

小结

- 迁移学习将从源数据集学到的知识迁移到目标数据集上。微调是迁移学习的一种常用技术。
- 目标模型复制了源模型上除了输出层外的所有模型设计及其参数，并基于目标数据集微调这些参数。而目标模型的输出层需要从头训练。
- 一般来说，微调参数会使用较小的学习率，而从头训练输出层可以使用较大的学习率。

注：除代码外本节与原书基本相同，[原书传送门](#)

9.3 目标检测和边界框

在前面的一些章节中，我们介绍了诸多用于图像分类的模型。在图像分类任务里，我们假设图像里只有一个主体目标，并关注如何识别该目标的类别。然而，很多时候图像里有多个我们感兴趣的目标，我们不仅想知道它们的类别，还想得到它们在图像中的具体位置。在计算机视觉里，我们将这类任务称为目标检测（object detection）或物体检测。

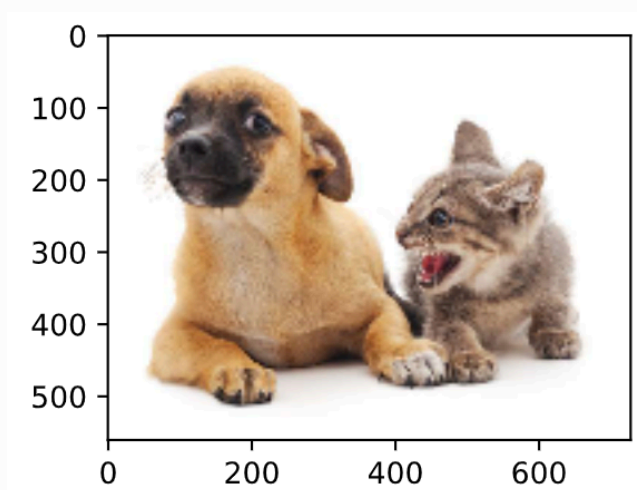
目标检测在多个领域中被广泛使用。例如，在无人驾驶里，我们需要通过识别拍摄到的视频图像里的车辆、行人、道路和障碍的位置来规划行进线路。机器人也常通过该任务来检测感兴趣的目标。安防领域则需要检测异常目标，如歹徒或者炸弹。

在接下来的几节里，我们将介绍目标检测里的多个深度学习模型。在此之前，让我们了解目标位置这个概念。先导入实验所需的包或模块。

```
1 %matplotlib inline
2 from PIL import Image
3
4 import sys
5 sys.path.append("../")
6 import d2lzh_pytorch as d2l
```

下面加载本节将使用的示例图像。可以看到图像左边是一只狗，右边是一只猫。它们是这张图像里的两个主要目标。

```
1 d2l.set_figsize()
2 img = Image.open('../img/catdog.jpg')
3 d2l.plt.imshow(img); # 加分号只显示图
```



9.3.1 边界框

在目标检测里，我们通常使用边界框（bounding box）来描述目标位置。边界框是一个矩形框，可以由矩形左上角的 x 和 y 轴坐标与右下角的 x 和 y 轴坐标确定。我们根据上面的图的坐标信息来定义图中狗和猫的边界框。图中的坐标原点在图像的左上角，原点往右和往下分别为 x 轴和 y 轴的正方向。

（x,y）
图片
（x₁, y₁）
（x₂, y₂）

```
1 # bbox是bounding box的缩写
2 dog_bbox, cat_bbox = [60, 45, 378, 516], [400, 112, 655, 493]
```

