NEURAL NETWORKS AND DEEP LEARNING
(ECS7026P)

Vinoth Murugan - 210958005

1. Read the dataset and create data loaders:

KMNIST is called as kuzushiji MNIST and it is substituted from the MNIST dataset. It is a large dataset of handwriting and, used for image processing. Kuzushiji-49 name suggests, is a larger unbalanced dataset with 48 Hiragana characters and one Hiragana iteration mark. There are 49 classes 28x28 grayscale, 270,912 pictures.

2. Create a model:

Stem:

The input image is split into non-overlapping patches by the model's stem layer. With patches 7 and feature size as 300 and having 256 batch size. Whereas patches*patches with 10 output class.

Backbone:

The backbone is initialized by the b1 and b2 that has multi-layer perceptron. The tensor is reversed to the first Multi-layer perceptron and sent to the second multi-layer perceptron. The patch size is 7*7 the output feature is set to 300.
Then after determining the mean feature, and it is passed to the classifier.

```
Net(
  (Unfold): Unfold(kernel_size=7, dilation=1, padding=0, stride=7)
  (Stem): Linear(in_features=49, out_features=300, bias=True)
  (b1): Sequential(
    (0): Linear(in_features=300, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=300, bias=True)
  )
  (b2): Sequential(
    (0): Linear(in_features=16, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=300, bias=True)
  )
  (classifier): Linear(in_features=300, out_features=10, bias=True)
)
```

3. Create the loss and optimizer:

The numerical instabilities are avoided by the implementation of the softmax cross Entropy. Adam as the optimization method, with a 0.5 learning rate.

```
loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.0025)
```

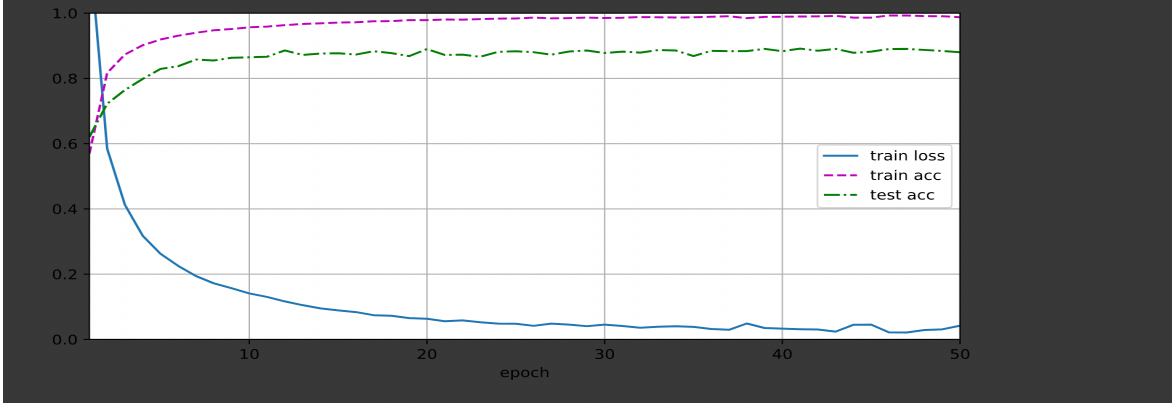
4. Write the training script to train the model

The curves with The evolution of the loss:

```
epochs = 50
fn = train_ch3(net, train_iter, test_iter, loss, epochs, optimizer)

with open('results.json', 'w') as fp:
    json.dump(fn, fp)
```

```
{'epoch:1': {'train acc': 0.5692833333333334, 'test acc': 0.6206, 'train loss': 1.2094193515459697}, 'epoch:2
```



In the above graph the evolution of loss(blue) was first above 1.0 and eventually it came to below 0.2

The curves for the evolution of the training and validation (test) accuracies

As shown in above graph, the training accuracy increases from the 0.6 and finally reaches to 1.0 and the testing accuracy same as training accuracy but finally reaches 0.83 and varies between 0.83 and 0.85

All training details including hyper-parameters used

The hyper-parameters number of epochs, learning rate, feature size, patches and size of image.

5. Final model accuracy on KMNIST validation set

The executed accuracies are shown in below image

```
epoch:50': {'train acc': 0.9875333333333334, 'test acc': 0.8801, 'train loss': 0.041999349459757404}}
```

Test accuracy – 88.01%
Training accuracy – 98.75%
Train loss – 0.041999