1. Introduction

This project aims to create two types of neural network, Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) that can solve the Extended MNIST (EMNIST) classification problem. The EMNIST is a set of handwritten character digits from the NIST Special Database. Each character digit is converted to a 28x28 pixel image format where its dataset structure matches the MNIST dataset. The datasets include 6 different splits, and this project will focus on the “Balanced” dataset that has 112,800 training samples, 18,800 testing samples and 47 balanced classes.

The basic structure of the proposed MLP is of 4 hidden layers with input layers of 784 neurons and output size of 47, indicating the 47 classes in the dataset. The network structure is built using nn.Sequential that acts as a container where modules can be added and links the output to inputs sequentially for each subsequent modules.

On the other hand, the basic structure of the proposed CNN is made up of first applying a 2D convolution to the input image, followed by an activation function and a 2D max pooling to reduce the image dimensions and finally the layer is flattened and treated as a fully connected layer.

1.1 Designing the Neural Networks

Various techniques and hyperparameters were tested to obtain the set of possible combination that achieve the highest classification accuracies. Firstly, the techniques were chosen based on sequential testing and next, the hyperparameters were tuned based on the best combination of techniques found. The list of techniques explored in order are activation function, optimizer, batch normalization, regularization, dropout and finally, adaptive learning rate. A model is created for different types of each technique and the type of technique which gives the highest accuracy is chosen to be integrated with the neural network. The hyperparameters tuned thereafter are the number of hidden layers and number of hidden neurons for each hidden layer.

An initial MLP model was created which consisted of 4 hidden layers with ReLU activation function, hidden neuron size per hidden layer of 392 with constant learning rate, Stochastic Gradient Descent (SGD) optimizer, no regularization, no dropout, and no batch normalization.

The base CNN model is made up of two convolutional layers with a filter size of 5 x 5, ReLU activation function and max pooling layers of size 2 and stride 2. Subsequently, the data is transformed into a uni-dimensional layer that connects to the output layer.

Furthermore, an epoch size of 10 was chosen for both the models as it was computationally more expensive and time consuming to run a higher number of epochs. Limited GPU was used on Google Colab when exploring the various techniques and the CPU was used for executing the base and final models of MLP and CNN.

1.2 Performance of Techniques

The mean K- cross validation accuracy, where k = 5, is used to evaluate the performance of each of the techniques and hyperparameters.

The mean accuracy of the base MLP model proposed in Section 1.1 is approximately 56.2% and that for the base CNN model is 83.1%. The following section will explore how each technique will affect the performances of the base models.

1.2.1 Activation Function

The activation function between each hidden layer is explored and the mean accuracy of the model are presented.

Activation Function	ReLU	LeakyReLU	ELU	Tanh
MLP Accuracy (%)	74.2	74.2	**74.3**	72.7
CNN Accuracy (%)	83.5	83.4	**84.0**	80.4

Table 1: Activation functions and their accuracy

The base model activation ReLU performs better than Tanh but slightly worse than LeakyReLU and ELU. LeakyReLU performs better than ReLU. Using ReLU, no learning happens when the new weight from backpropagation is the same as the old weight when the output value from the layer is negative. However, for LeakyReLU, it introduces a small slope for negative inputs, allowing the model to continue learning. From Table 1, ELU is the best activation function for both MLP and CNN. ELU performs better than LeakyReLU, although slightly, on the specific data and the model.

1.2.2 Optimizer

The activation function between each hidden layer used is ELU as determined in Section 1.2.1. Three different optimizers are explored, and the mean accuracy of the models are presented in Table 2.

Optimizers	SGD	Adam	RMSprop
MLP Accuracy (%)	67.2	**83.7**	83.6
CNN Accuracy (%)	84.2	86.5	**86.7**

Table 2: Optimizers and their accuracy

After each iteration, SGD computes the gradient of the loss function with respect to the weights and updates the weights in the opposite direction of the gradient to minimise the loss. RMSprop extends SGD and modifies its learning rate by taking a moving average of the squared gradient values. Table 2 shows that Adam is the best activation function for MLP while RMSprop is the best for CNN. Adam combines the advantages of SGD with RMSprop. It uses momentum to converge faster, which is an advantage of adopting SGD. The Adam optimizer additionally uses the gradient's moving average to adaptively change the learning rate of each rate, preventing the learning rate from inflating, which is a property of RMSprop.

