

图像增广

在5.6节（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。图像增广（image 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。

首先，导入实验所需的包或模块。

In [1]:

```
import os
os.listdir("/home/kesci/input/img2083/")
```

Out[1]:

```
['img']
```

In [2]:

```
%matplotlib inline
import os
import time
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torchvision
import sys
from PIL import Image

sys.path.append("/home/kesci/input/")
#置当前使用的GPU设备仅为0号设备
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

import d2lzh1981 as d2l

# 定义device, 是否使用GPU, 依据计算机配置自动会选择
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(torch.__version__)
print(device)

1.3.0
cpu
```

9.1.1 常用的图像增广方法

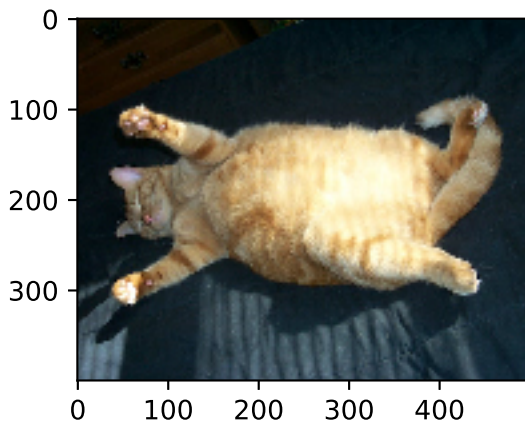
我们来读取一张形状为 400×500 （高和宽分别为400像素和500像素）的图像作为实验的样例。

In [3]:

```
d2l.set_figsize()
img = Image.open('/home/kesci/input/img2083/img/cat1.jpg')
d2l.plt.imshow(img)
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x7f8dae7aa198>
```



下面定义绘图函数show_images。

In [4]:

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
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
```

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。

In [5]:

```
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)
```

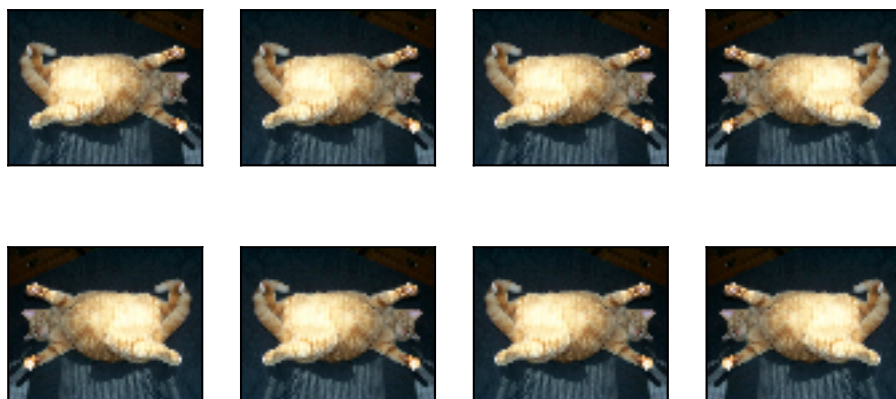
9.1.1.1 翻转和裁剪

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。

In [6]:

左右翻转

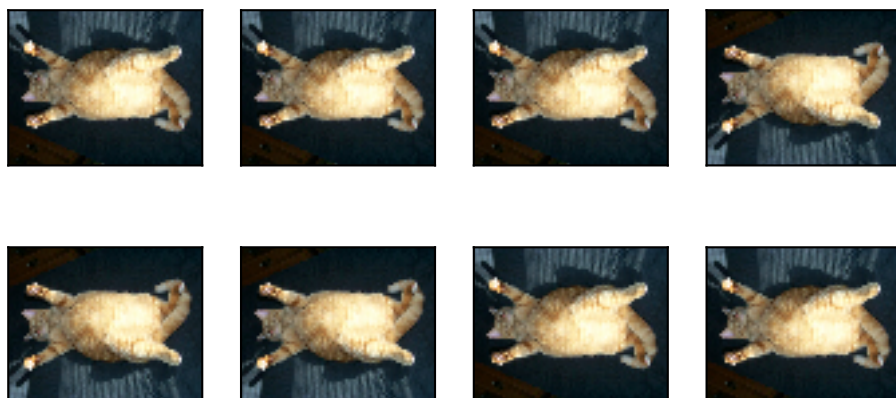
```
apply(img, torchvision.transforms.RandomHorizontalFlip())
```



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。

In [7]:

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

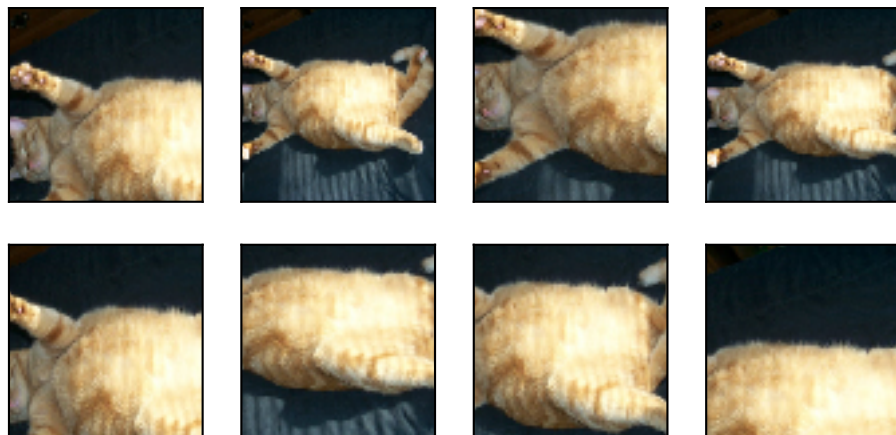


在我们使用的样例图像里，猫在图像正中间，但一般情况下可能不是这样。在5.4节（池化层）里我们解释了池化层能降低卷积层对目标位置的敏感度。除此之外，我们还可以通过对图像随机裁剪来让物体以不同的比例出现在图像的不同位置，这同样能够降低模型对目标位置的敏感性。

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。

In [8]:

```
shape_aug = torchvision.transforms.RandomResizedCrop(200, scale=(0.1, 1), ratio=(0.5, 1.5))  
apply(img, shape_aug)
```

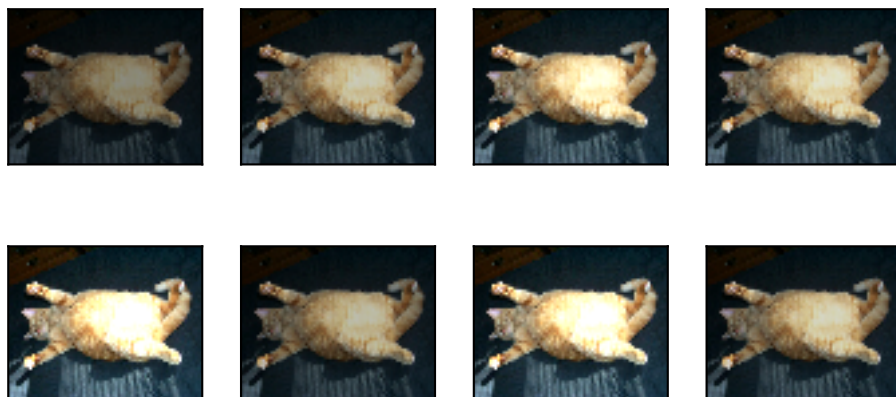


9.1.1.2 变化颜色

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)**。

In [9]:

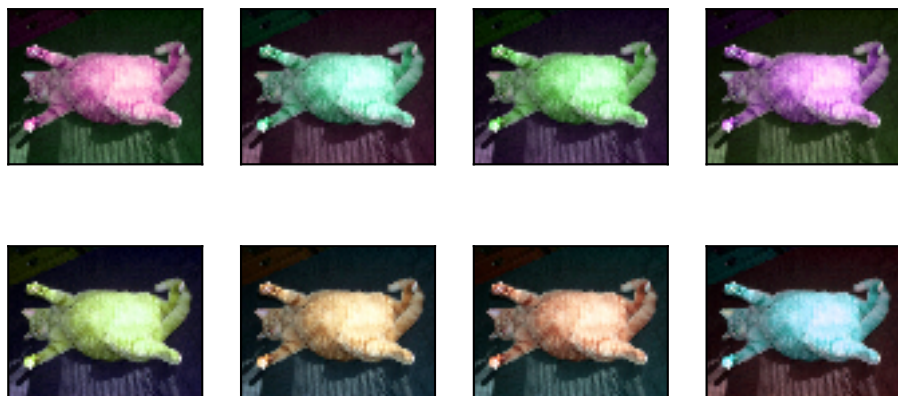
```
apply(img, torchvision.transforms.ColorJitter(brightness=0.5, contrast=0, saturation=0, hue=0))
```



我们也可以随机变化图像的色调。

In [10]:

```
apply(img, torchvision.transforms.ColorJitter(brightness=0, contrast=0, saturation=
```



类似地，我们也可以随机变化图像的对比度。

In [11]:

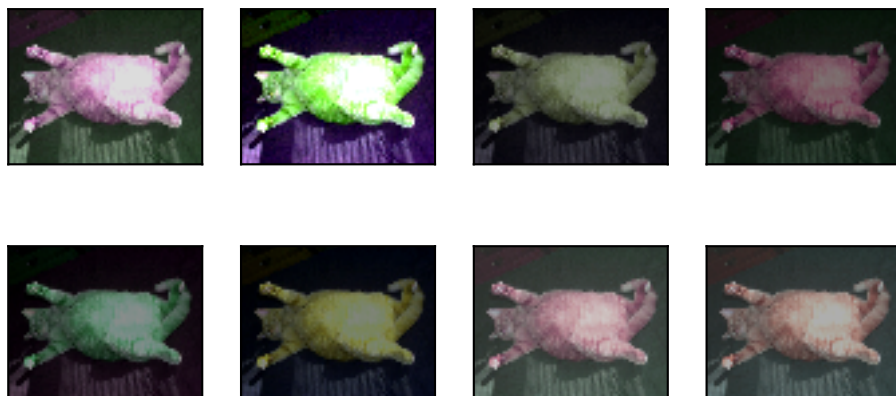
```
apply(img, torchvision.transforms.ColorJitter(brightness=0, contrast=0.5, saturatio
```



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）。

In [12]:

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

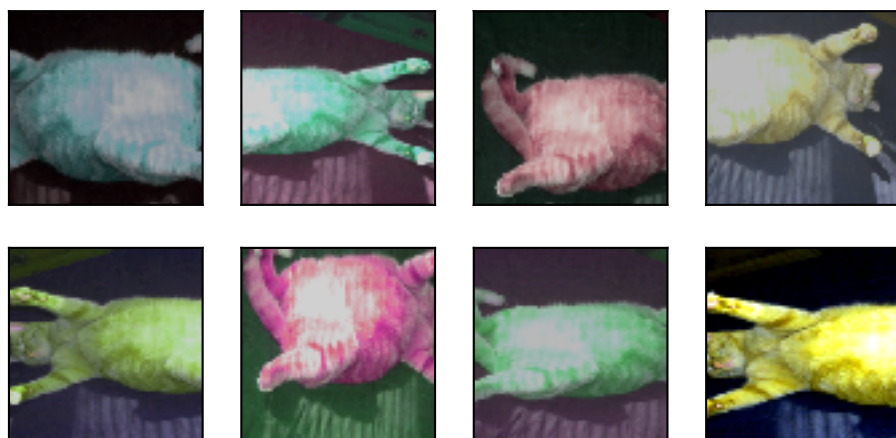


9.1.1.3 叠加多个图像增广方法

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。

In [13]:

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



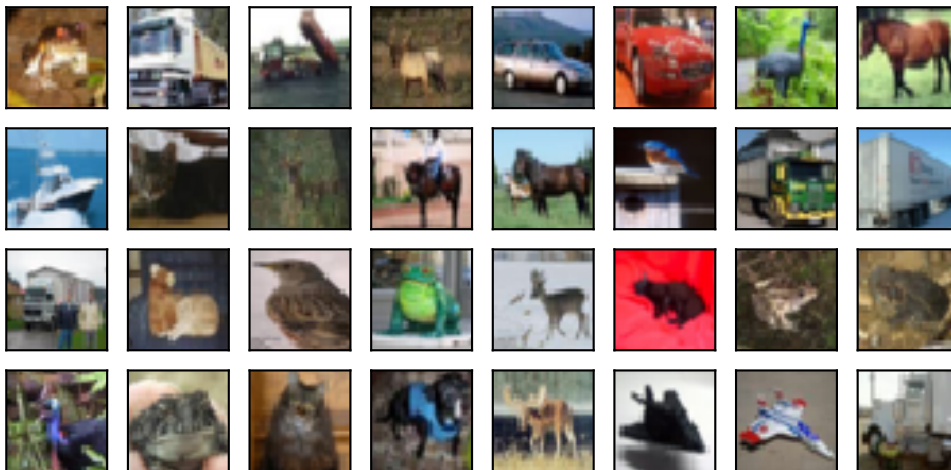
9.1.2 使用图像增广训练模型

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，而不是之前我们一直使用的Fashion-MNIST数据集。这是因为Fashion-MNIST数据集中物体的位置和尺寸都已经经过归一化处理，而CIFAR-10数据集中物体的颜色和大小区别更加显著。下面展示了CIFAR-10数据集中前32张训练图像。

In [15]:

```
CIFAR_ROOT_PATH = '/home/kesci/input/cifar102021'  
all_imges = torchvision.datasets.CIFAR10(train=True, root=CIFAR_ROOT_PATH, download  
# 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);
```

Files already downloaded and verified



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。

In [16]:

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

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。有关DataLoader的详细介绍，可参考更早的3.5节图像分类数据集(Fashion-MNIST)。

In [17]:

```
num_workers = 0 if sys.platform.startswith('win32') else 4
def load_cifar10(is_train, augs, batch_size, root=CIFAR_ROOT_PATH):
    dataset = torchvision.datasets.CIFAR10(root=root, train=is_train, transform=aug
    return DataLoader(dataset, batch_size=batch_size, shuffle=is_train, num_workers
```

9.1.2.1 使用图像增广训练模型

我们在CIFAR-10数据集上训练5.11节（残差网络）中介绍的ResNet-18模型。

我们先定义train函数使用GPU训练并评价模型。

In [18]:

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
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,
```

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。

```
%% Below, type any markdown to display in the Graffiti tip. %% Then run this cell to save it. train_iter =
load_cifar10(True, train_augs, batch_size) test_iter = load_cifar10(False, test_augs, batch_size)
```


In [19]:

```
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。

In [20]:

```
train_with_data_aug(flip_aug, no_aug)
```

training on cpu

```
epoch 1, loss 1.3790, train acc 0.504, test acc 0.554, time 195.8 sec  
epoch 2, loss 0.4992, train acc 0.646, test acc 0.592, time 192.5 sec  
epoch 3, loss 0.2821, train acc 0.702, test acc 0.657, time 193.7 sec  
epoch 4, loss 0.1859, train acc 0.739, test acc 0.693, time 195.4 sec  
epoch 5, loss 0.1349, train acc 0.766, test acc 0.688, time 192.6 sec  
epoch 6, loss 0.1022, train acc 0.786, test acc 0.701, time 200.2 sec  
epoch 7, loss 0.0797, train acc 0.806, test acc 0.720, time 191.8 sec  
epoch 8, loss 0.0633, train acc 0.825, test acc 0.695, time 198.6 sec  
epoch 9, loss 0.0524, train acc 0.836, test acc 0.693, time 192.1 sec  
epoch 10, loss 0.0437, train acc 0.850, test acc 0.769, time 196.3 sec
```

In []: