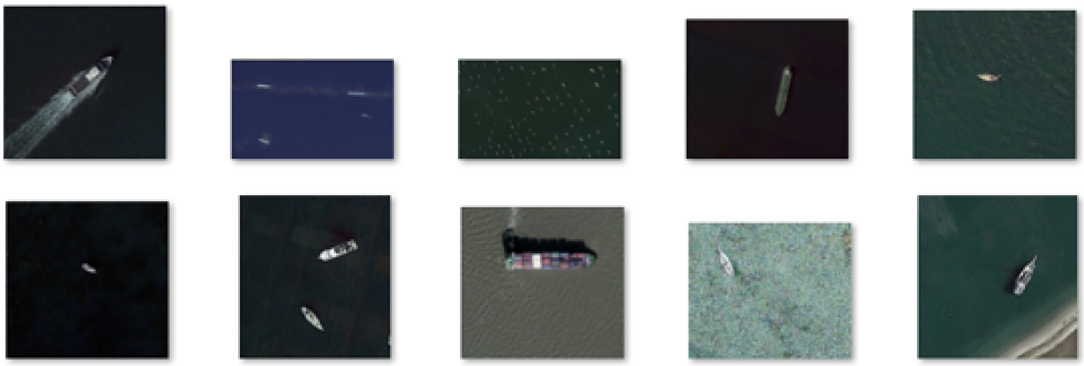
基于Faster R-CNN进行目标检测

运行

更新于 2020-08-26 09:59:10 3 2

目标检测是计算机视觉领域的重要分支，主要目标是从静态图片，或视频序列中检测并定位感兴趣的目标物体，需要确定目标的类别和位置。 本案例中，我们将采用Faster R-CNN模型，使用PyTorch，检测ShipsPascalVOC数据集中，船只在图像中的位置。



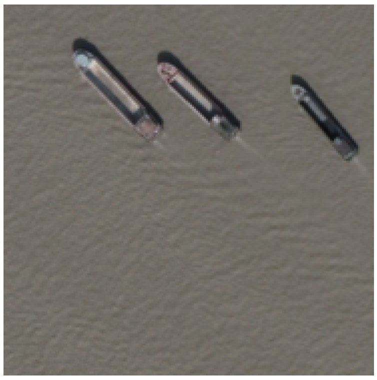
目录

1 数据集简介

本案例采用的ShipsPascalVOC数据集大小为121MB，包含621张 png 格式的图像，每张图像的大小不完全相同，基本在 400×300 像素左右。

图像中包含不同大小、不同颜色、不同形态和不同关照下的船只。数据集中已经对船只的位置用边框进行了标注，并在 `shiplabels.csv` 文件中注明了每个边框的坐标，便于进行目标识别训练。





2 划分数据集

2.1 分析数据集

在读取数据集之前，我们需要加载案例中使用到的文件和库。首先导入 `utils.py` 、 `transforms.py` 等文件，文件中定义了很多需要用到的辅助函数。如 `transform.py` 中定义了水平翻转函数，可以在接下来对训练集进行水平翻转，达到图像增强的效果。

```
# 导入包含辅助函数的python文件
%%bash
git clone https://github.com/pytorch/vision.git
cd vision
git checkout v0.3.0
cp references/detection/utils.py ../

cp references/detection/transforms.py ../

cp references/detection/coco_eval.py ../
cp references/detection/engine.py ../
cp references/detection/coco_utils.py ../
```

```
fatal: destination path 'vision' already exists and is not an empty directory.
HEAD is now at be37608 version check against PyTorch's CUDA version
```

接下来导入需要使用的库。

```
# 导入库
import zipfile
```

```
import zipfile
import numpy as np
import torch
import torch.utils.data
from PIL import Image
import pandas as pd
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from engine import train_one_epoch, evaluate
import utils
import transforms as T
from PIL import Image
import pycocotools
import pandas as pd
import os
import torchvision
```

加载好需要的文件和库后，首先读取ShipsPascalVOC数据集。

```
# 解压zip格式的数据集
zipName = "/content/ShipsPascalVOC.zip"
fileZip = zipfile.ZipFile(zipName)
fileZip.extractall()
fileZip.close()
```

