```
import torch.nn as nn
import torch.utils.model_zoo as model_zoo
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.optim as optim

def conv3x3(in_planes, out_planes, stride=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride, padding=1, bias=False)


def conv1x1(in_planes, out_planes, stride=1):
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False)


class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out


class CifarResNet(nn.Module):

    def __init__(self, block, layers, num_classes=100):
        super(CifarResNet, self).__init__()
        self.inplanes = 16
        self.conv1 = conv3x3(3, 16)
        self.bn1 = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)

        self.layer1 = self._make_layer(block, 16, layers[0])
        self.layer2 = self._make_layer(block, 32, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 64, layers[2], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(64 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride),
                nn.BatchNorm2d(planes * block.expansion),
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return

# Code added : To Tune hyper-parameters
_num_epoch = 42
_lr = 0.001
_momentum = 0.9
_weight_decay = 1e-4

# Code added to update learing rate to decay lr after each epoch
def update_lr(optimizer, lr):
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
```

```
class cifar_resnet20(nn.Module):
    def __init__(self):
        super(cifar_resnet20, self).__init__()
        ResNet20 = CifarResNet(BasicBlock, [3, 3, 3])
        url ='https://github.com/chenyaofo/pytorch-cifar-models/releases/download/resnet/cifar100_resnet20-23dac2f

        ResNet20.load_state_dict(model_zoo.load_url(url))
        modules = list(ResNet20.children())[:-1]
        backbone = nn.Sequential(*modules)
        self.backbone = nn.Sequential(*modules)
        self.fc = nn.Linear(64, 10)

    def forward(self, x):
        out = self.backbone(x)
        out = out.view(out.shape[0], -1)
        out = self.fc(out)
        return out


if __name__ == '__main__':
    model = cifar_resnet20().cuda()

    transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize(mean=(0.499, 0.499, 0.499),std=(0.1
    trainset = datasets.CIFAR10('./data', download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True, num_workers=2)

    # Code added : To create testset and testloader
    testset = datasets.CIFAR10(root='./data', train=False,download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=4,shuffle=False, num_workers=2)


    criterion = nn.CrossEntropyLoss()

    # Code added : To add L2 regularizer by setting weight_decay
    optimizer = optim.SGD(list(model.fc.parameters()), lr = _lr, momentum = _momentum,weight_decay = _weight_decay

    ## Do the training
    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
            inputs, labels = data
            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs.cuda())
            loss = criterion(outputs, labels.cuda())
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % 1000 == 999:    # print every 1000 mini-batches
                print('[%d, %5d] loss: %.3f' %
                    (epoch + 1, i + 1, running_loss / 1000))
                running_loss = 0.0

        # Decay learning rate
        if (epoch+1) % 20 == 0:
          _lr /= 3
          update_lr(optimizer, _lr)



    print('====== Finished Training ======')

    # Code added : To calculate the accuracy rate on test error and to save best model
    print('Testing 10000 images for : '+ str(epoch + 1) +" epoch(s)")
    best_acc = 0
    correct = 0
    total = 0
    with torch.no_grad():
      for i,data in enumerate(testloader, 0):
        images, labels = data
        outputs = model(images.cuda())
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted.cuda() == labels.cuda()).sum().item()
    print('Accuracy :'+ str((100 * correct / total)))
```

```
    if(best_acc < (100 * correct / total)):
      best_acc = (100 * correct / total)
      torch.save(model,'model_best.pth')
      print("New Best Model Saved")
```

```
Downloading: "https://github.com/chenyaofo/pytorch-cifar-models/releases/download/resnet/cifar100_resnet20-23
100%                                    1.11M/1.11M [00:00<00:00, 3.51MB/s]
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
                                   170499072/? [00:04<00:00, 57689996.77it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
[1,  1000] loss: 1.313
[2,  1000] loss: 1.013
[3,  1000] loss: 0.976
[4,  1000] loss: 0.974
[5,  1000] loss: 0.967
[6,  1000] loss: 0.966
[7,  1000] loss: 0.956
[8,  1000] loss: 0.955
[9,  1000] loss: 0.961
[10,  1000] loss: 0.955
[11,  1000] loss: 0.955
[12,  1000] loss: 0.959
[13,  1000] loss: 0.952
[14,  1000] loss: 0.960
[15,  1000] loss: 0.950
[16,  1000] loss: 0.947
[17,  1000] loss: 0.947
[18,  1000] loss: 0.947
[19,  1000] loss: 0.958
[20,  1000] loss: 0.955
[21,  1000] loss: 0.944
[22,  1000] loss: 0.941
[23,  1000] loss: 0.942
[24,  1000] loss: 0.942
[25,  1000] loss: 0.952
[26,  1000] loss: 0.943
[27,  1000] loss: 0.945
[28,  1000] loss: 0.943
[29,  1000] loss: 0.944
[30,  1000] loss: 0.942
[31,  1000] loss: 0.948
[32,  1000] loss: 0.946
[33,  1000] loss: 0.945
[34,  1000] loss: 0.950
[35,  1000] loss: 0.948
```

✓ 42m 13s completed at 22:43

✓ 42m 13s completed at 22:48