Neural Networks Project 1

Ahmed Ashraf Mohamed, 2103134　　Ahmed Yousri Ali, 2103108

1 Introduction

In this project, we aim to build a neural network from scratch using PyTorch and train it using the MNIST dataset. The goal is to accurately classify handwritten digits using deep learning techniques. To improve the neural network's performance, we will be adding a dropout and normalization layer to the model architecture. Additionally, we will be hyper-tuning the optimizer's parameters to optimize the model's accuracy. By implementing these techniques, we hope to achieve state-of-the-art results in image classification tasks and provide insights into the best practices for building deep learning models.

1.1 Brief about Adam Optimizer

Adam optimizer is a popular and widely used optimization algorithm in deep learning, specifically in training artificial neural networks. It is an adaptive learning rate optimization algorithm that is designed to achieve fast convergence and improve the quality of the model. Adam stands for "Adaptive Moment Estimation," which means that this algorithm adapts the learning rate of each parameter based on the estimates of the first and second moments of the gradients. This algorithm is a combination of two other optimization algorithms: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam optimizer is known for its efficiency, simplicity, and robustness, and it is widely used in various deep learning frameworks such as TensorFlow, Keras, and PyTorch.

1.2 Brief about Cross-Entropy Loss

Cross-entropy loss is a commonly used loss function in machine learning and deep learning for classification tasks. It measures the difference between the predicted probability distribution and the actual probability distribution of the target variable. This loss function is widely used because it is computationally efficient and provides a strong signal to update the model parameters during the training process. Cross-entropy loss is particularly useful when dealing with imbalanced classes, as it can effectively penalize the model for misclassifying the minority class. As a software engineer, understanding cross-entropy loss and its applications is important for developing accurate and efficient machine learning models.

2 Brief about Step 1

We obtained the Kaggle MNIST train dataset and split it into training and validation sets using an 80/20 split. We then saved the resulting train and validation sets to separate files and stored them in memory for use with our Fully-Connected Neural Network model.

3 Brief about Step 2 in the project

We utilized a Fully-Connected Neural Network (FCNN) model with 784 input neurons, 2 hidden layers, and 10 output neurons. We used Adam optimizer, a learning rate of 0.01, and Cross Entropy loss. After 10 epochs, the model achieved a training accuracy of 89% and a validation accuracy of 87%, as shown in the accompanying plot:

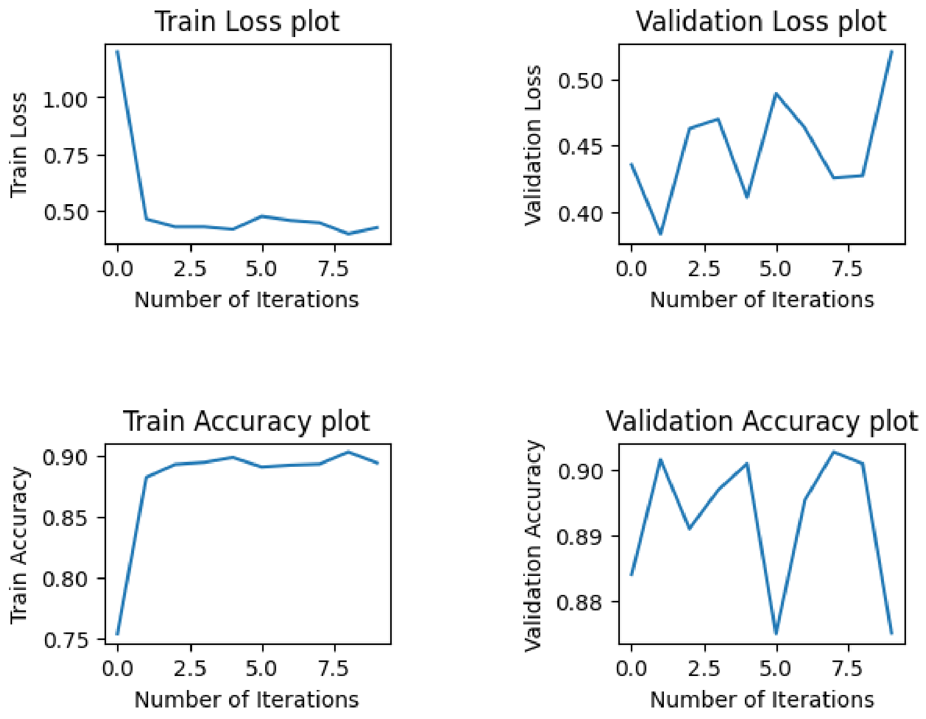

Figure 1: FCNN

4 Brief about Step 3

In the third step, we used the FCNN model with an added normalization layer and a dropout layer of 20%. After running the model for 10 epochs, we achieved a training accuracy of approximately 98% and a validation accuracy of approximately 97% as shown in the accompanying plot:

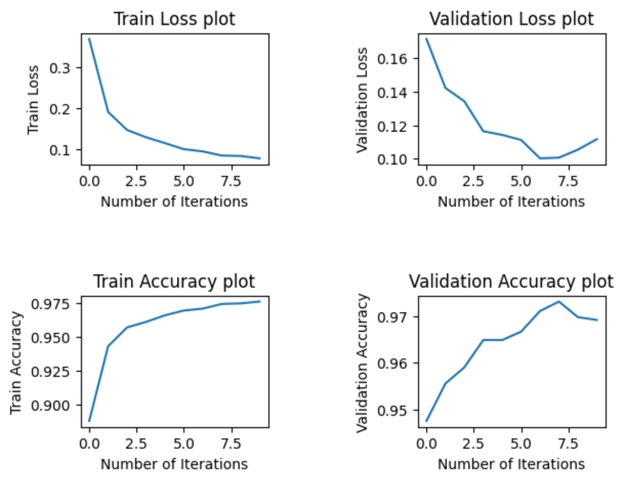

Figure 2: FCNN with Normalization and Dropout

This is an improvement over the FCNN model without a dropout layer, as shown in the accompanying plot:

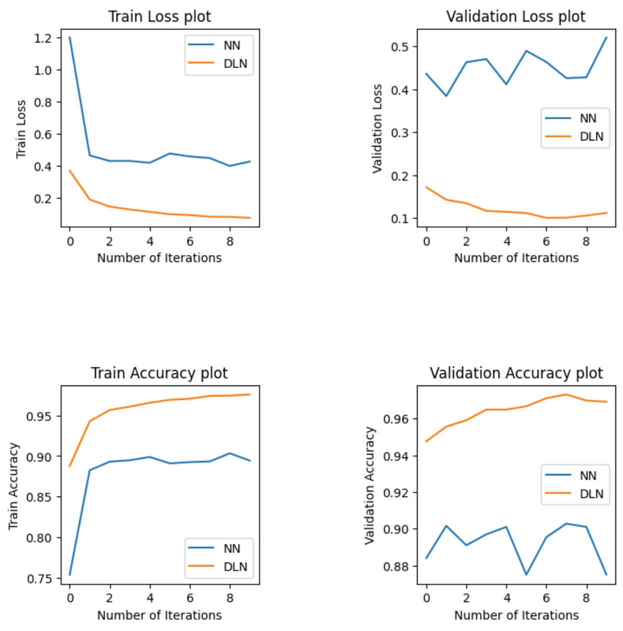

Figure 3: Comparison

5 Brief about Step 4

During the fourth step, we trained three models: the first model had a learning rate of 0.001 and a dropout probability of 20%, the second had a learning rate of 0.0005 and a dropout probability of 40%, and the third had a learning rate of 0.0001 and a dropout probability of 60%. After comparing the three models with the FCNN model, we found that the first model had the highest train and validation accuracy. On the other hand, the third model had the worst train accuracy due to overshooting, and the FCNN model had the worst validation accuracy.

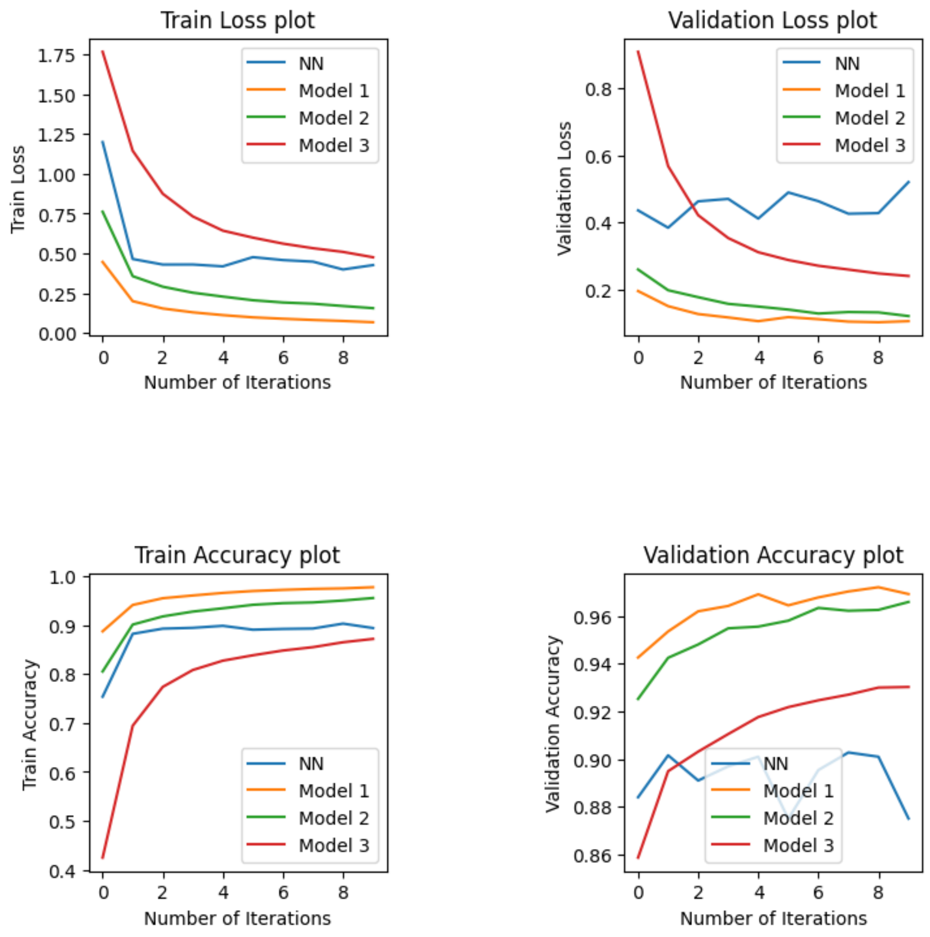

Figure 4: FCNN,DLModel-1,DLModel-2,DLModel-3