接下来读取 `shiplabels.csv` ，其中包含了图像信息。 下列表格解读了 `shiplabels.csv` 文件中各列的含义。

列名	含义
filename	图像的名称
width	图像的宽，和 `height` 一起表示图像的像素大小
height	图像的高
class	图像中包含需要检测的物体的类别，本案例中只检测船只一种物体，故该列值为"boat"
xmin	边框左下角的横坐标，确定边框的位置
ymin	边框左下角的纵坐标
xmax	边框右上角的横坐标
ymax	边框右上角的纵坐标

```
# 改变当前工作目录到指定的路径
os.chdir("/content")
# 读取边框数据
labels = pd.read_csv("/content/shiplabels.csv")
labels.head()
```

	filename	width	height	class	xmin	ymin	xmax	ymax
0	boat0.png	466	418	boat	282	205	333	273

1	boat1.png	400	362	boat	194	128	266	185
2	boat1.png	400	362	boat	169	170	189	187
3	boat10.png	400	243	boat	236	19	249	32
4	boat10.png	400	243	boat	244	68	253	83

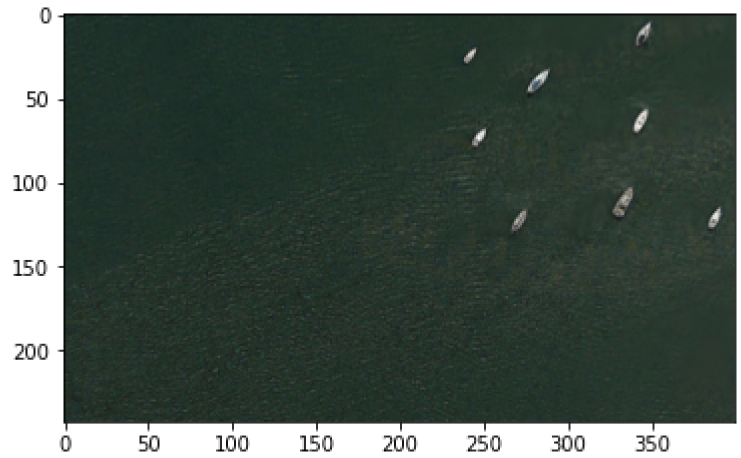
接下来我们通过定义函数，在图像上展示 `shiplabels.csv` 中边框的信息。

```
# 返回图像中物体的边框,每个边框包含四个参数,分别是左下角和右上角的x轴和y轴的坐标
def parse_one_annot(path_to_data_file, filename):
    data = pd.read_csv(path_to_data_file)
    boxes_array = data[data["filename"] == filename][["xmin", "ymin",
    "xmax", "ymax"]].values

    return boxes_array
```

以"boat10.png"为例，我们读取 `shiplabels.csv` 中的数据，画出图像中包含船只的边框。

```
# 画出"boat10.png"图像
img = cv2.imread("/content/images/boat10.png")
img_rgb = img[:, :, ::-1] # 将图像转换为RGB
plt.imshow(img_rgb)
plt.show()
```



读取 `shiplabels.csv` 中"boat10.png"的边框数据，因为上图中有8个白色船只，都是需要检测到的物体。因此返回 8×4 的数组，表示图中8个船只的边框位置。

```
parse_one_annot("/content/shiplabels.csv", "boat10.png")
```

```
array([[236,  19, 249,  32],
       [244,  68, 253,  83],
```

```
       [276,  33, 290,  52],
       [341,   6, 351,  22],
       [340,  58, 350,  73],
       [327, 103, 341, 124],
       [266, 117, 278, 133],
       [384, 115, 394, 131]])
```

如下图，将边框的坐标展现在图像上，可以看到每个需要检测的船只都被红色边框标识出来。

```
image = Image.open("/content/images/boat10.png")
draw = ImageDraw.Draw(image)
coordinates = parse_one_annot("/content/shiplabels.csv", "boat10.png")
for i in coordinates:
    draw.rectangle([(i[0], i[1]), (i[2], i[3])], outline ="red")
image
```

