In [ ]:

```
!pip install d2l==v1.0.0-alpha1.post0
!pip install pytorch-ignite
!pip install torchviz
!pip install Graphviz
!pip install colormap
```

The following additional libraries are needed to run this notebook. Note that running on Colab is experimental, please report a Github issue if you have any problem.

In [ ]:

```
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l
from torchvision import transforms
import torchvision
import time
import numpy as np
import matplotlib.pyplot as plt
import graphviz

from functools import partial
```

In [ ]:

```
(torch.__version__)
```

Out[ ]:

```
'2.0.1+cu118'
```

In [ ]:

```
#
device= torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [ ]:

```
transform1 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),transforms.Pad(4),transforms.RandomErasing(),transforms.RandomHoriz


transform2 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 128

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform2)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:13<00:00, 13086138.35it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

In [ ]:

```
testloader
```

Out[ ]:

```
<torch.utils.data.dataloader.DataLoader at 0x7fa1e956aec0>
```

Using a different trainer from d2l

Trainer adjusted

In [ ]:

```
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize):

    net = net.to(device)
    loss=nn.CrossEntropyLoss()
    trainer=torch.optim.SGD(net.parameters(), 0.01)


    timer, num_batches = d2l.Timer(), len(trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train loss', 'train acc', 'test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i
        metric = d2l.Accumulator(4)
        for i, (features, labels) in enumerate(trainloader):
            features=features.to(device)
            labels=labels.to(device)
            timer.start()
            l, acc = train_batch(
                net, features, labels, loss, trainer, device)
            metric.add(l, acc, labels.shape[0], labels.numel())
            timer.stop()
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (metric[0] / metric[2], metric[1] / metric[3],
                              None))
        test_acc = d2l.evaluate_accuracy_gpu(net, testloader)
        animator.add(epoch + 1, (None, None, test_acc))



    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')

def train_batch(net,X, y, loss, trainer, device):

      net.train()
      trainer.zero_grad()
      pred = net(X)
      l = loss(pred, y)
      l.sum().backward()
      trainer.step()
      train_loss_sum = l.sum()
      train_acc_sum = d2l.accuracy(pred, y)
      return train_loss_sum, train_acc_sum
```

In [ ]:

```
class Residual(nn.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)

        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()
        self.bn3 = nn.LazyBatchNorm2d()
        self.maxpool = nn.MaxPool2d(kernel_size=2)

        self.ident1 =nn.Identity()
        self.ident2 =nn.Identity()

    def forward(self, X):
        Y = F.celu(self.bn1(self.maxpool(self.conv1(X))),0.075)
        X=  self.ident1(Y)
        Y = F.celu(self.bn2(self.conv2(X)),0.075)
        Y = F.celu(self.bn3(self.conv3(Y)),0.075)
        Y= self.ident2(Y)
        Y  = X+Y
        return F.celu(Y,0.075)
```

In [ ]:

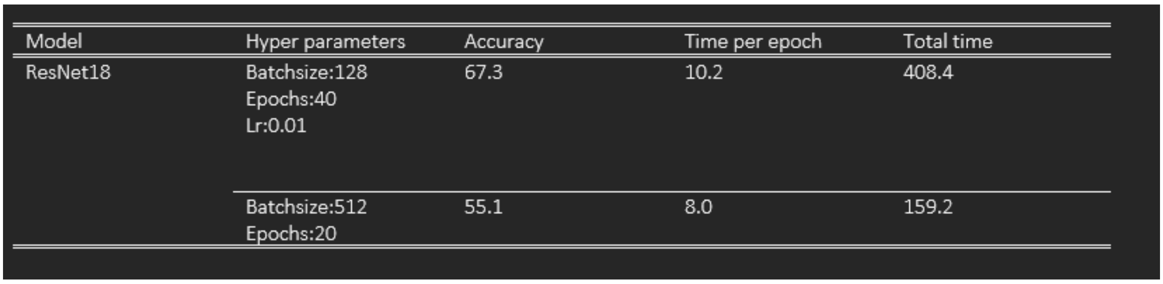
```
class ResNet(d2l.Classifier):

    def __init__(self, arch, lr=0.1, num_classes=10):
      super().__init__()
      self.save_hyperparameters()
      self.net = nn.Sequential(self.b1())



#Adding blocks
      for i, b in enumerate(arch):
          if i == 1:
            self.net.add_module(f'b{i+2}',nn.Sequential(
                nn.LazyConv2d(256, kernel_size=3, padding=1,stride=1),
                nn.MaxPool2d(kernel_size=2),
                nn.BatchNorm2d(256),
                nn.CELU(alpha=0.075)
                ))
          else:
            self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))


##Final block
      self.net.add_module('last', nn.Sequential(
          nn.MaxPool2d(4), nn.Flatten(),nn.LazyLinear(num_classes),nn.Identity()))
      self.net.apply(d2l.init_cnn)


    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64), nn.CELU(alpha=0.075) )


    def block(self, num_residuals, num_channels, first_block=False):
      blk = []
      for i in range(num_residuals):
              blk.append(Residual( num_channels))
      return nn.Sequential(*blk)




    def layer_summary(self, X_shape):
        """Defined in :numref:`sec_lenet`"""
        X = d2l.randn(*X_shape)
        for layer in self.net:
            X = layer(X)
            print(layer.__class__.__name__, 'output shape:\t', X.shape)
```

