import os  
import datetime  
  
import torch  
import torchvision  
import scipy.io  
from torchvision.models.detection.anchor\_utils import AnchorGenerator  
  
import transforms  
from network\_files import FasterRCNN, AnchorsGenerator  
from backbone import MobileNetV2, vgg  
from my\_dataset import VOCDataSet  
from train\_utils import GroupedBatchSampler, create\_aspect\_ratio\_groups  
from train\_utils import train\_eval\_utils as utils  
  
  
def create\_model(num\_classes):  
 # https://download.pytorch.org/models/vgg16-397923af.pth  
 # 如果使用vgg16的话就下载对应预训练权重并取消下面注释，接着把mobilenetv2模型对应的两行代码注释掉  
 # vgg\_feature = vgg(model\_name="vgg16", weights\_path="./backbone/vgg16.pth").features  
 # backbone = torch.nn.Sequential(\*list(vgg\_feature.\_modules.values())[:-1]) # 删除features中最后一个Maxpool层  
 # backbone.out\_channels = 512  
  
  
 # # https://download.pytorch.org/models/mobilenet\_v2-b0353104.pth  
 # backbone = MobileNetV2(weights\_path="./backbone/mobile-model-24.pth").features  
 # backbone.out\_channels = 1280 # 设置对应backbone输出特征矩阵的channels  
 #  
 # anchor\_generator = AnchorsGenerator(sizes=((32, 64, 128, 256, 512),),  
 # aspect\_ratios=((0.5, 1.0, 2.0),))  
 #  
 # roi\_pooler = torchvision.ops.MultiScaleRoIAlign(featmap\_names=['0'], # 在哪些特征层上进行roi pooling  
 # output\_size=[7, 7], # roi\_pooling输出特征矩阵尺寸  
 # sampling\_ratio=2) # 采样率  
 #  
 # model = FasterRCNN(backbone=backbone,  
 # num\_classes=num\_classes,  
 # rpn\_anchor\_generator=anchor\_generator,  
 # box\_roi\_pool=roi\_pooler)  
  
 # 创建 MobileNetV2 模型，并仅使用 features 部分  
 custom\_model = MobileNetV2()  
 # custom\_model.load\_state\_dict(torch.load("./backbone/mobilenet\_v2.pth"), False)  
 custom\_model.load\_state\_dict(torch.load(r"./save\_weights\_tr/defect\_1.pth"), False)  
 backbone = custom\_model.features  
  
  
 # 创建一个模型，只包含 features 部分，以获取输出通道数  
 dummy\_input = torch.randn(1, 3, 224, 224) # 假设输入大小为 (1, 3, 224, 224)  
 dummy\_output, out\_channels = None, None  
  
 # 遍历 features，找到最后一层的输出通道数  
 for layer in backbone:  
 dummy\_input = layer(dummy\_input)  
 if isinstance(dummy\_input, tuple):  
 dummy\_output, \_ = dummy\_input  
 else:  
 dummy\_output = dummy\_input  
  
 # 获取输出通道数  
 out\_channels = dummy\_output.size(1)  
  
 # 设置对应 backbone 输出特征矩阵的 channels  
 backbone.out\_channels = out\_channels  
  
 # 其他模型组件的设置  
 anchor\_generator = AnchorsGenerator(sizes=((32, 64, 128, 256, 512),),  
 aspect\_ratios=((0.5, 1.0, 2.0),))  
  
 roi\_pooler = torchvision.ops.MultiScaleRoIAlign(featmap\_names=['0'],  
 output\_size=[7, 7],  
 sampling\_ratio=2)  
  
 # 创建 FasterRCNN 模型  
 model = FasterRCNN(backbone=backbone,  
 num\_classes=num\_classes,  
 rpn\_anchor\_generator=anchor\_generator,  
 box\_roi\_pool=roi\_pooler)  
 return model  
  
  
def main():  
 device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  
 print("Using {} device training.".format(device.type))  
  
