CA Assignment 2
MLP & CNNs

ASSIGNMENT REPORT SUBMITTED FOR ASSSESMENT OF COMP534 (Applied Artificial Intelligence)

BY

SANTHOSH ARUNAGIRI -201586816
VISHAL - 201678285

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF LIVERPOOL
LIVERPOOL L69 3BX

INTRODUCTION

Extended- MINST Dataset is a collection of handwritten characters in the format of image that were originally extracted from MINST dataset. The EMNIST dataset includes a total of 814,255 images, with 47 balanced classes representing uppercase and lowercase letters, digits, and symbols.
This dataset was created to give a larger and diverse version of MINST dataset.
The EMNIST dataset is divided into six subsets, each with a different combination of classes and number of images. For this project, we will use the "balanced" subset, which contains 47 classes with a total of 112,800 images.
The EMNIST dataset is often used in research on character recognition, machine learning, and computer vision. The dataset is also useful for testing and comparing different algorithms and models, such as neural networks and support vector machines, for character recognition tasks.

DESIGN OF MLP

- This multi-layer perceptron (MLP) neural network model is implemented using PyTorch.
- The model comprises four completely linked layers, each having 128 nodes, 64 nodes, 32 nodes, and 47 nodes. A flattened image tensor with 28*28=784 features serve as the model's input.
- While initialising the model, hyperparameters for the activation function, dropout rate, batch normalisation, and L1 regularisation can be provided.
- The forward function uses the input tensor's linear transformation and activation functions to complete the forward run through the model.

DESIGN OF CNN

- The CNN class is defined as a subclass of nn.Module, which is a base class for all neural network modules in PyTorch.
- Two convolutional layers (conv1 and conv2), two fully connected layers (fc1 and fc2), and perhaps batch normalisation layers (bn1, bn2, and bn3) are all initialised by the __init__ method of the network, depending on the value of the batch_normalization parameter. The dropout_rate parameter indicates the dropout rate used in the dropout layer, whereas the activation parameter determines the kind of activation function that is utilised.
- The network's forward pass is specified by the forward procedure. The layers are applied to an input tensor x in the order specified in __init__. The tensor is flattened and then passed through the two fully connected layers after going through the two convolutional layers. The network's final output is that of the last fully connected layer.

HYPERPARAMETERS AND ACCURACIES

- A model function, a training dataset, a criterion function, and an optional quantity of cross-validation folds are all passed to the find_best_hyperparameters function.
- Cross-validation is used in conjunction with a grid search strategy to investigate various model hyperparameters.

- Learning rate schedulers, activation functions, optimizers, L1 and L2 regularisation, dropout rates, and batch normalisation are some of the hyperparameters investigated.
- The function outputs the best accuracy and hyperparameters found during the search

HYPERPARAMETERS OR TECHNIQUES

Activation function:

The three activation functions used by the model are ReLU, LeakyReLU, and ELU. Selecting the best activation function enhances the performance of the model because each activation function has advantages of its own. As an illustration, ReLU is a frequently utilised activation function due to its computing effectiveness and ability to solve the vanishing gradient problem. ReLU and LeakyReLU are comparable, however LeakyReLU can avoid the "dying ReLU" issue by allowing a modest gradient when the input is negative. ELU has been demonstrated to be effective in deep neural networks and can prevent the vanishing gradient issue.

Dropout rate:

Dropout is a regularization technique that can prevent overfitting by randomly dropping out (i.e., setting to zero) some of the nodes in the network during training. The likelihood of dropping out a node is controlled by the dropout_rate hyperparameter. A higher dropout rate can minimise the likelihood of overfitting while also increasing the likelihood of underfitting by lowering the model's capacity.

Batch normalisation:

Batch normalisation is a technique that during training normalises the inputs to a layer for each batch. It has been demonstrated to increase training stability and accelerate the learning curve. Whether to use batch normalisation in the model is controlled by the hyperparameter batch_normalization.

L1 regularisation

This method forces the model to have sparse weights by adding a penalty term to the loss function. The strength of the L1 regularisation is controlled by the l1_reg hyperparameter. L1 regularisation can reduce overfitting and improve the generalizability of the model.

Optimizer

The optimizer is responsible for updating the weights of the model during training. SGD, Adam, and RMSprop are the three optimizers that the code uses. The performance of the model can be enhanced by selecting the best optimizer from among those available.

Learning rate

The optimizer's learning rate is controlled by the lr hyperparameter. In addition to speeding up training, a faster learning rate can also cause the loss to diverge or change. While a lower learning rate can stop the loss from diverging, it can also make training take longer.

