**COURSE WORK: MSc. Big Data Science with Machine Learning**

**Name-** Dhruv Verma

**Subject-** Neural Networks and Deep Learning ECS7026P

**Student ID-** 210879364

**KUZUSHIJI-MNIST CLASSIFICATION**

**TASK 1: DATA LOADERS**

We have used modified functions from *d2l.py* to load the required datasets. The data loader was modified to fetch data from *torchvision.datasets.KMNIST*. The data loader takes in batch size and picture resize (in our case 32x32) as argument inputs. We will experiment with different batch sizes and compare the results. The loader primarily converts the obtained images into tensors and splits them into training and testing sets. The training set is also shuffled randomly during loading. We allocated 2 CPU threads to the data loader workers to download data efficiently.

**TASK 2: CREATE THE MODEL**

**STEM:** The function of the stem is to convert input tensor images of size **H\*W** (32\*32 in our case) into smaller-sized patches of **K\*K** (4\*4 in our case). Also, the number of patches is Np which is non-overlapping because we passed argument *stride=4* which ensures that the new patch is always 4 pixels ahead of starting of the previous one. Each patch gets reshaped into a vector, then transformed into a feature vector of dimensions d: **16\*4**. Due to the behaviour of *unfold* library [1], unfolded data is transposed before getting transferred to further modules. The data is further passed through a batch normalisation, linear layer, and activation function ReLu, in that order.

**BACKBONE:** The backbone blocks consist of two Multilayer Perceptrons or MLPs. Each MLP consists of 2 linear layers with batch normalisation and a non-linear activation function.

Each MLP gets transposed input from the *stem* class to ensure the same dimensionality throughout the block. Lastly a third non-linear activation function ReLu is applied before the final output.

**NETWORK AND CLASSIFIER- DVNET:** The DVNET class calls the *stem* and 4 *backbone* blocks (5 blocks were considered as one of the experiments). We are progressively increasing hidden layers in each *backbone* block passed starting from 64 up to 512. Lastly, the output is passed through the last classification layer consisting of *flatten* layer and *linear* layer outputting to 10 final classes.

**TASK 3: LOSS & OPTIMIZER**

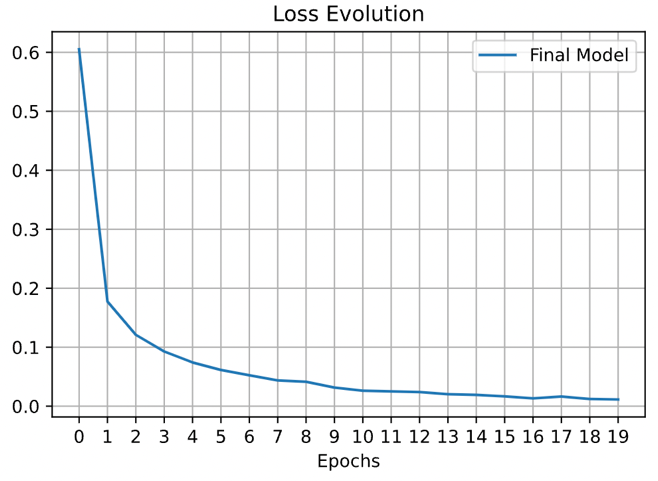
For loss, we are using Cross Entropy Loss as it gave the best-observed results. For optimizer Stochastic Gradient Descent (SGD) was used.

**TASK 4: EVALUATION**

For evaluation, we have used code snippets from Lab 5 & Lab 7 which allow us to plot the live training graph and produce training loss, training accuracy and testing accuracy in each epoch. The environment used to train the model was JHub provided by the EECS department which gives access to NVIDIA A40 GPU. Given the limited number of resources, changes were made to every hyperparameter possible and get the best results. The output parameters value, i.e, Loss, Training Accuracy and Testing Accuracy for each epoch are stored in a dataframe which helps with analysis and visualisation. Variations of the learning rate, batch size and layers were done to obtain different results.