我们再定义一个函数，让图像和 `shiplabels.csv` 中的边框数据一一匹配，返回图像中边框的面积、坐标、边框重合度以及边框的个数。

```
# 将图像和csv文件中边框坐标的数据一一对应
class BoatDataset(torch.utils.data.Dataset):
    def __init__(self, root, data_file, transforms=None):
        self.root = root
        self.transforms = transforms

        # 读取图片名
        self.imgs = sorted(os.listdir(os.path.join(root, "images")))
        self.path_to_data_file = data_file

    # 加载图片数据和对应的边框位置
    def __getitem__(self, idx):
        # 加载图片
        img_path = os.path.join(self.root, "images", self.imgs[idx])
        img = Image.open(img_path).convert("RGB")
```

```
        # 读取边框位置
        box_list = parse_one_annot(self.path_to_data_file,self.imgs[idx])
        # 将边框位置转化为Tensor格式
        boxes = torch.as_tensor(box_list, dtype=torch.float32)

        # 图像中包含的物体的个数
        num_objs = len(box_list)
        # 需要检测的物体只有一个类别：案例中是船只
        labels = torch.ones((num_objs,), dtype=torch.int64)
        image_id = torch.tensor([idx])
        # 每个边框的面积大小
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,0])
        # 假设所有的边框都不是重合度很高的(拥挤的)
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        # 变量target中包含图像中边框的面积area、边框的坐标boxes、图片的名称image_id、边框重合度(拥挤度)iscrowd、边框的个数labels
        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        # 图像变换
        if self.transforms is not None:
            img, target = self.transforms(img, target)
        return img, target

    # 图像的个数
    def __len__(self):
        return len(self.imgs)
```

我们仍以"boat10.png"图像为例，看 `BoatDataset` 的输出结果的形式。

```
# 以上一张图像boat10.png为例，看输出结果
dataset = BoatDataset(root= '/content',data_file= "/content/shiplabels.csv")
dataset.__getitem__(2)
```

```
(<PIL.Image.Image image mode=RGB size=400x243 at 0x7FDE225A2390>,
 {'area': tensor([169., 135., 266., 160., 150., 294., 192., 160.]),
  'boxes': tensor([[236.,  19., 249.,  32.],
          [244.,  68., 253.,  83.],
          [276.,  33., 290.,  52.],
          [341.,   6., 351.,  22.],
          [340.,  58., 350.,  73.],
          [327., 103., 341., 124.],
          [266., 117., 278., 133.],
          [384., 115., 394., 131.]]),
  'image_id': tensor([2]),
  'iscrowd': tensor([0, 0, 0, 0, 0, 0, 0, 0]),
  'labels': tensor([1, 1, 1, 1, 1, 1, 1, 1])})
```

2.2 划分训练集和测试集

接下来，我们随机抽取数据集中的图像，划分数据集。划分比例为，训练集：测试集 $= 7:3$。故测试集有 $621 \times 0.3 = 186$ 张图像，训练集有 $621 - 186 = 435$ 张图像。

在划分数据集之前，我们需要定义一个函数，对图像随机进行水平翻转的处理，已对训练集进行图像增强。图像增强可以增加模型的鲁棒性，有利于模型识别各种场景的图像。同时需要注意的是,不用对测试集进行图像增强。

```
def get_transform(train):
    transforms = []
    # 将图像数据转换为Tensor格式
    transforms.append(T.ToTensor())
    if train:
        # 在训练过程中，对训练集随机抽取数据，进行图像和边框的水平翻转，进行图像增强
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

```
dataset = BoatDataset(root= "/content",
          data_file= "/content/shiplabels.csv",
          transforms = get_transform(train=True))
dataset_test = BoatDataset(root= "/content",
               data_file= "/content/shiplabels.csv",
               transforms = get_transform(train=False))

# 分割训练集和测试集
# 设置随机种子
torch.manual_seed(1)

# 随机抽取435张测试集, 186张测试集
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-186])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-186:])
```

接下来，我们将训练集和测试集转换为 `DataLoader` 格式的数据，方便在模型训练和测试的时候，按照批次(batch)加载数据。

