We train an simple neural network to do task of image recognition, the data we used is CIFAR10 data set. Following the code.

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [2]:
```
#read data
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./da
ta/cifar-10-python.tar.gz
100%|████████████████████████| 170498071/170498071 [00:06<00:00, 28396235.45
it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

In [3]:
```
import torch.nn as nn
import torch.nn.functional as F
```

In [4]:
```
#define the network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [5]:
```
net = Net()
```

In [6]:
```
#define loss function and optimizer
import torch.optim as optim
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [7]:

```
#train the network with 2 epoch
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.226
[1,  4000] loss: 1.916
[1,  6000] loss: 1.730
[1,  8000] loss: 1.619
[1, 10000] loss: 1.563
[1, 12000] loss: 1.478
[2,  2000] loss: 1.426
[2,  4000] loss: 1.371
[2,  6000] loss: 1.320
[2,  8000] loss: 1.316
[2, 10000] loss: 1.296
[2, 12000] loss: 1.266
Finished Training
```