Deep Learning

Assignment 1

Adithya Krishna
Student ID: 25155591
Date: 26-03-2025

Table of Contents

1. Introduction

This project aims to solve a multi-classification problem with the use of deep neural networks. Taking a simple baseline model as a starting point, complex and advanced neural networks are built to do the classification accurately and efficiently. The model suffered from overfitting in the beginning and struggled to generalise well. However, with careful fine-tuning and testing, later models overcame this issue and produced good results. The accuracy metric is used to evaluate the model performance on the test set, while the loss over training and validation sets are used for hyperparameter tuning. Confusion matrices also provide deeper insights into the performance results and areas of improvement.

2. The Data

The Japanese MNIST dataset is used for this project. This medium-sized dataset contains 70000 images of handwritten Hiragana characters. There are 10 different classes in the target variable.

Right when the data is downloaded as npz compressed files, we get the split version as training with 60k images and test with 10k images. Later, for advanced operations, the train split is further divided in (80:20) ratio to make the validation set.

```
# Print dataset sizes
print(f"Training set size: {x_train.shape[0]}")
print(f"Validation set size: {x_val.shape[0]}")
print(f"Test set size: {x_test.shape[0]}")
```

```
Training set size: 48000
Validation set size: 12000
Test set size: 10000
```

Each image in the data is of dimension 28*28. These are flattened to form vectors of size (784,1) and then used for training as a normal structured dataset.

3. The Approach

The core objectives of this multi-classification deep learning network are as follows:-

- have above 80% accuracy
- prevent over-fitting
- maintain simplicity by avoiding unnecessary complexity
- can only use fully connected and dropout layers (CNN not permitted)

3.1 Challenges

This entire project is done in Google Colab under the free tier. Hence the GPU units [Google T4 GPU] run out after a while and only the CPU will be available which is quite slow for deep learning. Several measures were taken to overcome this, which include:-

- a dynamic “device” variable which chooses GPU [“cuda”] whenever available
- saving model (parameters and weights) just after training
- saving train and validation set losses

Even when the GPU is unavailable, the saved model can be loaded to the CPU and used for performance analysis. Also, the train and validation set losses can be loaded to plot the learning curve and assess the overfitting issue.

3.2 Data Pre-processing

The following data pre-processing steps were done in order before feeding into the model:-

- Reshaped to have the dimensions [row num, height, width, channel].
- Change datatype to np.float32- it reduces memory usage while maintaining sufficient precision for training.
- Standardise the values in the dataset so that they fall in the range of [0,1] - it helps the model to learn effectively.
- Splitting train dataset to generate a validation dataset- this will later be used for performance analysis and tuning.
- Converting the datasets from NumPy arrays to Tensors allows efficient computation on GPUs, supports automatic differentiation, and integrates seamlessly with PyTorch models and optimisers.
- A data loader is used, which neatly integrates with PyTorch’s training loop. Mini batches improve efficiency and enable batch gradient descent. Also, the shuffle=True argument helps in randomising the order of training data in each epoch to prevent the model from learning unintended patterns.

3.3 Experiment 1: Baseline Model

The baseline model is a simple 2-layer feedforward network with one hidden layer and is suitable for basic classification tasks.

Feature	Details
Architecture	Fully Connected Neural Network (FCNN)
Input Size	784 (Flattened 28×28 images)
Hidden Layer	1 layer with 128 neurons
Activation	ReLU (Rectified Linear Unit)
Output Size	10 (for classification into 10 classes)
Output Activation	None (Handled by CrossEntropyLoss)
Loss Function	CrossEntropyLoss (Combines Softmax and Negative Log Likelihood)
Optimizer	Adam (Adaptive Moment Estimation)
Learning Rate	0.001
Trainable Parameters	(784×128+128)+(128×10+10)=101,770(784 \times 128 + 128) + (128 \times 10 + 10) = 101,770(784×128+128)+(128×10+10)=101,77

CrossEntropyLoss automatically applies the softmax function inside the loss calculation, and hence, an explicit activation function is not needed in the output layer.

ReLu activation function is used here as it is fast, simple, and effective. It helps neural networks train faster and avoid vanishing gradients.

