Summary

- Some DL tricks and summary

数据增广技术

- 在深度卷积神经网络里我们提到过，大规模数据集是成功应用深度神经网络的前提。数据增广（data augmentation）技术通过对训练图像做一系列随机改变，来产生相似但又不同的训练样本，从而扩大训练数据集的规模。图像增广的另一种解释是，随机改变训练样本可以降低模型对某些属性的依赖，从而提高模型的泛化能力。例如，我们可以对图像进行不同方式的裁剪，使感兴趣的物体出现在不同位置，从而减轻模型对物体出现位置的依赖性。我们也可以调整亮度、色彩等因素来降低模型对色彩的敏感度。可以说，在当年AlexNet的成功中，图像增广技术功不可没。本节我们将讨论这个在计算机视觉里被广泛使用的技术

常用的图像增广方法

- 我们来读取一张形状为400×500（高和宽分别为400像素和500像素）的图像作为实验的样例。

d2l.set_figsize() img = Image.open('../img/cat1.jpg')

d2l.plt.imshow(img)

常用的图像增广方法

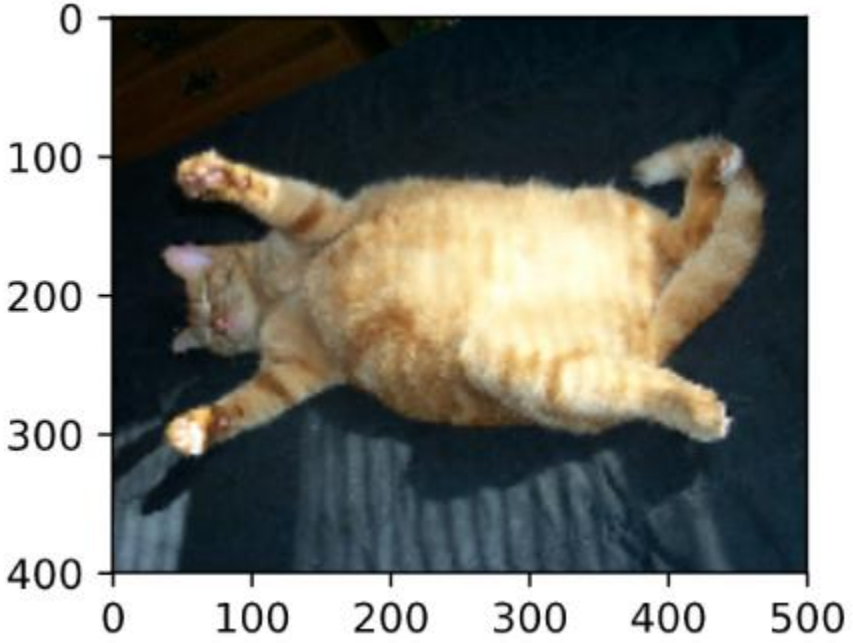
- 下面定义绘图函数show_images。

```
def show_images(imgs, num_rows, num_cols, scale=2):
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
    for i in range(num_rows):
        for j in range(num_cols):
            axes[i][j].imshow(imgs[i * num_cols + j])
            axes[i][j].axes.get_xaxis().set_visible(False)
            axes[i][j].axes.get_yaxis().set_visible(False)
    return axes
```

常用的图像增广方法

- 大部分图像增广方法都有一定的随机性。为了方便观察图像增广的效果，接下来我们定义一个辅助函数apply。这个函数对输入图像img多次运行图像增广方法aug并展示所有的结果。

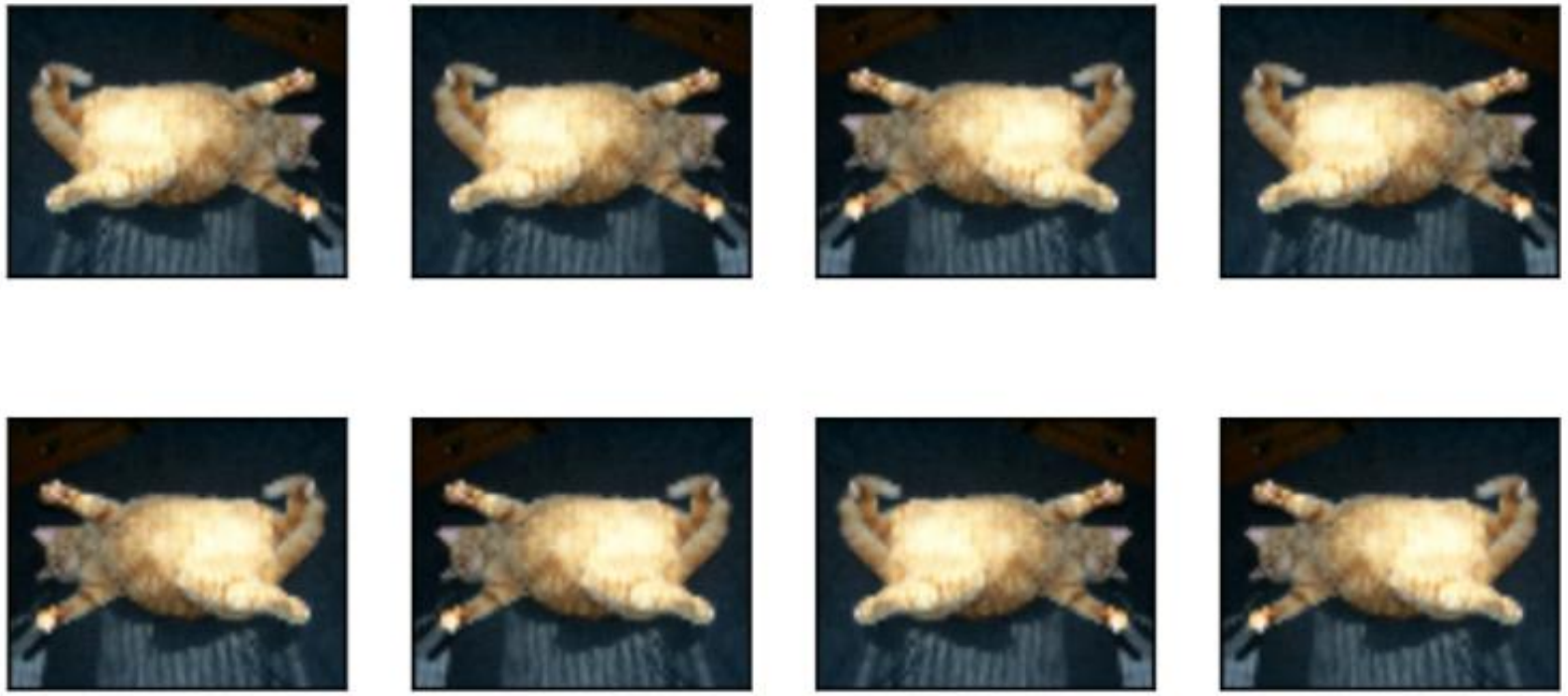
```
def apply(img, aug, num_rows=2, num_cols=4, scale=1.5):
    Y = [aug(img) for _ in range(num_rows * num_cols)]
    show_images(Y, num_rows, num_cols, scale)
```