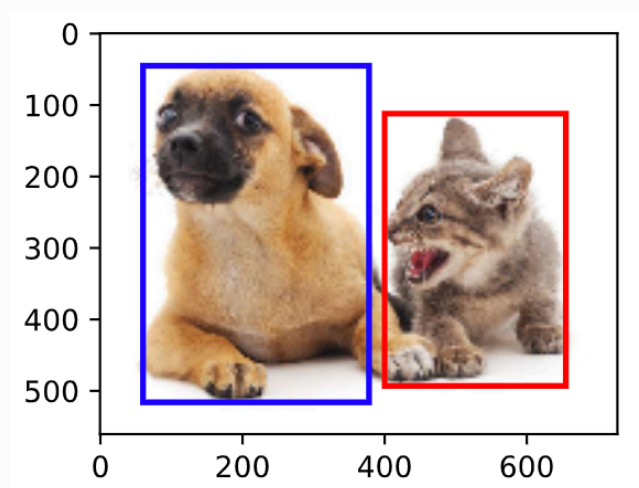
我们可以在图中将边界框画出来，以检查其是否准确。画之前，我们定义一个辅助函数 `bbox_to_rect`。它将边界框表示成matplotlib的边界框格式。

```
1 def bbox_to_rect(bbox, color): # 本函数已保存在d2lzh_pytorch中方便以后使用
2     # 将边界框(左上x, 左上y, 右下x, 右下y)格式转换成matplotlib格式:
3     # ((左上x, 左上y), 宽, 高)
4     return d2l.plt.Rectangle(
5         xy=(bbox[0], bbox[1]), width=bbox[2]-bbox[0],
6         height=bbox[3]-bbox[1],
7         fill=False, edgecolor=color, linewidth=2)
```

我们将边界框加载在图像上，可以看到目标的主要轮廓基本在框内。

```
1 fig = d2l.plt.imshow(img)
2 fig.axes.add_patch(bbox_to_rect(dog_bbox, 'blue'))
3 fig.axes.add_patch(bbox_to_rect(cat_bbox, 'red'));
```

输出：



小结

- 在目标检测里不仅需要找出图像里面所有感兴趣的目标，而且要知道它们的位置。位置一般由矩形边界框来表示。

注：除代码外本节与原书基本相同，[原书传送门](#)

9.4 锚框

目标检测算法通常会在输入图像中采样大量的区域，然后判断这些区域中是否包含我们感兴趣的目标，并调整区域边缘从而更准确地预测目标的真实边界框（ground-truth bounding box）。不同的模型使用的区域采样方法可能不同。这里我们介绍其中的一种方法：它以每个像素为中心生成多个大小和宽高比（aspect ratio）不同的边界框。这些边界框被称为锚框（anchor box）。我们将在后面基于锚框实践目标检测。

首先，导入本节需要的包或模块。这里我们新引入了 `contrib` 包，并修改了NumPy的打印精度。由于 `NDArray` 的打印实际调用NumPy的打印函数，本节打印出的 `NDArray` 中的浮点数更简洁一些。

```
1 %matplotlib inline
2 import d2lzh as d2l
3 from mxnet import contrib, gluon, image, nd
4 import numpy as np
5 np.set_printoptions(2)
```

9.4.1 生成多个锚框

就会为 $h \times w$ 个像素点就会生成不同大小的锚框

假设输入图像高为 h ，宽为 w 。我们分别以图像的每个像素为中心生成不同形状的锚框。设大小为 $s \in (0, 1]$ 且宽高比为 $r > 0$ ，那么锚框的宽和高将分别为 $ws\sqrt{r}$ 和 hs/\sqrt{r} 。当中心位置给定时，已知宽和高的锚框是确定的。

对于每一个像素都有

下面我们分别设定好一组大小 s_1, \dots, s_n 和一组宽高比 r_1, \dots, r_m 。如果以每个像素为中心时使用所有的大小与宽高比的组合，输入图像将一共得到 $whnm$ 个锚框。虽然这些锚框可能覆盖了所有的真实边界框，但计算复杂度容易过高。因此，我们通常只对包含 s_1 或 r_1 的大小与宽高比的组合感兴趣，即

$$(s_1, r_1), (s_1, r_2), \dots, (s_1, r_m), (s_2, r_1), (s_3, r_1), \dots, (s_n, r_1).$$