 # 用来保存coco\_info的文件  
 results\_file = "results{}.txt".format(datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))  
  
 # 检查保存权重文件夹是否存在，不存在则创建  
 if not os.path.exists("save\_weights"):  
 os.makedirs("save\_weights")  
  
 data\_transform = {  
 "train": transforms.Compose([transforms.ToTensor(),  
 transforms.RandomHorizontalFlip(0.5)]),  
 "val": transforms.Compose([transforms.ToTensor()])  
 }  
  
 VOC\_root = "./" # VOCdevkit  
 aspect\_ratio\_group\_factor = 3  
 batch\_size = 4  
 amp = False # 是否使用混合精度训练，需要GPU支持  
  
 # # check voc root  
 # if os.path.exists(os.path.join(VOC\_root, "VOCdevkit")) is False:  
 # raise FileNotFoundError("VOCdevkit dose not in path:'{}'.".format(VOC\_root))  
  
 # load train data set  
 # VOCdevkit -> VOC2012 -> ImageSets -> Main -> train.txt  
 train\_dataset = VOCDataSet(VOC\_root, "2012", data\_transform["train"], "train.txt")  
 train\_sampler = None  
  
 # 是否按图片相似高宽比采样图片组成batch  
 # 使用的话能够减小训练时所需GPU显存，默认使用  
 if aspect\_ratio\_group\_factor >= 0:  
 train\_sampler = torch.utils.data.RandomSampler(train\_dataset)  
 # 统计所有图像高宽比例在bins区间中的位置索引  
 group\_ids = create\_aspect\_ratio\_groups(train\_dataset, k=aspect\_ratio\_group\_factor)  
 # 每个batch图片从同一高宽比例区间中取  
 train\_batch\_sampler = GroupedBatchSampler(train\_sampler, group\_ids, batch\_size)  
  
 nw = min([os.cpu\_count(), batch\_size if batch\_size > 1 else 0, 8]) # number of workers  
 print('Using %g dataloader workers' % nw)  
  
 # 注意这里的collate\_fn是自定义的，因为读取的数据包括image和targets，不能直接使用默认的方法合成batch  
 if train\_sampler:  
 # 如果按照图片高宽比采样图片，dataloader中需要使用batch\_sampler  
 train\_data\_loader = torch.utils.data.DataLoader(train\_dataset,  
 batch\_sampler=train\_batch\_sampler,  
 pin\_memory=True,  
 num\_workers=nw,  
 collate\_fn=train\_dataset.collate\_fn)  
 else:  
 train\_data\_loader = torch.utils.data.DataLoader(train\_dataset,  
 batch\_size=batch\_size,  
 shuffle=True,  
 pin\_memory=True,  
 num\_workers=nw,  
 collate\_fn=train\_dataset.collate\_fn)  
  
 # load validation data set  
 # VOCdevkit -> VOC2012 -> ImageSets -> Main -> val.txt  
 val\_dataset = VOCDataSet(VOC\_root, "2012", data\_transform["val"], "val.txt")  
 val\_data\_loader = torch.utils.data.DataLoader(val\_dataset,  
 batch\_size=1,  
 shuffle=False,  
 pin\_memory=True,  
 num\_workers=nw,  
 collate\_fn=val\_dataset.collate\_fn)  
  
 # create model num\_classes equal background + 20 classes  
 model = create\_model(num\_classes=2)  
 # print(model)  
  
 model.to(device)  
  
 scaler = torch.cuda.amp.GradScaler() if amp else None  
  
 train\_loss = []  
 learning\_rate = []  
 val\_map = []  
  
 # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #  
 # first frozen backbone and train 5 epochs #  
 # 首先冻结前置特征提取网络权重（backbone），训练rpn以及最终预测网络部分 #  
 # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #  
 for param in model.backbone.parameters():  
 param.requires\_grad = False  
  