Adam Optimiser combines the benefits of two optimisers (Momentum and RMSProp). The default values for Adam's hyperparameters typically work well in practice without requiring significant tuning making it user-friendly and less time-consuming.

3.4 Experiment 2: (Hidden Layer + Dropout)

Model 2 is a fully connected neural network with 3 layers- an input layer, two hidden layers with ReLU activations, and an output layer.

Feature	Description
Model Name	ExperimentModel2
Input Size	784 (flattened 28x28 image from MNIST dataset)
Layer 1 (fc1)	Fully connected layer (input_size = 784, hidden_size1 = 256)
Activation Function 1 (ReLU)	Applies the ReLU (Rectified Linear Unit) activation after fc1
Dropout 1	Dropout layer with a probability of 0.3 after fc1 and ReLU to reduce overfitting
Layer 2 (fc2)	Fully connected layer (hidden_size1 = 256, hidden_size2 = 128)
Activation Function 2 (ReLU)	Applies the ReLU activation after fc2
Dropout 2	Dropout layer with a probability of 0.3 after fc2 and ReLU to reduce overfitting
Layer 3 (fc3)	Fully connected output layer (hidden_size2 = 128, output_size = 10, for classification)
Output Layer	10 outputs corresponding to class probabilities (no softmax, as CrossEntropyLoss applies it)
Loss Function	nn.CrossEntropyLoss() for multi-class classification
Optimizer	optim.Adam(model2.parameters(), lr=0.001) for adaptive learning rate
Dropout Rate	30% (0.3) drop probability after the first and second layers

It includes the features of the baseline model with two main changes-

- **an additional hidden layer**- this deeper architecture enables better feature extraction and potentially higher accuracy.
- **dropout Layers-** it helps in reducing overfitting by randomly setting a fraction of inputs to 0 during training.

3.5 Experiment 3: (Batch Norm + Aggressive Dropout + AdamW + StepLR)

Model 3 is an advanced neural network with additional techniques for improved performance and regularization. It drops the additional hidden layer added in Model 2 to reduce overfitting and also takes a more aggressive dropout approach. It incorporates batch normalisation for stable learning and uses the AdamW optimiser with L2 regularisation, which further helps in reducing overfitting.

Finally, early stopping is manually implemented to train the model after selecting the optimum number of epochs which gives the best generalisation.

Feature	Description
Architecture	Fully connected neural network with 3 layers (2 hidden layers)
Input Size	784 (corresponding to the flattened 28x28 images)
Hidden Layer 1	`nn.Linear(input_size, hidden_size)` (Input to hidden layer with 128 neurons)
Activation (Hidden 1)	`ReLU` (Rectified Linear Unit activation function)
Batch Normalization	`nn.BatchNorm1d(hidden_size)` (Normalization applied to the output of hidden layer 1)
Dropout	`nn.Dropout(0.5)` (50% dropout probability after each hidden layer)
Hidden Layer 2	`nn.Linear(hidden_size, hidden_size)` (128 to 128 neurons)
Activation (Hidden 2)	`ReLU` (Rectified Linear Unit activation function)
Output Layer	`nn.Linear(hidden_size, output_size)` (Final output layer with 10 neurons)
Output Activation	None (Handled by `CrossEntropyLoss` which includes softmax internally)
Optimizer	`AdamW` (with L2 regularization via `weight_decay=1e-4`)
Learning Rate Scheduler	`StepLR` (Reduces the learning rate by 50% every 50 epochs)
Loss Function	`CrossEntropyLoss` (For multi-class classification)

