锚框

- 提出多个被称为锚框的区域
- 预测每个锚框里面是都含有关注的物体
- 如果有，预测这个锚框到真是边缘框的偏移

IoU(交并比)

$$IoU = \frac{A \cap B}{A \cup B}$$

NMS

- 只在预测的时候才会使用NMS，因为在训练的时候会给每个锚框选择一个真实框进行匹配
    - 在训练的时候进行两次预测，一是这个锚框里是不是背景，而是如果有背景和真实框差多少
- 预测的时候会生成很多的预测框，使用NMS去掉冗余框使得输出更干净
- NMS流程：
    - 选出非背景类的最大的框
    - 计算所有其他的框和这个框的IOU，如果大于某个阈值θ则去掉这个框
    - 反复直到没有符合条件的框

In [62]:
```
%matplotlib inline
import torch
from torch import nn
from d2l import torch as d2l
import matplotlib.pyplot as plt

# 精简输出精度
torch.set_printoptions(2)
```

- 我们以图像的每个像素为中心生成不同形状的锚框，缩放比为s，宽高比为r,
- 为了降低计算复杂度在实践中我们只考虑s_1和r_1的组合

In [63]:
```
def multibox_prior(data, sizes, ratios):
    """生成以每个像素为中心具有不同形状的锚框"""
    in_height, in_width = data.shape[-2:]
    device, num_sizes, num_ratios = data.device, len(sizes), len(ratios)
    boxes_per_pixel = (num_sizes + num_ratios - 1)
    size_tensor = torch.tensor(sizes, device=device)
    ratio_tensor = torch.tensor(ratios, device=device)

    # 每个像素宽高为1，把重心放到像素中间
    offset_w, offset_h = 0.5, 0.5
    steps_h = 1.0 / in_height # 进行宽高归一化表示，用于处理不同分辨率的图片
    steps_w = 1.0 / in_width

    center_h = (torch.arange(in_height, device=device) + offset_h) * steps_h
```

```python
    center_w = (torch.arange(in_width, device=device) + offset_w) * steps_w
    # 生成所有像素点的中心网格
    shift_y, shift_x = torch.meshgrid(center_h, center_w, indexing="ij")
    shift_y, shift_x = shift_y.reshape(-1), shift_x.reshape(-1)

    # 生成锚框的高宽
    # [box_per_pixel]
    w = torch.cat((size_tensor * torch.sqrt(ratio_tensor[0]), size_tensor[0] * t
    h = torch.cat((size_tensor / torch.sqrt(ratio_tensor[0]), size_tensor[0] / t

    # 生成半高和半宽
    anchor_manipulations = torch.stack((-w, -h, w, h)).T.repeat(in_height * in_w

    # 每个中心点都有boxes_per_pixel个框
    out_grid = torch.stack([shift_x, shift_y, shift_x, shift_y], 1).repeat_inter
    output = out_grid + anchor_manipulations
    return output.unsqueeze(0)
```

In [64]:
```python
x = torch.tensor([1, 2, 3, 4])
y = torch.tensor([5, 6, 7, 8])
torch.stack((x, y), 0).repeat(1, 2), torch.stack((x, y), 1).repeat(1, 2)
```

Out[64]:
```
(tensor([[1, 2, 3, 4, 1, 2, 3, 4],
         [5, 6, 7, 8, 5, 6, 7, 8]]),
 tensor([[1, 5, 1, 5],
         [2, 6, 2, 6],
         [3, 7, 3, 7],
         [4, 8, 4, 8]]))
```

In [65]:
```python
img = plt.imread('../img/catdog.jpg')
h, w = img.shape[:2]
print(h, w)
X = torch.rand((1, 3, h, w))
Y = multibox_prior(X, sizes=[0.75, 0.5, 0.25], ratios=[1, 2, 0.5])
Y.shape
```

```
561 728
```

Out[65]:
```
torch.Size([1, 2042040, 4])
```

- 整合成以像素点为单位的形式

In [66]:
```python
boxes = Y.reshape(h, w, 5, 4)
print(f"boxes.shape:{boxes.shape}")
boxes[250, 250, :, :], boxes[250, 250, : , :].shape
```

```
boxes.shape:torch.Size([561, 728, 5, 4])
```

Out[66]:
```
(tensor([[ 0.06,  0.07,  0.63,  0.82],
         [ 0.15,  0.20,  0.54,  0.70],
         [ 0.25,  0.32,  0.44,  0.57],
         [-0.06,  0.18,  0.75,  0.71],
         [ 0.14, -0.08,  0.55,  0.98]]),
 torch.Size([5, 4]))
```