1.2.3 Batch Normalization

	With Normalization	**Without Normalization**
MLP Accuracy (%)	**84.3**	83.7
CNN Accuracy (%)	**87.3**	86.7

Table 3: Batch normalization and their accuracy

Batch normalisation addresses substantial changes in network input values by standardising the input values to each layer by subtracting the batch mean and dividing by the batch regular deviation. This stabilizes the training step and leads to faster and more stable convergence to a minima, hence the accuracy of both the models with batch normalization are better than without. The difference between the train and validation accuracy of the batch normalization is small and thus suggesting that the model with batch normalization performs well.

1.2.4 Regularization

Three sets of L1 and L2 regularization weights were explored.

Weight Combinations	**L1 = 0.001, L2 = 0.005**	**L1 = 0.002, L2 = 0.006**	**L1 = 0.003, L2 = 0.007**	**Without Regularization**
MLP Accuracy (%)	72.7	71.1	69.5	**84.3**
CNN Accuracy (%)	83.2	82.2	82.2	**87.3**

Table 4: Regularization weights and their accuracy

Although the use of regularization is to prevent overfitting by reducing the model complexity, preventing fitting to noise, and improving its generalizability, from Table 4, it shows that the accuracy without regularization performs better than when regularization is used for MLP and CNN. This could be due to a good initial model fit from the batch normalization in the previous step.

1.2.5 Dropout

	With Dropout	**Without Dropout**
MLP Accuracy (%)	81.6	**84.3**
CNN Accuracy (%)	87.3	**87.3**

Table 5: Dropout and their accuracy

Dropout is another regularisation strategy that randomly removes neurons from the forward layer during training to avoid overfitting. A high dropout rate may result in underfitting, and its utilisation is also influenced by L1 and L2 regularisation. According to Table 5, the accuracy of MLP without dropout is higher than that of dropout. As a result, dropout will no longer be used in the construction of the MLP model. Using dropout, on the other hand, has no effect on the performance of the CNN model. As a result, dropout is not used in the CNN model.

1.2.6 Adaptive Learning Rate

	StepLR	**ExponentialLR**
MLP Accuracy (%)	84.3	**84.9**
CNN Accuracy (%)	87.7	**87.9**

Table 6: Adaptive learning rates and their accuracy

Pytorch provides two adaptive learning rate approaches for model tuning: StepLR and ExponentialLR. StepLR employs a fixed learning rate for a fixed number of epochs, with the learning rate lowered by a factor at regular intervals. This strategy, however, may result in a slower convergence on the ideal solution. ExponentialLR, on the other hand, can assist in escaping from local minima by allowing the model to make a greater initial jump in weight space and then decreasing exponentially when the weights settle into a satisfactory solution. For both models, a step size of 3 was used and gamma was taken as 0.75 to reduce the step learning rate while a gamma of 0.825 was used for the exponential learning rate. From Table 6, both learning rates are comparable in accuracy for the MLP and CNN implementation, with ExponentialLR having a slight edge. Hence, ExponentialLR will be used in the final MLP and CNN models.

1.3 Hyperparameter Tuning

The CNN model's hyperparameters were not tuned due to the time-consuming nature of running the programme combined with unpredictable internet connections. Experimenting with hyperparameters was futile without better hardware and infrastructure. The ELU activation function, Adam optimizer, batch normalisation, and ExponentialLR were the model tuning approaches used to produce the MLP model after exploring with the various techniques. For the final CNN model, the ELU activation function, RMSprop optimization, batch normalization and ExponentialLR were used. The number of hidden layers and the number of hidden neurons for each hidden layer were the two hyperparameters that were tuned.

1.3.1 Number of hidden layers

We were unable to test for more hidden layers consistently due to infrastructure and inconsistent internet constraints, despite acquiring extra GPU from Google Colab while training the CNN model. Table 7 below shows the tested layers for the MLP model.

Number of hidden layers	Accuracy (%)
3 Layers	84.7
4 Layers	**84.9**

Table 7: Number of hidden layers and their accuracy

Table 7 shows that 4 hidden layers are just slightly better than 3 hidden layers, thus 4 hidden layers were chosen for the final model.

