In [1]:

```
import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim

train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')

def accuracy(model,testloader):
    correct = 0
    total = 0
    for data in testloader:
            images, labels = data
            if train_on_gpu:
                images, labels = images.cuda(), labels.cuda()
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print('acc = %.3f' %(correct/total))

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv_layer = nn.Sequential(
        nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
        nn.Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
        nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
        nn.Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
        nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
```

```
            nn.Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
```

```python
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
        nn.AvgPool2d(kernel_size=1, stride=1, padding=0),
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(512,10)
        )

    def forward(self, x):
        # conv layers
        x = self.conv_layer(x)

        # flatten
        x = x.view(x.size(0), -1)

        # fc layer
        x = self.fc_layer(x)

        return x


def main():
    # load and transform dataset
    transform = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        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=128,
                                              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=100,
                                             shuffle=False, num_workers=2)

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

    # TODO: Define your optimizer and criterion.
    criterion = nn.CrossEntropyLoss()
    model = CNN()
    if train_on_gpu:
        model.cuda()
```

```python
optimizer = optim.SGD(model.parameters(), lr=1e-1,
                      momentum=0.9, weight_decay=5e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)

num_epoch = 200
for epoch in range(num_epoch):  # 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
        if train_on_gpu:
            inputs, labels = inputs.cuda(), labels.cuda()

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(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: %.5f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
    print('Epoch = %d | loss = %.4f'%(epoch+1,running_loss/len(trainloader)))
    scheduler.step()
    accuracy(model,testloader)

print('Finished Training')

PATH = './model.pth'
torch.save(model.state_dict(), PATH)

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        if train_on_gpu:
            images, labels = images.cuda(), labels.cuda()
        outputs = model(images)
```

```
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 10000 test images: %d %%' % (
            100 * correct / total))


if __name__ == "__main__":
    main()
```

```
CUDA is available!  Training on GPU ...
Files already downloaded and verified