每一个像素都有这么多锚框

也就是说，以相同像素为中心的锚框的数量为 $n + m - 1$ 。对于整个输入图像，我们将一共生成 $wh(n + m - 1)$ 个锚框。



以上生成锚框的方法已实现在 `MultiBoxPrior` 函数中。指定输入、一组大小和一组宽高比，该函数将返回输入的所有锚框。

```
1 img = image.imread('../img/catdog.jpg').asnumpy()
2 h, w = img.shape[0:2]
3
4 print(h, w)
5 X = nd.random.uniform(shape=(1, 3, h, w)) # 构造输入数据
6 Y = contrib.nd.MultiBoxPrior(X, sizes=[0.75, 0.5, 0.25], ratios=[1,
7   2, 0.5]) # n=3, m=2
8 Y.shape
```

最后生成锚框数: $h \cdot w \cdot (n + m - 1)$

torch.Size([1, 2042040, 4])

表格记录, 每个坐标由4个元素表示。

我们看到，返回锚框变量`y`的形状为（批量大小，锚框个数，4）。将锚框变量`y`的形状变为（图像高，图像宽，以相同像素为中心的锚框个数，4）后，我们就可以通过指定像素位置来获取所有以该像素为中心的锚框了。下面的例子里我们访问以（250，250）为第一个锚框。它有4个元素，分别是锚框左上角的 x 和 y 轴坐标和右下角的 x 和 y 轴坐标，其中 x 和 y 轴的坐标值分别已除以图像的宽和高，因此值域均为0和1之间。

这儿四个维度的顺序不能变，因为在MultiBoxPrior中的把坐

标用meshgrid分成行列。

```
1 boxes = Y.reshape((h, w, 5, 4))
2 boxes[250, 250, 0, :]
```

坐标点(250, 250) 第0个框

```
1 # 展示某个像素点的 anchor
2 boxes = Y.reshape((h, w, 5, 4))
3 boxes[250, 250, 0, :] # torch.tensor([w, h, w, h], dtype=torch.float32)
4 # 第一个 size 和 ratio 分别为 0.75 和 1，则宽高均为 0.75 * 0.7184 = 0.5316 和 0.8206 = 0.6708
tensor([-0.0316, 0.0706, 0.7184, 0.8206])
```

为了描绘图像中以某个像素为中心的所有锚框，我们先定义 `show_bboxes` 函数以便在图像上画出多个边界框。

```
1 # 本函数已保存在d2lzh包中方便以后使用
2 def show_bboxes(axes, bboxes, labels=None, colors=None):
3     def _make_list(obj, default_values=None):
4         if obj is None:
5             obj = default_values
6         elif not isinstance(obj, (list, tuple)):
```

```

7         obj = [obj]
8         return obj
9
10        labels = _make_list(labels)
11        colors = _make_list(colors, ['b', 'g', 'r', 'm', 'c'])
12        for i, bbox in enumerate(bboxes):
13            color = colors[i % len(colors)]
14            rect = d2l.bbox_to_rect(bbox.asnumpy(), color)
15            axes.add_patch(rect)
16            if labels and len(labels) > i:
17                text_color = 'k' if color == 'w' else 'w'
18                axes.text(rect.xy[0], rect.xy[1], labels[i],
19                          va='center', ha='center', fontsize=9,
20                          color=text_color,
21                          bbox=dict(facecolor=color, lw=0))

```

刚刚我们看到，变量 `bboxes` 中 x 和 y 轴的坐标值分别已除以图像的宽和高。在绘图时，我们需要恢复锚框的原始坐标值，并因此定义了变量 `bbox_scale`。现在，我们可以画出图像中以 (250, 250) 为中心的所有锚框了。可以看到，大小为 0.75 且宽高比为 1 的锚框较好地覆盖了图像中的狗。

```

1 d2l.set_figsize() # 控制图片的大小, 3.5x2.5大小
2 bbox_scale = nd.array((w, h, w, h))
3 fig = d2l.plt.imshow(img)
4 show_bboxes(fig.axes, boxes[250, 250, :, :] * bbox_scale,
5             5个框的标签 ['s=0.75, r=1', 's=0.5, r=1', 's=0.25, r=1', 's=0.75,
6             r=2',
7             's=0.75, r=0.5'])