翻转和裁剪

- 左右翻转图像通常不改变物体的类别。它是最早也是最广泛使用的一种图像增广方法。下面我们通过torchvision.transforms模块创建RandomHorizontalFlip实例来实现一半概率的图像水平（左右）翻转。

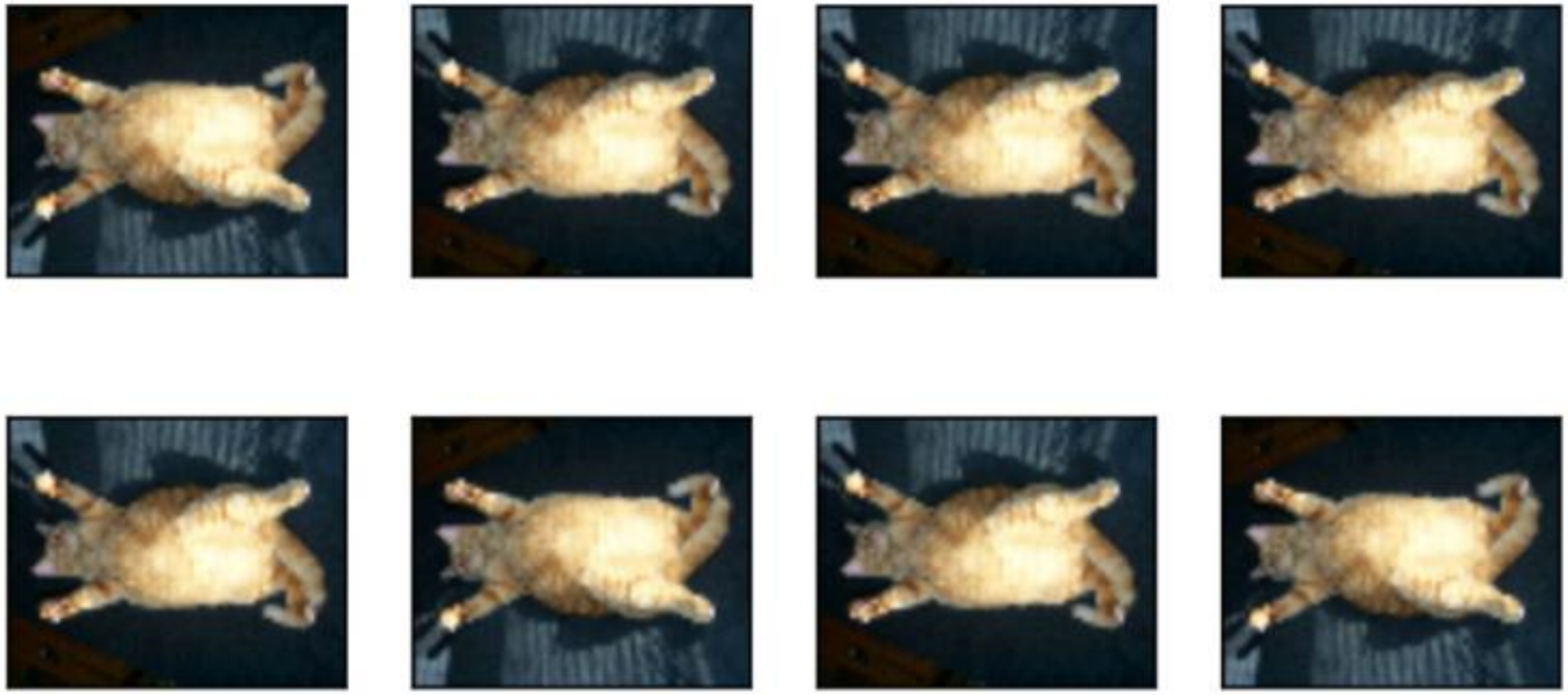
apply(img, torchvision.transforms.RandomHorizontalFlip())



翻转和裁剪

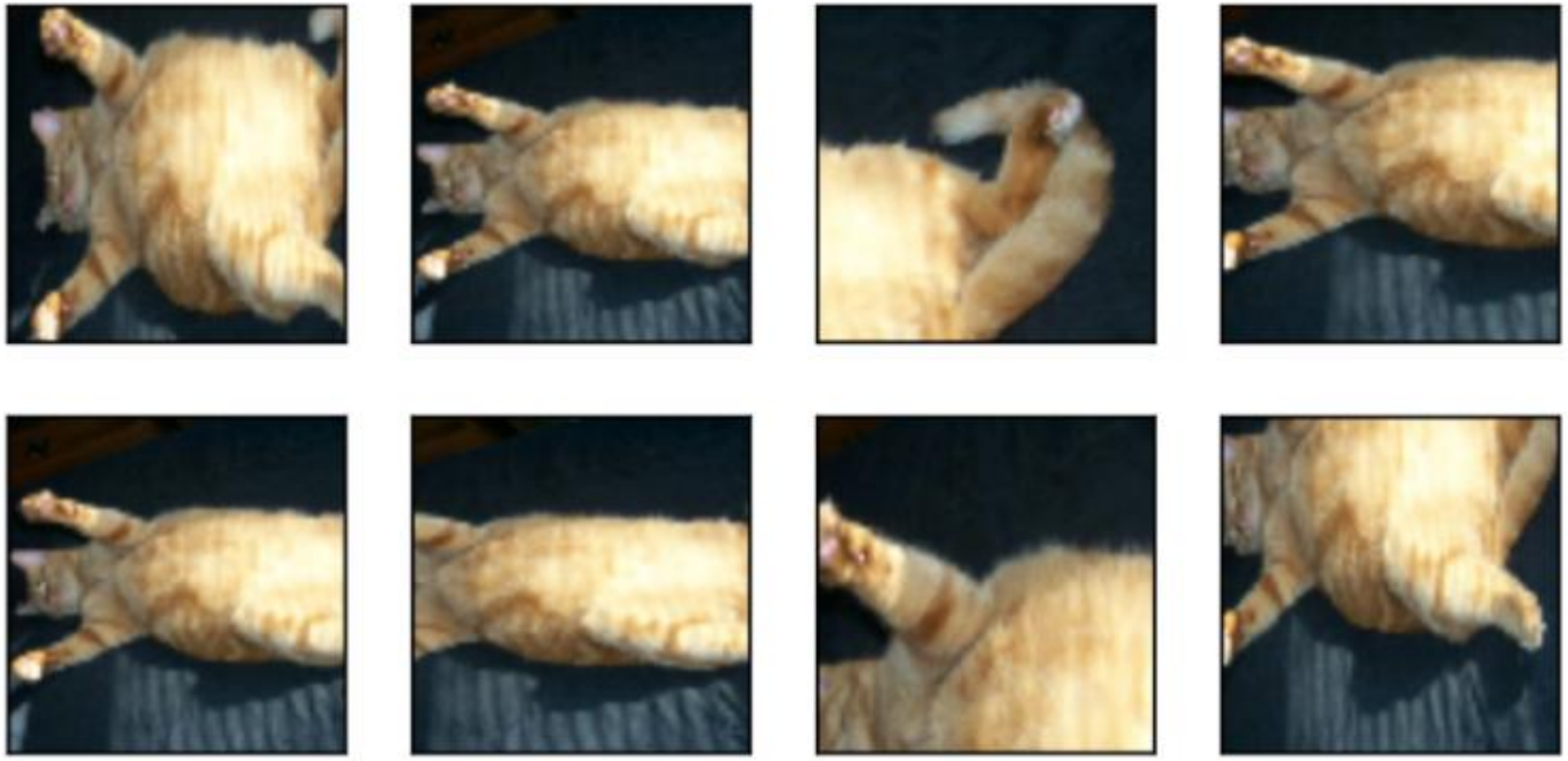
- 上下翻转不如左右翻转通用。但是至少对于样例图像，上下翻转不会造成识别障碍。下面我们创建RandomVerticalFlip实例来实现一半概率的图像垂直（上下）翻转。

