In [20]:

```
import torch
import torchvision
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as dset
from torch.utils.data import DataLoader
from torch.autograd import Variable

'''
STEP 1: LOADING DATASET
'''

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (

trainset = torchvision.datasets.CIFAR10(root='./data', train = True, download = True, transfo

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download = True, transform

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

'''
STEP 2: MAKING DATASET ITERABLE
'''

batch_size = 100
n_iters = 3000
num_epochs = 20
#num_epochs = n_iters / (len(trainset) / batch_size)
#num_epochs = int(num_epochs)

print(len(trainset), num_epochs)

trainloader = torch.utils.data.DataLoader(trainset, batch_size = batch_size, shuffle = True, n

testloader = torch.utils.data.DataLoader(testset, batch_size = batch_size, shuffle = False, nu

'''
STEP 3: CREATE MODEL CLASS
'''
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()

        # Convolution 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, paddin
        self.relu1 = nn.ReLU()
        self.batchNorm1 = nn.BatchNorm2d(16)

        # Max pool 1
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        # Convolution 2
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, paddi
        self.relu2 = nn.ReLU()
        self.batchNorm2 = nn.BatchNorm2d(32)

        # Max pool 2
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

```

```
        # Convolution 3

        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, paddi
        self.relu3 = nn.ReLU()
        self.batchNorm3 = nn.BatchNorm2d(64)

        # Max pool 3
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)

        # Convolution 4
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padd
        self.relu4 = nn.ReLU()
        self.batchNorm4 = nn.BatchNorm2d(128)

        # Max pool 4
        self.maxpool4 = nn.MaxPool2d(kernel_size=2)

        # Fully connected 1 (readout)
        self.fc1 = nn.Linear(512, 512)

        self.layer = nn.Sequential(
            nn.Linear(512,256),
            nn.ReLU(),
            nn.Linear(256,128),
            nn.ReLU(),
            nn.Linear(128,10)
        )

    def forward(self, x):
        # Convolution 1
        out = self.conv1(x)
        out = self.relu1(out)
        out = self.batchNorm1(out)

        # Max pool 1
        out = self.maxpool1(out)

        # Convolution 2
        out = self.conv2(out)
        out = self.relu2(out)
        out = self.batchNorm2(out)

        # Max pool 2
        out = self.maxpool2(out)

        # Convolution 3
        out = self.conv3(out)
        out = self.relu3(out)
        out = self.batchNorm3(out)

        # Max pool 3
        out = self.maxpool3(out)


        # Convolution 4
        out = self.conv4(out)
        out = self.relu4(out)
        out = self.batchNorm4(out)

        # Max pool 4
        out = self.maxpool4(out)
```

```python

        # Resize
        # Original size: (100, 32, 7, 7)
        # out.size(0): 100
        # New out size: (100, 32*7*7)
        out = out.view(out.size(0), -1)

        # Linear function (readout)
        out = self.fc1(out)
        out = self.layer(out)

        return out

'''
STEP 4: INSTANTIATE MODEL CLASS
'''

model = CNNModel().cuda()

#######################
#  USE GPU FOR MODEL  #
#######################

# device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")



# model.to(device)

'''
STEP 5: INSTANTIATE LOSS CLASS
'''
criterion = nn.CrossEntropyLoss()


'''
STEP 6: INSTANTIATE OPTIMIZER CLASS
'''
learning_rate = 0.0015

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.002)


```

```
Files already downloaded and verified
Files already downloaded and verified
50000 20
```

In [21]:

```
'''
STEP 7: TRAIN THE MODEL
'''
iter = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(trainloader):

        images = Variable(images).cuda()
        labels = Variable(labels).cuda()

        # Clear gradients w.r.t. parameters
        optimizer.zero_grad()

        # Forward pass to get output/logits
        outputs = model(images)

        # Calculate Loss: softmax --> cross entropy loss
        loss = criterion(outputs, labels)

        # Getting gradients w.r.t. parameters
        loss.backward()

        # Updating parameters
        optimizer.step()

        iter += 1

        if iter % 200 == 0:
            # Calculate Accuracy
            correct = 0
            total = 0
            # Iterate through test dataset
            for images, labels in testloader:
                #######################
                #  USE GPU FOR MODEL  #
                #######################
                #images = Variable(images).to(device)
                #labels = Variable(labels).to(device)
                images = Variable(images).cuda()
                labels = Variable(labels).cuda()

                # Forward pass only to get logits/output
                outputs = model(images)

                # Get predictions from the maximum value
                _, predicted = torch.max(outputs.data, 1)

                # Total number of labels
                total += labels.size(0)

                #######################
                #  USE GPU FOR MODEL  #
                #######################
                # Total correct predictions
                if torch.cuda.is_available():
                    correct += (predicted.cpu() == labels.cpu()).sum()
                else:
                    correct += (predicted == labels).sum()

