```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt


from torch.optim import lr_scheduler


batch_size=16
learning_rate=0.002
num_epoch=30


cifar_train=dset.CIFAR10("CIFAR10/", train=True, transform=transforms.Compose([transforms.ToTensor(),
cifar_test=dset.CIFAR10("CIFAR10/", train=False, transform=transforms.Compose([transforms.ToTensor(),
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
print ("cifar_train 길이:", len(cifar_train))
print ("cifar_test 길이:", len(cifar_test))

image, label = cifar_train.__getitem__(1)
print ("image data 형태:", image.size())
print ("label:", label)

img=image.numpy()
r,g,b = img[0,:,:], img[1,:,:], img[2,:,:]
img2=np.zeros((img.shape[1], img.shape[2], img.shape[0]))
img2[:,:,0], img2[:,:,1], img2[:,:,2]=r,g,b

plt.title("label: %d" %label)
plt.imshow(img2, interpolation='bicubic')
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
cifar_train 길이: 50000
cifar_test 길이: 10000
image data 형태: torch.Size([3, 32, 32])
label: 9
```


```
def ComputeAccr(dloader, imodel):
    correct=0
    total=0

    for j, [imgs, labels] in enumerate(dloader):
        img=Variable(imgs, volatile=True).cuda()
        label=Variable(labels).cuda()
        output=imodel.forward(img)
        _, output_index=torch.max(output, 1)

        total+=label.size(0)
        correct+=(output_index == label).sum().float()
    print ("Accuracy of Test Data: {}".format(100*correct/total))


train_loader=torch.utils.data.DataLoader(list(cifar_train)[:], batch_size=batch_size, shuffle=True, nur
test_loader=torch.utils.data.DataLoader(cifar_test, batch_size=batch_size, shuffle=False, num_workers=2

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer=nn.Sequential(
            nn.Conv2d(3,16,3,padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Conv2d(16,32,3,padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2,2),
            nn.Conv2d(32,64,3,padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2,2)
        )
        self.fc_layer=nn.Sequential(
            nn.Linear(64*8*8,100),
            nn.ReLU(),
            nn.Linear(100, 10)
        )
```

```
    def forward(self, x):
        out=self.layer(x)
        out=out.view(batch_size, -1)
        out=self.fc_layer(out)

        return out
model=CNN().cuda()


loss_func=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(), lr=learning_rate)
model.train()
for i in range(num_epoch):
    for j, [image, label] in enumerate(train_loader):
        x=Variable(image).cuda()
        y_=Variable(label).cuda()

        optimizer.zero_grad()
        output=model.forward(x)
        loss=loss_func(output, y_)
        loss.backward()
        optimizer.step()

        if j%1000==0:
            print(j, loss)
```

```
0 tensor(0.3786, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.2060, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0276, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0105, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0741, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0379, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0022, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.1837, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0099, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.2757, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.1074, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.1500, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.3528, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.3879, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0260, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0686, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0388, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0954, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0549, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0802, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0655, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0221, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0284, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0035, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.3655, device='cuda:0', grad_fn=<NllLossBackward>)

1000 tensor(0.1856, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0291, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.5429, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0363, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
0 tensor(0.0565, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.4476, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0065, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0046, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0397, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.1968, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.1185, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.1260, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.1441, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.1214, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0217, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.1196, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.4039, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0118, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0023, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.2491, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0056, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0548, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0146, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0067, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0265, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.4302, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.6744, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0332, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.1844, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.3340, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0122, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.6808, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0299, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.3765, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
model.eval()
ComputeAccr(test_loader, model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: volatile was remove

Accuracy of Test Data: 73.79000091552734
```

#pkl파일을 저장하기 위해서 추가한 코드입니다.

```
from google.colab import drive
drive.mount('/content/gdrive')

netname='/content/gdrive/My Drive/Colab Notebooks/final.pkl'
torch.save(model, netname, )
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/con
```

#pkl파일을 로드하여 제대로 저장됐는지 테스트 하기 위해서 추가한 코드입니다.

```
netname='/content/gdrive/My Drive/Colab Notebooks/final.pkl'
model=torch.load(netname)
ComputeAccr(test_loader, model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: volatile was remove

Accuracy of Test Data: 73.79000091552734
```

✓ 3초 오후 11:46에 완료됨