```

上面的内容用pytorch实现如下：

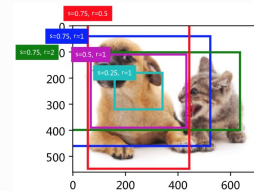
```

1 # 展示 250 250 像素点的 anchor
2 d2l.set_figsize()
3 fig = d2l.plt.imshow(img)
4 boxes[250, 250, :, :] = torch.tensor([w, h, w, h], dtype=torch.float32)
5 show_bboxes(fig.axes, boxes[250, 250, :, :] * bbox_scale,
6             ['s=0.75, r=1', 's=0.75, r=2', 's=0.75, r=0.5', 's=0.5, r=1', 's=0.25, r=1'])

```

可见其产生的tensor是归一化过的长度，所以分别为每个位置乘上w、h

9.4.2 交并比



我们刚刚提到某个锚框较好地覆盖了图像中的狗。如果该目标的真实边界框已知，这里的“较好”该如何量化呢？一种直观的方法是衡量锚框和真实边界框之间的相似度。我们知道，Jaccard系数 (Jaccard index) 可以衡量两个集合的相似度。给定集合 \mathcal{A} 和 \mathcal{B} ，它们的Jaccard系数即二者交集大小除以二者并集大小：

$$J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}.$$

实际上，我们可以把边界框内的像素区域看成是像素的集合。如此一来，我们可以用两个边界框的像素集合的Jaccard系数衡量这两个边界框的相似度。当衡量两个边界框的相似度时，我们通常将Jaccard系数称为交并比 (Intersection over Union, IoU)，即两个边界框相交面积与相并面积之比，如图9.2所示。交并比的取值范围在0和1之间：0表示两个边界框无重合像素，1表示两个边界框相等。

0 ~ 1

在本节的剩余部分，我们将使用交并比来衡量锚框与真实边界框以及锚框与锚框之间的相似度。

9.4.3 标注训练集的锚框

在训练集中，我们将每个锚框视为一个训练样本。为了训练目标检测模型，我们需要为每个锚框标注两类标签：一是锚框所含目标的类别，简称类别；二是真实边界框相对锚框的偏移量，简称偏移量（offset）。在目标检测时，我们首先生成多个锚框，然后为每个锚框预测类别以及偏移量，接着根据预测的偏移量调整锚框位置从而得到预测边界框，最后筛选需要输出的预测边界框。

我们知道，在目标检测的训练集中，每个图像已标注了真实边界框的位置以及所含目标的类别。在生成锚框之后，我们主要依据与锚框相似的真实边界框的位置和类别信息为锚框标注。那么，该如何为锚框分配与其相似的真实边界框呢？