The following table summarises the key improvements in model 3 over model 2:-

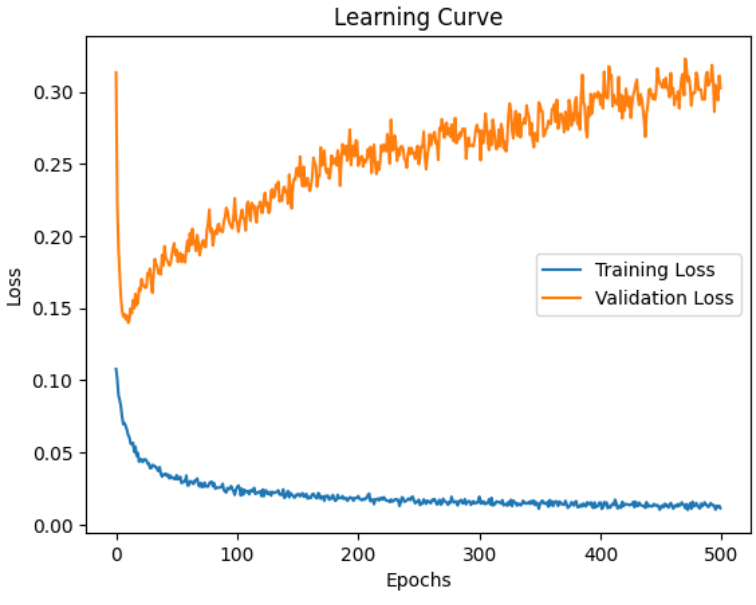
Feature	ExperimentModel2	ExperimentModel3	Benefit
Batch Normalization	No	Yes (BatchNorm1d)	Faster training, stable activations
Dropout	0.3	0.5	Stronger regularization
Hidden Layers	256 → 128 → 10	128 → 128 → 10	Simpler, less overfitting
Optimizer	Adam	AdamW	Better weight decay, better generalization
Learning Rate Scheduling	No	Yes (StepLR)	Controlled convergence

4. The Results

The following table shows the accuracy comparison of the 3 models:-

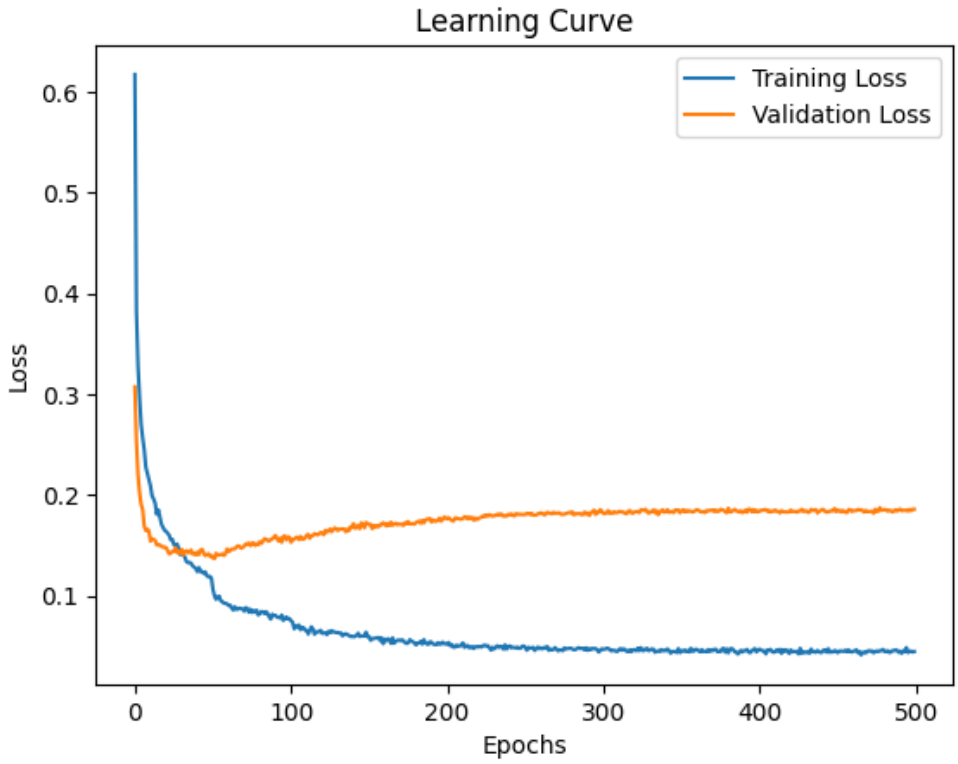
Model	Test Accuracy
Model 1	89.62%
Model 2	90.42%
Model 3	89.70%
Model 3 + early stopping	90.05%

Model 1 has a good accuracy score but it suffers from overfitting.
For experiment 2, a new split of the dataset is introduced to effectively track the model performance- the validation set. Model 2 had an additional hidden layer, and dropout layers were introduced to prevent it from overfitting. Following is the learning curve of Model 2 on train and validation sets:-



The training loss is decreasing, but the validation loss is increasing after the initial dip. This again suggests overfitting. The model fails to generalise well with unseen data.