1.3.2 Number of hidden neurons

Skorch, a wrapper which enables to wrap neural network models built using Pytorch with the capabilities and the ease of use of Scikit-learn, was used to wrap the MLP network structure and the essential network requirements such as batch size and optimizers into a simple callable. GridSearchCV was used to determine the best combination of number of neurons in each hidden layer from a combination of hidden neurons of sizes 392, 294, 196, with a cross-validation of 4. The number of hidden neurons used in the models since the baseline model was a constant 392 for each hidden layer.

Number of hidden neurons (Layer 1, Layer 2, Layer 3, Layer 4)	Accuracy (%)
392, 392, 392, 392	84.7
392, 392, 196, 196	**84.8**

Table 8: Number of hidden neurons for each hidden layer and their accuracy

From Table 8, the number of hidden neurons used for hidden layers 1 to 4 are 392, 392, 196 and 196 respectively in the final model as it has a slight advantage in accuracy compared to the number of neurons in the base model. The final MLP model is obtained by adding these findings to the model obtained thus far.

2. Results

2.1 MLP Results

The average training time of an epoch for the MLP model is 22.04 seconds.

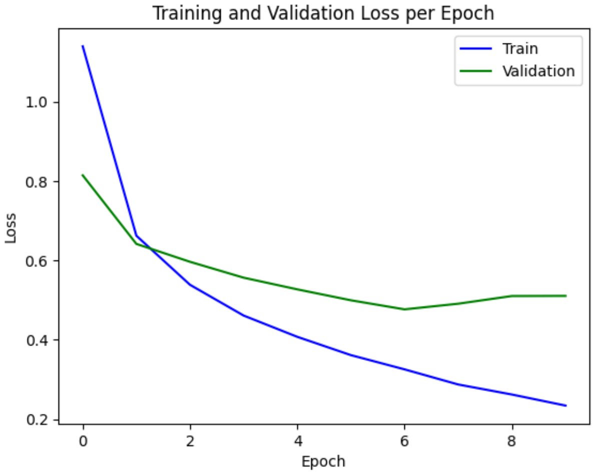

Figure 1: Graph of Loss vs Epoch

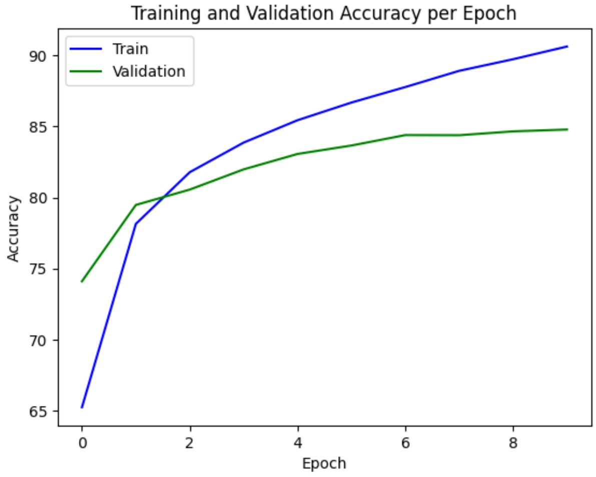

Figure 2: Graph of Accuracy vs Epoch

Figure 1 shows that the loss of the training and validation dataset decreases as the number of epochs increases, however we also see that the model is overfitted as it performs better on the training dataset but poorly on the validation dataset. Figure 2 shows that the training and validation dataset is relatively proportional to the increase of the epoch. However, the training accuracy is greater than the validation accuracy at epoch = 2, and the graphs continue to diverge as the epochs increase. This suggests that the model might be overfitted and that the model is not able to generalise very well with external data.