apply(img, torchvision.transforms.RandomVerticalFlip())



- 在我们使用的样例图像里，猫在图像正中间，但一般情况下可能不是这样。在池化层里我们解释了池化层能降低卷积层对目标位置的敏感度。除此之外，我们还可以通过对图像随机裁剪来让物体以不同的比例出现在图像的不同位置，这同样能够降低模型对目标位置的敏感性。
- 在下面的代码里，我们每次随机裁剪出一块面积为原面积 10% ~ 100% 的区域，且该区域的宽和高之比随机取自 0.5 ~2 ，然后再将该区域的宽和高分别缩放到200像素。若无特殊说明，本节中 a 和 b 之间的随机数指的是从区间 [a,b] 中随机均匀采样所得到的连续值。

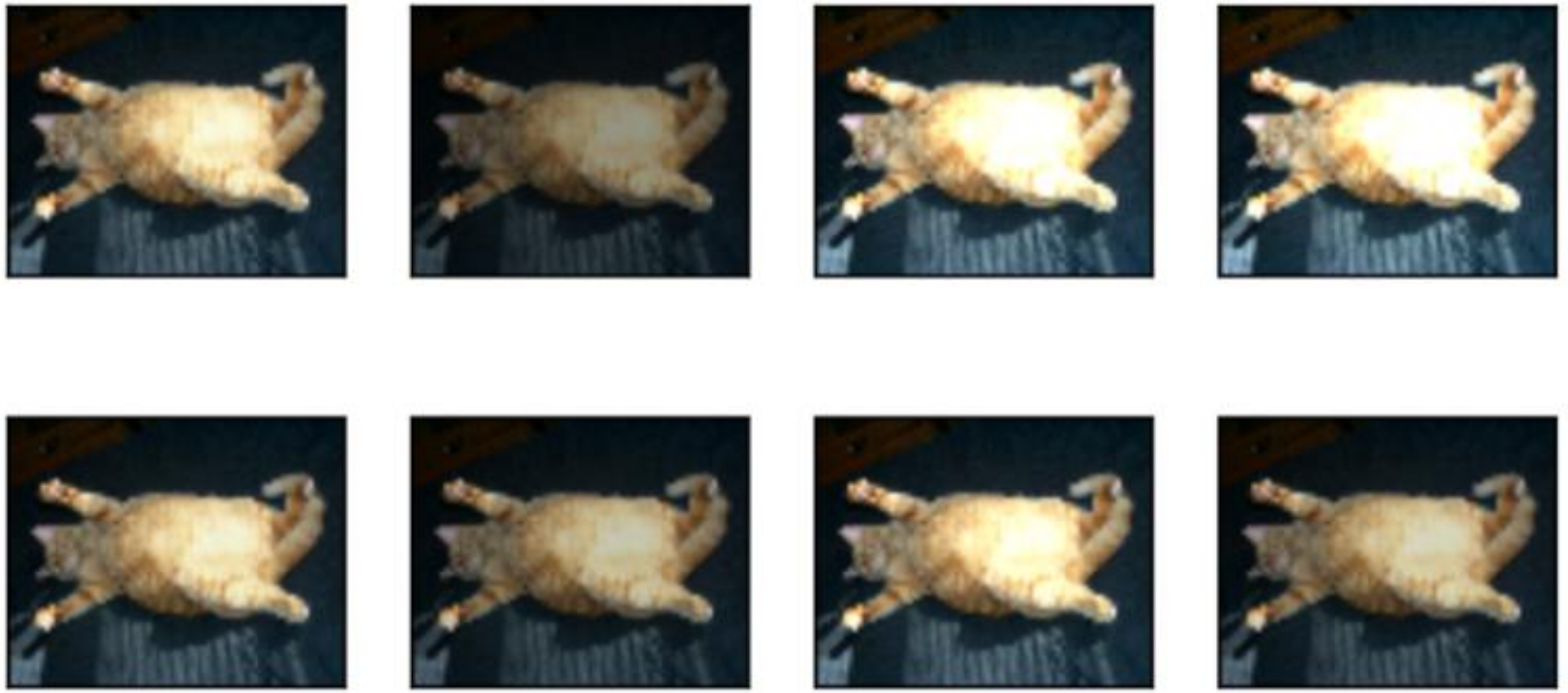
```
shape_aug = torchvision.transforms.RandomResizedCrop(200, scale=(0.1, 1), ratio=(0.5, 2))

apply(img, shape_aug)
```



变化颜色

- 另一类增广方法是变化颜色。我们可以从4个方面改变图像的颜色：亮度（brightness）、对比度（contrast）、饱和度（saturation）和色调（hue）。在下面的例子里，我们将图像的亮度随机变化为原图亮度的 50%（1-0.5）~150%（1+0.5）。

apply(img, torchvision.transforms.ColorJitter(brightness=0.5))



