Semantic segmentation

语义分割

如果一张图像中有两只猫和一条狗，语义分割会标记出图像中哪些像素属于猫，哪些像素属于狗，哪些属于背景。

目标检测

如果一张图像中有两只猫和一条狗，目标检测会返回两个猫的边界框及标签、一个狗的边界框及标签，而不会细致到每个像素级别。

区别

粒度：语义分割是逐像素的精细分类，而目标检测只检测物体的边界框

目标：语义分割关注图像中每个像素的类别，而目标检测只关心物体的位置和类别，不关心物体的具体形状或细节。

复杂度：语义分割通常更复杂，因为它需要生成像素级别的输出；而目标检测只需生成一组边界框和分类。

应用场景

语义分割：

自动驾驶（道路、行人、车辆的分割）

医学影像（器官、病灶区域的分割）

遥感图像分析（地形、植被、建筑物等的分割）

自动驾驶：

安全监控（检测和识别人物或物体）

人脸识别（在人群中检测出人脸）

自动驾驶（检测路上的行人、车辆、交通标志）

Code

```python
import os
import torch
import torchvision
import matplotlib.pyplot as plt
from d2l import torch as d2l
from d2l.torch import show_images

# 正常是需要这么下载并解压的，但是我这里下载不了，所以我把数据集下载到了本地，并把路径改成了本地路径
# 下载链接为http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
# d2l.DATA_HUB['voc2012'] = (d2l.DATA_URL + 'VOCtrainval_11-May-2012.tar',
#                            '4e443f8a2eca6b1dac8a6c57641b67dd40621a49')
```

```python
#
# voc_dir = d2l.download_extract('voc2012','VOCdevkit/VOC2012')
voc_dir = 'D:\\data\\voc_dir\\VOCdevkit\\VOC2012'

def read_voc_images(voc_dir, is_train=True):
    txt_fname = os.path.join(voc_dir, 'ImageSets', 'Segmentation',
                             'train.txt' if is_train else 'val.txt')
    mode = torchvision.io.image.ImageReadMode.RGB
    # Split file
    with open(txt_fname, 'r') as f:
        images = f.read().split()
    features, labels = [], []
    for i, fname in enumerate(images):
        features.append(torchvision.io.read_image(os.path.join(voc_dir, 'JPEGImages',
f'{fname}.jpg'))) # Read images
        labels.append(torchvision.io.read_image(os.path.join(voc_dir,
'SegmentationClass', f'{fname}.png'), mode)) # Read labels
    return features, labels

train_features, train_labels = read_voc_images(voc_dir, True)

n = 5
imgs = train_features[0:n] + train_labels[0:n]
imgs = [img.permute(1,2,0) for img in imgs]

# 库函数不生效，所以新自定义了show_image函数，如下行代码可运行则无需定义
# show_images(imgs,2,n) # first | images, second | labels

def show_images(imgs, num_rows, num_cols, scale=1.5):
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()
    for i, (ax, img) in enumerate(zip(axes, imgs)):
        ax.imshow(img)
        ax.axis('off')  # Turn off axis lines
    plt.show()
show_images(imgs, 2, n)  # first | images, second | labels

VOC_COLORMAP = [[0,0,0],[128,0,0],[0,128,0],[128,128,0],
                [0,0,128],[128,0,128],[0,128,128],[128,128,128],
                [64,0,0],[192,0,0],[64,128,0],[192,128,0],
                [64,0,128],[192,0,128],[64,128,128],[192,128,128],
                [0,64,0],[128,64,0],[0,192,0],[128,192,0],
                [0,64,128]]

VOC_CLASSES = ['background','aeroplane','bicycle','bird','boat','bottle','bus',
               'car','cat','chair','cow','diningtable','dog','horse','motorbike',
               'person','potted plant','sheep','sofa','train','tv/monitor']

# Searching class index
# Map RGB TO VOC
def voc_colormap2label():
    # 创建一个全零的张量，大小为256的三次方，因为RGB颜色的每个通道有256种可能的值，所以总共
有256^3种可能的颜色组合。数据类型设为long
    colormap2label = torch.zeros(256**3, dtype=torch.long)
    for i, colormap in enumerate(VOC_COLORMAP):
        # 计算颜色值的一维索引，并将这个索引对应的位置设为i。这样，给定一个颜色值，我们就可
以通过这个映射找到对应的类别索引
```

```python
        colormap2label[(colormap[0] * 256 + colormap[1]) * 256 + colormap[2]] = i
    return colormap2label

# Map VOC TO RGB
def voc_label_indices(colormap, colormap2label):
    # 将输入的colormap的通道维度移到最后一维，并将其转换为numpy数组，然后转换为int32类型。这是因为我们需要使用numpy的高级索引功能
    colormap = colormap.permute(1,2,0).numpy().astype('int32')
    # 计算colormap中每个像素的颜色值对应的一维索引。这里的索引计算方式和上一个函数中的是一致的
    idx = ((colormap[:,:,0] * 256 + colormap[:,:,1]) * 256 + colormap[:,:,2])
    return colormap2label[idx]

# 调用上面定义的两个函数，将训练数据集中的第一个标签图像的RGB颜色值转换为对应的类别索引，并将结果保存在变量y中
y = voc_label_indices(train_labels[0], voc_colormap2label())
# 打印变量y中的一小部分，即第105行到115行，第130列到140列的部分。这里是为了查看转换后的类别索引是否正确
print(y[105:115,130:140])
# 打印VOC_CLASSES列表中的第二个类别名（索引为1）。这里是为了查看第二个类别名是什么
print(VOC_CLASSES[1])

def voc_rand_crop(feature, label, height, width):
    # Crop both feature and label images with the same random crop
    rect = torchvision.transforms.RandomCrop.get_params(feature,(height,width))
    # Get target crop
    feature = torchvision.transforms.functional.crop(feature, *rect)
    # 根据生成的裁剪框，对标签图像进行裁剪。注意，我们是在同一个裁剪框下裁剪特征图像和标签图像，以保证它们对应的位置仍然是对齐的
    label = torchvision.transforms.functional.crop(label,*rect)
    return feature, label
'''
imgs = []
torch不好使，使用matplotlib的show_images函数
for _ in range(n):
    imgs += voc_rand_crop(train_features[0],train_labels[0],200,300)
imgs = [img.permute(1,2,0) for img in imgs]
d2l.show_images(imgs[::2] + imgs[1::2],2,n)
'''
# 使用matplotlib的show_images函数绘制裁剪后的图像
imgs = []
for _ in range(n):
    cropped_feature, cropped_label = voc_rand_crop(train_features[0], train_labels[0], 200, 300)
    imgs.append(cropped_feature)
    imgs.append(cropped_label)
fig, axes = plt.subplots(nrows=2, ncols=n, figsize=(15, 6))
for i, ax in enumerate(axes.flat):
    img = imgs[i].permute(1, 2, 0).numpy()
    ax.imshow(img)
    ax.axis('off')
plt.tight_layout()
plt.show()

# Traditional training approach
class VOCSegDataset(torch.utils.data.Dataset):
    def __init__(self, is_train, crop_size, voc_dir):
```

```
        self.transform = torchvision.transforms.Normalize(mean=
[0.485,0.456,0.406],std=[0.229,0.224,0.225])
        self.crop_size = crop_size
        features, labels = read_voc_images(voc_dir, is_train = is_train)
        self.features = [self.normalize_image(feature) for feature in
self.filter(features)]
        self.labels = self.filter(labels)
        self.colormap2label = voc_colormap2label()
        print('read ' + str(len(self.features)) + ' examples')

    def normalize_image(self, img):
        return self.transform(img.float())

    def filter(self, imgs):
        return [img
                for img in imgs
                if (img.shape[1] >= self.crop_size[0] and img.shape[2] >=
self.crop_size[1] ) ]

    def __getitem__(self, idx):
        # 调用之前定义的voc_rand_crop函数，对指定索引的特征图像和标签图像进行随机裁剪
        feature, label =
voc_rand_crop(self.features[idx],self.labels[idx],*self.crop_size)
        # 调用voc_label_indices函数，将裁剪后的标签图像转换为类别索引，并返回裁剪和转换后
的特征图像和标签图像
        return (feature, voc_label_indices(label,self.colormap2label))

    def __len__(self):
        # 返回特征图像列表的长度，即数据集的长度
        return len(self.features)

crop_size = (320, 480)
voc_train = VOCSegDataset(True, crop_size, voc_dir)
voc_test = VOCSegDataset(False, crop_size, voc_dir)

batch_size = 64
train_iter =
torch.utils.data.DataLoader(voc_train,batch_size,shuffle=True,drop_last=True,
                                          num_workers=0)
for X,Y in train_iter:
    print(X.shape)
    print(Y.shape)
    break

# 整合所有组件
def load_data_voc(batch_size, crop_size):
    voc_dir = 'D:\\data\\voc_dir\\VOCdevkit\\VOC2012'
    num_workers = d2l.get_dataloader_workers()
    train_iter = torch.utils.data.DataLoader(VOCSegDataset(True, crop_size, voc_dir),
batch_size,shuffle=True,
                                              drop_last = True,
num_workers=num_workers)
    test_iter = torch.utils.data.DataLoader(VOCSegDataset(False, crop_size, voc_dir),
batch_size, drop_last=True,
                                             num_workers=num_workers)
    return train_iter, test_iter
```