Introduction of Dataset

The EMNIST1 (Extended MNIST) dataset is a set of handwritten character digits that have been taken from the NIST Special Database 19 and formatted into a standard size of 28x28 pixels. With the addition of capital and lowercase letters, as well as special characters, the dataset is intended to resemble the well-known MNIST dataset.

There are six distinct divides in the EMNIST dataset, each with a little different focus. The "Balanced" dataset, which we are utilising for this project, is intended to minimise misclassification mistakes brought on by capital and lowercase letters. It has an equal amount of samples for each class. There are 131,600 photos in all, 112,800 of which are in the training set and 18,800 in the test set. The dataset contains 10 digits, 26 letters, and 11 special characters in a total of 47 classes.

For training and evaluating image classification models, the EMNIST dataset is frequently used in machine learning research. The dataset presents a number of difficulties for classification systems, including the variety in handwriting styles and the similarity between some characters.

Structure of MLP and CNN

MLP

On the EMNIST dataset, the code defines and trains a multi-layer perceptron (MLP). Four neurons and a ReLU activation function make up the single hidden layer of the MLP, which is implemented using PyTorch. Using the Adam optimizer and a learning rate of 0.001, the model is trained over a period of 20 epochs.

After training the MLP, the code calculates the test accuracy and generates a confusion matrix to visualize the model's performance on each class.

With various hyperparameters, such as the number of hidden layers, the number of neurons in each layer, the learning rate, and the optimizer, to enhance the model's performance. A alternate neural network architecture, like a convolutional neural network (CNN), which is frequently employed for image classification applications, might also be tried.

Additionally, data augmentation techniques could be used to artificially increase the size of the training set and prevent overfitting. For example, the random rotations, translations, and other transformations to the input images to create new training examples.

CNN

The code begins by importing the required packages, which include Scikit-learn, Matplotlib, PyTorch, and NumPy. Following that, it specifies the batch size, learning rate, and number of epochs for the CNN model's hyperparameters.

The CNN model is then defined by the programme using PyTorch's nn.Module class. The model consists of two fully connected layers, two convolutional layers, and a max pooling layer after each. For both the fully connected and convolutional layers, the model employs the Sigmoid activation function.

The confusion matrix plotting function is then defined by the programme, and it is used subsequently to show how well the CNN performed on the test set.

The programme constructs data loaders for the training and test sets and loads the EMNIST dataset. The CNN model is then initialised, the loss function and optimizer are defined, and the CNN model is trained for the required number of epochs.

The programme computes the training loss and accuracy and adds them to a list during training. After training, the programme calculates the test loss and accuracy, as well as the precision, recall, F1 score, and confusion matrix to assess how well the CNN performed on the test set. The programme then depicts the confusion matrix, training and test losses, and accuracies over the epochs.

Design on MLP and CNN

To optimise the performance of the model in machine learning, a number of hyperparameters must be adjusted. In this instance, we used both multilayer perceptron (MLP) and convolutional neural network (CNN) models, experimenting with different settings of batch size, learning rate, number of iterations, and number of layers.

Batch size refers to the number of samples that are propagated through the model before updating the model parameters. We have found that the batch size of 128 for CNN and 900 for MLP provide the best results compared to other values less than that. However, increasing the batch size may take longer to execute but may result in better performance.

The learning rate is another important hyperparameter that determines the step size at each iteration while moving toward a minimum of a loss function.We have used a learning rate of 0.001 for both CNN and MLP to overcome the overshoot issue, where the model may perform poorly if the learning rate is too high. The learning rate helps the model to learn based on the given parameter. The number of iterations is the number of times the model is trained on the dataset. We have used 20 iterations for both CNN and MLP to obtain good results, as higher iterations may take longer to execute and may produce only slight differences in the output. The number of iterations helps the model to train again and again until it reaches the required optimal value.

The number of layers in a neural network determines the depth of the network and allows for the function to make specific transformations of the data. We have used 2 layers of neural networks in CNN and 4 layers in MLP to obtain good results.