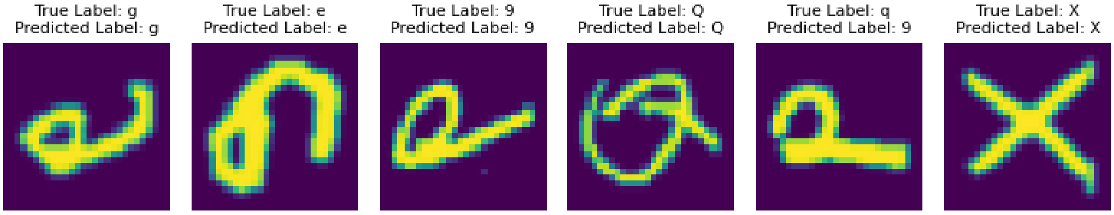

Figure 3: Prediction of 6 samples with true and predicted labels

Figure 3 depicts the first six data objects from the testing dataset and indicates that the MLP model performed well in categorising the handwritten characters.

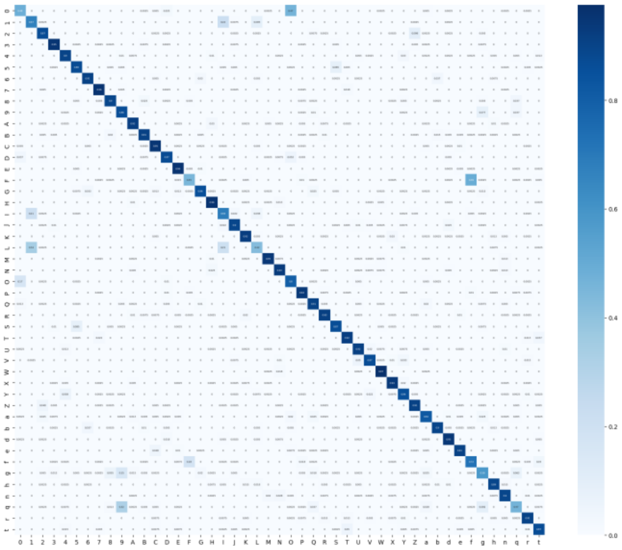

Figure 4: Confusion Matrix of MLP model

From the confusion matrix, the average precision score, recall, f1-score, and accuracy are as follows: 0.843, 0.842, 0.840, 0.842.

2.2 CNN Results

The average training time of an epochs for the CNN model is 170.54 seconds.

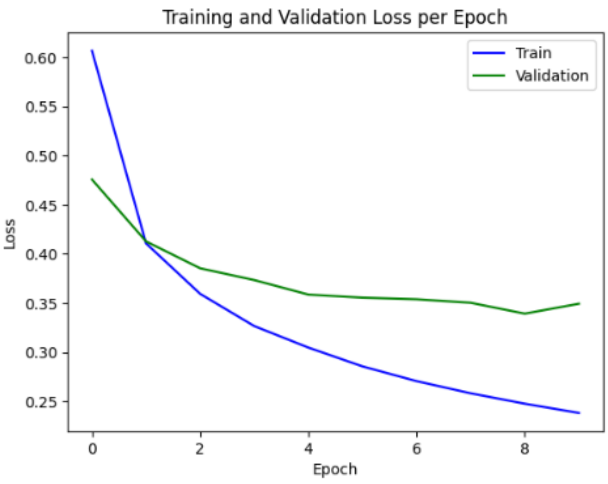

Figure 5: Graph of Loss vs Epoch

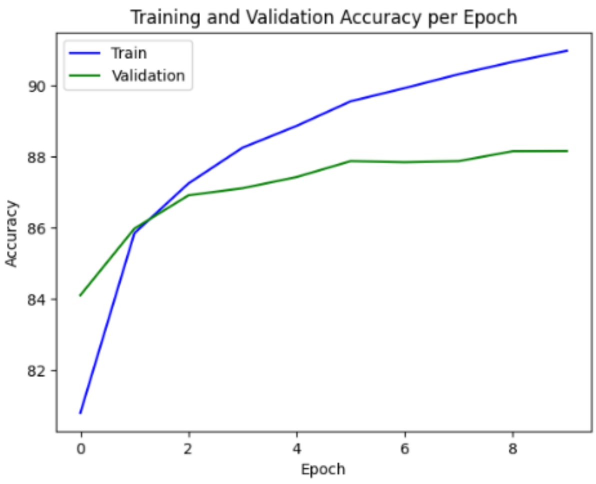

Figure 6: Graph of Accuracy vs Epoch

Figures 5 and 6 have very similar performance as compared to the MLP model. MLP has a higher validation loss and lower train loss as compared to the CNN model, while the CNN has a higher validation accuracy compared to the MLP model. This goes in direction with the expectation that CNN models perform much better than MLPs on image classification tasks. However, as can be clearly seen from the graphs of both models, the models are overfitted to the dataset even at such a low number of epochs of training.