变化颜色

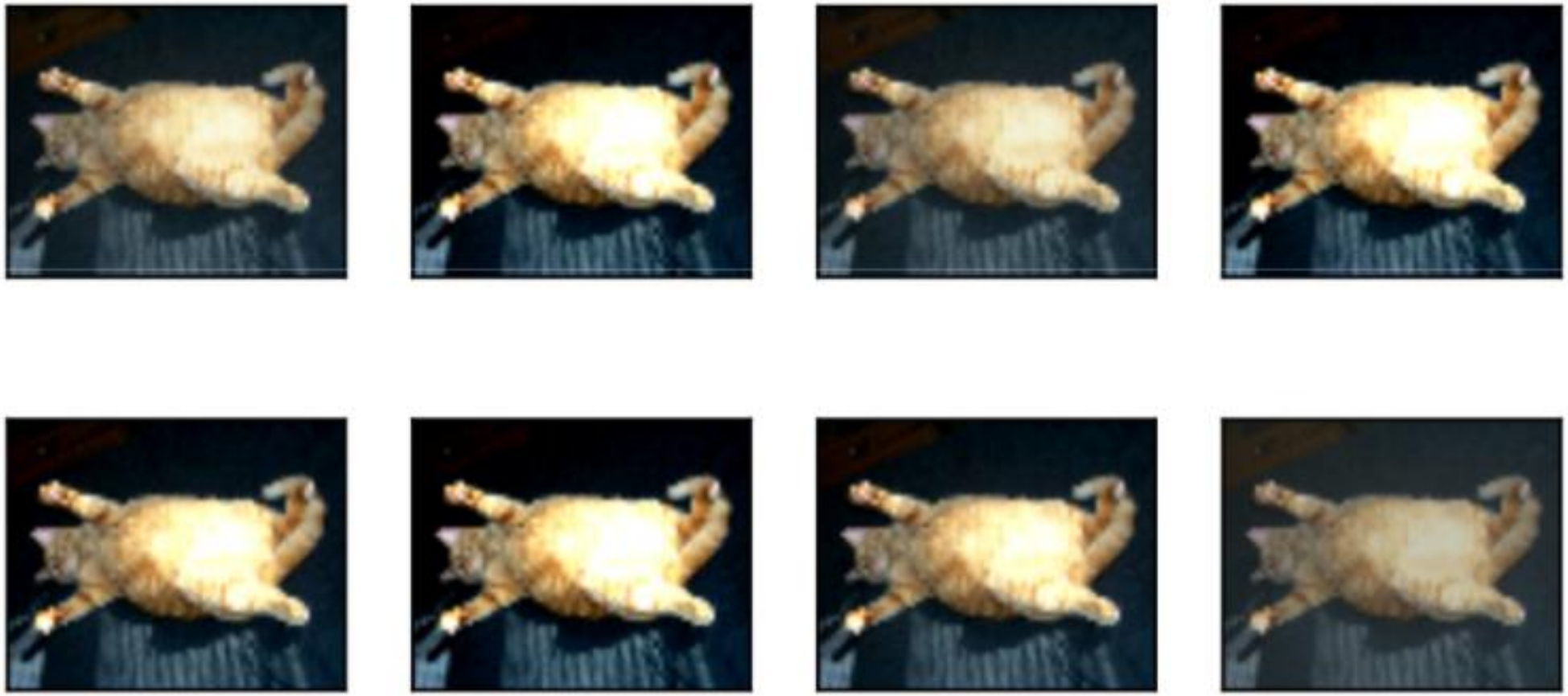
- 我们也可以随机变化图像的色调。

apply(img, torchvision.transforms.ColorJitter(hue=0.5))

变化颜色

- 类似地，我们也可以随机变化图像的对比度。

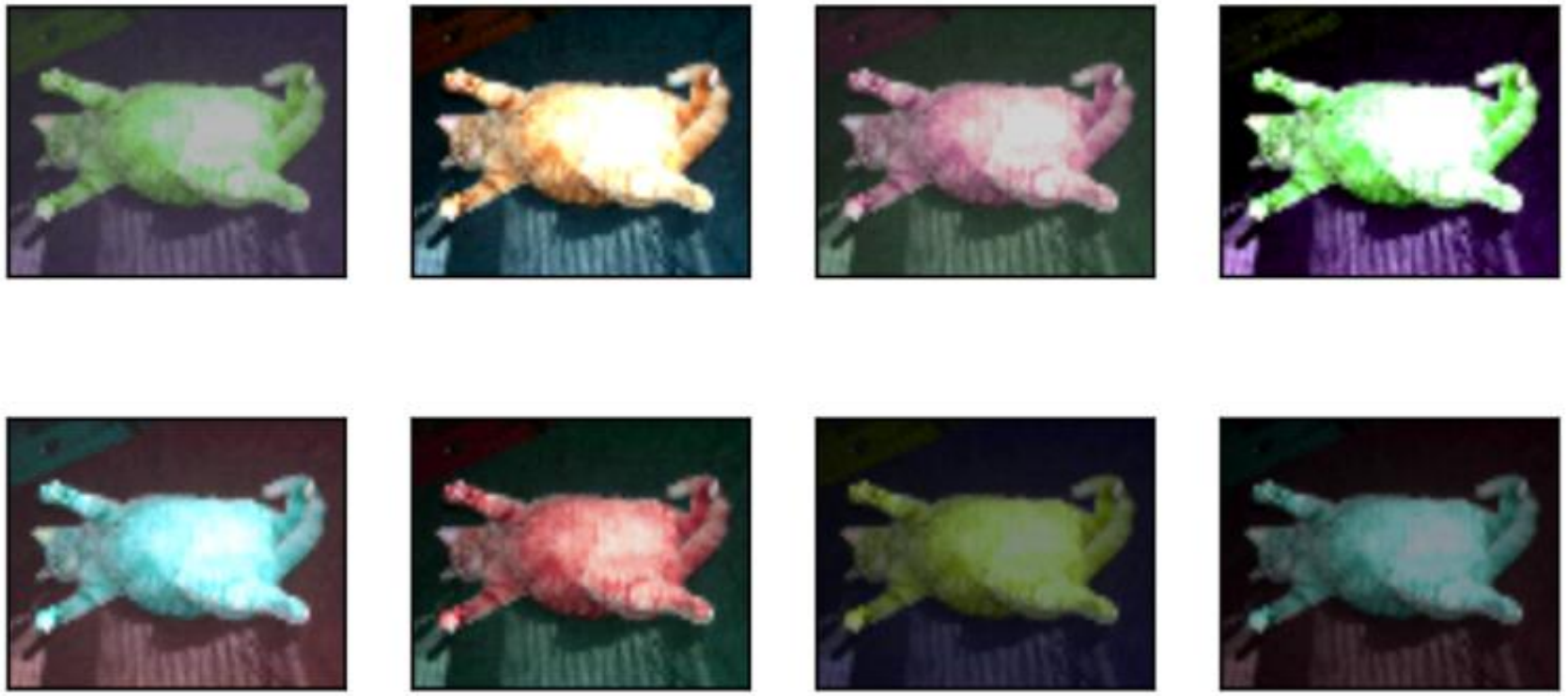
apply(img, torchvision.transforms.ColorJitter(contrast=0.5))