BaseLine restnet 18

In [ ]:

```
transform1 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),transforms.Pad(4)])


transform2 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 128

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform2)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)
```

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

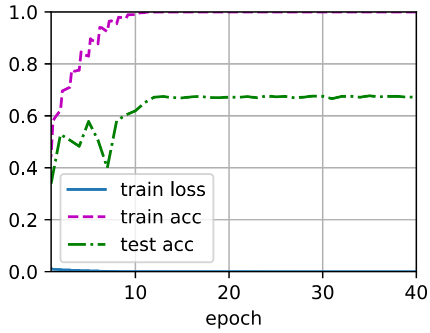


Model	Hyper parameters	Accuracy	Time per epoch	Total time
ResNet18	Batchsize:128 Epochs:40 Lr:0.01	67.3	10.2	408.4
	Batchsize:512 Epochs:20	55.1	8.0	159.2

In [ ]:

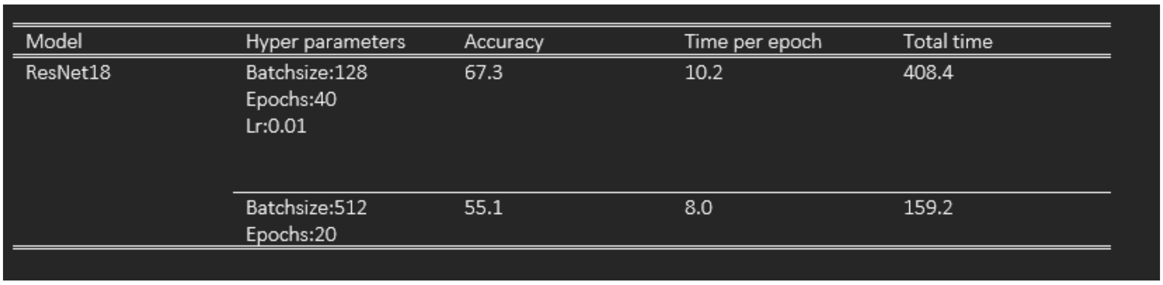
```
net=d2l.resnet18(10,3)
net(next(iter(trainloader))[0])
net=net.to(device)
train_ch13(net, trainloader,testloader, 40,
               device,128)
```

```
total test accuracy 0.673
Total training time per epoch 10.2
408.4 total training time
cuda:0
```



Clearly overfitting

2ND resnet18

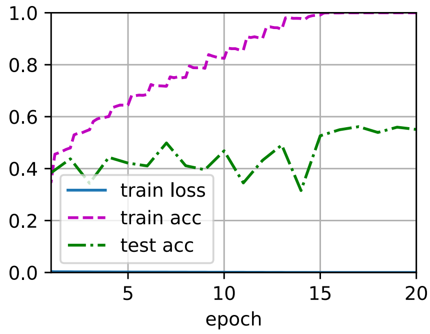


Model	Hyper parameters	Accuracy	Time per epoch	Total time
ResNet18	Batchsize:128 Epochs:40 Lr:0.01	67.3	10.2	408.4
	Batchsize:512 Epochs:20	55.1	8.0	159.2

In [ ]:

```
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform2)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)
net=d2l.resnet18(10,3)
net(next(iter(trainloader))[0])
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512)
```

```
total test accuracy 0.551
Total training time per epoch 8.0
159.2 total training time
cuda:0
```



Still Clearly overfitting

In [ ]:

```
class ResNet9(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((1, 128), (1, 256), (1, 512)),
                       lr, num_classes)
ResNet18().layer_summary((1,3,32,32))
```

```
Sequential output shape:         torch.Size([1, 64, 32, 32])
Sequential output shape:         torch.Size([1, 128, 16, 16])
Sequential output shape:         torch.Size([1, 256, 8, 8])
Sequential output shape:         torch.Size([1, 512, 4, 4])
Sequential output shape:         torch.Size([1, 10])
```