Hence, Model 3 was built with the main aim of reducing overfitting. As the first step, the additional hidden layer included as part of Model 2 was removed. Stronger dropout layers and additional techniques were employed to produce an 89.70% accuracy and the following learning curve:-

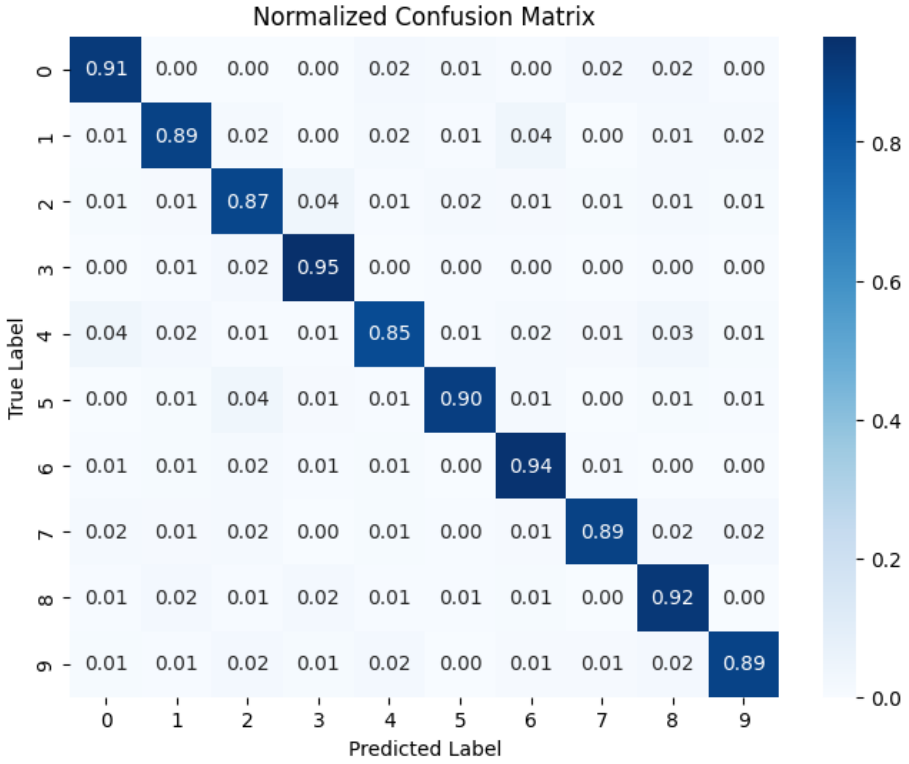


Notice that the best loss (least) is at 50 epochs, after which the validation loss increases. If we stop training at this stage, the model will have better generalisation ability. Such a model is trained as an extension of model 3 which produces an accuracy of 90.05%.

This model is the final and recommended solution for the Japanese MNIST dataset multi-class classification problem.

5. Future Work and Recommendations

Following is the confusion matrix from the final chosen model:-



There are classes where the model needs improvement. For example, only 85% of class 4 values are predicted correctly by the machine. Such errors should be studied on a case-by-case basis and the model needs to be re-trained by giving special consideration to error cases.

Also, with various techniques and early stopping, over-fitting is reduced to a great extent but it is not completely eliminated. More tweaks can be done to further reduce overfitting without affecting the performance.

6. Conclusion

This project is successful in delivering a solution that checks all the core objectives which include high accuracy, eliminating over-fitting and maintaining simplicity. The final model has an accuracy of 90.05% and most of the over-fitting is eliminated. The addition of a validation set proved to be very useful in analysing the model performances and fine-tuning the parameters.

7. References

OpenAI. (2023). ChatGPT (March 2023 version) [Large language model]. https://openai.com/chatgpt