```
# 保存为训练集和测试集
data_loader = torch.utils.data.DataLoader(
              dataset, batch_size=2, shuffle=True, num_workers=4,
              collate_fn=utils.collate_fn)
data_loader_test = torch.utils.data.DataLoader(
         dataset_test, batch_size=1, shuffle=False, num_workers=4,
         collate_fn=utils.collate_fn)
```

```
print("数据集共包含: {} 张图像; 训练集: {} 张; 测试集: {} 张".format(len(indices), len(dataset), len(dataset_test)))
```

数据集共包含：621 张图像；训练集：435 张；测试集：186 张

3 建立Faster R-CNN模型

接下来我们将构建Faster R-CNN模型。

3.1 读取预训练模型

为了节省模型训练时间，并且因为本案例采用的数据集较小，为了得到更好的训练结果，我们通过迁移学习训练模型。下面我们通过定义函数，加载在COCO数据集上训练好的Faster R-CNN模型。

因为在进行迁移学习的时候，如果需要训练的数据集较小，且与原数据集的相似程度不高，我们需要微调Faster R-CNN模型较前面的层。因为前面的层所提取的特征更具有一般性，后面的层则更加具体，更具有原始数据集的特征。

```
def get_model(num_classes):
    # 加载在COCO数据集上进行预训练得到的目标检测模型
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

    # 获取输入分类器的特征的数量
    in_features = model.roi_heads.box_predictor.cls_score.in_features

    # 修改预训练模型较前面的层
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,num_classes)
    return model
```

3.2 构建Faster R-CNN的网络结构

下面打印出了我们采用的Faster R-CNN的网络结构。

```
# 判断是否用GPU训练模型
torch.cuda.is_available()
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# 输出 Faster R-CNN 网络的结构
# 数据集中只包含两类图像：包含船只的图像、不包含船只的图像
num_classes = 2

# 用辅助函数获得预训练模型
```

```
model = get_model(num_classes)

# 用合适的设备训练模型
model.to(device)
```

```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3,
3), bias=False)
      (bn1): FrozenBatchNorm2d(64)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, cei
l_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
            (1): FrozenBatchNorm2d(256)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64)
```

```
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
            (1): FrozenBatchNorm2d(512)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128)
```

```
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
      )
      (layer3): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
            (1): FrozenBatchNorm2d(1024)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
```

```
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=Fa
lse)
            (1): FrozenBatchNorm2d(2048)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512)
```

```
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
        )
      )
    )
    (fpn): FeaturePyramidNetwork(
      (inner_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
        (2): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (layer_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
      (extra_blocks): LastLevelMaxPool()
    )
  )
  (rpn): RegionProposalNetwork(
    (anchor_generator): AnchorGenerator()
    (head): RPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (roi_heads): RoIHeads(
    (box_roi_pool): MultiScaleRoIAlign()
    (box_head): TwoMLPHead(
      (fc6): Linear(in_features=12544, out_features=1024, bias=True)
      (fc7): Linear(in_features=1024, out_features=1024, bias=True)
    )
    (box_predictor): FastRCNNPredictor(
```

```
        (cls_score): Linear(in_features=1024, out_features=2, bias=True)
        (bbox_pred): Linear(in_features=1024, out_features=8, bias=True)
      )
    )
  )
```

同时，我们还需要为模型选择优化算法和学习率调整方式。

这里我们选择随机梯度下降(SGD)算法作为参数优化算法。并将学习率的调整方式设定为：一次epoch代表对所有样本进行一次训练，每进行3次epoch，学习率下降10倍。

