### Importing libraries

```
from __future__ import print_function #importing print function from the __future__ module. This module is used to inherit
import torch #importing pytorch
import torch.nn as nn #importing nn module from pytorch. it provides basic classes for building a newural network
import torch.nn.functional as F #importing FUnctional submodule from nn module, It provides functions for performing neura
import torch.optim as optim #this module provides various optimisers like sgd, adam
from torchvision import datasets, transforms #importing datasets and transformers module. to load datasets from pytorch an
```

### Network

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1) #input-28 OUtput-28 RF-3
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1) #input-28 OUtput-28 RF-5
        self.pool1 = nn.MaxPool2d(2, 2) #input-28 OUtput14 RF-10
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1) #input-14 OUtput-14 RF-12
        self.conv4 = nn.Conv2d(128, 256, 3, padding=1) #input-14 OUtput-14 RF-14
        self.pool2 = nn.MaxPool2d(2, 2) #input-14 OUtput-7 RF-28
        self.conv5 = nn.Conv2d(256, 512, 3) #input-7 OUtput-5 RF-30
        self.conv6 = nn.Conv2d(512, 1024, 3) #input-5 OUtput-3 RF-32
        self.conv7 = nn.Conv2d(1024, 10, 3) #input-3 OUtput-1 RF-34

    def forward(self, x):
        x = self.pool1(F.relu(self.conv2(F.relu(self.conv1(x))))) #first convolution bloch (conv1-->relu-->conv2-->relu-->
        x = self.pool2(F.relu(self.conv4(F.relu(self.conv3(x))))) #second convolution bloch (conv3-->relu-->conv4-->relu--
        x = F.relu(self.conv6(F.relu(self.conv5(x)))) #third convolution bloch (conv5-->relu-->conv6-->relu)
        x·=·self.conv7(x) #second convolution bloch (conv7-->relu) # relu is reoved as it is not required in last layer
        x = x.view(-1, 10) #reducing output to number of classes
        return F.log_softmax(x) #applying log softmax to get likelihood of each class
```

### Getting model summary

```
!pip install torchsummary #installing torchsummary module to get network summary like number of parameters, output shape a
from torchsummary import summary #importing torchsummary
use_cuda = torch.cuda.is_available() #check if GPU is available
device = torch.device("cuda" if use_cuda else "cpu") #make divice GPU if it is available
model = Net().to(device) #move model to GPU if it is available
summary(model, input_size=(1, 28, 28)) #print summary
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torchsummary in /usr/local/lib/python3.8/dist-packages (1.5.1)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             320
            Conv2d-2           [-1, 64, 28, 28]          18,496
         MaxPool2d-3           [-1, 64, 14, 14]               0
            Conv2d-4          [-1, 128, 14, 14]          73,856
            Conv2d-5          [-1, 256, 14, 14]         295,168
         MaxPool2d-6            [-1, 256, 7, 7]               0
            Conv2d-7            [-1, 512, 5, 5]       1,180,160
            Conv2d-8           [-1, 1024, 3, 3]       4,719,616
            Conv2d-9             [-1, 10, 1, 1]          92,170
================================================================
Total params: 6,379,786
Trainable params: 6,379,786
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.51
Params size (MB): 24.34
Estimated Total Size (MB): 25.85
----------------------------------------------------------------
<ipython-input-9-1799a9071da3>:20: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change
  return F.log_softmax(x) #applying log softmax to get likelihood of each class
```

```
torch.manual_seed(1) #fixing a seed so that we don't get different results everytime we run the code
batch_size = 128 #number of images in each batch

kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
```

```
                    transform=transforms.Compose([                  #transforming the train data using the mean and STD of
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ])),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))      #transforming the test data using the mean and STD
                    ])),
    batch_size=batch_size, shuffle=True, **kwargs)



from tqdm import tqdm #tqdm shows the progress in beautiful way
def train(model, device, train_loader, optimizer, epoch):
    model.train() #telling model that model is going to be trained
    pbar = tqdm(train_loader)
    for batch_idx, (data, target) in enumerate(pbar):
        data, target = data.to(device), target.to(device) #moving data to GPU
        optimizer.zero_grad() #making grads as zero
        output = model(data) #passing input to model and getting output
        loss = F.nll_loss(output, target) # calculating loss by comparing original and predicted output
        loss.backward() #performing backpropagation
        optimizer.step() #changing weights
        pbar.set_description(desc= f'loss={loss.item()} batch_id={batch_idx}') #show description


def test(model, device, test_loader):
    model.eval() #telling torch that model will be evaluated here.
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data) #passing test data to model and getting output
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item() #find number of correct outputs

    test_loss /= len(test_loader.dataset) #finding test loss

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))   #printing all the logs



model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

for epoch in range(1, 20):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

```
Test set: Average loss: 0.0339, Accuracy: 9899/10000 (99%)

loss=0.010990120470523834 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 26.06it/s]

Test set: Average loss: 0.0268, Accuracy: 9911/10000 (99%)

loss=0.0008172564557753503 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.25it/s]
```

```
Test set: Average loss: 0.0393, Accuracy: 9915/10000 (99%)

loss=0.0001445007510483265 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.12it/s]

Test set: Average loss: 0.0409, Accuracy: 9907/10000 (99%)

loss=0.00043609319254755974 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.05it/s]

Test set: Average loss: 0.0354, Accuracy: 9919/10000 (99%)

loss=0.006890381220728159 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.19it/s]

Test set: Average loss: 0.0351, Accuracy: 9919/10000 (99%)

loss=0.0006936820573173463 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.15it/s]

Test set: Average loss: 0.0347, Accuracy: 9927/10000 (99%)

loss=0.0008094091899693012 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 27.12it/s]

Test set: Average loss: 0.0340, Accuracy: 9923/10000 (99%)

loss=0.00027849155594594777 batch_id=468: 100%|██████████| 469/469 [00:18<00:00, 25.95it/s]

Test set: Average loss: 0.0333, Accuracy: 9927/10000 (99%)

loss=0.0003454394463915378 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 26.92it/s]

Test set: Average loss: 0.0390, Accuracy: 9914/10000 (99%)
```