In [ ]:

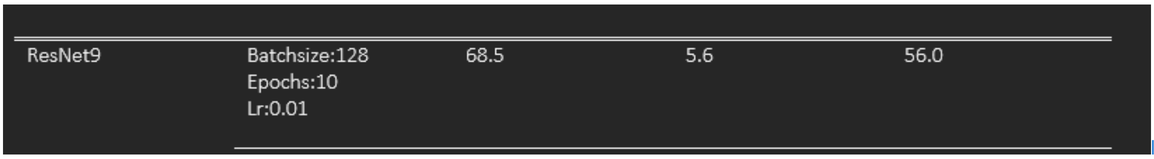
```
def train_batch(net,X, y, loss, trainer, device):
    net.train()
    trainer.zero_grad()
    pred = net(X)
    l = loss(pred, y)
    l.sum().backward()
    trainer.step()
    train_loss_sum = l.sum()
    train_acc_sum = d2l.accuracy(pred, y)
    return train_loss_sum, train_acc_sum
```

In [ ]:

```
device= torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [ ]:

```
batch_size = 128


trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0,pin_memory=True)

testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0,pin_memory=True)
```



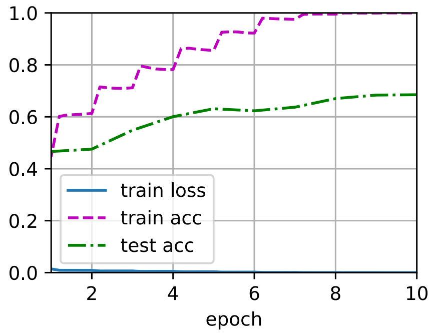
ResNet9	Batchsize:128 Epochs:10 Lr:0.01	68.5	5.6	56.0

link text

In [ ]:

```
net=ResNet18()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)

train_ch13(net, trainloader,testloader, 10,device,128)
```

```
total test accuracy 0.685
Total training time per epoch 5.6
56.0 total training time
cuda:0
```



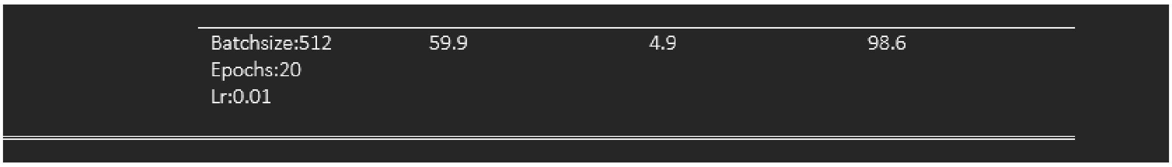
In [ ]:

```
len(trainloader)
```

Out[ ]:

```
391
```

more epoch, and larger batch size.

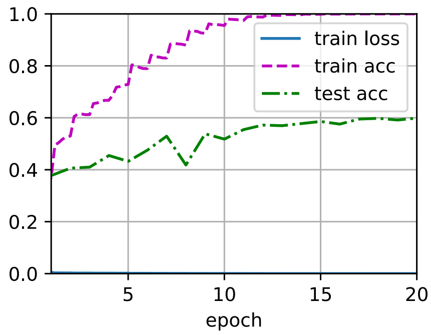


Batchsize:512 Epochs:20 Lr:0.01	59.9	4.9	98.6

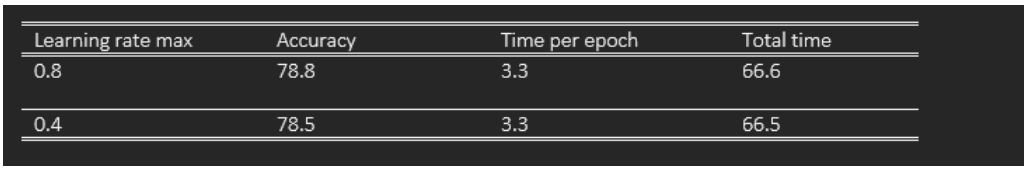
In [ ]:

```
batch_size = 512


trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)




net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512)
```

```
total test accuracy 0.599
Total training time per epoch 4.9
98.6 total training time
cuda:0
```

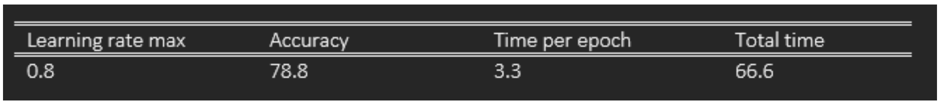


Just increasing batch size results in large accuracy losses but substantial training time reductions.



Learning rate max	Accuracy	Time per epoch	Total time
0.8	78.8	3.3	66.6
0.4	78.5	3.3	66.5

```
In [ ]:
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize,lr):



    trainer=torch.optim.SGD(net.parameters(), lr=lr)#, weight_decay = 0.000001momentum= 0.9)
    loss=nn.CrossEntropyLoss()

    LRsched = torch.optim.lr_scheduler.OneCycleLR(trainer ,
                                                         max_lr=0.8,
                                                         epochs=num_epochs,
                                                         steps_per_epoch=len(trainloader),
                                                         anneal_strategy = "linear")


    timer, num_batches = d2l.Timer(), len( trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train acc','test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i

        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate( trainloader):
            timer.start()
            trainer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            with torch.no_grad():
              l.backward()
              metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])


            trainer.step()
            LRsched.step()
            timer.stop()

            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (metric[1] / metric[2],
                              None))
        test_acc = d2l.evaluate_accuracy_gpu(net, testloader)
        animator.add(epoch + 1, ( None, test_acc))



    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')
```

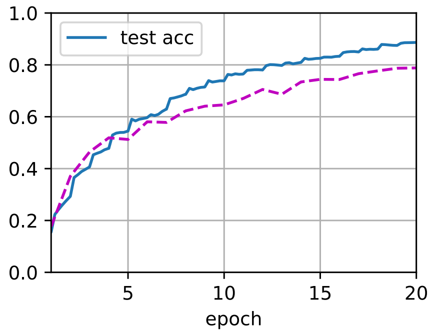


Learning rate max	Accuracy	Time per epoch	Total time
0.8	78.8	3.3	66.6

In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                               ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])
     ])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])

batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

```
total test accuracy 0.788
Total training time per epoch 3.3
66.6 total training time
cuda:0
```



In [ ]:

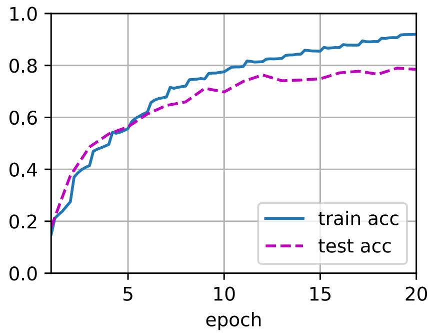
```
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize,lr):



    trainer=torch.optim.SGD(net.parameters(), lr=lr)#, weight_decay = 0.000001momentum= 0.9)
    loss=nn.CrossEntropyLoss()

    LRsched = torch.optim.lr_scheduler.OneCycleLR(trainer ,
                                                     max_lr=0.4,
                                                     epochs=num_epochs,
                                                     steps_per_epoch=len(trainloader),
                                                     anneal_strategy = "linear")


    timer, num_batches = d2l.Timer(), len( trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train acc','test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i

        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate( trainloader):
            timer.start()
            trainer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            with torch.no_grad():
              l.backward()
              metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])


            trainer.step()
            LRsched.step()
            timer.stop()

            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (metric[1] / metric[2],
                              None))
        test_acc = d2l.evaluate_accuracy_gpu(net, testloader)
        animator.add(epoch + 1, ( None, test_acc))



    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')
```

```
0.4                    78.5                    3.3                    66.5
```

In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                                   ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])
     ])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])

batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

```
total test accuracy 0.785
Total training time per epoch 3.3
66.5 total training time
cuda:0
```



Decay and momentum

Add weight decay and momentum to SGD optimiser. Generalise better and smoothes the SGD line making optiimiser more effecient (quicker to convergence).Input linear scheduling. Also removed the plotting of training accuracy and training loss.

In [ ]:

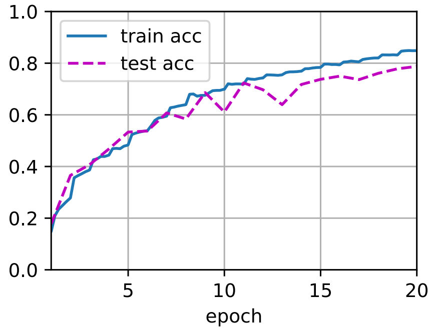
```
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize,lr):



    trainer=torch.optim.SGD(net.parameters(), lr=lr, momentum= 0.9, weight_decay = 0.00001)#
    loss=nn.CrossEntropyLoss()

    LRsched = torch.optim.lr_scheduler.OneCycleLR(trainer ,
                                                         max_lr=0.8,
                                                         epochs=num_epochs,
                                                         steps_per_epoch=len(trainloader),
                                                         anneal_strategy = "linear")


    timer, num_batches = d2l.Timer(), len( trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train acc','test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i

        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate( trainloader):
            timer.start()
            trainer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            with torch.no_grad():
              l.backward()
              metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])


            trainer.step()
            LRsched.step()
            timer.stop()

            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (metric[1] / metric[2],
                              None))

        test_acc = d2l.evaluate_accuracy_gpu(net,  testloader)
        animator.add(epoch + 1, (None, test_acc))



    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')
```

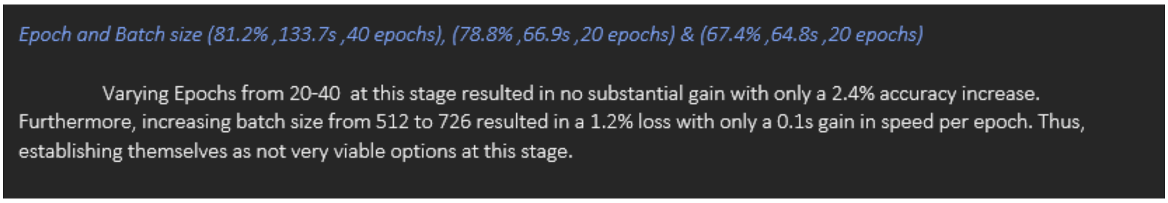
In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                                   ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010]),
     transforms.RandomErasing()])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
```

```
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

```
total test accuracy 0.787
Total training time per epoch 3.3
66.9 total training time
cuda:0
```





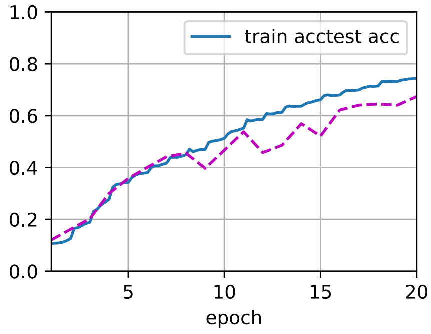
Epoch and Batch size (81.2% ,133.7s ,40 epochs), (78.8% ,66.9s ,20 epochs) & (67.4% ,64.8s ,20 epochs)

Varying Epochs from 20-40 at this stage resulted in no substantial gain with only a 2.4% accuracy increase. Furthermore, increasing batch size from 512 to 726 resulted in a 1.2% loss with only a 0.1s gain in speed per epoch. Thus, establishing themselves as not very viable options at this stage.

In [ ]:

```
batch_size = 726

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,726,0.01)
```

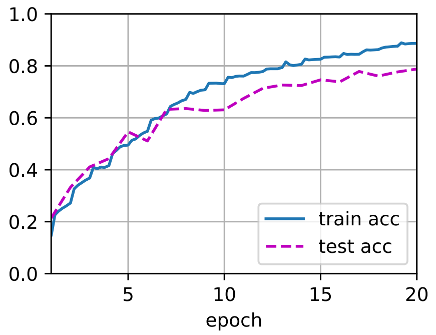
```
total test accuracy 0.674
Total training time per epoch 3.2
64.8 total training time
cuda:0
```



In [ ]:

```
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

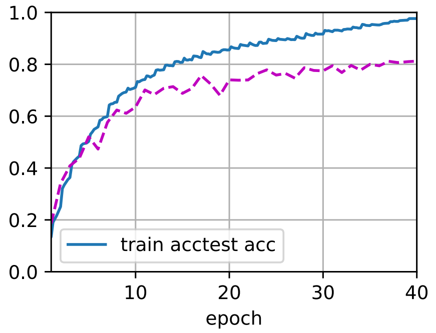
```
total test accuracy 0.788
Total training time per epoch 3.3
66.9 total training time
cuda:0
```



In [ ]:

```
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)


net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 40,
               device,512,0.01)
```

```
total test accuracy 0.812
Total training time per epoch 3.3
133.7 total training time
cuda:0
```



Inceasing epochs does not seem like a feasible option for increasing accuracy.

In [ ]:

```
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize,lr):



    trainer=torch.optim.SGD(net.parameters(), lr=lr, momentum= 0.9, weight_decay = 0.00001)#
    loss=nn.CrossEntropyLoss(label_smoothing=0.2)

    LRsched = torch.optim.lr_scheduler.OneCycleLR(trainer ,
                                                         max_lr=0.8,
                                                         epochs=num_epochs,
                                                         steps_per_epoch=len(trainloader),
                                                         anneal_strategy = "linear")


    timer, num_batches = d2l.Timer(), len( trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train acc','test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i

        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate( trainloader):
            timer.start()
            trainer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = torch.mul(net(X),0.125)
            l = loss(y_hat, y)
            with torch.no_grad():
              l.backward()
              metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])


            trainer.step()
            LRsched.step()
            timer.stop()
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                            animator.add(epoch + (i + 1) / num_batches,
                                        (metric[1] / metric[2],
                                          None))

        test_acc = d2l.evaluate_accuracy_gpu(net,  testloader)
        animator.add(epoch + 1, ( None, test_acc))



    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')
```

In [ ]:

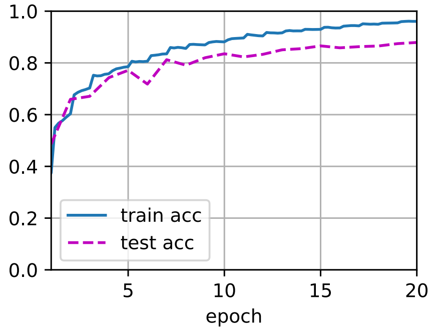
```
class ResNet(d2l.Classifier):

    def __init__(self, arch, lr=0.1, num_classes=10):
      super().__init__()
      self.save_hyperparameters()
      self.net = nn.Sequential(self.b1())



#Adding blocks
      for i, b in enumerate(arch):
          if i == 1:
            self.net.add_module(f'b{i+2}',nn.Sequential(
                nn.LazyConv2d(256, kernel_size=3, padding=1,stride=1),
                nn.MaxPool2d(kernel_size=2),
                nn.BatchNorm2d(256,eps=1),
                nn.CELU(alpha=0.075)
                ))
          else:
            self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))


##Final block
      self.net.add_module('last', nn.Sequential(
          nn.MaxPool2d(4), nn.Flatten(),nn.LazyLinear(num_classes),nn.Identity()))
      self.net.apply(d2l.init_cnn)


    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64,eps=1), nn.CELU(alpha=0.075) )


    def block(self, num_residuals, num_channels, first_block=False):
      blk = []
      for i in range(num_residuals):
              blk.append(Residual( num_channels))
      return nn.Sequential(*blk)





    def layer_summary(self, X_shape):
        """Defined in :numref:`sec_lenet`"""
        X = d2l.randn(*X_shape)
        for layer in self.net:
            X = layer(X)
            print(layer.__class__.__name__, 'output shape:\t', X.shape)

class Residual(nn.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)

        self.bn1 = nn.LazyBatchNorm2d(1)
        self.bn2 = nn.LazyBatchNorm2d(1)
        self.bn3 = nn.LazyBatchNorm2d(1)
        self.maxpool = nn.MaxPool2d(kernel_size=2)
        self.ident1 =nn.Identity()
        self.ident2 =nn.Identity()

    def forward(self, X):
        Y = F.celu(self.bn1(self.maxpool(self.conv1(X))),0.075)
        X=  self.ident1(Y)
        Y = F.celu(self.bn2(self.conv2(X)),0.075)
        Y = F.celu(self.bn3(self.conv3(Y)),0.075)
        Y= self.ident2(Y)
        Y  = X+Y
        return F.celu(Y,0.075)
```

Label smoothing & Scaling batch norm scales (87.9%, 68.8s, 20 epoch)

In [ ]:

```
net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

```
total test accuracy 0.879
Total training time per epoch 3.4
68.8 total training time
cuda:0
```



In [ ]:

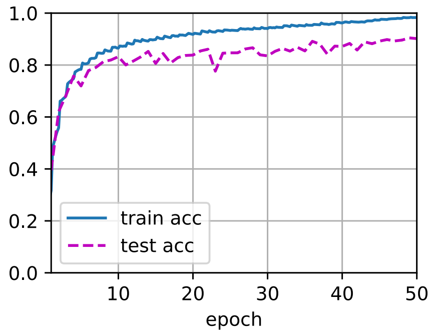
```
def train_ch13(net, trainloader, testloader,num_epochs,
               device,batchsize,lr):



    trainer=torch.optim.SGD(net.parameters(), lr=lr, momentum= 0.9, weight_decay = 0.00005)#
    loss=nn.CrossEntropyLoss(label_smoothing=0.2)

    LRsched = torch.optim.lr_scheduler.OneCycleLR(trainer ,
                                                         max_lr=0.9,
                                                         epochs=num_epochs,
                                                         steps_per_epoch=len(trainloader),
                                                         anneal_strategy = "linear")


    timer, num_batches = d2l.Timer(), len( trainloader)
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0, 1],
                            legend=['train acc','test acc'])
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples,
        # no. of predictions,LRsched, i

        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate( trainloader):
            timer.start()
            trainer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = torch.mul(net(X),0.125)
            l = loss(y_hat, y)
            with torch.no_grad():
              l.backward()
              metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])


            trainer.step()
            LRsched.step()
            timer.stop()
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                            animator.add(epoch + (i + 1) / num_batches,
                                        (metric[1] / metric[2],
                                          None))

        test_acc = d2l.evaluate_accuracy_gpu(net,  testloader)
        animator.add(epoch + 1, ( None, test_acc))
    print(f'total test accuracy {test_acc:.3f}')
    print(f'Total training time per epoch {timer.sum()/num_epochs:.1f}')
    print(f'{ timer.sum():.1f} total training time ')
    print(f'{str(device)}')
```

Epoch and Batch size (90.2%, 191.1s, 50 epochs)

In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                                   ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010]),
     transforms.RandomErasing()])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])
batch_size = 256

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)

device= torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((1, 128), (1, 256), (1, 512)),
                       lr, num_classes)
ResNet18().layer_summary((1,3,32,32))

net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 50,
               device,256,0.01)
```

```
total test accuracy 0.902
Total training time per epoch 3.8
191.1 total training time
cuda:0
```



Architecture

In [ ]:

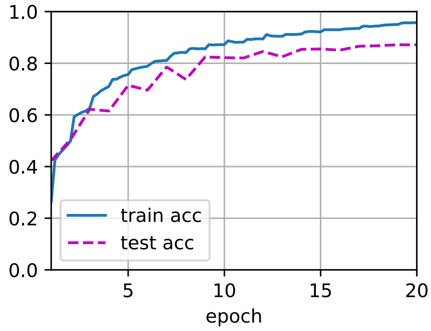
```
class ResNet(d2l.Classifier):

    def __init__(self, arch, lr=0.1, num_classes=10):
      super().__init__()
      self.save_hyperparameters()
      self.net = nn.Sequential(self.b1())



#Adding blocks
      for i, b in enumerate(arch):
          if i == 1:
            self.net.add_module(f'b{i+2}',nn.Sequential(
                nn.LazyConv2d(256, kernel_size=3, padding=1,stride=1),
                nn.BatchNorm2d(256,eps=1),
                nn.MaxPool2d(kernel_size=2),
                nn.CELU(alpha=0.075)
                ))
          else:
            self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))


##Final block
      self.net.add_module('last', nn.Sequential(
          nn.MaxPool2d(4), nn.Flatten(),nn.LazyLinear(num_classes),nn.Identity()))
      self.net.apply(d2l.init_cnn)


    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64,eps=1), nn.CELU(alpha=0.075) )


    def block(self, num_residuals, num_channels, first_block=False):
      blk = []
      for i in range(num_residuals):
              blk.append(Residual( num_channels))
      return nn.Sequential(*blk)




    def layer_summary(self, X_shape):
        """Defined in :numref:`sec_lenet`"""
        X = d2l.randn(*X_shape)
        for layer in self.net:
            X = layer(X)
            print(layer.__class__.__name__, 'output shape:\t', X.shape)

class Residual(nn.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)

        self.bn1 = nn.LazyBatchNorm2d(1)
        self.bn2 = nn.LazyBatchNorm2d(1)
        self.bn3 = nn.LazyBatchNorm2d(1)
        self.maxpool = nn.MaxPool2d(kernel_size=2)

        self.ident1 =nn.Identity()
        self.ident2 =nn.Identity()

    def forward(self, X):
        Y = F.celu(self.maxpool(self.bn1(self.conv1(X))),0.075)
        X=  self.ident1(Y)
        Y = F.celu(self.bn2(self.conv2(X)),0.075)
        Y = F.celu(self.bn3(self.conv3(Y)),0.075)
        Y= self.ident2(Y)
        Y  = X+Y
        return F.celu(Y,0.075)
```

In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                               ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010]),
     transforms.RandomErasing()])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
```