Files already downloaded and verified
Epoch = 1 | loss = 2.5475
acc = 0.113
Epoch = 2 | loss = 2.2827
acc = 0.115
Epoch = 3 | loss = 2.2759
acc = 0.130
Epoch = 4 | loss = 2.0480
acc = 0.254
Epoch = 5 | loss = 1.7250
acc = 0.376
Epoch = 6 | loss = 1.4172
acc = 0.547
Epoch = 7 | loss = 1.1246
acc = 0.654
Epoch = 8 | loss = 0.9400
acc = 0.693
Epoch = 9 | loss = 0.8232
acc = 0.721
Epoch = 10 | loss = 0.7495
acc = 0.745
Epoch = 11 | loss = 0.6979
acc = 0.772
Epoch = 12 | loss = 0.6562
acc = 0.770
Epoch = 13 | loss = 0.6165
acc = 0.794
Epoch = 14 | loss = 0.5913
acc = 0.780
Epoch = 15 | loss = 0.5747
acc = 0.800
Epoch = 16 | loss = 0.5609
acc = 0.806
Epoch = 17 | loss = 0.5490
acc = 0.798
Epoch = 18 | loss = 0.5303
acc = 0.816
Epoch = 19 | loss = 0.5139
acc = 0.792
Epoch = 20 | loss = 0.5089
acc = 0.812
Epoch = 21 | loss = 0.5027
acc = 0.831
Epoch = 22 | loss = 0.4858
```

```
acc = 0.818
Epoch = 23 | loss = 0.4814
acc = 0.819
Epoch = 24 | loss = 0.4790
acc = 0.824
Epoch = 25 | loss = 0.4702
acc = 0.818
Epoch = 26 | loss = 0.4634
acc = 0.822
Epoch = 27 | loss = 0.4557
acc = 0.827
Epoch = 28 | loss = 0.4589
acc = 0.834
Epoch = 29 | loss = 0.4507
acc = 0.834
Epoch = 30 | loss = 0.4454
acc = 0.828
Epoch = 31 | loss = 0.4357
acc = 0.818
Epoch = 32 | loss = 0.4363
acc = 0.827
Epoch = 33 | loss = 0.4263
acc = 0.834
Epoch = 34 | loss = 0.4264
acc = 0.843
Epoch = 35 | loss = 0.4204
acc = 0.830
Epoch = 36 | loss = 0.4202
acc = 0.843
Epoch = 37 | loss = 0.4149
acc = 0.836
Epoch = 38 | loss = 0.4106
acc = 0.836
Epoch = 39 | loss = 0.4197
acc = 0.843
Epoch = 40 | loss = 0.4067
acc = 0.849
Epoch = 41 | loss = 0.4081
acc = 0.845
Epoch = 42 | loss = 0.3951
acc = 0.847
Epoch = 43 | loss = 0.3911
acc = 0.838
Epoch = 44 | loss = 0.3902
acc = 0.843
Epoch = 45 | loss = 0.3936
```

```
acc = 0.839
Epoch = 46 | loss = 0.3917
acc = 0.842
Epoch = 47 | loss = 0.3889
acc = 0.852
Epoch = 48 | loss = 0.3836
acc = 0.855
Epoch = 49 | loss = 0.3826
acc = 0.853
Epoch = 50 | loss = 0.3808
acc = 0.844
Epoch = 51 | loss = 0.3758
acc = 0.859
Epoch = 52 | loss = 0.3722
acc = 0.844
Epoch = 53 | loss = 0.3694
acc = 0.850
Epoch = 54 | loss = 0.3666
acc = 0.855
Epoch = 55 | loss = 0.3707
acc = 0.852
Epoch = 56 | loss = 0.3625
acc = 0.851
Epoch = 57 | loss = 0.3704
acc = 0.843
Epoch = 58 | loss = 0.3645
acc = 0.856
Epoch = 59 | loss = 0.3566
acc = 0.860
Epoch = 60 | loss = 0.3469
acc = 0.854
Epoch = 61 | loss = 0.3486
acc = 0.864
Epoch = 62 | loss = 0.3479
acc = 0.861
Epoch = 63 | loss = 0.3429
acc = 0.851
Epoch = 64 | loss = 0.3445
acc = 0.848
Epoch = 65 | loss = 0.3490
acc = 0.850
Epoch = 66 | loss = 0.3428
acc = 0.847
Epoch = 67 | loss = 0.3290
acc = 0.864
Epoch = 68 | loss = 0.3363
```

```
acc = 0.863
Epoch = 69 | loss = 0.3330
acc = 0.850
Epoch = 70 | loss = 0.3328
acc = 0.859
Epoch = 71 | loss = 0.3190
acc = 0.860
Epoch = 72 | loss = 0.3197
acc = 0.860
Epoch = 73 | loss = 0.3193
acc = 0.859
Epoch = 74 | loss = 0.3230
acc = 0.855
Epoch = 75 | loss = 0.3179
acc = 0.859
Epoch = 76 | loss = 0.3104
acc = 0.864
Epoch = 77 | loss = 0.3071