```

```
            accuracy = 100 * correct / total

            # Print Loss
            print('Iteration: {}. Loss: {}. Accuracy: {}'.format(iter, loss.item(), accuracy))
```

```
Iteration: 200. Loss: 1.0802453756332397. Accuracy: 54
Iteration: 400. Loss: 1.0591685771942139. Accuracy: 61
Iteration: 600. Loss: 1.2498900890350342. Accuracy: 63
Iteration: 800. Loss: 0.9499179124832153. Accuracy: 68
Iteration: 1000. Loss: 1.098762035369873. Accuracy: 70
Iteration: 1200. Loss: 0.7513238787651062. Accuracy: 69
Iteration: 1400. Loss: 0.84354567527771. Accuracy: 71
Iteration: 1600. Loss: 0.5185913443565369. Accuracy: 71
Iteration: 1800. Loss: 0.9087273478507996. Accuracy: 71
Iteration: 2000. Loss: 0.6466696262359619. Accuracy: 73
Iteration: 2200. Loss: 0.5459758639335632. Accuracy: 73
Iteration: 2400. Loss: 0.6114209890365601. Accuracy: 73
Iteration: 2600. Loss: 0.5416773557662964. Accuracy: 74
Iteration: 2800. Loss: 0.7845168113708496. Accuracy: 74
Iteration: 3000. Loss: 0.673625648021698. Accuracy: 75
Iteration: 3200. Loss: 0.48281776905059814. Accuracy: 74
Iteration: 3400. Loss: 0.6193227171897888. Accuracy: 74
Iteration: 3600. Loss: 0.5311362743377686. Accuracy: 75
Iteration: 3800. Loss: 0.7146061658859253. Accuracy: 73
Iteration: 4000. Loss: 0.6406211853027344. Accuracy: 76
Iteration: 4200. Loss: 0.4911739230155945. Accuracy: 76
Iteration: 4400. Loss: 0.6070870161056519. Accuracy: 76
Iteration: 4600. Loss: 0.5238867998123169. Accuracy: 76
Iteration: 4800. Loss: 0.3915778398513794. Accuracy: 76
Iteration: 5000. Loss: 0.5295023322105408. Accuracy: 77
Iteration: 5200. Loss: 0.43360623717308044. Accuracy: 76
Iteration: 5400. Loss: 0.40151968598365784. Accuracy: 77
Iteration: 5600. Loss: 0.3985865116119385. Accuracy: 76
Iteration: 5800. Loss: 0.6029594540596008. Accuracy: 76
Iteration: 6000. Loss: 0.42858999967575073. Accuracy: 77
Iteration: 6200. Loss: 0.32337379455566406. Accuracy: 78
Iteration: 6400. Loss: 0.3180921673774719. Accuracy: 78
Iteration: 6600. Loss: 0.32018253207206726. Accuracy: 78
Iteration: 6800. Loss: 0.5380628705024719. Accuracy: 78
Iteration: 7000. Loss: 0.5449371933937073. Accuracy: 78
Iteration: 7200. Loss: 0.4668041467666626. Accuracy: 78
Iteration: 7400. Loss: 0.3947407007217407. Accuracy: 79
Iteration: 7600. Loss: 0.33736830949783325. Accuracy: 78
Iteration: 7800. Loss: 0.3849654793739319. Accuracy: 78
Iteration: 8000. Loss: 0.47112786769866943. Accuracy: 78
Iteration: 8200. Loss: 0.38789236545562744. Accuracy: 79
Iteration: 8400. Loss: 0.4590197503566742. Accuracy: 79
Iteration: 8600. Loss: 0.2545468807220459. Accuracy: 79
Iteration: 8800. Loss: 0.40812888741493225. Accuracy: 79
Iteration: 9000. Loss: 0.35889342427253723. Accuracy: 80
Iteration: 9200. Loss: 0.362052321434021. Accuracy: 79
Iteration: 9400. Loss: 0.4707458019256592. Accuracy: 79
Iteration: 9600. Loss: 0.3114205300807953. Accuracy: 80
Iteration: 9800. Loss: 0.4438943564891815. Accuracy: 79
Iteration: 10000. Loss: 0.4702972173690796. Accuracy: 80
```

In [ ]:

```

```