```
# 选择SGD优化器算法
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

# 确定调整学习率的方式：每进行3次epoch降低10倍学习率
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

4 训练模型

在定义好上述函数后，我们调用这些函数，在ShipsPascalVOC数据集上进行模型的训练。

```
# 训练5次epoch
num_epochs = 5
for epoch in range(num_epochs):
    # 打印每次训练的结果
    train_one_epoch(model, optimizer, data_loader, device, epoch,
                    print_freq=100)
    # 更新学习率
    lr_scheduler.step()
    # 看模型在测试集上训练的效果
    evaluate(model, data_loader_test, device=device)
```

```
Epoch: [0]  [  0/218]  eta: 0:05:34  lr: 0.000028  loss: 0.0443 (0.0443)  loss
_classifier: 0.0136 (0.0136)  loss_box_reg: 0.0301 (0.0301)  loss_objectness:
0.0001 (0.0001)  loss_rpn_box_reg: 0.0006 (0.0006)  time: 1.5324  data: 0.2911
max mem: 4598
Epoch: [0]  [100/218]  eta: 0:02:38  lr: 0.002330  loss: 0.0461 (0.1145)  loss
_classifier: 0.0149 (0.0295)  loss_box_reg: 0.0287 (0.0625)  loss_objectness:
0.0002 (0.0043)  loss_rpn_box_reg: 0.0007 (0.0182)  time: 1.2581  data: 0.0109
max mem: 4598
Epoch: [0]  [200/218]  eta: 0:00:23  lr: 0.004632  loss: 0.0519 (0.1061)  loss
_classifier: 0.0159 (0.0271)  loss_box_reg: 0.0313 (0.0587)  loss_objectness:
0.0001 (0.0038)  loss_rpn_box_reg: 0.0013 (0.0165)  time: 1.3309  data: 0.0094
max mem: 4598
Epoch: [0]  [217/218]  eta: 0:00:01  lr: 0.005000  loss: 0.0607 (0.1039)  loss
```

```
_classifier: 0.0189 (0.0267)  loss_box_reg: 0.0406 (0.0583)  loss_objectness: 0.0001 (0.0036)  loss_rpn_box_reg: 0.0007 (0.0153)  time: 1.2824  data: 0.0090  max mem: 4598
Epoch: [0] Total time: 0:04:49 (1.3264 s / it)
creating index...
index created!
Test:  [  0/186]  eta: 0:01:59  model_time: 0.3562 (0.3562)  evaluator_time: 0.0027 (0.0027)  time: 0.6401  data: 0.2743  max mem: 4598
Test:  [100/186]  eta: 0:00:27  model_time: 0.2796 (0.3011)  evaluator_time: 0.0013 (0.0049)  time: 0.2983  data: 0.0050  max mem: 4598
Test:  [185/186]  eta: 0:00:00  model_time: 0.2841 (0.3027)  evaluator_time: 0.0013 (0.0050)  time: 0.3152  data: 0.0048  max mem: 4598
Test: Total time: 0:00:58 (0.3157 s / it)
Averaged stats: model_time: 0.2841 (0.3027)  evaluator_time: 0.0013 (0.0050)
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.465
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.837
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.432
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.282
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.702
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.859
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.250
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.390
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.510
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.357
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.753
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.885
Epoch: [1]  [  0/218]  eta: 0:06:04  lr: 0.005000  loss: 0.0282 (0.0282)  loss_classifier: 0.0145 (0.0145)  loss_box_reg: 0.0136 (0.0136)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0002 (0.0002)  time: 1.6724  data: 0.3879  max mem: 4598
Epoch: [1]  [100/218]  eta: 0:02:37  lr: 0.005000  loss: 0.0528 (0.1240)  loss_classifier: 0.0150 (0.0322)  loss_box_reg: 0.0360 (0.0668)  loss_objectness: 0.0001 (0.0040)  loss_rpn_box_reg: 0.0009 (0.0210)  time: 1.3174  data: 0.0096  max mem: 4598
Epoch: [1]  [200/218]  eta: 0:00:23  lr: 0.005000  loss: 0.0608 (0.1183)  loss_classifier: 0.0176 (0.0298)  loss_box_reg: 0.0404 (0.0641)  loss_objectness: 0.0002 (0.0041)  loss_rpn_box_reg: 0.0008 (0.0203)  time: 1.2939  data: 0.0097  max mem: 4598
Epoch: [1]  [217/218]  eta: 0:00:01  lr: 0.005000  loss: 0.0528 (0.1152)  loss_classifier: 0.0143 (0.0294)  loss_box_reg: 0.0313 (0.0626)  loss_objectness:
```

```
_classifier: 0.0113 (0.0261)  loss_box_reg: 0.0318 (0.0528)  loss_objectness:
0.0001 (0.0039)  loss_rpn_box_reg: 0.0011 (0.0193)  time: 1.2486  data: 0.0092
max mem: 4598
Epoch: [1] Total time: 0:04:47 (1.3167 s / it)
creating index...