 # define optimizer  
 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)  
 init\_epochs = 5  
 for epoch in range(init\_epochs):  
 # train for one epoch, printing every 10 iterations  
 mean\_loss, lr = utils.train\_one\_epoch(model, optimizer, train\_data\_loader,  
 device, epoch, print\_freq=50,  
 warmup=True, scaler=scaler)  
 train\_loss.append(mean\_loss.item())  
 learning\_rate.append(lr)  
 # evaluate on the test dataset  
 coco\_info = utils.evaluate(model, val\_data\_loader, device=device)  
  
 # write into txt  
 with open(results\_file, "a") as f:  
 # 写入的数据包括coco指标还有loss和learning rate  
 result\_info = [f"{i:.4f}" for i in coco\_info + [mean\_loss.item()]] + [f"{lr:.6f}"]  
 txt = "epoch:{} {}".format(epoch, ' '.join(result\_info))  
 f.write(txt + "\n")  
  
 val\_map.append(coco\_info[1]) # pascal mAP  
  
 torch.save(model.state\_dict(), "./save\_weights/pretrain.pth")  
  
 # # # # # # # # # # # # # # # # # # # # # # # # # # # #  
 # second unfrozen backbone and train all network #  
 # 解冻前置特征提取网络权重（backbone），接着训练整个网络权重 #  
 # # # # # # # # # # # # # # # # # # # # # # # # # # # #  
  
 # 冻结backbone部分底层权重  
 for name, parameter in model.backbone.named\_parameters():  
 split\_name = name.split(".")[0]  
 if split\_name in ["0", "1", "2", "3"]:  
 parameter.requires\_grad = False  
 else:  
 parameter.requires\_grad = True  
  
 # define optimizer  
 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)  
 # learning rate scheduler  
 lr\_scheduler = torch.optim.lr\_scheduler.StepLR(optimizer,  
 step\_size=3,  
 gamma=0.33)  
 num\_epochs = 20  
 for epoch in range(init\_epochs, num\_epochs+init\_epochs, 1):  
 # train for one epoch, printing every 50 iterations  
 mean\_loss, lr = utils.train\_one\_epoch(model, optimizer, train\_data\_loader,  
 device, epoch, print\_freq=50,  
 warmup=True, scaler=scaler)  
 train\_loss.append(mean\_loss.item())  
 learning\_rate.append(lr)  
  
 # update the learning rate  
 lr\_scheduler.step()  
  
 # evaluate on the test dataset  
 coco\_info = utils.evaluate(model, val\_data\_loader, device=device)  
  
 # write into txt  
 with open(results\_file, "a") as f:  
 # 写入的数据包括coco指标还有loss和learning rate  
 result\_info = [f"{i:.4f}" for i in coco\_info + [mean\_loss.item()]] + [f"{lr:.6f}"]  
 txt = "epoch:{} {}".format(epoch, ' '.join(result\_info))  
 f.write(txt + "\n")  
  
 val\_map.append(coco\_info[1]) # pascal mAP  
  
 # save weights  
 # 仅保存最后5个epoch的权重  
 if epoch in range(num\_epochs+init\_epochs)[-1:]:  
 save\_files = {  
 'model': model.state\_dict(),  
 'optimizer': optimizer.state\_dict(),  
 'lr\_scheduler': lr\_scheduler.state\_dict(),  
 'epoch': epoch}  
 torch.save(save\_files, r"E:/transfer learning/myself/faster\_rcnn/save\_weights\_tr/mobile-model-{}.pth".format(epoch))  
  
 # plot loss and lr curve  
 if len(train\_loss) != 0 and len(learning\_rate) != 0:  
 from plot\_curve import plot\_loss\_and\_lr  
 plot\_loss\_and\_lr(train\_loss, learning\_rate)  
  
 # plot mAP curve  
 if len(val\_map) != 0:  
 from plot\_curve import plot\_map  
 plot\_map(val\_map)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()