NN & Deep learning Project 2022 - Report.

1.We assign a batch size of 512 and load our fashion MNIST dataset using my utils.

```
[7]  batch_size = 512    # setting up the batch size and loading the data.
     train_iter, test_iter = mu.load_data_fashion_mnist(batch_size)

     /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: U
       cpuset_checked))
```

2.Also using Einops we have performed the process of attaining non-overlapping patches and we have transformed it into an feature vector and all features are stored in the input matrix "X".
I have built the backbone using two Multilayer perceptron's per block and the input of the second MLP is the output of the first MLP with weights and an non-linear activation function. In this case I have used RELU(ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time).

Finally our Linear classifier takes the input from my last MLP's output and computes the mean feature.

```
def forward(self,x):
  x = self.to_patch_embedding(x)

  x = self.Linear1(x)
  x = self.relu(x)

  x = self.Linear2(x)

  x = self.relu(x)

  x = x.transpose(1,2)


  x = self.Linear3(x)
  x = self.relu(x)

  x = self.Linear4(x)
  x = self.relu(x)


  x = torch.mean(x,dim=1)
  out = self.classifier(x)
  return out
```

3.For the loss function I have used **cross entropy loss** and it is mainly useful as it can describe how likely a model is and the error function of each data point. It can also be used to describe a predicted outcome compared to the true outcome. Also for optimizer I have used ADAM OPTIMIZER because Adam converges faster compared to SGD.

ADAM uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.
Adam optimizer are generally better than every other optimization algorithms, have Adam has faster computation time, and require fewer parameters for tuning.

```
[36] loss = nn.CrossEntropyLoss()   #Defining the loss function
     lr = 0.01

     optimizer = torch.optim.Adam(net.parameters(), lr=lr, weight_decay=0)  #Using ADAM optimizer
```

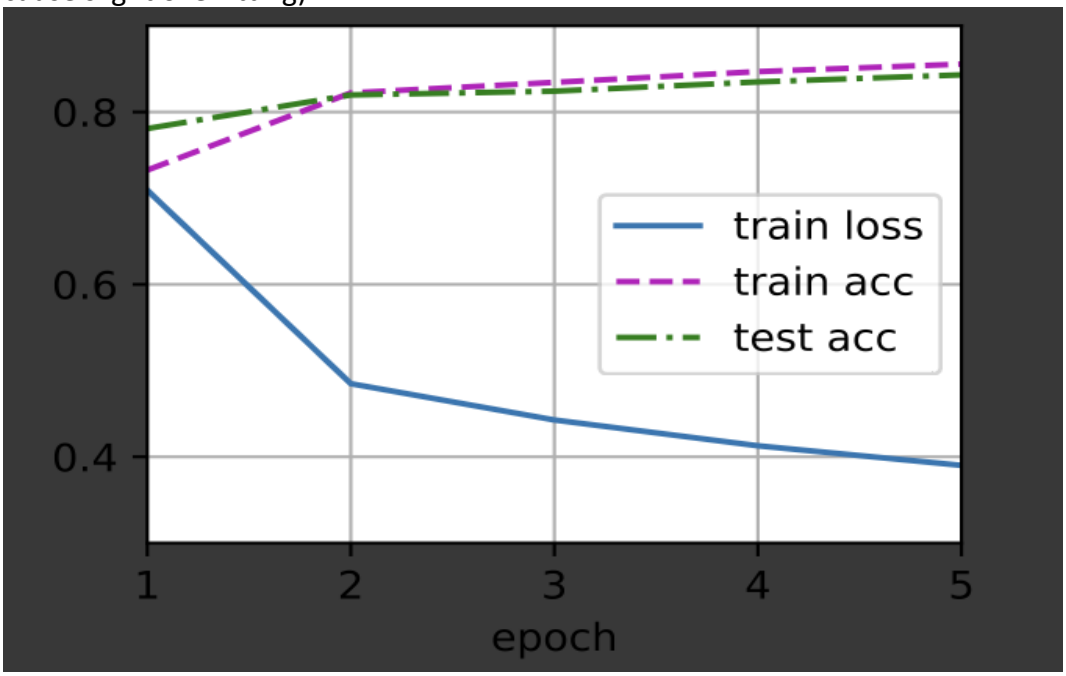
4.

- **Curves for evolution of loss :**

We can clearly see that the curves of the training loss decreases gradually as the number of epochs increase. This indicates that after each epoch we are approaching towards the global minima.

- **Curves for the evolution of training and test accuracies:**

The training accuracy and test accuracy gradually increases and however we can note that the training accuracy is slightly higher than the test accuracy. (This may cause slight overfitting).



- **Training Details with all the hyper parameters used:**

To train our model we consider the number of epochs to be **5 (i.e** One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only Once.)
Also considering the other hyperparameters to fine tune our model I have used the **learning rate as 0.01** and the **weight decay as 0** for our ADAM optimizer.
Also I setup the learning rate to 0.01 in order to increase the efficiency of our model by using small steps.
Our **Relu** activation function also acts as a hyperparameter.

```
[42] def train_ch3(net, train_iter, test_iter, loss, num_epochs, optimizer):
         """Train a model."""
         animator = Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0.1, 0.9],
                             legend=['train loss', 'train acc', 'test acc'])
         for epoch in range(num_epochs):
             train_metrics = train_epoch_ch3(net, train_iter, loss, optimizer)
             test_acc = evaluate_accuracy(net, test_iter)
             animator.add(epoch + 1, train_metrics + (test_acc,))
         train_loss, train_acc = train_metrics

[43] num_epochs = 5 # learning rate 0.01
     mu.train_ch3(net, train_iter, test_iter, loss, num_epochs, optimizer) #Training our model
```

5.Final Model accuracy :
With all the above steps performed the final model accuracy that I obtained on the Fashion Mnist validation set is : **84.31**

```
[44] evaluate_accuracy(net, test_iter)  # Final Model accuracy on the validation set.

     /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarn
       cpuset_checked))
     0.8431
```

x----------x