变化颜色

- 我们也可以同时设置如何随机变化图像的亮度（brightness）、对比度（contrast）、饱和度（saturation）和色调（hue）。

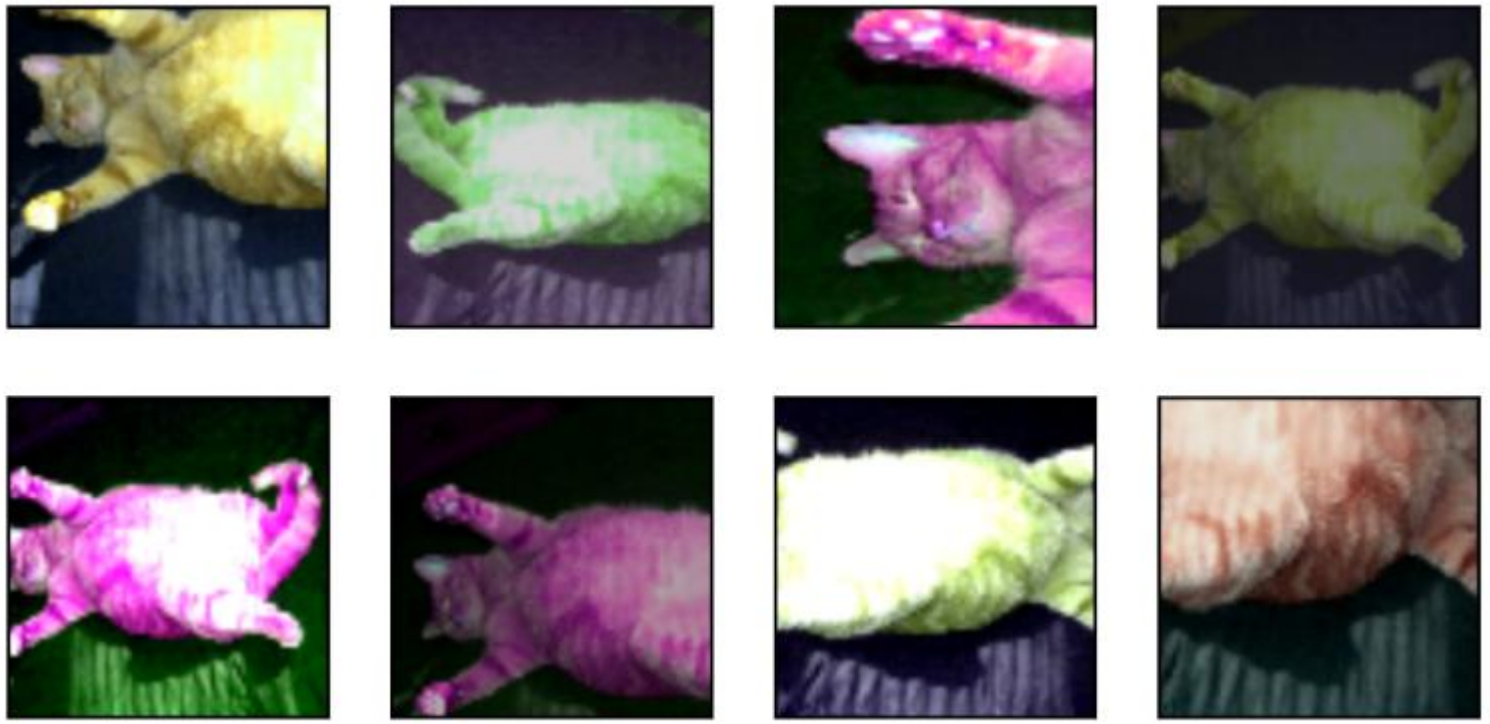
```
color_aug = torchvision.transforms.ColorJitter( brightness=0.5, contrast=0.5,
  saturation=0.5, hue=0.5)
apply(img, color_aug)
```



叠加多个图像增广方法

- 实际应用中我们会将多个图像增广方法叠加使用。我们可以通过Compose实例将上面定义的多个图像增广方法叠加起来，再应用到每张图像之上。

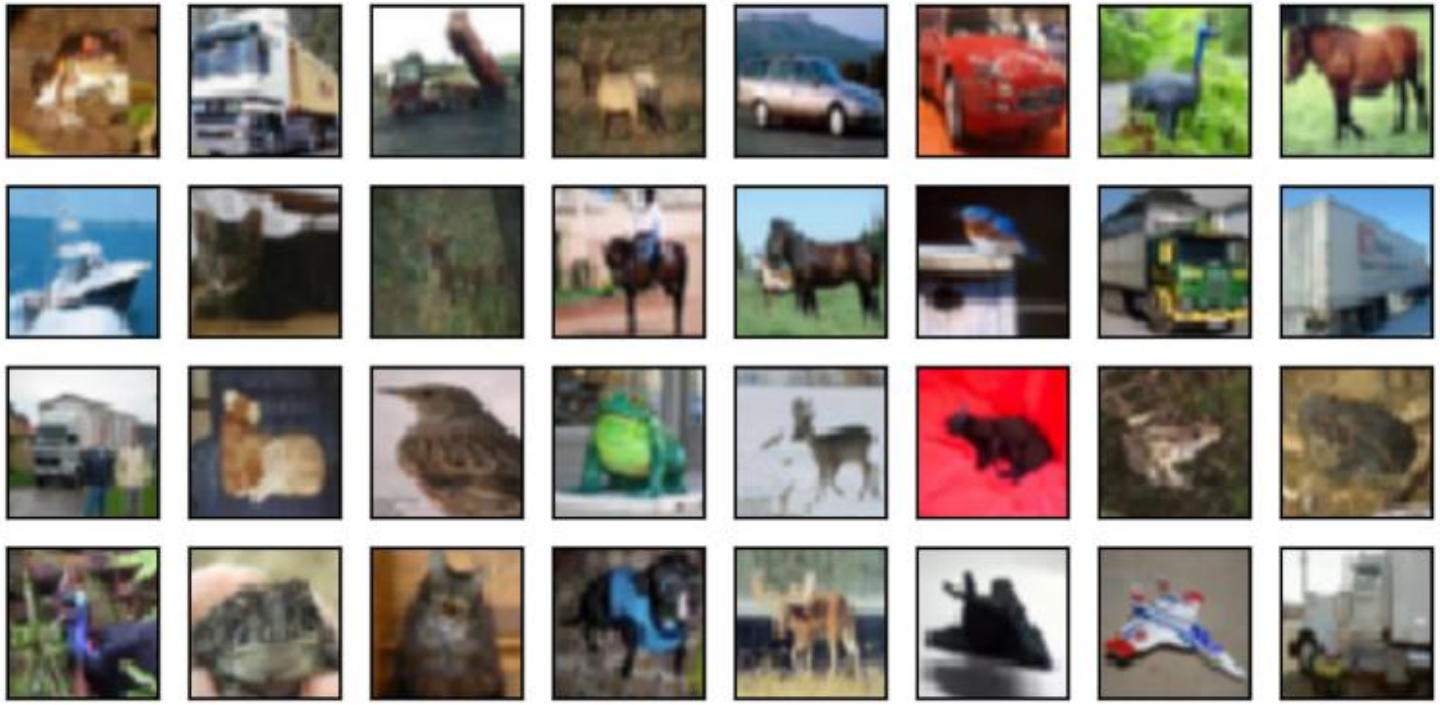
```
augs = torchvision.transforms.Compose([ torchvision.transforms.RandomHorizontalFlip(),
    color_aug, shape_aug])
apply(img, augs)
```



