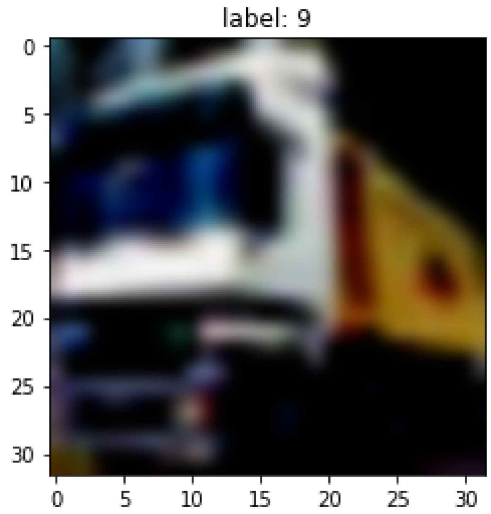
```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt

from torch.optim import lr_scheduler

batch_size=16
learning_rate=0.002
num_epoch=1
```

```
#cifar_train=dset.CIFAR10("CIFARD10/",train=True, transform=transforms.ToTensor(), target_transform=None, downlo

# (2) Data augmentation
'''cifar_train=dset.CIFAR10("CIFARD10/", train=True,
                        transform=transforms.Compose([
                        transforms.Scale(36),
                        transforms.CenterCrop(32),
                        transforms.RandomHorizontalFlip(),
                        transforms.Lambda(lambda x: x.rotate(90)),
                        transforms.ToTensor()
                        ]))'''

#cifar_test=dset.CIFAR10("CIFARD10/",train=False, transform=transforms.ToTensor(), target_transform=None, downlo

# (4) Data Normalization
cifar_train=dset.CIFAR10("CIFARD10/", train=True,
                        transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean=(0.5,0.5,0.5), std=(0.5,0.5,0.5)),
                        ])
                        , target_transform=None, download=False)

cifar_test=dset.CIFAR10("CIFARD10/", train=False,
                        transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean=(0.5,0.5,0.5), std=(0.5,0.5,0.5)),
                        ])
                        , target_transform=None, download=False)
```

```
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..255
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, num_workers=
test_loader=torch.utils.data.DataLoader(cifar_test, batch_size=batch_size, shuffle= False, num_workers=2, drop_l

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), # (1) drop out
        nn.BatchNorm2d(16), # (5) batch normalization
        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*8,100),
        nn.ReLU(),
       #nn.Dropout2d(0.2),
       nn.BatchNorm1d(100),
        nn.Linear(100,10)
    )

  # (3) weight initialization
    '''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.kaiming_normal(m.weight.data)
        m.bias.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()
```

```
loss_func=nn.CrossEntropyLoss()
optimizer=torch.optim.SGD(model.parameters(), lr=learning_rate)
#optimizer=torch.optim.Adam(model.parameters(), lr=learning_rate) # (6) Adam optimizer
#scheduler = lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.2) # (7) learing rate decay

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(0.4769, device='cuda:0', grad_fn=<NllLossBackward>)
1000 tensor(0.3109, device='cuda:0', grad_fn=<NllLossBackward>)
2000 tensor(0.2336, device='cuda:0', grad_fn=<NllLossBackward>)
3000 tensor(0.3342, 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 rem

Accuracy of Test Data: 76.88999938964844
```

```
1 netname='./nets/my_net_final.pkl'
2 torch.save(model, netname, )
```

```
1 netname = './nets/my_net_final.pkl'
2 model = torch.load(netname)
3
4 ComputeAccr(test_loader, model)
```

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

Accuracy of Test Data: 77.27999877929688
```

```
1
```