acc = 0.856
Epoch = 78 | loss = 0.3103
acc = 0.853
Epoch = 79 | loss = 0.3007
acc = 0.867
Epoch = 80 | loss = 0.3028
acc = 0.859
Epoch = 81 | loss = 0.2977
acc = 0.863
Epoch = 82 | loss = 0.2925
acc = 0.856
Epoch = 83 | loss = 0.3020
acc = 0.868
Epoch = 84 | loss = 0.2918
acc = 0.876
Epoch = 85 | loss = 0.2850
acc = 0.866
Epoch = 86 | loss = 0.2881
acc = 0.868
Epoch = 87 | loss = 0.2829
acc = 0.869
Epoch = 88 | loss = 0.2837
acc = 0.865
Epoch = 89 | loss = 0.2760
acc = 0.865
Epoch = 90 | loss = 0.2781
acc = 0.880
Epoch = 91 | loss = 0.2717
```

```
acc = 0.876
Epoch = 92 | loss = 0.2670
acc = 0.880
Epoch = 93 | loss = 0.2654
acc = 0.861
Epoch = 94 | loss = 0.2618
acc = 0.877
Epoch = 95 | loss = 0.2548
acc = 0.867
Epoch = 96 | loss = 0.2549
acc = 0.871
Epoch = 97 | loss = 0.2544
acc = 0.879
Epoch = 98 | loss = 0.2475
acc = 0.875
Epoch = 99 | loss = 0.2507
acc = 0.872
Epoch = 100 | loss = 0.2434
acc = 0.881
Epoch = 101 | loss = 0.2422
acc = 0.873
Epoch = 102 | loss = 0.2380
acc = 0.875
Epoch = 103 | loss = 0.2343
acc = 0.879
Epoch = 104 | loss = 0.2323
acc = 0.884
Epoch = 105 | loss = 0.2265
acc = 0.878
Epoch = 106 | loss = 0.2237
acc = 0.878
Epoch = 107 | loss = 0.2272
acc = 0.871
Epoch = 108 | loss = 0.2201
acc = 0.885
Epoch = 109 | loss = 0.2145
acc = 0.881
Epoch = 110 | loss = 0.2146
acc = 0.875
Epoch = 111 | loss = 0.2064
acc = 0.887
Epoch = 112 | loss = 0.2020
acc = 0.887
Epoch = 113 | loss = 0.1998
acc = 0.884
Epoch = 114 | loss = 0.1979
```

```
acc = 0.883
Epoch = 115 | loss = 0.1952
acc = 0.885
Epoch = 116 | loss = 0.1894
acc = 0.881
Epoch = 117 | loss = 0.1911
acc = 0.888
Epoch = 118 | loss = 0.1811
acc = 0.881
Epoch = 119 | loss = 0.1856
acc = 0.886
Epoch = 120 | loss = 0.1800
acc = 0.892
Epoch = 121 | loss = 0.1738
acc = 0.889
Epoch = 122 | loss = 0.1717
acc = 0.887
Epoch = 123 | loss = 0.1693
acc = 0.885
Epoch = 124 | loss = 0.1644
acc = 0.896
Epoch = 125 | loss = 0.1599
acc = 0.891
Epoch = 126 | loss = 0.1586
acc = 0.896
Epoch = 127 | loss = 0.1545
acc = 0.887
Epoch = 128 | loss = 0.1543
acc = 0.892
Epoch = 129 | loss = 0.1483
acc = 0.893
Epoch = 130 | loss = 0.1420
acc = 0.892
Epoch = 131 | loss = 0.1385
acc = 0.892
Epoch = 132 | loss = 0.1320
acc = 0.895
Epoch = 133 | loss = 0.1348
acc = 0.898
Epoch = 134 | loss = 0.1336
acc = 0.899
Epoch = 135 | loss = 0.1286
acc = 0.897
Epoch = 136 | loss = 0.1215
acc = 0.897
Epoch = 137 | loss = 0.1122
```

```
acc = 0.898
Epoch = 138 | loss = 0.1141
acc = 0.899
Epoch = 139 | loss = 0.1112
acc = 0.896
Epoch = 140 | loss = 0.1102
acc = 0.896
Epoch = 141 | loss = 0.1068
acc = 0.901
Epoch = 142 | loss = 0.0979
acc = 0.902
Epoch = 143 | loss = 0.0960
acc = 0.902
Epoch = 144 | loss = 0.0942
acc = 0.902
Epoch = 145 | loss = 0.0907
acc = 0.904
Epoch = 146 | loss = 0.0887
acc = 0.905
Epoch = 147 | loss = 0.0874
acc = 0.897
Epoch = 148 | loss = 0.0820
acc = 0.904
Epoch = 149 | loss = 0.0764
acc = 0.902
Epoch = 150 | loss = 0.0738
acc = 0.902
Epoch = 151 | loss = 0.0672
acc = 0.900
Epoch = 152 | loss = 0.0684
acc = 0.907
Epoch = 153 | loss = 0.0617