使用图像增广训练模型

- 下面我们来看一个将图像增广应用在实际训练中的例子。这里我们使用CIFAR-10数据集，因为其物体颜色和大小区别更显著。下面展示了该数据集前32张训练图像。

```
all_imges = torchvision.datasets.CIFAR10(train=True, root="~/Datasets/CIFAR", download=True)
# all_imges的每一个元素都是(image, label)
show_images([all_imges[i][0] for i in range(32)], 4, 8, scale=0.8);
```



使用图像增广训练模型

- **为了在预测时得到确定的结果，我们通常只将图像增广应用在训练样本上，而不在预测时使用含随机操作的图像增广**。在这里我们只使用最简单的随机左右翻转。此外，我们使用ToTensor将小批量图像转成PyTorch需要的格式，即形状为(批量大小, 通道数, 高, 宽)、值域在0到1之间且类型为32位浮点数。

```
flip_aug = torchvision.transforms.Compose([
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor()])

no_aug = torchvision.transforms.Compose([ torchvision.transforms.ToTensor()])
```

使用图像增广训练模型

- **接下来我们定义一个辅助函数来方便读取图像并应用图像增广。**

```
num_workers = 0 if sys.platform.startswith('win32') else 4
def load_cifar10(is_train, augs, batch_size, root="~/Datasets/CIFAR"):
    dataset = torchvision.datasets.CIFAR10(root=root, train=is_train, transform=augs, download=True)
    return DataLoader(dataset, batch_size=batch_size, shuffle=is_train, num_workers=num_workers)
```

- 我们在CIFAR-10数据集上训练残差网络中介绍的ResNet-18模型。

```
def train(train_iter, test_iter, net, loss, optimizer, device, num_epochs):
    net = net.to(device)
    print("training on ", device)
    batch_count = 0
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n, start = 0.0, 0.0, 0, time.time()
        for X, y in train_iter:
            X = X.to(device)
            y = y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            train_l_sum += l.cpu().item()
            train_acc_sum += (y_hat.argmax(dim=1) == y).sum().cpu().item()
            n += y.shape[0]
            batch_count += 1
        test_acc = d2l.evaluate_accuracy(test_iter, net)
        print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f, time %.1f sec'
              % (epoch + 1, train_l_sum / batch_count, train_acc_sum / n, test_acc, time.time() - start))
```

- 然后就可以定义train_with_data_aug函数使用图像增广来训练模型了。该函数使用Adam算法作为训练使用的优化算法，然后将图像增广应用于训练数据集之上，最后调用刚才定义的train函数训练并评价模型。

```
def train_with_data_aug(train_augs, test_augs, lr=0.001):
    batch_size, net = 256, d2l.resnet18(10)
    optimizer = torch.optim.Adam(net.parameters(), lr=lr)
    loss = torch.nn.CrossEntropyLoss()
    train_iter = load_cifar10(True, train_augs, batch_size)
    test_iter = load_cifar10(False, test_augs, batch_size)
    train(train_iter, test_iter, net, loss, optimizer, device, num_epochs=10)
```

- 下面使用随机左右翻转的图像增广来训练模型。

train_with_data_aug(flip_aug, no_aug)
- 输出：

training on cuda
epoch 1, loss 1.3615, train acc 0.505, test acc 0.493, time 123.2 sec
epoch 2, loss 0.5003, train acc 0.645, test acc 0.620, time 123.0 sec
epoch 3, loss 0.2811, train acc 0.703, test acc 0.616, time 123.1 sec
epoch 4, loss 0.1890, train acc 0.735, test acc 0.686, time 123.0 sec
epoch 5, loss 0.1346, train acc 0.765, test acc 0.671, time 123.1 sec
epoch 6, loss 0.1029, train acc 0.787, test acc 0.674, time 123.1 sec
epoch 7, loss 0.0803, train acc 0.804, test acc 0.749, time 123.1 sec
epoch 8, loss 0.0644, train acc 0.822, test acc 0.717, time 123.1 sec
epoch 9, loss 0.0526, train acc 0.836, test acc 0.750, time 123.0 sec
epoch 10, loss 0.0433, train acc 0.851, test acc 0.754, time 123.1 sec

微调

- 我们可以在只有6万张图像的Fashion-MNIST训练数据集上训练模型。我们还了解学术界当下使用最广泛的大规模图像数据集ImageNet，它有超过1,000万的图像和1,000类的物体。我们平常接触到数据集的规模通常在这两者之间。
- 假设我们想从图像中识别出不同种类的椅子，然后将购买链接推荐给用户。一种可能的方法是先找出100种常见的椅子，为每种椅子拍摄1,000张不同角度的图像，然后在收 集到的图像数据集上训练一个分类模型。这个椅子数据集虽然可能比Fashion-MNIST数据集要庞大，但样本数仍然不及ImageNet数据集中样本数的十分之一。这可能会导致适用于ImageNet数据集的复杂模型在这个椅子数据集上过拟合。同时，因为数据量有限，最终训练得到的模型的精度也可能达不到实用的要求。