假设图像中锚框分别为 A_1, A_2, \dots, A_{n_a} ，真实边界框分别为 B_1, B_2, \dots, B_{n_b} ，且 $n_a \geq n_b$ 。定义矩阵 $\mathbf{X} \in \mathbb{R}^{n_a \times n_b}$ ，其中第 i 行第 j 列的元素 x_{ij} 为锚框 A_i 与真实边界框 B_j 的交并比。首先，我们找出矩阵 \mathbf{X} 中最大元素，并将该元素的行索引与列索引分别记为 i_1, j_1 。我们为锚框 A_{i_1} 分配真实边界框 B_{j_1} 。显然，锚框 A_{i_1} 和真实边界框 B_{j_1} 在所有的“锚框—真实边界框”的配对中相似度最高。接下来，将矩阵 \mathbf{X} 中第 i_1 行和第 j_1 列上的所有元素丢弃。找出矩阵 \mathbf{X} 中剩余的最大元素，并将该元素的行索引与列索引分别记为 i_2, j_2 。我们为锚框 A_{i_2} 分配真实边界框 B_{j_2} ，再将矩阵 \mathbf{X} 中第 i_2 行和第 j_2 列上的所有元素丢弃。此时矩阵 \mathbf{X} 中已有两行两列的元素被丢弃。依此类推，直到矩阵 \mathbf{X} 中所有 n_b 列元素全部被丢弃。这个时候，我们已为 n_b 个锚框各分配了一个真实边界框。接下来，我们只遍历剩余的 $n_a - n_b$ 个锚框：给定其中的锚框 A_i ，根据矩阵 \mathbf{X} 的第 i 行找到与 A_i 交并比最大的真实边界框 B_j ，且只有当该交并比大于预先设定的阈值时，才为锚框 A_i 分配真实边界框 B_j 。

如图9.3（左）所示，假设矩阵 \mathbf{X} 中最大值为 x_{23} ，我们将为锚框 A_2 分配真实边界框 B_3 。然后，丢弃矩阵中第2行和第3列的所有元素，找出剩余阴影部分的最大元素 x_{71} ，为锚框 A_7 分配真实边界框 B_1 。接着如图9.3（中）所示，丢弃矩阵中第7行和第1列的所有元素，找出剩余阴影部分的最大元素 x_{54} ，为锚框 A_5 分配真实边界框 B_4 。最后如图9.3（右）所示，丢弃矩阵中第5行和第4列的所有元素，找出剩余阴影部分的最大元素 x_{92} ，为锚框 A_9 分配真实边界框 B_2 。之后，我们只需遍历除去 A_2, A_5, A_7, A_9 的剩余锚框，并根据阈值判断是否为剩余锚框分配真实边界框。

现在我们可以标注锚框的类别和偏移量了。如果一个锚框 A 被分配了真实边界框 B ，将锚框 A 的类别设为 B 的类别，并根据 B 和 A 的中心坐标的相对位置以及两个框的相对大小为锚框 A 标注偏移量。由于数据集中各个框的位置和大小各异，因此这些相对位置和相对大小通常需要一些特殊变换，才能使偏移量的分布更均匀从而更容易拟合。设锚框 A 及其被分配的真实边界框 B 的中心坐标分别为 (x_a, y_a) 和 (x_b, y_b) ， A 和 B 的宽分别为 w_a 和 w_b ，高分别为 h_a 和 h_b ，一个常用的技巧是将 A 的偏移量标注为

$$\left(\frac{\frac{x_b - x_a}{w_a} - \mu_x}{\sigma_x}, \frac{\frac{y_b - y_a}{h_a} - \mu_y}{\sigma_y}, \frac{\log \frac{w_b}{w_a} - \mu_w}{\sigma_w}, \frac{\log \frac{h_b}{h_a} - \mu_h}{\sigma_h} \right),$$

其中常数的默认值为 $\mu_x = \mu_y = \mu_w = \mu_h = 0, \sigma_x = \sigma_y = 0.1, \sigma_w = \sigma_h = 0.2$ 。如果一个锚框没有被分配真实边界框，我们只需将该锚框的类别设为背景。类别为背景的锚框通常被称为负类锚框，其余则被称为正类锚框。

