文章编号：2095-2163(2023)08-0045-08 中图分类号：TP391 文献标志码：A

一种基于目标检测的无人零售商品识别算法

王军祥
(福建船政交通职业学院 信息与智慧交通学院，福州 350007)

摘 要：随着近几年深度学习技术飞速发展，深度卷积神经网络在图像分类等任务的水准已经高于人类的水平，这为无人零售带来了新的可能。本文通过目标检测数据集制作、基于翻转的目标检测数据扩充，基于开源目标检测框架 MMDetection 构建了一种基于深度神经网络的无人零售商品定位、识别模型，并最终实现了一个基于深度学习目标检测算法的无人零售商品识别系统。文末，依据目标检测常用性能衡量指标，采用目标检测常用度量方法，对本系统训练好的模型进行验证，结果表明，系统对商品识别速度快，整体性能良好。

关键词：深度学习；图像处理；目标检测；数据扩充

An algorithm for unmanned retail product recognition based on object detection

WANG Junxiang
(School of Information and Intelligent Transportation, Fujian Chuanzheng Communications College, Fuzhou 350007, China)

【Abstract】 With the rapid development of deep learning technology in recent years, the level of deep convolution neural network in image classification and other tasks has been higher than that of human beings, which brings new possibilities for unmanned retail. Through the production of target detection data set, the expansion of target detection data based on flipping, and based on the open source target detection framework MMDetection, this paper constructs an unmanned retail commodity location and recognition model based on deep neural network, and realizes an unmanned retail commodity recognition system based on deep learning target detection algorithm. Therefore, according to the commonly used performance indicators of target detection, the commonly used measurement methods of target detection are used to verify the trained model of the system. The results show that the commodity recognition speed is fast and the overall performance is good.

【Key words】 deep learning; image processing; target detection; data expansion

0 引 言

各类新零售模式及智能服务平台的不断发展，要求系统能精准识别出商品类型以完成销售服务，而商品信息识别技术是这项功能得以实现的重要基础[1]。传统自动贩售货柜成本较高，使用称重仪的自动贩售柜则不支持同时贩售重量相同、但价格不同的商品。无人零售作为人工智能技术在零售业的典型应用场景，因其具备无人化、低成本、智能化的应用优势，在日常生活中深受欢迎。利用人工智能技术优势，结合国内全球领先的移动支付态势，无人零售作为新零售的实践样本得到了极大关注[2]。

本文实现了一个基于深度学习目标检测算法的零售商品定位、识别模型。首先采集商品图片数据，使用开源的目标检测标注工具进行数据标注，介绍了一种快速清洗数据的方法，讨论了目标检测数据集扩充方法，提出了 COCO 数据集格式以及从 Labelme 格式转换成 COCO 格式的方法。分析了训练过程中的日志，并在介绍目标检测常用性能指标后对训练结果进行了分析。

1 相关工具综述

1.1 深度学习框架 Pytorch

Pytorch 是一个由 Facebook 人工智能研究院研发的开源机器学习框架，提供了既可以存在于 CPU 上、也可以存在于 GPU 上的张量，可以极大地加快计算速度。Pytorch 基于反向模式自动微分技术，使得用户可以构建动态的神经网络模型。用户可以像使用

基金项目：福建省中青年课题(科技类)(JAT210704)。

作者简介：王军祥(1975-)，男，教授，主要研究方向：软件技术、人工智能与大数据技术应用。

收稿日期：2023-02-09

Numpy 一样自然地使用 Pytorch。与 Torch 或其他一些替代方案相比,PyTorch 中的内存使用非常高效。

1.2 目标检测框架 MMDetection

MMDetection[3] 是一个基于 PyTorch 的开源目标检测工具箱,具有以下特点:

(1)模块化设计。开发团队将目标检测框架解构成几个不同的部分,用户可以通过组合不同的模块来搭建自定义的目标检测模型。

(2)热门模型开箱即用。工具箱支持直接使用各种热门的、当下流行的目标检测模型,比如 Faster RCNN、Mask RCNN、RetinaNet 等。

(3)高性能。所有基本的外接框和掩膜操作都能在 GPU 上执行,训练速度比其它代码库快,或者至少是可比较的、业内最先进的技术水平。

2 总体设计

监督学习是指通过让机器学习大量带有标签的样本数据,训练出一个模型,使该模型可以根据输入得到相应输出结果的过程。本文任务是训练一个能够定位、识别零售商品的神经网络。首要任务是采集足够多的图片,并对这些图片进行标注。因为这是一个目标检测任务,所以给图片中的每一个商品标注位置、大小和类别。本文使用开源目标检测框架 MMDetection 搭建 Faster RCNN 网络。最后,为了判断哪一阶段的模型精度最高,本文衡量每一阶段的模型在验证集上的表现情况。