微调

- 为了应对上述问题，一个显而易见的解决办法是收集更多的数据。然而，收集和标注数据会花费大量的时间和资金。例如，为了收集ImageNet数据集，研究人员花费了数百万美元的研究经费。虽然目前的数据采集成本已降低了不少，但其成本仍然不可忽略。
- 另外一种解决办法是应用迁移学习（transfer learning），将从源数据集学到的知识迁移到目标数据集上。例如，虽然ImageNet数据集的图像大多跟椅子无关，但在该数据集上训练的模型可以抽取较通用的图像特征，从而能够帮助识别边缘、纹理、形状和物体组成等。这些类似的特征对于识别椅子也可能同样有效。

微调

- 我们介绍迁移学习中的一种常用技术：微调（fine tuning），由4步构成。

1. 在源数据集（如ImageNet数据集）上预训练一个神经网络模型，即源模型。
2. 创建一个新的神经网络模型，即目标模型。它复制了源模型上除了输出层外的所有模型设计及其参数。我们假设这些模型参数包含了源数据集上学习到的知识，且这些知识同样适用于目标数据集。我们还假设源模型的输出层跟源数据集的标签紧密相关，因此在目标模型中不予采用。
3. 为目标模型添加一个输出大小为目标数据集类别个数的输出层，并随机初始化该层的模型参数。
4. 在目标数据集（如椅子数据集）上训练目标模型。我们将从头训练输出层，而其余层的参数都是基于源模型的参数微调得到的。

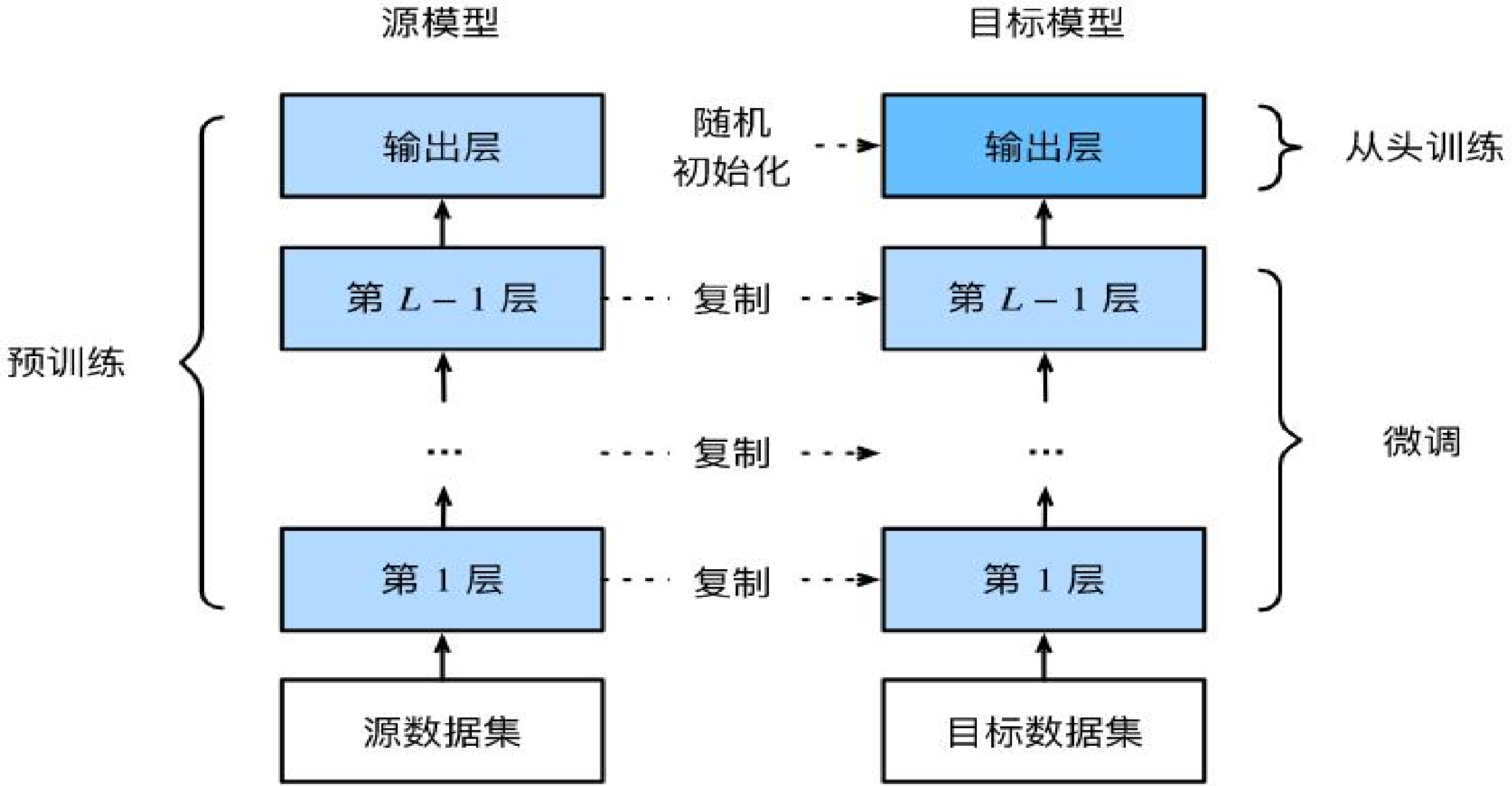

源模型
目标模型
输出层
随机
初始化
输出层
从头训练
第 $L-1$ 层
复制
第 $L-1$ 层
预训练
微调
...
复制
...
第 1 层
复制
第 1 层
源数据集
目标数据集

图9.1 微调

热狗识别

- 接下来我们来实践一个具体的例子：热狗识别。我们将基于一个小数据集对在ImageNet数据集上训练好的ResNet模型进行微调。该小数据集含有数千张包含热狗和不包含热狗的图像。我们将使用微调得到的模型来识别一张图像中是否包含热狗。
- 首先，导入实验所需的包或模块。torchvision的models包提供了常用的预训练模型。如果希望获取更多的预训练模型，可以使用使用pretrained-models.pytorch仓库。
- 我们使用的热狗数据集是从网上抓取的，它含有1400张包含热狗的正类图像，和同样多包含其他食品的负类图像。各类的1000张图像被用于训练，其余则用于测试。

获取数据集

- 我们首先将压缩后的数据集下载到路径data_dir之下，然后在该路径将下载好的数据集解压，得到两个文件夹hotdog/train和hotdog/test。这两个文件夹下面均有hotdog和not-hotdog两个类别文件夹，每个类别文件夹里面是图像文件。

```
data_dir = '/S1/CSCL/tangss/Datasets'
os.listdir(os.path.join(data_dir, "hotdog")) # ['train', 'test']
```