index created!
Test:  [  0/186]  eta: 0:02:02  model_time: 0.3643 (0.3643)  evaluator_time:
0.0025 (0.0025)  time: 0.6571  data: 0.2886  max mem: 4598
Test:  [100/186]  eta: 0:00:27  model_time: 0.2838 (0.3016)  evaluator_time:
0.0013 (0.0054)  time: 0.3000  data: 0.0050  max mem: 4598
Test:  [185/186]  eta: 0:00:00  model_time: 0.2843 (0.3032)  evaluator_time:
0.0015 (0.0053)  time: 0.3152  data: 0.0050  max mem: 4598
Test: Total time: 0:00:58 (0.3165 s / it)
Averaged stats: model_time: 0.2843 (0.3032)  evaluator_time: 0.0015 (0.0053)
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.49
1
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.85
7
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.49
9
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.31
8
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.71
9
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.82
2
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.25
4
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.40
6
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.54
5
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.40
9
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.77
0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.85
6
Epoch: [2]  [  0/218]  eta: 0:06:37  lr: 0.000500  loss: 0.2067 (0.2067)  loss
_classifier: 0.0668 (0.0668)  loss_box_reg: 0.1217 (0.1217)  loss_objectness:
0.0147 (0.0147)  loss_rpn_box_reg: 0.0035 (0.0035)  time: 1.8238  data: 0.4054
max mem: 4598
Epoch: [2]  [200/218]  eta: 0:00:23  lr: 0.000500  loss: 0.0442 (0.0993)  loss
_classifier: 0.0152 (0.0261)  loss_box_reg: 0.0304 (0.0543)  loss_objectness:
0.0003 (0.0028)  loss_rpn_box_reg: 0.0004 (0.0161)  time: 1.3852  data: 0.0094
max mem: 4598
Epoch: [2]  [217/218]  eta: 0:00:01  lr: 0.000500  loss: 0.0395 (0.0962)  loss
_classifier: 0.0123 (0.0254)  loss_box_reg: 0.0236 (0.0530)  loss_objectness:
0.0001 (0.0026)  loss_rpn_box_reg: 0.0003 (0.0152)  time: 1.2655  data: 0.0091
max mem: 4598
Epoch: [2] Total time: 0:04:48 (1.3251 s / it)
creating index...
index created!
```

```
Test:  [  0/186]  eta: 0:01:52  model_time: 0.3477 (0.3477)  evaluator_time:
0.0026 (0.0026)  time: 0.6025  data: 0.2507  max mem: 4598
Test:  [100/186]  eta: 0:00:27  model_time: 0.2822 (0.3025)  evaluator_time:
0.0013 (0.0051)  time: 0.3010  data: 0.0050  max mem: 4598
Test:  [185/186]  eta: 0:00:00  model_time: 0.2873 (0.3045)  evaluator_time:
0.0014 (0.0052)  time: 0.3192  data: 0.0049  max mem: 4598
Test: Total time: 0:00:59 (0.3179 s / it)
Averaged stats: model_time: 0.2873 (0.3045)  evaluator_time: 0.0014 (0.0052)
Accumulating evaluation results...
DONE (t=0.04s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.50
3
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.85
6
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.50
2
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.32
2
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.74
1
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.86
3
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.26
4
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.41
7
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.55
7
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.41
2
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.79
3
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.89
4
Epoch: [3]  [  0/218]  eta: 0:06:02  lr: 0.000500  loss: 0.0459 (0.0459)  loss
_classifier: 0.0175 (0.0175)  loss_box_reg: 0.0274 (0.0274)  loss_objectness:
0.0001 (0.0001)  loss_rpn_box_reg: 0.0009 (0.0009)  time: 1.6618  data: 0.4468
max mem: 4598
Epoch: [3]  [100/218]  eta: 0:02:38  lr: 0.000500  loss: 0.0347 (0.0886)  loss
_classifier: 0.0125 (0.0235)  loss_box_reg: 0.0225 (0.0466)  loss_objectness:
0.0001 (0.0026)  loss_rpn_box_reg: 0.0003 (0.0159)  time: 1.2835  data: 0.0103
max mem: 4598
Epoch: [3]  [200/218]  eta: 0:00:24  lr: 0.000500  loss: 0.0302 (0.0910)  loss