L2 Regularisation

Weight decay (L2 regularisation) is a method for enabling the model to have tiny weights by adding a penalty term to the loss function. The strength of the weight decay is controlled by the l2_reg hyperparameter. L2 regularisation can reduce overfitting and improve the ability to generalise of the model.

Learning rate scheduler

The learning rate scheduler is responsible for adjusting the learning rate during training. StepLR and ExponentialLR, two learning rate schedulers, are used in the code. Every step_size epochs, StepLR reduces the learning rate by a factor of gamma. Every epoch, ExponentialLR lowers the learning rate by a factor of gamma. By allowing the learning rate to adjust to the training progress, a learning rate scheduler enhances the performance of the model.

OVERFITTING AND UNDERFITTING:

In machine learning models, overfitting and underfitting are frequent problems. When a model performs exceptionally well on the training dataset but poorly on the validation or testing datasets, this is known as overfitting. Underfitting, on the other hand, happens when the model does not perform well on both the training and the validation or testing datasets. there is no overfitting problem. However, the difference between the training and testing accuracies is not large, which indicates that the model might be underfitting the data.

To deal with underfitting, we need to try increasing the complexity of the model, for example, by adding more layers or increasing the number of neurons in the existing layers. Also can explore different activation functions, optimisers, and learning rates.

since this model is simple, increasing the number of neurons in the layers, adding more layers, or trying different activation functions would be good idea.

LOSS AND ACCURACY:

we can observe that the training loss decreases as the number of epochs increases, which is expected as the model learns from the training data. Particularly, we can observe a notable reduction in the training loss from Epoch 1 to Epoch 2, followed by a more steady reduction over the course of the following epochs.

The testing loss also decreases as the number of epochs increases, which indicates that the model is improving in its ability to generalize to new, unseen data. This is demonstrated by the testing loss values' declining trend from Epoch 1 to Epoch 10.

As the model improves and learns to base its predictions more accurately on the testing data, the model's testing accuracy also increases with time. The validation accuracy rises from Epoch 1 to Epoch 5, after which it levels out and makes little progress.

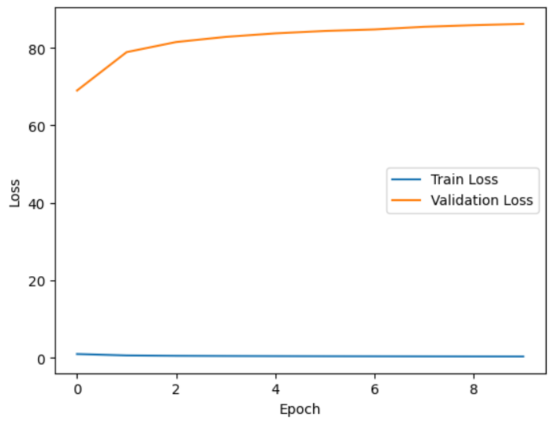
Overall, we can conclude that the MLP model is learning from the training data and is able to generalize well to new, unseen data, as evidenced by the decreasing trend in testing loss and increasing trend in testing accuracy.

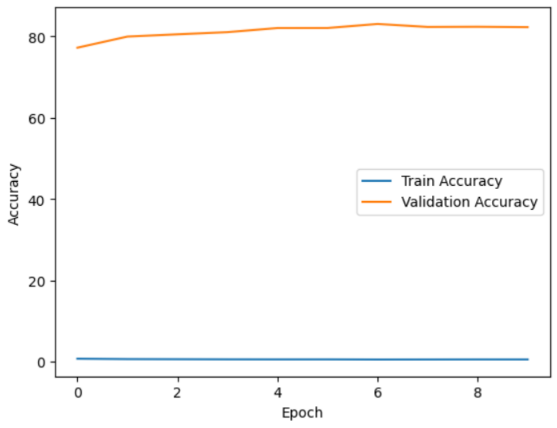
RESULTS - MULTILAYER PERCEPTRON

The best hyper parameters for Multilayer perceptron are,

- Optimizer: ADAM
- Learning rate: 0.01,
- L2 regularisation: 0,
- L1 regularisation: 0,
- Learning rate Scheduler: StepLR,
- Activation: elu,
- Dropout rate: 0,
- Batch normalization: True,
- Iterations: 10

Train accuracy	76.71%
Test accuracy	82.34%
Precision	0.83
Recall	0.82
F1 Score	0.82