- 我们创建两个ImageFolder实例来分别读取训练数据集和测试数据集中的所有图像文件。

```
train_imgs = ImageFolder(os.path.join(data_dir, 'hotdog/train'))
test_imgs = ImageFolder(os.path.join(data_dir, 'hotdog/test'))
```

获取数据集

- 下面画出前8张正类图像和最后8张负类图像。可以看到，它们的大小和高宽比各不相同。

```
hotdogs = [train_imgs[i][0] for i in range(8)]
not_hotdogs = [train_imgs[-i - 1][0] for i in range(8)]
d2l.show_images(hotdogs + not_hotdogs, 2, 8, scale=1.4);
```

获取数据集

- 在训练时，我们先从图像中裁剪出随机大小和随机高宽比的一块随机区域，然后将该区域缩放为高和宽均为224像素的输入。测试时，我们将图像的高和宽均缩放为256像素，然后从中裁剪出高和宽均为224像素的中心区域作为输入。此外，我们对RGB（红、绿、蓝）三个颜色通道的数值做标准化：每个数值减去该通道所有数值的平均值，再除以该通道所有数值的标准差作为输出。

```
# 指定RGB三个通道的均值和方差来将图像通道归一化
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
train_augs = transforms.Compose([
        transforms.RandomResizedCrop(size=224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize
    ])

test_augs = transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        normalize
    ])
```

定义和初始化模型

- 我们使用在ImageNet数据集上预训练的ResNet-18作为源模型。这里指定pretrained=True来自动下载并加载预训练的模型参数。在第一次使用时需要联网下载模型参数。

```
pretrained_net = models.resnet18(pretrained=True)
```

- 下面打印源模型的成员变量fc。作为一个全连接层，它将ResNet最终的全局平均池化层输出变换成ImageNet数据集上1000类的输出。

```
print(pretrained_net.fc)
```

- 输出：

```
Linear(in_features=512, out_features=1000, bias=True)
```

定义和初始化模型

- 可见此时pretrained_net最后的输出个数等于目标数据集的类别数1000。所以我们应该将最后的fc成修改我们需要的输出类别数:

```
pretrained_net.fc = nn.Linear(512, 2)
print(pretrained_net.fc)
```

输出：

```
Linear(in_features=512, out_features=2, bias=True)
```

定义和初始化模型

- 此时，pretrained_net的fc层就被随机初始化了，但是其他层依然保存着预训练得到的参数。由于是在很大的ImageNet数据集上预训练的，所以参数已经足够好，因此一般只需使用较小的学习率来微调这些参数，而fc中的随机初始化参数一般需要更大的学习率从头训练。PyTorch可以方便的对模型的不同部分设置不同的学习参数，我们在下面代码中将fc的学习率设为已经预训练过的部分的10倍。

```
output_params = list(map(id, pretrained_net.fc.parameters()))
feature_params = filter(lambda p: id(p) not in output_params, pretrained_net.parameters())
lr = 0.01
optimizer = optim.SGD([{'params': feature_params},
                       {'params': pretrained_net.fc.parameters(), 'lr': lr * 10}],
                       lr=lr, weight_decay=0.001)
```

微调模型

- 我们先定义一个使用微调的训练函数train_fine_tuning以便多次调用。

```
def train_fine_tuning(net, optimizer, batch_size=128, num_epochs=5):
    train_iter = DataLoader(ImageFolder(os.path.join(data_dir, 'hotdog/train'), transform=train_augs),
                            batch_size, shuffle=True)
    test_iter = DataLoader(ImageFolder(os.path.join(data_dir, 'hotdog/test'), transform=test_augs),
                           batch_size)
    loss = torch.nn.CrossEntropyLoss()
    d2l.train(train_iter, test_iter, net, loss, optimizer, device, num_epochs)
```

微调模型

根据前面的设置，我们将以10倍的学习率从头训练目标模型的输出层参数。

```
train_fine_tuning(pretrained_net, optimizer)
```

输出：

```
training on  cuda
epoch 1, loss 3.1183, train acc 0.731, test acc 0.932, time 41.4 sec
epoch 2, loss 0.6471, train acc 0.829, test acc 0.869, time 25.6 sec
epoch 3, loss 0.0964, train acc 0.920, test acc 0.910, time 24.9 sec
epoch 4, loss 0.0659, train acc 0.922, test acc 0.936, time 25.2 sec
epoch 5, loss 0.0668, train acc 0.913, test acc 0.929, time 25.0 sec
```

微调模型

- 作为对比，我们定义一个相同的模型，但将它的所有模型参数都初始化为随机值。由于整个模型都需要从头训练，我们可以使用较大的学习率。

```
scratch_net = models.resnet18(pretrained=False, num_classes=2)
lr = 0.1
optimizer = optim.SGD(scratch_net.parameters(), lr=lr, weight_decay=0.001)
train_fine_tuning(scratch_net, optimizer)
```

微调模型

- 输出：

training on cuda
epoch 1, loss 2.6686, train acc 0.582, test acc 0.556, time 25.3 sec
epoch 2, loss 0.2434, train acc 0.797, test acc 0.776, time 25.3 sec
epoch 3, loss 0.1251, train acc 0.845, test acc 0.802, time 24.9 sec
epoch 4, loss 0.0958, train acc 0.833, test acc 0.810, time 25.0 sec
epoch 5, loss 0.0757, train acc 0.836, test acc 0.780, time 24.9 sec

可以看到，微调的模型因为参数初始值更好，往往在相同迭代周期下取得更高的精度。

Summary

Classes

- #1 Intro slide | swi-prolog
- #2 Reasoning slide | Source Code
- #3 Searching-I slide
- #4 Searching-II slide
- #5 Machine Learning Intro slide
- #6 Classification slide
- #7 Linear Models slide
- #8 Trees and Forests slide
- #9 Dimension Reduction slide
- #10 CNN slide
- #11 RNN slide
- #12 AutoEncoder slide

Projects

- Task1(AutoDriving)：
- Task2(ReDoPaper):
- Task3(arXivAPP)：
- Task4(SWI)：
- Task5(AIBooks)：
- Task6(miniWatson)：
- Task7(AIWriter):
- Task8(ResearchGraph):
- Task9(HumanEye):
- Task10(Brainstorm):
- Task11(NewsRec):

Final Report

- 12/18 and 12/25
- Five min. report with one min. discussion
- Focus on your **own** work

Survey

- https://www.wjx.cn/jq/100566452.aspx