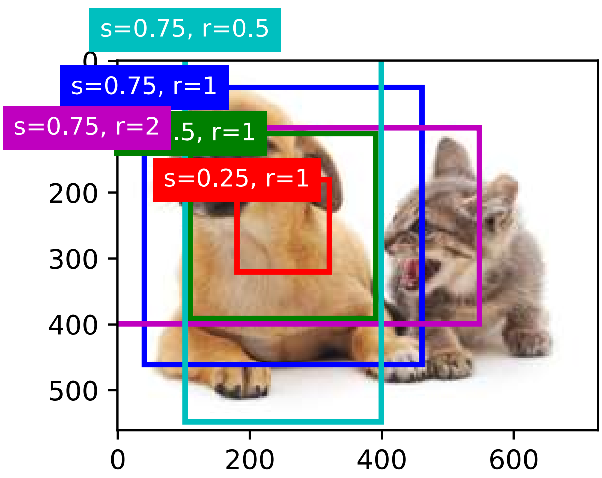
- 定义绘制边界框的函数

In [67]:
```python
def show_bboxes(axes, bboxes, labels=None, colors=None):
    """显示所有边界框"""
```

```
    def _make_list(obj, default_values=None):
        if obj is None:
            obj = default_values
        elif not isinstance(obj, (list, tuple)):
            obj = [obj]
        return obj
    labels = _make_list(labels)
    colors = _make_list(colors, ['b', 'g', 'r', 'm', 'c'])
    for i, bbox in enumerate(bboxes):
        color = colors[i % len(colors)]
        rect = d2l.bbox_to_rect(bbox, color)
        axes.add_patch(rect)
        if labels and len(labels) > i:
            text_color = 'k' if color == 'w' else 'w'
            axes.text(rect.xy[0], rect.xy[1], labels[i], va='center', ha='center
                      fontsize=9, color=text_color, bbox=dict(facecolor=color, l
```

In [68]:

```
d2l.set_figsize()
bbox_scale = torch.tensor((w, h, w, h))
fig = plt.imshow(img)
show_bboxes(fig.axes, boxes[250, 250, :, :] * bbox_scale,
            ['s=0.75, r=1', 's=0.5, r=1', 's=0.25, r=1', 's=0.75, r=2','s=0.75,
```



- 定义IoU函数计算bbox的交并比

In [69]:

```
def box_iou(boxes1, boxes2):
    """计算锚框的交并比"""
    # boxes1[N, 4]
    # boxes2[M, 4]
    # 返回一个[N, M]的tensor，计算boxes1与boxes2的每个框的交并比
    box_area = lambda boxes: ((boxes[:, 2] - boxes[:, 0]) * (boxes[:, 3] - boxes
    # 张量的形状
    # boxes1:[n_boxes1, 4]
    # boxes2:[n_boxes2, 4]
    # area1:[n_boxes1, ] 访问列维度会少一维
    # area2:[n_boxes2, ]
    areas1 = box_area(boxes1)
    areas2 = box_area(boxes2)
    # 计算boxes1与boxes2的每个框的交集的左上角和右下角
    inter_upperleft = torch.max(boxes1[:, None, :2], boxes2[:, :2])
    inter_lowerright = torch.min(boxes1[:, None, 2:], boxes2[:, 2:])
    inters = (inter_lowerright - inter_upperleft).clamp(min=0)
    inter_areas = inters[:, :, 0] * inters[:, :, 1] # [n_boxes1, n_boxes2]
```

```python
    union_areas = areas1[:, None] + areas2 - inter_areas
    return inter_areas / union_areas

test_box1 = boxes[250, 250]
test_box2 = boxes[249, 249]
x = torch.tensor([1, 2, 3, 4])
y = torch.tensor([0, 2, 4, 1])
temp = torch.stack((x, y))
torch.max(x, y)
# y = torch.max(test_box1[:, None, :2], test_box2[:, :2])
torch.max(torch.stack((x, y)), dim=1)
# torch.nonzero(torch.stack((x, y)))
torch.argmax(temp), temp
```