下面演示一个具体的例子。我们为读取的图像中的猫和狗定义真实边界框，其中第一个元素为类别（0为狗，1为猫），剩余4个元素分别为左上角的 x 和 y 轴坐标以及右下角的 x 和 y 轴坐标（值域在0到1之间）。这里通过左上角和右下角的坐标构造了5个需要标注的锚框，分别记为 A_0, \dots, A_4 （程序中索引从0开始）。先画出这些锚框与真实边界框在图像中的位置。

```
1 ground_truth = nd.array([[0, 0.1, 0.08, 0.52, 0.92],
2                           [1, 0.55, 0.2, 0.9, 0.88]])
3 anchors = nd.array([[0, 0.1, 0.2, 0.3], [0.15, 0.2, 0.4, 0.4],
4                     [0.63, 0.05, 0.88, 0.98], [0.66, 0.45, 0.8,
5                     0.8],
6                     [0.57, 0.3, 0.92, 0.9]])
7
8 fig = d2l.plt.imshow(img)
9 show_bboxes(fig.axes, ground_truth[:, 1:] * bbox_scale, ['dog',
10 'cat'], 'k')
11 show_bboxes(fig.axes, anchors * bbox_scale, ['0', '1', '2', '3',
12 '4']);
```

我们可以通过 `contrib.nd` 模块中的 `MultiBoxTarget` 函数来为锚框标注类别和偏移量。该函数将背景类别设为0，并令从零开始的目标类别的整数索引自加1（1为狗，2为猫）。我们通过 `expand_dims` 函数为锚框和真实边界框添加样本维，并构造形状为(批量大小, 包括背景类别个数, 锚框数)的任意预测结果。

```
1 labels = contrib.nd.MultiBoxTarget(anchors.expand_dims(axis=0),
2                                     ground_truth.expand_dims(axis=0),
3                                     nd.zeros((1, 3, 5)))
```

返回的结果里有3项，均为 `NDArray`。第三项表示为锚框标注的类别。

```
1 labels[2]
```

我们根据锚框与真实边界框在图像中的位置来分析这些标注的类别。首先，在所有的“锚框—真实边界框”的配对中，锚框 A_4 与猫的真实边界框的交并比最大，因此锚框 A_4 的类别标注为猫。不考虑锚框 A_4 或猫的真实边界框，在剩余的“锚框—真实边界框”的配对中，最大交并比的配对为锚框 A_1 和狗的真实边界框，因此锚框 A_1 的类别标注为狗。接下来遍历未标注的剩余3个锚框：与锚框 A_0 交并比最大的真实边界框的类别为狗，但交并比小于阈值（默认为0.5），因此类别标注为背景；与锚框 A_2 交并比最大的真实边界框的类别为猫，且交并比大于阈值，因此类别标注为猫；与锚框 A_3 交并比最大的真实边界框的类别为猫，但交并比小于阈值，因此类别标注为背景。

返回值的第二项为掩码（mask）变量，形状为（批量大小，锚框个数的四倍）。掩码变量中的元素与每个锚框的4个偏移量一一对应。由于我们不关心对背景的检测，有关负类的偏移量不应影响目标函数。通过按元素乘法，掩码变量中的0可以在计算目标函数之前过滤掉负类的偏移量。

```
1 labels[1]
```

返回的第一项是为每个锚框标注的四个偏移量，其中负类锚框的偏移量标注为0。

```
1 labels[0]
```

9.4.4 输出预测边界框

在模型预测阶段，我们先为图像生成多个锚框，并为这些锚框一一预测类别和偏移量。随后，我们根据锚框及其预测偏移量得到预测边界框。当锚框数量较多时，同一个目标上可能会输出较多相似的预测边界框。为了使结果更加简洁，我们可以移除相似的预测边界框。常用的方法叫作非极大值抑制（non-maximum suppression, NMS）。

我们来描述一下非极大值抑制的工作原理。对于一个预测边界框 B ，模型会计算各个类别的预测概率。设其中最大的预测概率为 p ，该概率所对应的类别即 B 的预测类别。我们也将 p 称为预测边界框 B 的置信度。在同一图像上，我们将预测类别非背景的预测边界框按置信度从高到低排序，得到列表 L 。从 L 中选取置信度最高的预测边界框 B_1 作为基准，将所有与 B_1 的交并比大于某阈值的非基准预测边界框从 L 中移除。这里的阈值是预先设定的超参数。此时， L 保留了置信度最高的预测边界框并移除了与其相似的其他预测边界框。接下来，从 L 中选取置信度第二高的预测边界框 B_2 作为基准，将所有与 B_2 的交并比大于某阈值的非基准预测边界框从 L 中移除。重复这一过程，直到 L 中所有的预测边界框都曾作为基准。此时 L 中任意一对预测边界框的交并比都小于阈值。最终，输出列表 L 中的所有预测边界框。