```
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)

device= torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

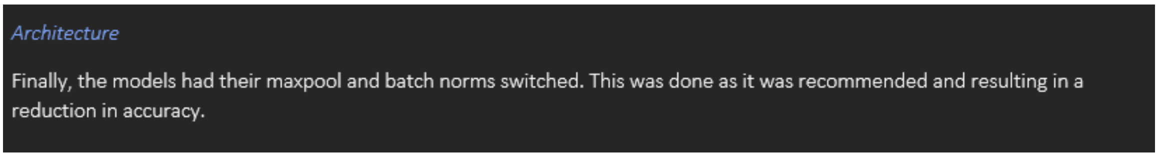
```
total test accuracy 0.871
Total training time per epoch 3.7
73.0 total training time
cuda:0
```



Drop layers

In [ ]:

```
class ResNet(d2l.Classifier):

    def __init__(self, arch, lr=0.1, num_classes=10):
      super().__init__()
      self.save_hyperparameters()
      self.net = nn.Sequential(self.b1())



#Adding blocks
      for i, b in enumerate(arch):
          if i == 1:
            self.net.add_module(f'b{i+2}',nn.Sequential(
                nn.Dropout(p=0.2),
                nn.LazyConv2d(256, kernel_size=3, padding=1,stride=1),
                nn.MaxPool2d(kernel_size=2),
                nn.BatchNorm2d(256,eps=1),
                nn.CELU(alpha=0.075)
                ))
          else:
            self.net.add_module(f'b{i+2}', self.block(*b, first_block=(i==0)))


##Final block
      self.net.add_module('last', nn.Sequential(nn.Dropout(0.7),
          nn.MaxPool2d(4), nn.Flatten(),nn.LazyLinear(num_classes),nn.Identity()))
      self.net.apply(d2l.init_cnn)


    def b1(self):
        return nn.Sequential(
            nn.LazyConv2d(64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64,eps=1), nn.CELU(alpha=0.075) )


    def block(self, num_residuals, num_channels, first_block=False):
      blk = []
      for i in range(num_residuals):
              blk.append(Residual( num_channels))
      return nn.Sequential(*blk)




    def layer_summary(self, X_shape):
        """Defined in :numref:`sec_lenet`"""
        X = d2l.randn(*X_shape)
        for layer in self.net:
            X = layer(X)
            print(layer.__class__.__name__, 'output shape:\t', X.shape)

class Residual(nn.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=1)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                      stride=1)
        self.conv3 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                      stride=1)

        self.bn1 = nn.LazyBatchNorm2d(1)
        self.bn2 = nn.LazyBatchNorm2d(1)
        self.bn3 = nn.LazyBatchNorm2d(1)
        self.maxpool  = nn.MaxPool2d(kernel_size=2)
        self.Dropout=nn.Dropout(0.2)
        self.ident1 =nn.Identity()
        self.ident2 =nn.Identity()

    def forward(self, X):
        Y= self.Dropout(X)
        Y = F.celu(self.bn1(self.maxpool(self.conv1(X))),0.075)
        X=  self.ident1(Y)
        Y = F.celu(self.bn2(self.conv2(X)),0.075)
        Y = F.celu(self.bn3(self.conv3(Y)),0.075)
        Y= self.ident2(Y)
        Y  = X+Y
        return F.celu(Y,0.075)
```



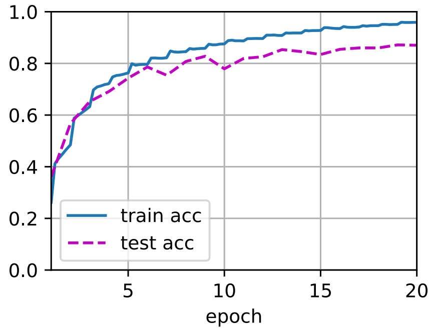
Architecture

Finally, the models had their maxpool and batch norms switched. This was done as it was recommended and resulting in a reduction in accuracy.

In [ ]:

```
transform1 = transforms.Compose(
    [

    torchvision.transforms.Resize(40),
    torchvision.transforms.RandomResizedCrop(32, scale=(0.64, 1.0),
                                               ratio=(1.0, 1.0)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),

    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010]),
     transforms.RandomErasing()])


transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.4914, 0.4822, 0.4465],
                                     [0.2023, 0.1994, 0.2010])])
batch_size = 512

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform= transform1)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=0)

device= torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net=ResNet9()
net.apply_init([next(iter(trainloader))[0]], d2l.init_cnn)
net=net.to(device)
train_ch13(net, trainloader,testloader, 20,
               device,512,0.01)
```

```
total test accuracy 0.870
Total training time per epoch 3.5
69.4 total training time
cuda:0
```


epoch
train acc
test acc