2024

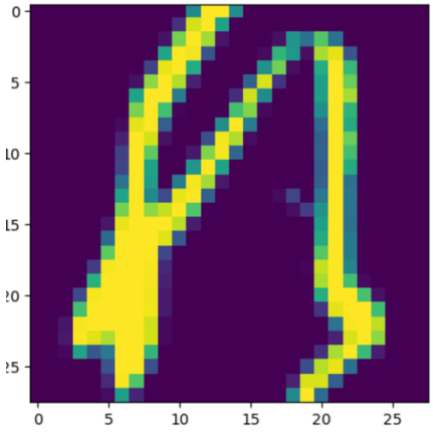
1. Presentation of Data

In this report and project, we will be working with the Japanese MNIST dataset, which comprises 70,000 images of handwritten Hiragana characters. Each image is of dimension 28 by 28 pixels. Our goal is to train a custom neural network model to classify these images into one of 10 different classes corresponding to the different characters.

We will pre-process the images by flattening them into vectors of dimension (784, 1) and then train our neural network using fully connected layers and dropout regularization. The training process will involve optimizing the model's parameters to minimize the classification error.

To ensure that our model generalizes well to unseen data, we will conduct four experiments, tweaking hyperparameters and architectural choices as necessary. Our aim is to achieve an accuracy of at least 80% on the test set while minimizing overfitting.

The training dataset, including training images and training labels, contains 60,000 variables, while the test dataset contains 10,000 variables. The first image of the training set is represented below:



2. Presentation of Different Architectures and Hyperparameters Tested

With our target variable, which consists of labels for 10 different classes, representing multiclassification, we will use cross-entropy with softmax as the criterion. The neural network architecture of each experiment is represented in the table below:

Experiment	Hyperparameters			Model summary				
	Optimizer	Learning rate	Weight decay	Input size	Hidden size	Number of classes	Dropout rate	ReLu (time(s))
1	Adam	0.0001	0.001	784	128	10	0.5	1
2	Adam	0.0001	0.001	784	128	10	0.1	1
3	SGD (momentum=0.9)	0.001	0.001	784	256,128	10	0.1	2
4	SGD (momentum=0.9)	0.001	0.001	784	512,256,128	10	0.1	3

Experiment 1:

Experiment 1 was designed to establish a baseline performance using a simple neural network architecture with one hidden layer. The rationale behind this choice is to assess the model's capability to learn the task without introducing unnecessary complexity.

In this experiment, we employed a basic linear model with one hidden layer comprising 128 units. We incorporated a dropout layer with a rate of 0.5 and ReLU activation function to introduce non-linearity and prevent overfitting. Additionally, weight decay of 0.001 was applied to regularize the model further and mitigate overfitting. We opted for the Adam optimizer due to its effectiveness in optimizing neural networks. However, to counteract a potentially rapid learning curve, we deliberately set a lower learning rate of 0.0001, as initial observations suggested that a higher learning rate may hinder the model's ability to converge to an optimal solution.

Experiment 2:

Experiment 2 retains the same number of hidden layers as Experiment 1, with only one hidden layer consisting of 128 units. However, a dropout layer with a dropout rate of 0.1 was introduced after this hidden layer to address overfitting.

Despite maintaining the architecture's simplicity by retaining a single hidden layer, the addition of dropout regularization aimed to enhance the model's generalization ability. By randomly deactivating 10% of the neurons during training, overfitting tendencies were mitigated, allowing the model to better generalize to unseen data.

Through Experiment 2, we aimed to evaluate whether the regularization technique would lead to improved performance on both the training and test datasets compared to Experiment 1.

Experiment 3:

Experiment 3 aimed to explore the effects of increasing model complexity by adding more hidden layers and increasing the size of these layers. The decision to use two hidden layers with sizes 256 and 128, respectively, was made to allow the model to capture more intricate patterns in the data.

Moreover, the choice of SGD with Momentum optimizer, combined with a relatively higher learning rate of 0.001, aimed to balance the trade-off between convergence speed and generalization. By incorporating momentum, the optimizer was expected to navigate the parameter space more efficiently and potentially escape local minima.

Despite the increased complexity, dropout regularization with a dropout rate of 0.1 was utilized after each hidden layer to prevent overfitting.

Experiment 4:

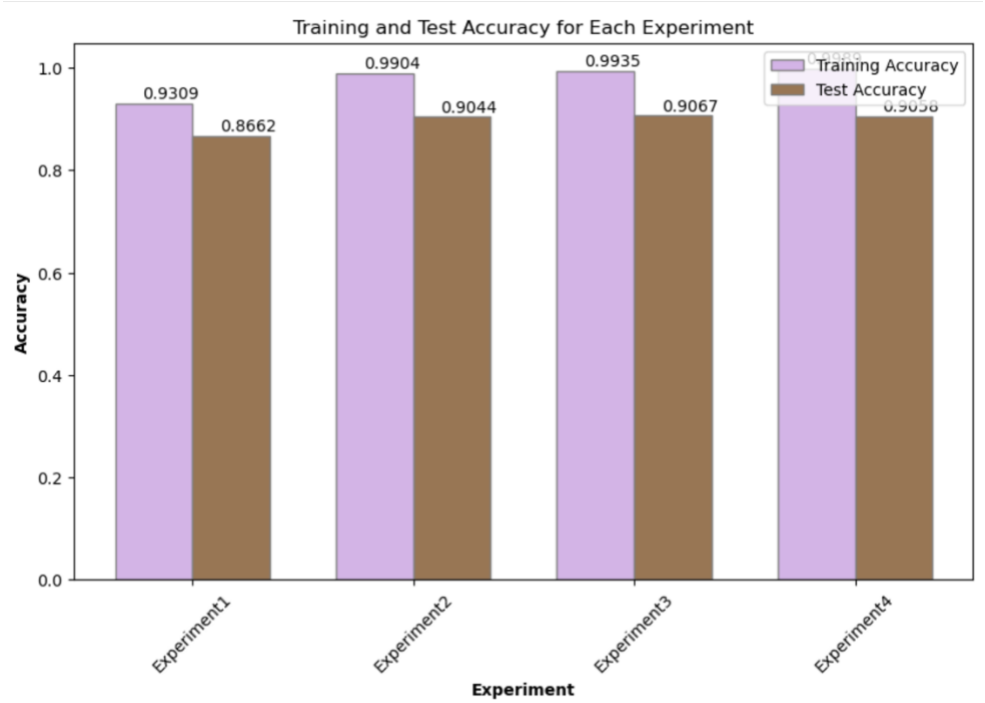
Experiment 4 aimed to evaluate the impact of increasing model depth by adding another hidden layer compared to Experiment 3. This increased depth was intended to enable the neural network to capture even more complex relationships within the data.