下面来看一个具体的例子。先构造4个锚框。简单起见，我们假设预测偏移量全是0：预测边界框即锚框。最后，我们构造每个类别的预测概率。

```
1 anchors = nd.array([[0.1, 0.08, 0.52, 0.92], [0.08, 0.2, 0.56,  
2               [0.15, 0.3, 0.62, 0.91], [0.55, 0.2, 0.9,  
0.95]],  
0.88]))  
3 offset_preds = nd.array([0] * anchors.size)  
4 cls_probs = nd.array([0] * 4, # 背景的预测概率  
5               [0.9, 0.8, 0.7, 0.1], # 狗的预测概率  
6               [0.1, 0.2, 0.3, 0.9])) # 猫的预测概率
```

在图像上打印预测边界框和它们的置信度。

```
1 fig = d2l.plt.imshow(img)  
2 show_bboxes(fig.axes, anchors * bbox_scale,  
3             ['dog=0.9', 'dog=0.8', 'dog=0.7', 'cat=0.9'])
```


我们使用 `contrib.nd` 模块的 `MultiBoxDetection` 函数来执行非极大值抑制并设阈值为0.5。这里为 `NDArray` 输入都增加了样本维。我们看到，返回的结果的形状为(批量大小, 锚框个数, 6)。其中每一行的6个元素代表同一个预测边界框的输出信息。第一个元素是索引从0开始计数的预测类别（0为狗，1为猫），其中-1表示背景或在非极大值抑制中被移除。第二个元素是预测边界框的置信度。剩余的4个元素分别是预测边界框左上角的 x 和 y 轴坐标以及右下角的 x 和 y 轴坐标（值域在0到1之间）。

```
1 output = contrib.ndarray.MultiBoxDetection(  
2     cls_probs.expand_dims(axis=0), offset_preds.expand_dims(axis=0),  
3     anchors.expand_dims(axis=0), nms_threshold=0.5)  
4 output
```

我们移除掉类别为-1的预测边界框，并可视化非极大值抑制保留的结果。

```
1 fig = d2l.plt.imshow(img)  
2 for i in output[0].asnumpy():  
3     if i[0] == -1:  
4         continue  
5     label = ('dog=', 'cat=')[int(i[0])] + str(i[1])  
6     show_bboxes(fig.axes, [nd.array(i[2:]) * bbox_scale], label)
```

实践中，我们可以在执行非极大值抑制前将置信度较低的预测边界框移除，从而减小非极大值抑制的计算量。我们还可以筛选非极大值抑制的输出，例如，只保留其中置信度较高的结果作为最终输出。

小结

- 以每个像素为中心，生成多个大小和宽高比不同的锚框。
- 交并比是两个边界框相交面积与相并面积之比。
- 在训练集中，为每个锚框标注两类标签：一是锚框所含目标的类别；二是真实边界框相对锚框的偏移量。
- 预测时，可以使用非极大值抑制来移除相似的预测边界框，从而令结果简洁。

练习

- 改变 `MultiBoxPrior` 函数中 `sizes` 和 `ratios` 的取值，观察生成的锚框的变化。
- 构造交并比为0.5的两个边界框，观察它们的重合度。
- 按本节定义的为锚框标注偏移量的方法（常数采用默认值），验证偏移量 `labels[0]` 的输出结果。
- 修改“标注训练集的锚框”与“输出预测边界框”两小节中的变量 `anchors`，结果有什么变化？