```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms


dropout_value = 0.05
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.convblock1 = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=16, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Dropout(dropout_value))  # input:28, output:26, receptive field:3

        self.convblock2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=20, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(20),
            nn.ReLU(),
            nn.Dropout(dropout_value))  # input:26 , output:24, receptive field:5

        self.convblock3 = nn.Sequential(
            nn.Conv2d(in_channels=20, out_channels=24, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(24),
            nn.ReLU(),
            nn.Dropout(dropout_value))  # input:24 , output:22 receptive field:7

        self.pool1 = nn.MaxPool2d(2, 2)     # input:22 , output:11 receptive field:14



        self.convblock4 = nn.Sequential(
            nn.Conv2d(in_channels=24, out_channels=20, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(20),
            nn.ReLU(),
            nn.Dropout(dropout_value))   # input:11 , output:9, receptive field:16

        self.convblock5 = nn.Sequential(
            nn.Conv2d(in_channels=20, out_channels=16, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Dropout(dropout_value))  # input:9 , output:7 receptive field:18

        # OUTPUT BLOCK
        self.gap = nn.Sequential(
            nn.AvgPool2d(kernel_size=2)
        ) # input:7 , output:3 receptive field:18

        self.convblock6 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=10, kernel_size=(3, 3), padding=0, bias=False),
            nn.BatchNorm2d(10),
            nn.ReLU())  # input:3 , output:1 receptive field:20


    def forward(self, x):
        x = self.convblock1(x)
        x = self.convblock2(x)
        x = self.convblock3(x)
        x = self.pool1(x)
        x = self.convblock4(x)
        x = self.convblock5(x)
        x = self.gap(x)
        x = self.convblock6(x)
        x = x.view(-1, 10)
        return F.log_softmax(x)


!pip install torchsummary
from torchsummary import summary
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
model = Net().to(device)
summary(model, input_size=(1, 28, 28))
```

```
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, 16, 26, 26]             144
       BatchNorm2d-2           [-1, 16, 26, 26]              32
              ReLU-3           [-1, 16, 26, 26]               0
           Dropout-4           [-1, 16, 26, 26]               0
            Conv2d-5           [-1, 20, 24, 24]           2,880
       BatchNorm2d-6           [-1, 20, 24, 24]              40
              ReLU-7           [-1, 20, 24, 24]               0
           Dropout-8           [-1, 20, 24, 24]               0
            Conv2d-9           [-1, 24, 22, 22]           4,320
      BatchNorm2d-10           [-1, 24, 22, 22]              48
             ReLU-11           [-1, 24, 22, 22]               0
          Dropout-12           [-1, 24, 22, 22]               0
        MaxPool2d-13           [-1, 24, 11, 11]               0
           Conv2d-14             [-1, 20, 9, 9]           4,320
      BatchNorm2d-15             [-1, 20, 9, 9]              40
             ReLU-16             [-1, 20, 9, 9]               0
          Dropout-17             [-1, 20, 9, 9]               0
           Conv2d-18             [-1, 16, 7, 7]           2,880
      BatchNorm2d-19             [-1, 16, 7, 7]              32
             ReLU-20             [-1, 16, 7, 7]               0
          Dropout-21             [-1, 16, 7, 7]               0
        AvgPool2d-22             [-1, 16, 3, 3]               0
           Conv2d-23             [-1, 10, 1, 1]           1,440
      BatchNorm2d-24             [-1, 10, 1, 1]              20
             ReLU-25             [-1, 10, 1, 1]               0
================================================================
Total params: 16,196
Trainable params: 16,196
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.13
Params size (MB): 0.06
Estimated Total Size (MB): 1.20
----------------------------------------------------------------
<ipython-input-8-6d2d074856ac>:61: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change
  return F.log_softmax(x)
```

```
torch.manual_seed(1)
batch_size = 128

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([
                       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,))
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)
```

```
from tqdm import tqdm
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    pbar = tqdm(train_loader)
    for batch_idx, (data, target) in enumerate(pbar):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        pbar.set_description(desc= f'loss={loss.item()} batch_id={batch_idx}')


def test(model, device, test_loader):
    model.eval()
    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)
            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()

    test_loss /= len(test_loader.dataset)

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


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

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

```
Test set: Average loss: 0.0251, Accuracy: 9932/10000 (99.3200%)

EPOCH: 10
loss=0.012725134380161762 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.24it/s]

Test set: Average loss: 0.0242, Accuracy: 9933/10000 (99.3300%)

EPOCH: 11
loss=0.038086201995611119 batch_id=468: 100%|██████████| 469/469 [00:17<00:00, 26.74it/s]

Test set: Average loss: 0.0225, Accuracy: 9940/10000 (99.4000%)

EPOCH: 12
loss=0.038855623453855515 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.57it/s]

Test set: Average loss: 0.0212, Accuracy: 9943/10000 (99.4300%)

EPOCH: 13
loss=0.040390361100435260 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.62it/s]

Test set: Average loss: 0.0231, Accuracy: 9937/10000 (99.3700%)

EPOCH: 14
loss=0.022311633452773094 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.68it/s]

Test set: Average loss: 0.0196, Accuracy: 9948/10000 (99.4800%)

EPOCH: 15
loss=0.045131918042898180 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.53it/s]

Test set: Average loss: 0.0218, Accuracy: 9937/10000 (99.3700%)

EPOCH: 16
loss=0.016212640330195427 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.74it/s]

Test set: Average loss: 0.0209, Accuracy: 9942/10000 (99.4200%)

EPOCH: 17
loss=0.0048682489432394505 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.66it/s]

Test set: Average loss: 0.0218, Accuracy: 9946/10000 (99.4600%)

EPOCH: 18
loss=0.16152639687061310 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 30.58it/s]

Test set: Average loss: 0.0185, Accuracy: 9951/10000 (99.5100%)

EPOCH: 19
loss=0.027208864688873290 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 31.07it/s]

Test set: Average loss: 0.0200, Accuracy: 9936/10000 (99.3600%)

EPOCH: 20
loss=0.018307654187083244 batch_id=468: 100%|██████████| 469/469 [00:15<00:00, 31.09it/s]

Test set: Average loss: 0.0200, Accuracy: 9948/10000 (99.4800%)
```

✓ 5m 34s completed at 08:38