The best results were obtained at a **learning rate = 0.09, epochs = 20, batch size = 128** with **4 backbone blocks** consisting of **2 batch normalisation** layers followed by **ReLu**.Below we can see the evolution of loss and training/testing accuracy graphs for final model:

|  |  |
| --- | --- |
|  | **Final Model** |
| **Training Loss** | 0.011 |
| **Training Accuracy** | 99.59 |
| **Testing Accuracy** | 94.06 |

Chart, line chart

Description automatically generated

Chart, line chart

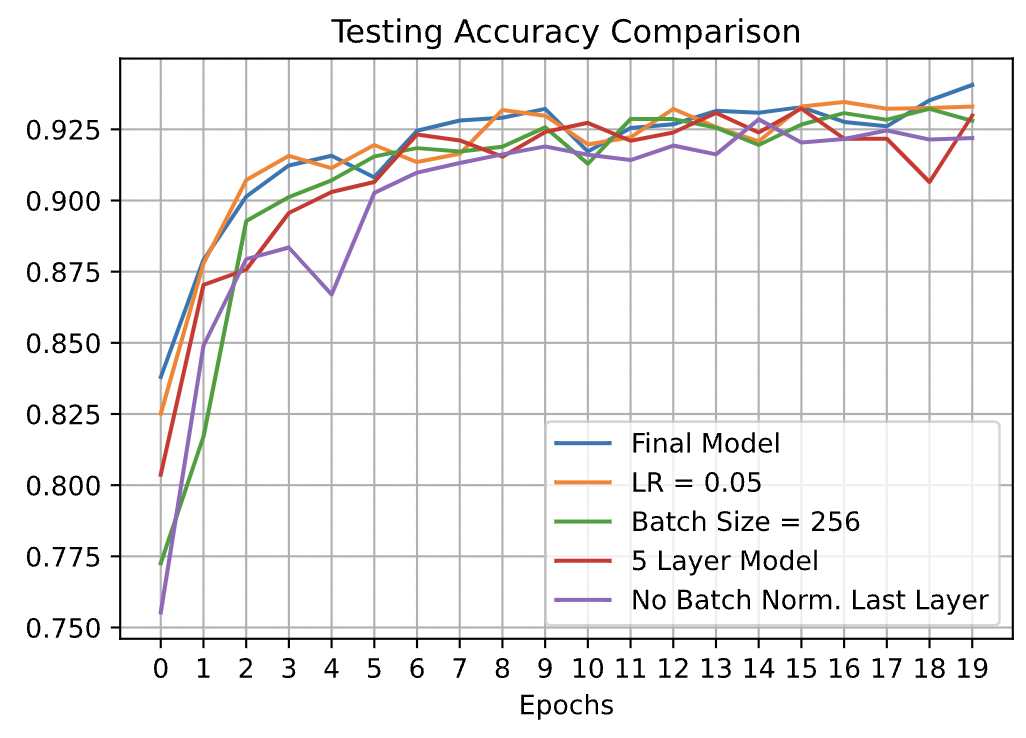
Description automatically generated

Chart

Description automatically generatedBefore getting to the final model, 4 different models were also executed for 20 epochs with varying parameters which included **learning rate = 0.05, batch size = 256, 5 backbone blocks and no batch normalisation in last layer.** The data obtained from each model was stored in dataframes and comparative graphs were produced for all evolution of loss and training/testing accuracies. The following graphs were obtained:

Chart, line chart

Description automatically generated



From the above graphs we can observe that ‘Evolution of Loss’ and ‘Training Accuracy’ were close to same for every model with ‘No Batch Norm. Last Layer’ showing the worst performance in every case. Only in ‘Testing Accuracy’ we can observe can meaningful variations and it is our most important metric for performance.

We can see that our final model with learning rate of 0.09 gives best test accuracy followed by ‘LR=0.05’ and ‘5 Layer Model’. Even though they are close in accuracy, it is not feasible to run 5 Backbone model as it doesn’t give any significant boost to accuracy, inverse happens. Lack of batch normalisation in last layer gives the worst testing accuracy and we can clearly observe that in all graphs. In depth details can be observed from the stored dataframes in the notebook we can provide more insights.

Thus, the final accuracy achieved is **94.06%,** here we have achieved satisfactory result and finalised our network. Further improvements can be made using a bigger dataset, transfer learning, etc.

**REFERENCES:**

[1] Pytorch.org. 2022. *Unfold — PyTorch 1.12 documentation*. [online] Available at: <https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html?highlight=unfold#torch.nn.Unfold> [Accessed 29 July 2022].