acc = 0.905
Epoch = 154 | loss = 0.0652
acc = 0.910
Epoch = 155 | loss = 0.0606
acc = 0.908
Epoch = 156 | loss = 0.0537
acc = 0.905
Epoch = 157 | loss = 0.0507
acc = 0.908
Epoch = 158 | loss = 0.0482
acc = 0.915
Epoch = 159 | loss = 0.0468
acc = 0.907
Epoch = 160 | loss = 0.0416
```

```
acc = 0.915
Epoch = 161 | loss = 0.0383
acc = 0.914
Epoch = 162 | loss = 0.0383
acc = 0.913
Epoch = 163 | loss = 0.0308
acc = 0.913
Epoch = 164 | loss = 0.0320
acc = 0.914
Epoch = 165 | loss = 0.0264
acc = 0.913
Epoch = 166 | loss = 0.0245
acc = 0.916
Epoch = 167 | loss = 0.0238
acc = 0.917
Epoch = 168 | loss = 0.0223
acc = 0.923
Epoch = 169 | loss = 0.0184
acc = 0.920
Epoch = 170 | loss = 0.0158
acc = 0.919
Epoch = 171 | loss = 0.0152
acc = 0.925
Epoch = 172 | loss = 0.0146
acc = 0.925
Epoch = 173 | loss = 0.0134
acc = 0.923
Epoch = 174 | loss = 0.0098
acc = 0.925
Epoch = 175 | loss = 0.0091
acc = 0.923
Epoch = 176 | loss = 0.0060
acc = 0.926
Epoch = 177 | loss = 0.0070
acc = 0.928
Epoch = 178 | loss = 0.0051
acc = 0.926
Epoch = 179 | loss = 0.0055
acc = 0.927
Epoch = 180 | loss = 0.0044
acc = 0.929
Epoch = 181 | loss = 0.0036
acc = 0.929
Epoch = 182 | loss = 0.0034
acc = 0.932
Epoch = 183 | loss = 0.0030
```

```
acc = 0.928
Epoch = 184 | loss = 0.0023
acc = 0.930
Epoch = 185 | loss = 0.0022
acc = 0.928
Epoch = 186 | loss = 0.0021
acc = 0.933
Epoch = 187 | loss = 0.0020
acc = 0.930
Epoch = 188 | loss = 0.0018
acc = 0.932
Epoch = 189 | loss = 0.0015
acc = 0.934
Epoch = 190 | loss = 0.0022
acc = 0.930
Epoch = 191 | loss = 0.0015
acc = 0.932
Epoch = 192 | loss = 0.0019
acc = 0.932
Epoch = 193 | loss = 0.0016
acc = 0.934
Epoch = 194 | loss = 0.0015
acc = 0.933
Epoch = 195 | loss = 0.0016
acc = 0.932
Epoch = 196 | loss = 0.0015
acc = 0.935
Epoch = 197 | loss = 0.0017
acc = 0.934
Epoch = 198 | loss = 0.0015
acc = 0.934
Epoch = 199 | loss = 0.0014
acc = 0.934
Epoch = 200 | loss = 0.0015
acc = 0.934
Finished Training
Accuracy of the network on the 10000 test images: 93 %
```

After training the model I got the following results:
test accuracy: 93.1%

Hyperparameter tuning that I carried out to tune the model:
I Initially started with a model with blocks of convolution layers with 2 convolution layers per block. I tried tweaking the model parameters such as changing the channels of the convolution layers upto 512 channels but that did not result in good performance. I also tried testing out changing the optimizer such as using Adam optimizer and also tried multiple shedulers such as exponential LR and multi Step LR but that also did not work.
Then I switched to a resnet based approach with multiple optimizers and schedulers to test out my model but that was also stalling at around 85% accuracy or the model was getting too large to run. Hence finally I switched to a VGG based model with more convolution layers and more activation functions. I also switched to a cosine annealing LR as it was providing promising results. Finally I tweaked the number of epochs and batch sizes and ended up with accuracy over 90%