_classifier: 0.0107 (0.0242)  loss_box_reg: 0.0190 (0.0505)  loss_objectness:
0.0001 (0.0027)  loss_rpn_box_reg: 0.0003 (0.0135)  time: 1.3463  data: 0.0099
max mem: 4598
Epoch: [3]  [217/218]  eta: 0:00:01  lr: 0.000500  loss: 0.0279 (0.0902)  loss
_classifier: 0.0103 (0.0240)  loss_box_reg: 0.0161 (0.0498)  loss_objectness:
0.0001 (0.0026)  loss_rpn_box_reg: 0.0003 (0.0137)  time: 1.2636  data: 0.0104
max mem: 4598
Epoch: [3] Total time: 0:04:50 (1.3336 s / it)
creating index...
index created!
Test:  [  0/186]  eta: 0:01:52  model_time: 0.3774 (0.3774)  evaluator_time:
```

```
0.0026 (0.0026)  time: 0.6055  data: 0.2238  max mem: 4598
Test:  [100/186]  eta: 0:00:27  model_time: 0.2814 (0.3027)  evaluator_time: 
0.0013 (0.0051)  time: 0.3001  data: 0.0049  max mem: 4598
Test:  [185/186]  eta: 0:00:00  model_time: 0.2854 (0.3044)  evaluator_time: 
0.0014 (0.0051)  time: 0.3177  data: 0.0049  max mem: 4598
Test: Total time: 0:00:59 (0.3172 s / it)
Averaged stats: model_time: 0.2854 (0.3044)  evaluator_time: 0.0014 (0.0051)
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.50
8
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.86
6
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.51
2
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.32
7
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.74
0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.86
9
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.26
2
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.41
7
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.56
1
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.41
9
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.79
4
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.89
3
Epoch: [4]  [  0/218]  eta: 0:05:18  lr: 0.000500  loss: 0.0358 (0.0358)  loss
_classifier: 0.0121 (0.0121)  loss_box_reg: 0.0237 (0.0237)  loss_objectness: 
0.0000 (0.0000)  loss_rpn_box_reg: 0.0001 (0.0001)  time: 1.4631  data: 0.2642 
max mem: 4598
Epoch: [4]  [100/218]  eta: 0:02:34  lr: 0.000500  loss: 0.0380 (0.0790)  loss
_classifier: 0.0127 (0.0222)  loss_box_reg: 0.0249 (0.0452)  loss_objectness: 
0.0002 (0.0017)  loss_rpn_box_reg: 0.0004 (0.0099)  time: 1.2797  data: 0.0098 
max mem: 4598
Epoch: [4]  [200/218]  eta: 0:00:23  lr: 0.000500  loss: 0.0368 (0.0926)  loss
_classifier: 0.0121 (0.0243)  loss_box_reg: 0.0241 (0.0510)  loss_objectness: 
0.0001 (0.0027)  loss_rpn_box_reg: 0.0006 (0.0146)  time: 1.3716  data: 0.0104 
max mem: 4598
Epoch: [4]  [217/218]  eta: 0:00:01  lr: 0.000500  loss: 0.0262 (0.0887)  loss
_classifier: 0.0103 (0.0235)  loss_box_reg: 0.0179 (0.0490)  loss_objectness: 
0.0001 (0.0025)  loss_rpn_box_reg: 0.0003 (0.0136)  time: 1.3427  data: 0.0095 
max mem: 4598
Epoch: [4] Total time: 0:04:48 (1.3248 s / it)
creating index...
index created!
Test:  [  0/186]  eta: 0:01:54  model_time: 0.3720 (0.3720)  evaluator_time: 
0.0024 (0.0024)  time: 0.6182  data: 0.2419  max mem: 4598
```

```
Test:  [100/186]  eta: 0:00:27  model_time: 0.2812 (0.3038)  evaluator_time: 0.0013 (0.0051)  time: 0.3015  data: 0.0050  max mem: 4598
Test:  [185/186]  eta: 0:00:00  model_time: 0.2874 (0.3054)  evaluator_time: 0.0014 (0.0052)  time: 0.3189  data: 0.0050  max mem: 4598
Test: Total time: 0:00:59 (0.3186 s / it)
Averaged stats: model_time: 0.2874 (0.3054)  evaluator_time: 0.0014 (0.0052)
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.505
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.859
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.515
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.325
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.735
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.867
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.263
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.414
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.414
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.788
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.893
```