Out[69]:
```
(tensor(3),
 tensor([[1, 2, 3, 4],
         [0, 2, 4, 1]]))
```

- 将真实边界框分配给锚框

In [70]:
```python
def assign_anchor_to_bbox(ground_truth, anchors, device, iou_threshold=0.5):
    """将最接近的真实边界框分配给锚框"""
    # gt:[N, 4]
    # anchors:[M, 4]
    num_anchors, num_gt_boxes = anchors.shape[0], ground_truth.shape[0]
    # 位于第i行第j列的元素是anchor_i对于gt_j的IoU
    jaccard = box_iou(anchors, ground_truth)
    # 对于每个锚框，分配真实边界框， -1表示没有分配到的gt框
    anchors_bbox_map = torch.full((num_anchors, ), -1, dtype=torch.long, device=
    # 根据阈值，决定是否分配边界框
    # 找出jaccard中每行的最大值，并给出索引,并降一个维度，每行中是锚框与所有真实框
    # max_ious:[M, ]anchor与所有gt框产生的最大iou的值
    # indices:[M, ]与anchor产生最大iou的是哪个gt框
    # 先根据阈值决定是否分配边界框
    max_ious, indices = torch.max(jaccard, dim=1)
    anc_i = torch.nonzero(max_ious >= iou_threshold).reshape(-1) # 最大iou满足阈
    box_j = indices[max_ious >= iou_threshold] # 最大iou满足阈值的锚框对应的gt框
    anchors_bbox_map[anc_i] = box_j
    col_discard = torch.full((num_anchors,), -1)
    row_discard = torch.full((num_gt_boxes, ), -1)
    # 然后再进行循环操作
    for _ in range(num_gt_boxes):
        max_idx = torch.argmax(jaccard) # 将jaccard展平，看是第几个
        box_idx = (max_idx % num_gt_boxes).long()
        anc_idx = (max_idx / num_gt_boxes).long()
        anchors_bbox_map[anc_idx] = box_idx
        jaccard[:, box_idx] = col_discard
        jaccard[anc_idx, :] = row_discard
    return anchors_bbox_map
```

- 标记类别和偏移量，锚框为A， 真实框为B

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

其中默认值为$\mu_x=\mu_y=\mu_w=\mu_h=0,\sigma_x=\sigma_y=0.1,\sigma_w=\sigma_h=0.2$

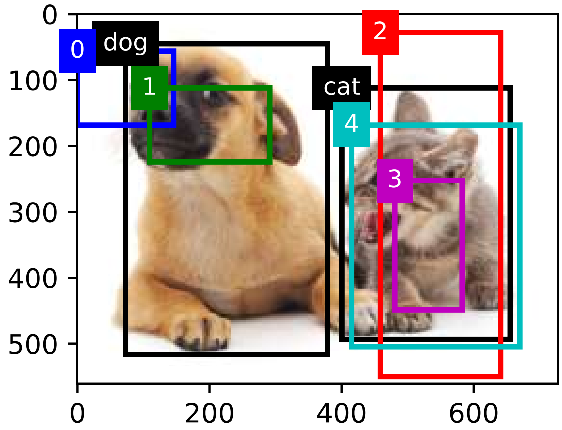
In [71]:
```
def offset_boxes(anchors, assigned_bb, eps=1e-6):
    # xyxy2xywh
    c_anc = d2l.box_corner_to_center(anchors)
    c_assigned_bb = d2l.box_corner_to_center(assigned_bb)
    offset_xy = 10 * (c_assigned_bb[:, :2] - c_anc[:, :2] / c_anc[:, 2:])
    offset_wh = 5 * torch.log(eps + c_assigned_bb[:, 2:] / c_anc[:, 2:])
    offset = torch.cat([offset_xy, offset_wh], dim=1)
    return offset
```

- 使用真实边界框标记锚框

In [72]:
```
def multibox_target(anchors, labels):
    # anchors[1, num_anchors, 4]
    # labels[batch_size, num_gt_boxes, 5]
    batch_size, anchors = labels.shape[0], anchors.squeeze(0) # 移除维度为0的维度
    batch_offset, batch_mask, batch_class_labels = [], [], []
    device, num_anchors = anchors.device, anchors.shape[0]
    for i in range(batch_size):
        label = labels[i, :, :] # [num_gt_boxes, 5]
        anchor_bbox_map = assign_anchor_to_bbox(label[:, 1:], anchors, device)
        bbox_mask = ((anchor_bbox_map >= 0).float().unsqueeze(-1)).repeat(1, 4)
        # 将类标签和分配的边界框坐标初始化为0
        class_labels = torch.zeros(num_anchors, dtype=torch.long,device=device)
        assigned_bb = torch.zeros((num_anchors, 4), device=device, dtype=torch.f
        # 使用真实边界框来标记锚框的类别
        # 如果一个框没有被分配，那么就标记为背景类
        indices_true = torch.nonzero(anchor_bbox_map >= 0)
        bb_idx = anchor_bbox_map[indices_true] # 被分配的真实框
        class_labels[indices_true] = label[bb_idx, 0].long() + 1 # 背景类是0，所
        assigned_bb[indices_true] = label[bb_idx, 1:]
        # 偏移量转换
        offset = offset_boxes(anchors, assigned_bb) * bbox_mask # 计算所有的偏移
        batch_offset.append(offset.reshape(-1))
        batch_mask.append(bbox_mask.reshape(-1))
        batch_class_labels.append(class_labels)
    bbox_offset = torch.stack(batch_offset)
    bbox_mask = torch.stack(batch_mask)
    class_labels = torch.stack(batch_class_labels)
    return (bbox_offset, bbox_mask, class_labels)
```

In [73]:
```
ground_truth = torch.tensor(
    [[0, 0.1, 0.08, 0.52, 0.92],
    [1, 0.55, 0.2, 0.9, 0.88]]
)
anchors = torch.tensor(
    [[0, 0.1, 0.2, 0.3],
     [0.15, 0.2, 0.4, 0.4],
     [0.63, 0.05, 0.88, 0.98],
     [0.66, 0.45, 0.8, 0.8],
     [0.57, 0.3, 0.92, 0.9]]
)

fig = plt.imshow(img)
show_bboxes(fig.axes, ground_truth[:, 1:] * bbox_scale, ['dog', 'cat'], 'k')
show_bboxes(fig.axes, anchors * bbox_scale, ['0', '1', '2', '3', '4'])
```



In [74]:
```
labels = multibox_target(anchors.unsqueeze(0), ground_truth.unsqueeze(0))
labels[2]
```

Out[74]:
```
tensor([[0, 1, 2, 0, 2]])
```

In [75]:
```
labels[1], labels[1].shape
```

Out[75]:
```
(tensor([[0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
1.,
          1., 1.]]),
 torch.Size([1, 20]))
```

In [76]:
```
labels[0], labels[0].shape
```

Out[76]:
```
(tensor([[-0.00e+00, -0.00e+00, -0.00e+00, -0.00e+00, -7.90e+00, -1.00e+01,
           2.59e+00,  7.18e+00, -2.30e+01, -1.38e-01,  1.68e+00, -1.57e+00,
          -0.00e+00, -0.00e+00, -0.00e+00, -0.00e+00, -1.40e+01, -4.60e+00,
           4.17e-06,  6.26e-01]]),
 torch.Size([1, 20]))
```

NMS输出实现

- 输入锚框和偏移量，进行逆变换返回预测的边界框坐标

In [77]:
```
def offset_inverse(anchors, offset_preds):
    """根据带有预测偏移量的锚框来预测边界框"""
    anc = d2l.box_corner_to_center(anchors)
    pred_bbox_xy = (offset_preds[:, :2] * anc[:, 2:] / 10) + anc[:, :2]
    pred_bbox_wh = torch.exp(offset_preds[:, 2:] / 5) * anc[:, 2:]
    pred_bbox = torch.cat((pred_bbox_xy, pred_bbox_wh), dim=1)
    predicted_bbox = d2l.box_center_to_corner(pred_bbox)
    return predicted_bbox

def nms(boxes, scores, iou_threshold):
    """对预测框的置信度进行排序"""
    B = torch.argsort(scores, dim=-1, descending=True) # 对最后一个维度进行降序扌
    keep = [] # 保留预测边界框的指标
    while B.numel() > 0:
        i = B[0]
        keep.append(i)
        if B.numel() == 1: break
        iou = box_iou(boxes[i, :].reshape(-1, 4), # 概率最大的框， 计算概率最大的
                      boxes[B[1:], :].reshape(-1)).reshape(-1)
        inds = torch.nonzero(iou <= iou_threshold).reshape(-1) # 选出iou小于阈值
```

```python
        B = B[inds + 1] # 因为inds的索引是基于B[1:]索引比原来少1，所以要+1
    return torch.tensor(keep, device=boxes.device)
```