Finally, we have used L2 regularization in the MLP model to improve the performance. Regularization techniques are used to prevent overfitting, where the model may perform well on the training set but poorly on the test set. L2 regularization adds a penalty term to the loss function, which shrinks the weights toward zero. We have used a lambda value of 0.001 to obtain the best regularization, resulting in the best performance of the model.

Overfitting and Under-Fitting Issue:

We faced overfitting issues as we increased the learning rate on the models, but after tuning it to the best value we got the best result in learning of 0.001 in both the models. And also, the number of iterations made a difference in the model training. The lower the number of iterations, the poorer the model gets trained. Hence after using numbers of iterations, we got this number to give the best. The number of hidden layers in MLP made a difference in result as we previously used 2 layers and now updated to 4, giving the best trained model.

Loss and Accuracy

MLP

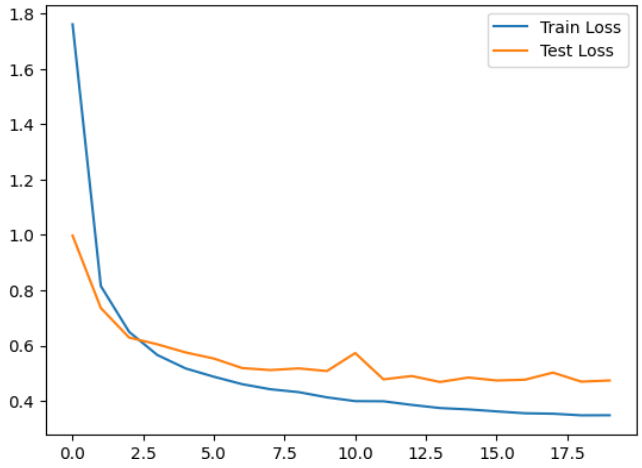
We have used ReLU, Leaky ReLU and Sigmoid as the activation function, SGD, Adagrad and Adam as optimizer. By comparing all these combination we got ReLU as the activation function and Adam as the optimizer to be the best test accuracy of **84.12%** with the minimum loss of **0.5085** using L2 regularisation.

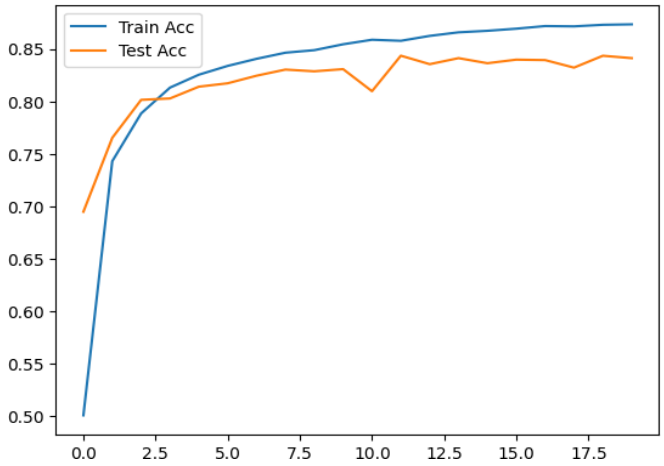
Training loss is the loss function generated using the training data during the training. It shows how effectively the model is absorbing the data. As more epochs are added to the example, the training loss falls from **1.7609** in the first epoch to **0.3488** in the final epoch, showing that the model is becoming more accurate.

Testing loss is the loss function using a different set of data from the training phase during the testing phase. To determine how successfully the model generalises to new data, the test loss is computed. The test loss falls from **0.9978** in the first epoch to **0.4744** in the last epoch, showing that the model is becoming more general.

Training accuracy is the number of correct predictions the model made using the training data divided by the total number of training data points. The training accuracy rises from **0.5005** in the first epoch to **0.8735** in the last epoch, demonstrating that the model is becoming more accurate at predicting the labels of the training data.

Testing accuracy measures how many right predictions the model made on test data in relation to all test data points. The model is improving as the test accuracy rises from **0.6947** in the first epoch to **0.8413** in the last epoch.





CNN

The outcomes display how well a machine learning model performed throughout training and testing. Given that it provides accuracy, precision, recall, and F1 score, and it's a classification model.