因为我们设定训练每一个epoch的过程中，每50张打印一次训练结果。训练结果包括：

在训练集中：

- `eta` ：每训练50张图像所需要的时间；
- `lr` ：学习率；
- `loss` ：损失函数；
- `loss_classifier` ：分类损失；
- `loss_box_reg` ：预测边框和真实边框的误差；
- `loss_objectness` ：边框包含物体的置信度损失；
- `loss_rpn_box_reg` ：计算RPN的边界损失.

在测试集中：

- `Average Precision` ：平均准确率；
- `IOU` ：预测边框和目标区域的交集除以并集；
- `IoU=0.50` ：重合比例大于0.5的算正例；

- `IoU=0.50:0.95` ：从0.50到0.95每隔0.5取一次作为阈值，计算一次，然后取均值。
- `area` ：区域大小， `small` 为小于 32×32 的区域； `medium` 为 32×32 和 96×96之间的区域； `large` 为大于 96×96 的区域；
- `maxDets` ：最多取目标区域的个数；
- `Average Recall` ：平均召回率。

因此，从输出结果我们可以看到，模型对较大的物体，即 `large` 区域的检测准确率较高，在 `IoU=0.50:0.95` 的情况下达到0.867。同时，在 `IoU=0.50` 的情况下，模型检测各种尺度的物体，达到0.859的平均准确率。

5 评价模型分类效果

接下来我们观察模型的目标检测效果如何。首先读取模型训练后得到的参数。

```
loaded_model = get_model(num_classes = 2)
loaded_model.load_state_dict(model.state_dict())
```

```
<All keys matched successfully>
```

下图展示了模型在随机抽取的一张图像上，对船只进行目标检测的结果。

```
idx = 2
img, _ = dataset_test[idx]
label_boxes = np.array(dataset_test[idx][1]["boxes"])
# 调用训练好的模型的参数，观测其检测物体的效果
loaded_model.eval()
with torch.no_grad():
    prediction = loaded_model([img])
image = Image.fromarray(img.mul(255).permute(1, 2,0).byte().numpy())
draw = ImageDraw.Draw(image)

# 画出真实边框和预测边框
for elem in range(len(label_boxes)):
    draw.rectangle([(label_boxes[elem][0], label_boxes[elem][1]),
    (label_boxes[elem][2], label_boxes[elem][3])],
    outline ="green", width =3)
for element in range(len(prediction[0]["boxes"])):
    boxes = prediction[0]["boxes"][element].cpu().numpy()
    score = np.round(prediction[0]["scores"][element].cpu().numpy(),
                     decimals= 4)
    if score > 0.8:
        draw.rectangle([(boxes[0], boxes[1]), (boxes[2], boxes[3])],
        outline ="red", width =3)
        draw.text((boxes[0], boxes[1]), text = str(score))
image
```

其中绿色的边框是真实的包含船只的边框，即 `shiplabels.csv` 中该图像对应的数据。红色的边框是由模型预测得到的。红色边框左上角的数字 0.9985 表示真实边框和预测边框的交并比。由于二者的交并比已经达到 0.9985，说明模型在这一张图像上的预测结果很好。

6 总结

本案例介绍了如何使用Faster R-CNN模型对ShipsPascalVOC数据集进行目标检测。虽然数据集图像较少，但通过迁移学习和微调的方法，我们仍获得了较好的精确率。并且通过随机抽取图像进行目标检测，看到模型的预测边框与真实边框重合度较高，检测结果较好。

发表您的讨论

竞赛平台 (http://cookdata.cn/competition/)
关于我们 (/footer?sub=0)
产品介绍 (/footer?sub=0)
服务条款 (/footer?sub=1)
帮助中心 (/footer?sub=2)
联系我们 (/footer?sub=3)