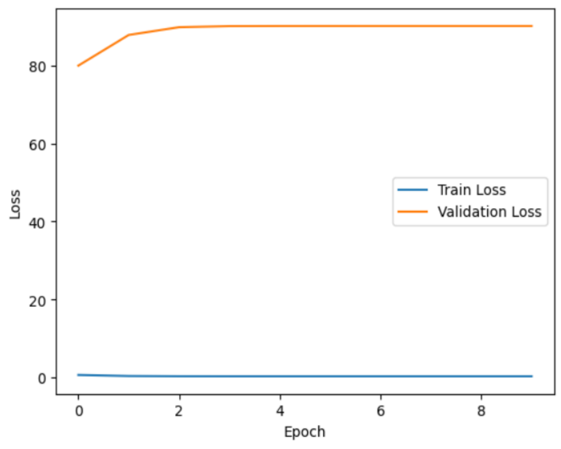
RESULTS – CONVOLUTIONAL NEURAL NETWORK

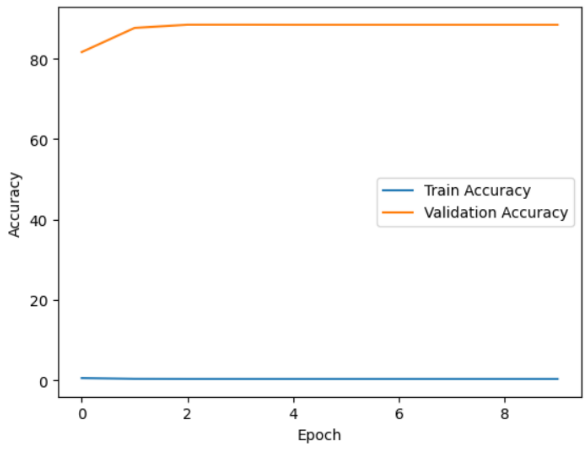
The best hyper parameters for Multilayer perceptron are,

- Optimizer: ADAM
- Learning rate: 0.01
- L2 Regularisation: 0.0001
- L1 Regularisation: 0

- Scheduler: ExponentialLR,
- Activation: relu
- Dropout rate: 0
- Batch Normalization: True
- Iterations: 10

Train accuracy	84.92%
Test accuracy	88.09%
Precision	0.88
Recall	0.88
F1 Score	0.88





OBSERVATION

The results from both the models show that both the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN) achieved reasonable accuracy on the EMNIST dataset, with the CNN achieving slightly better results than the MLP.

MLPs are a type of neural network that is commonly used for classification tasks. They are simple to implement and have fewer parameters than CNNs, making them faster to train. However, it is known that they may struggle with tasks that require the recognition of spatial patterns, such as image classification, which is where CNNs excel.

From our observation we can clearly understand CNN works well than MLP.

CONCLUSION

Through this models we can clearly state CNN is better than MLP in image recognisation over EMNIST dataset. But still we are able to find some underfitting, we believe it must be due to the simple implementation, we might need to try some more new parameters and add some more layers.
From this assignment we were able to learn how to use pytorch. Since it was a combination of many parameters it took a long time to run which impacted our work efficiancy.As this is our first time we were only able to implement a simple model. We believe we could work on getting more efficient accuracy next time by exploring different parameters.

REFERENCES

1. Shin, T. (2023). Pytorch-EMNIST-Classification. [online] GitHub. Available at: https://github.com/tw7366/Pytorch-EMNIST-Classification.git [Accessed 19 Apr. 2023].

2. Hill, A. (2023). *EMNIST-Convolutional-Neural-Net*. [online] GitHub. Available at: https://github.com/austin-hill/EMNIST-CNN.git [Accessed 19 Apr. 2023].
3. Analytics Vidhya. (2021). *Convolutional Neural Network with Implementation in Python*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/.
4. PyTorch Forums. (2021). *Unexpectedly high results on EMNIST-Letters with no augmentation and simple CNN*. [online] Available at: https://discuss.pytorch.org/t/unexpectedly-high-results-on-emnist-letters-with-no-augmentation-and-simple-cnn/120827 [Accessed 19 Apr. 2023].
5. GitHub. (2023). *machine-learning-scripts*. [online] Available at: https://github.com/CSCfi/machine-learning-scripts/blob/master/notebooks/pytorch-mnist-mlp.ipynb [Accessed 19 Apr. 2023].
6. Deeplearningbook.org. (2012). Available at: https://www.deeplearningbook.org/contents/mlp.html.