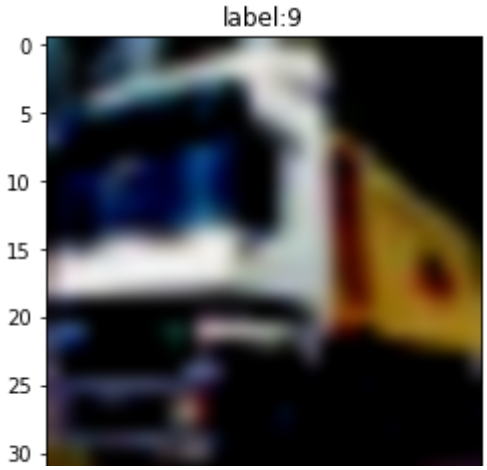
```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt

from torch.optim import lr_scheduler

batch_size=16
learning_rate=0.002
num_epoch=1


##cifar_train=dset.CIFAR10("CIFAR10/",train=True, transform=transforms.ToTensor(), target_transform
cifar_train=dset.CIFAR10("CIFAR10/",train=True, transform=transforms.Compose([transforms.ToTensor()

'''cifar_train=dset.CIFAR10("CIFAR10/",train=True, transform=transforms.Compose([transforms.Scale(3
                                                                                transforms.CenterCrop
                                                                                transforms.RandomHori
                                                                                transforms.Lambda(lam
                                                                                transforms.ToTensor()
                                                                                transforms.Normalize(
                                                                                ]))'''

##cifar_test=dset.CIFAR10("CIFAR10/",train=False, transform=transforms.ToTensor(), target_transform

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to CIFAR10/cifar-10-pyth
                                                                    170499072/? [00:14<00:00, 11510194.82it/s]

Extracting CIFAR10/cifar-10-python.tar.gz to CIFAR10/
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, :, :]
```

```
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 float
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,
test_loader=torch.utils.data.DataLoader(cifar_test, batch_size=batch_size, shuffle=False, num_worke

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.Dropout2d(0.2),
        nn.BatchNorm2d(16),
```

```
            nn.Conv2d(16,32,3,padding=1),
            nn.ReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2,2),
            nn.Conv2d(32,64,3,padding=1),
            nn.ReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2,2)
        )
        self.fc_layer=nn.Sequential(
            nn.Linear(64*8*8, 64*4*4),
            nn.PReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm1d(64*4*4),

            nn.Linear(64*4*4, 64*4),
            nn.PReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm1d(64*4),

            nn.Linear(64*4, 64*2),
            nn.PReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm1d(64*2),

            nn.Linear(64*2, 64),
            nn.PReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm1d(64),


            nn.Linear(64, 32),
            nn.PReLU(),
            ##nn.Dropout2d(0.2),
            nn.BatchNorm1d(32),



            nn.Linear(32,10)
        )

        for m in self.modules():
          if isinstance(m,nn.Conv2d):
            init.kaiming_normal(m.weight.data)
            m.bias.data.fill_(0)
            if isinstance(m,nn.Linear):
              init.kalming_normal(m.weight.data)
              m.blas.data.fill_(0)
    def forward(self,x):
        out=self.layer(x)
        out=out.view(batch_size,-1)
        out=self.fc_layer(out)
        return out
```

```
model=CNN().cuda()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:57: UserWarning: nn.init.kaiming
```

```
num_epoch = 100
loss_func=nn.CrossEntropyLoss()
##optimizer=torch.optim.SGD(model.parameters(), lr=learning_rate)
optimizer=torch.optim.Adam(model.parameters(), lr=learning_rate)

scheduler = lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.2)



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(0.0005, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0017, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(8.3147e-06, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(3.6063e-05, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.4453e-05, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(5.1557e-06, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0001, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(9.7824e-06, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(1.9220e-05, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.3865e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0355, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0587, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(5.5709e-05, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(4.9485e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(7.6741e-07, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(3.0355e-05, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(3.0480e-05, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0018, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0002, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(9.5440e-06, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(6.9290e-07, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(3.3963e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0027, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(5.7815e-06, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(9.1788e-06, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
1000 tensor(4.3958e-06, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(7.6491e-05, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(3.2037e-07, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0006, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.0594e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0002, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0004, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0005, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.0766e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(7.2939e-06, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.2410, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(2.3026e-05, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(4.7637e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(1.0207e-06, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0122, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0023, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.8924e-06, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(3.4273e-07, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(2.0121e-05, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(2.4140e-06, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.1180, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(2.3079e-05, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.0002, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.0001, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0147, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.2050, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(3.0919e-06, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(5.8116e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(6.1102e-05, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(3.7998e-07, device='cuda:0', grad_fn=<NllLossBackward>)
0 tensor(6.6977e-06, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(1.0110e-05, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.0002, 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 re

Accuracy of Test Data: 78.30999755859375
```

더블클릭 또는 Enter 키를 눌러 수정

```
netname = "./cifar10"
torch.save(model, netname,)
```