The choice of hidden layer sizes (512, 256, 128) was motivated by the desire to progressively decrease the number of neurons in each subsequent layer, allowing the network to extract increasingly abstract features while controlling the model's capacity to prevent overfitting.

Similar to Experiment 3, Experiment 4 utilized SGD with Momentum as the optimizer with a learning rate of 0.001 and weight decay of 0.001. By incorporating dropout regularization with a dropout rate of 0.1 after each hidden layer, Experiment 4 aimed to improve the model's generalization performance by reducing overfitting.

3. Analysis of Model Performance and Limitations

3.1 Model performance



The bar chart above illustrates the training and test accuracy for each experiment, revealing that all models achieved respectable training and test accuracies above 80%. Notably, Experiment 3 attained the highest test accuracy of 90.09%.

Dropout regularization effectively countered overfitting across all experiments, with lower dropout rates (ranging from 0.5 to 0.1) yielding improved accuracy. Experiment 3, featuring two hidden layers, showcased the optimal balance between model complexity and performance. Experiment 4, incorporating three hidden layers, failed to significantly surpass Experiment 3, indicating diminishing returns with heightened model complexity.

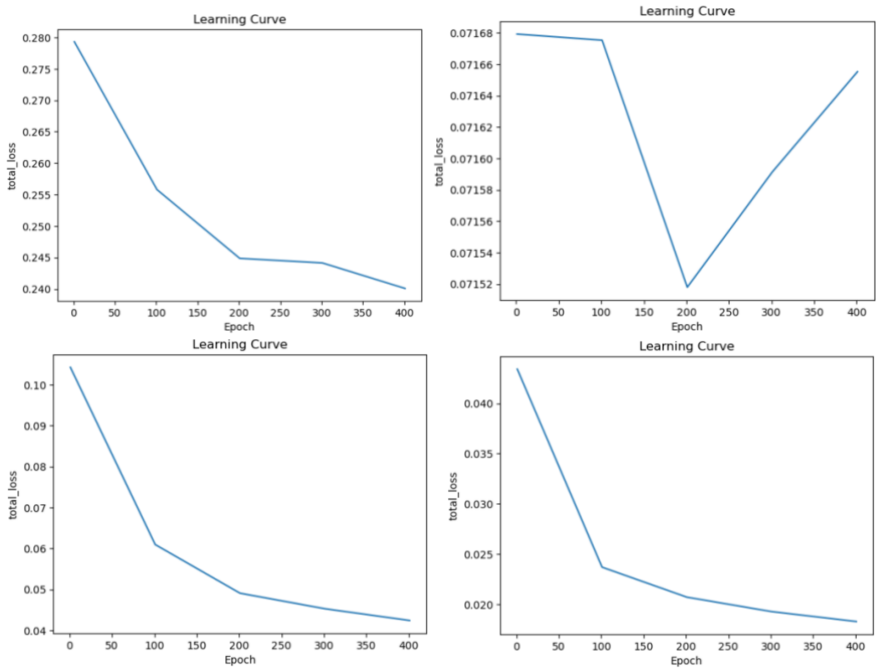
Hence, Experiment 3, employing the SGD optimizer with momentum=0.9, a learning rate of 0.001, and weight decay=0.001, along with two hidden layers sized 256 and 128, and a dropout rate of 0.1, yielded the most promising accuracy results.

3.2 Limitations

The provided code necessitates execution on Google Colab. However, due to CPU limitations, it's not feasible to download the .ipynb file with the private output. Consequently, we opted to execute it on Jupyter Lab and load the data from our local directory instead. Despite our attempts to execute all the code on Google Colab previously, the notebook still contains code for loading data from Google Colab. This limitation may affect the reproducibility of results and the ease of sharing the notebook with others who rely solely on Google Colab.

4. Presentation of Remaining Issues and Recommendations

The learning curve of Experiment 2 appears to underperform compared to other experiments, suggesting potential challenges in finding the optimal balance of hyperparameters and optimizers. Despite conducting several experiments, determining whether adjusting certain rates would enhance model performance remains inconclusive.



Furthermore, the learning curve displayed a straight-line pattern due to recording total loss values at intervals of every 100 epochs. This limited granularity resulted in a less detailed representation of the model's learning dynamics. To gain more nuanced insights, it's recommended to record loss values at shorter intervals, such as every epoch.

While the models achieved satisfactory performance, there is still room for improvement in accuracy. Exploring different activation functions, learning rates, and optimizer configurations could optimize performance further. Additionally, increasing the dataset size or implementing data augmentation techniques may enhance model generalization.

Regularly monitoring learning curves and validation metrics during training could offer insights into model convergence and potential issues. Lastly, experimenting with different network architectures, such as convolutional neural networks (CNNs), might better capture spatial dependencies in the image data.