Training loss is the loss function generated using the training data during the training. It shows how effectively the model is absorbing the data. As more epochs are added to the example, the training loss falls from **1.7502** in the first epoch to **0.2121** in the final epoch, showing that the model is becoming more accurate.

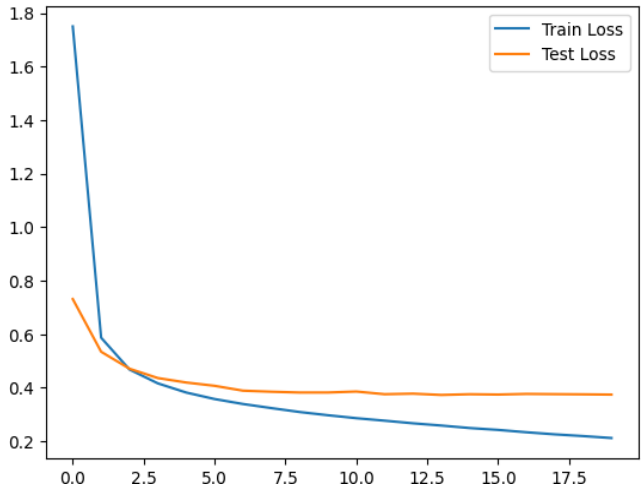
Testing loss is the loss function using a different set of data from the training phase during the testing phase. To determine how successfully the model generalises to new data, the test loss is computed. The test loss

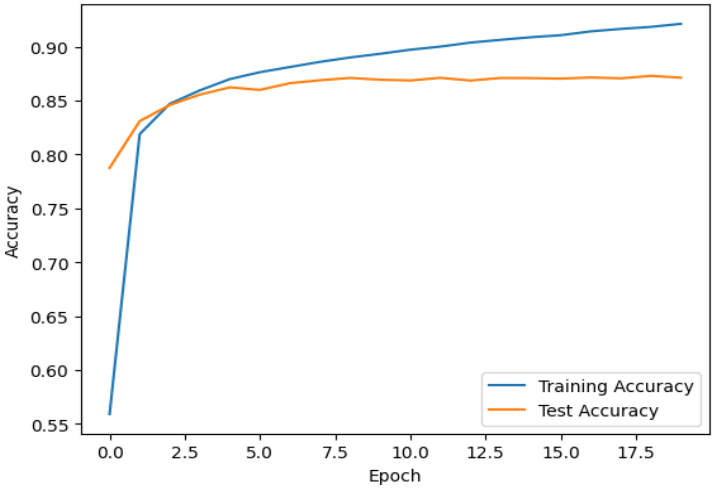
falls from **0.7313** in the first epoch to **0.3743** in the last epoch, showing that the model is becoming more general.

Training accuracy is the number of correct predictions the model made using the training data divided by the total number of training data points. The training accuracy rises from **0.5591** in the first epoch to **0.9212** in the last epoch, demonstrating that the model is becoming more accurate at predicting the labels of the training data.

Testing accuracy measures how many right predictions the model made on test data in relation to all test data points. The model is improving as the test accuracy rises from **0.7875** in the first epoch to **0.8712** in the last epoch.

We have used ReLU, Leaky ReLU and Sigmoid as the activation function, SGD, Adagrad and Adam as optimizer. By comparing all these combination we got Sigmoid as the activation function and Adam as the optimizer to be the best test accuracy of **87.12%** with the minimum loss of **0.3822** without using any other parameters.