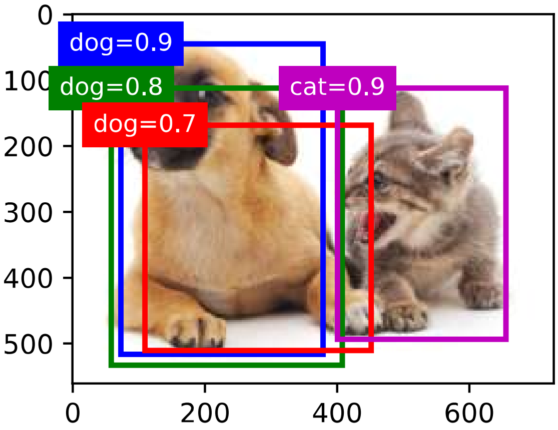
In [78]:
```python
def multibox_detection(cls_probs, offset_preds, anchors, nms_threshold=0.5, pos_
    """使用NMS来预测边界框"""
    # cls_probs, 每个锚框对于类别的预测概率[bs, num_classes + 1, num_anchors]
    device, batch_size = cls_probs.device, cls_probs.shape[0]
    anchors = anchors.squeeze(0)
    num_classes, num_anchors = cls_probs.shape[1], cls_probs.shape[2]
    out = []
    for i in range(batch_size):
        # cls_prob:[num_classes + 1, num_anchors]
        # offset_pred:[num_anchors, 4]
        cls_prob, offset_pred = cls_probs[i], offset_preds[i].reshape(-1, 4)
        # conf:[num_anchors, ] class_id:[num_anchors, ]
        conf, class_id = torch.max(cls_prob[1:], 0) # 算出每一列的最大值，并返回
        preddicted_bb = offset_inverse(anchors, offset_pred)
        keep = nms(preddicted_bb, conf, nms_threshold) # 保留的锚框的索引

        # 找到所有non_keep的索引，并将其种类设置为背景
        all_idx = torch.arange(num_anchors, dtype=torch.long, device=device)
        combined = torch.cat((keep, all_idx))
        # 数组中的唯一元素， 唯一元素出现的次数
        uniques, counts = combined.unique(return_counts=True)
        non_keep = uniques[counts == 1]
        all_id_sorted = torch.cat((keep, non_keep)) # 前面是保留的锚框的索引，后
        class_id[non_keep] = -1
        class_id = class_id[all_id_sorted]
        conf, preddicted_bb = conf[all_id_sorted], preddicted_bb[all_id_sorted]
        # pos_threshold是一个非背景预测的阈值
        below_min_idx = (conf < pos_threshold) # 置信度小于阈值的锚框索引
        class_id[below_min_idx] = -1
        conf[below_min_idx] = 1 - conf[below_min_idx]
        pred_info = torch.cat((class_id.unsqueeze(1), conf.unsqueeze(1), preddic
        out.append(pred_info)
    return out
```

- 一个代码demo，用于展示代码运作

In [79]:
```python
anchors = torch.tensor(
    [[0.1, 0.08, 0.52, 0.92],
     [0.08, 0.2, 0.56, 0.95],
    [0.15, 0.3, 0.62, 0.91],
    [0.55, 0.2, 0.9, 0.88]]
)
offset_preds = torch.tensor([0] * anchors.numel())
cls_probs = torch.tensor(
    [[0] * 4,  # 背景的预测概率
    [0.9, 0.8, 0.7, 0.1],  # 狗的预测概率
    [0.1, 0.2, 0.3, 0.9]] # 猫的预测概率
)
```

In [80]:
```python
fig = plt.imshow(img)
show_bboxes(fig.axes, anchors * bbox_scale, ['dog=0.9', 'dog=0.8', 'dog=0.7', 'c
```



- 调用这个函数，使得输出变得更干净，输出的形状为[batch_size, num_anchors, 6]
- 其中最后6个维度为$[class\ id, \quad conf, \quad xyxy]$

In [86]:
```
output = multibox_detection(cls_probs.unsqueeze(0), offset_preds.unsqueeze(0),
                            anchors.unsqueeze(0), nms_threshold=0.5)
output, output[0].shape
```

Out[86]:
```
([tensor([[ 0.00,  0.90,  0.10,  0.08,  0.52,  0.92],
          [ 1.00,  0.90,  0.55,  0.20,  0.90,  0.88],
          [-1.00,  0.80,  0.08,  0.20,  0.56,  0.95],
          [-1.00,  0.70,  0.15,  0.30,  0.62,  0.91]])],
 torch.Size([4, 6]))
```

- 执行绘图， 删除-1类别的预测框，非-1类别进行绘制

In [89]:
```
fig = plt.imshow(img)

for i in output[0].detach().numpy():
    if i[0] == -1:
        continue
    label = ('dog=', 'cat=')[int(i[0])] + str(i[1])
    show_bboxes(fig.axes, [torch.tensor(i[2:]) * bbox_scale], label)
```