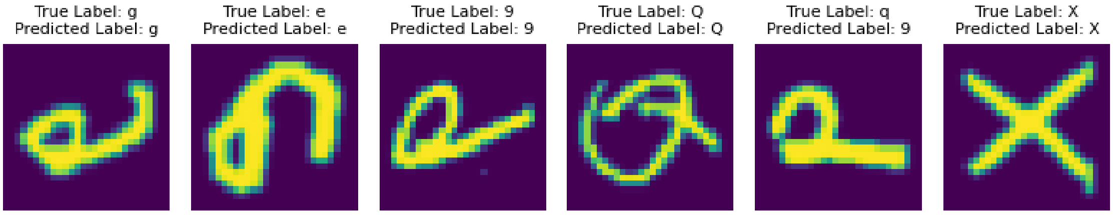

Figure 7: Prediction of 6 samples with true and predicted labels

Figure 7 shows that the CNN also shows good performance in classification of these handwritten character images.

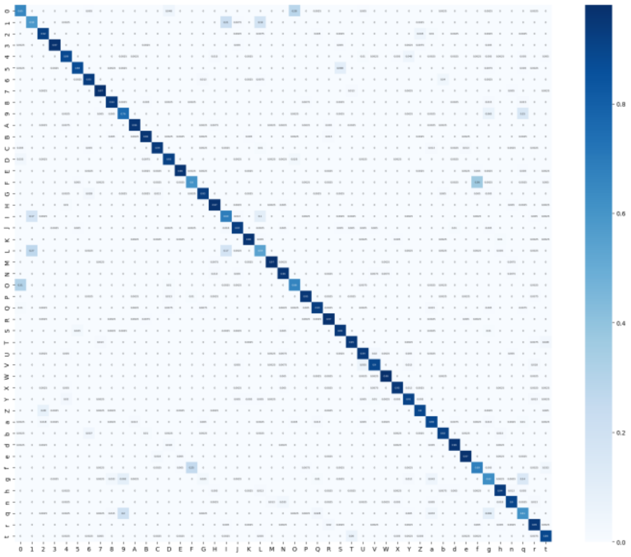

Figure 8: Confusion Matrix of CNN model

From the confusion matrix, the average precision score, recall, f1-score, and accuracy are as follows: 0.879, 0. 879, 0. 879, 0. 879.

2.3 Evaluation

MLP testing accuracy is 84.2% while the CNN testing accuracy is 87.6%. Furthermore, the average precision score, recall, f1-score of the CNN model is also consistently higher than that of the MLP model. The CNN model does perform better in classification of the EMNIST dataset as compared to the MLP model. The difference in time taken to train the MLP model and CNN model is 148.55 seconds. The CNN model is approximately 87% slower than the MLP model in training time. The difference of 3.4% of CNN performing better than the MLP model might not be significant enough to warrant the use of CNN model in this application as it is computationally expensive and time consuming to train the model to produce the results. The MLP model was easier to implement and quicker to train. However, with better hardware and an absence of a time constrain, the CNN model will be preferred in the classification of the EMNIST dataset.

3. Conclusion

Fine tuning and training the models is a time-consuming and complex procedure. There are numerous variations and procedures available to obtain the best model feasible. stronger processing power might have greatly decreased training time, allowing for the prospect of training a larger range of possibilities to develop a stronger model capable of solving the classification issue with more accuracy. There are tools available to aid in hyperparameter tuning, such as Optuna [1] and Ray Tunning [2], which could be considered in future works.

References

[1] “A hyperparameter optimization framework,” *Optuna*. [Online]. Available: https://optuna.org/. [Accessed: 16-Apr-2023].

[2] “Ray 2.3.1,” *Tune: Scalable Hyperparameter Tuning - Ray 2.3.1*. [Online]. Available: https://docs.ray.io/en/latest/tune/index.html. [Accessed: 16-Apr-2023].