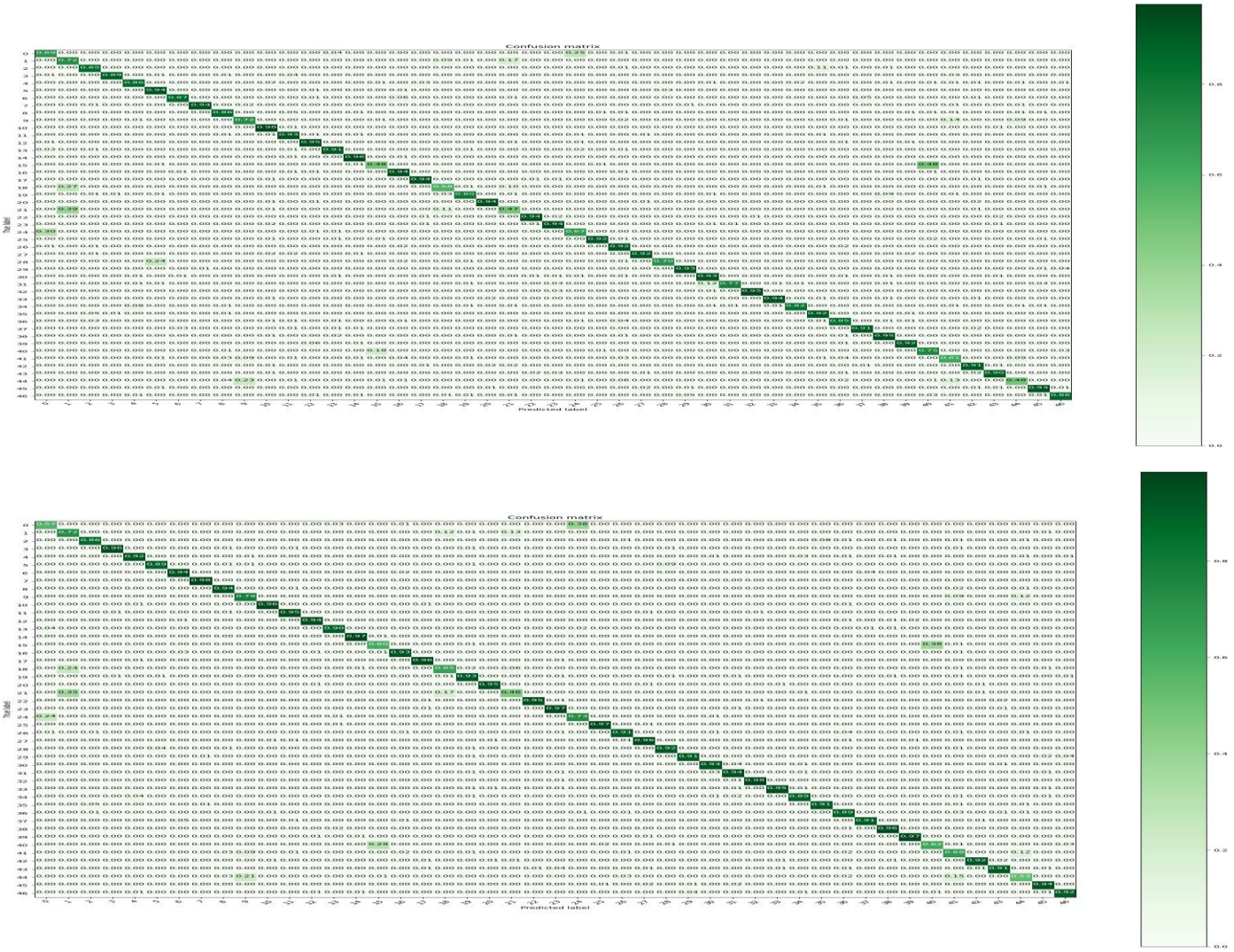
Result

As the final test accuracies of MLP and CNN, it is concluded that CNN performs best when compare to the MLP for the given dataset.

As predicted the accuracy of CNN is **87.12%** and the accuracy of MLP is **84.12%,** hence the image classification of CNN is predicting the best training and testing accuracies of the given dataset.

Below are the result of the confusion matrix:

MLP & CNN



Final conclusion

The set of code includes implementations for two distinct neural network models—a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN)—for training and testing on the EMNIST dataset. The handwritten characters in the EMNIST dataset include 10 numerals, 26 capital letters, and 11 lowercase letters. The CNN model has numerous convolutional and pooling layers followed by fully connected layers, whereas the MLP model has four hidden layers with ReLU activation. Both models employ an optimizer for optimisation and the cross-entropy loss function. The best accuracy is recorded while tracking the training and test losses and accuracies in distinct lists during training. The models' performance is further assessed using a confusion matrix function, which computes measures including precision, recall, F1-score, and accuracy using the features of scikit-learn. Overall, employing both MLP and CNN models, the code offers a complete solution for training and testing neural networks on the EMNIST dataset.