AlexNet

```
import os
os.getcwd()
```

```
'/content'
```

```
import torch as T
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

import torchvision
import torchvision.transforms as transforms

device = 'cuda' if T.cuda.is_available() else 'cpu'
device
```

```
'cuda'
```

```
transform = transforms.Compose(
    [transforms.Resize((224, 224)),
     transforms.ToTensor(),
     transforms.Normalize(mean = (0.5, 0.5, 0.5), std = (0.5, 0.5, 0.5))])

batch_size = 32
image_size = (32, 32, 3)

train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=t
train_loader = T.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True, num_wor

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=t
test_loader = T.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_work
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifa
                                                    170499072/? [00:11<00:00, 17129467.27it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
```

```
                        nn.ReLU(inplace=True),
                        nn.Conv2d(384,  256,  kernel_size=3,  padding=1),
                        nn.ReLU(inplace=True),
                        nn.Conv2d(256,  256,  kernel_size=3,  padding=1),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(kernel_size=3,  stride=2),
                )
                self.classifier  =  nn.Sequential(
                        nn.Dropout(),
                        nn.Linear(256  *  6  *  6,  4096),
                        nn.ReLU(inplace=True),
                        nn.Dropout(),
                        nn.Linear(4096,  4096),
                        nn.ReLU(inplace=True),
                        nn.Linear(4096,  num_classes),
                )

        def  forward(self,  x):
                x  =  self.features(x)
                x  =  x.view(x.size(0),  256  *  6  *  6)
                x  =  self.classifier(x)
                return  x


net  =  AlexNet()
net.to(device)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
criterion  =  nn.CrossEntropyLoss()

#  also  the  optimizer
```

```
# also the optimizer
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)


train_loss = []
test_loss = []
train_acc = []
test_acc = []

for epoch in range(1, 33):    #32 epoch

        running_loss = .0
        correct = 0
        total = 0
        for i, data in enumerate(train_loader):
                # get the inputs
                inputs, labels = data
                if device == 'cuda':
                        inputs, labels = inputs.to(device), labels.to(device)

                # reset the parameter gradients
                optimizer.zero_grad()

                # forward
                outputs = net(inputs)
                loss = criterion(outputs, labels)
                # backward
                loss.backward()
                # optimize
                optimizer.step()

                running_loss += loss.item()
                _, predicted = T.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        running_loss /= len(train_loader)
        train_loss.append(running_loss)
        running_acc = correct / total
        train_acc.append(running_acc)

        if epoch % 4 == 0:
                print('\nEpoch: {}'.format(epoch))
                print('Train Acc. => {:.3f}%'.format(100 * running_acc), end=' | ')
                print('Train Loss => {:.5f}'.format(running_loss))

        with T.no_grad():
                correct = 0
                total = 0
                test_running_loss = .0
                for data in test_loader:
                        inputs, labels = data
                        if device == 'cuda':
                                inputs, labels = inputs.to(device), labels.to(device)
                        outputs = net(inputs)
                        loss = criterion(outputs, labels)
```

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

            test_running_loss  /=  len(test_loader)
            test_loss.append(test_running_loss)
            test_running_acc  =  correct  /  total
            test_acc.append(test_running_acc)

            if  epoch  %  4  ==  0:
                print('Test  Acc.    =>  {:.3f}%'.format(100  *  test_running_acc),  end='
                print('Test  Loss    =>  {:.5f}'.format(test_running_loss))

print('Finished  Training')
```

```
Epoch: 4
Train Acc. => 55.230% | Train Loss => 1.23943
Test Acc.  => 57.490% | Test Loss  => 1.16939

Epoch: 8
Train Acc. => 73.354% | Train Loss => 0.76147
Test Acc.  => 72.400% | Test Loss  => 0.79087

Epoch: 12
Train Acc. => 81.438% | Train Loss => 0.53863
Test Acc.  => 76.890% | Test Loss  => 0.67115

Epoch: 16
Train Acc. => 86.848% | Train Loss => 0.37812
Test Acc.  => 80.370% | Test Loss  => 0.60078

Epoch: 20
Train Acc. => 90.918% | Train Loss => 0.25543
Test Acc.  => 81.050% | Test Loss  => 0.60969

Epoch: 24
Train Acc. => 93.958% | Train Loss => 0.16896
Test Acc.  => 81.040% | Test Loss  => 0.69141

Epoch: 28
Train Acc. => 95.906% | Train Loss => 0.11479
Test Acc.  => 82.110% | Test Loss  => 0.71463

Epoch: 32
Train Acc. => 97.228% | Train Loss => 0.07889
Test Acc.  => 82.770% | Test Loss  => 0.75348
Finished Training
```

```
T.save(AlexNet()  ,  './trained_model')
```

✓ 1 秒 完成時間：下午7:26