3 详细设计

3.1 采集环境

为了模拟零售场景,本文准备 12 个小件物品作为测试用商品,具体类别有:阿尔卑斯奶糖、可口可乐、芬达、雪碧、德芙巧克力、南孚电池、读卡器、西梅等。用一张灰色的桌子作为结算台,在桌子边缘放置一个相机支架,将摄像头固定在上面,调整相机位置到距离桌面 40 cm 的高度。在桌子边缘放置一个台灯,将台灯色温值调整至6 000 K的正白光,以保证拍摄环境不会偏黄或偏蓝。

3.2 采集方法

自己编写 Python 脚本,自动采集摄像头所拍摄的视频帧。用 USB 连接摄像头与电脑,将 cv2.VideoCapture 的参数设置为 1,表示使用操作系统能辨别的第二个摄像头。每隔 2 000 ms 读取一次图片,若读取成功,则在电脑屏幕上显示一个窗口,用于预览将要保存的图片,并将图片以 JPEG 的格式保存在“images”文件夹中,图片文件名为 4 位数字表示的图片 id,从“0000”开始编号,“0001”为第二张图片,以此递增。

脚本运行过程中,随机从 12 个商品中抽取 1~3 个摆放到桌面上,预览窗口中的图片。每刷新一次就用手挪动一次商品的位置,经过 90 min 的采集,共采集到3 000张图片。

3.3 数据集清洗

在采集过程中难免出现一些特殊情况,比如:

(1)调整商品位置时拍摄到了手。

(2)商品的一部分超出了图片边界。

(3)未及时移动商品导致拍摄了 2 张商品布局完全相同的图片。

在情况(1)中,手可能会遮挡商品,这会对识别产生干扰。

在情况(2)中,2 张布局相同的图片的标注是相同的,可以在数据扩充阶段自动生成,不必由标注员浪费时间手动标注 2 次。

在情况(3)中,虽然在大部分情况下,人眼依然可以从图片中辨认出该商品的所属类别,但商品超出图片边界多少才算不可辨认,很难有明确的标准,且人眼无法直接看出一个商品超出图片边界的具体的比值。

综上所述,在采集完成后需要删除包含有特殊情况的图片,以达到减少标注员工作量、加快标注进度的目标。清洗完成后剩余1 900张有效图片。

3.4 数据集标注

3.4.1 标注的意义

为了进行监督学习,需要许多训练示例,每个示例都由一个输入和一个期望的输出组成,在目标检测任务中,输入的是图片,输出的是每个目标的位置,现在需要人工标注图片中每个物体的位置。标注要求用矩形框将图片内的各个商品标注出来,标注框要求尽量与标注目标最小外接矩形贴合,每个标注框还应携带商品类别信息(商品名)。

3.4.2 标注信息的结构

标注完成后,标注信息会以 json 格式保存在“anno”文件夹中,每一张图片对应一个同名的 json 文件,其文件结构如下:

```
{   "version": "4.5.6",
"flags": {},
"imagePath": "../0000.jpg",
"imageData": null,
"imageHeight": 480,
```

"imageWidth"：640，

"shapes"：[ ]　}

version 是 labelme 的版本号，将来更新标注结构时可以根据版本号的不同选择不同的解析算法，以达到向后兼容的目的。flags 是图片层面的属性，一般用于图片分类任务。imagePath 表示图片对于标注文件的相对路径。imageData 保存着用 base64 编码的图片，但由于在启动 labelme 时加上了"—nodata"参数，所以没有在标签中额外保存图片数据。imageHeight 和 imageWidth 保存着图片的高宽信息。shapes 是标注框列表，一张图里可能有多个标注框，所以用列表来存储，列表中的每一个元素结构如下：

{　"label"："可口可乐"，

"points"：[[184，20]，[328，198]]，

"shape_type"："rectangle"，

"group_id"：null，

"flags"：{ }　}

label 是标注框对应的类别名。points 是一个列表，其数据表示标注的具体位置，要查看 shape_type 的值才能确定用哪种算法解析 points 里的数据。在目标检测任务中，shape_type 一般选择"rectangle"，表示 points 里的数据是一个矩形框 2 个对角顶点的坐标。要注意的是，2 个对角的顶点顺序不是固定的，有可能是(左上，右下)、(右上，左下)，(右下，左上)，(左下，右上)这 4 种可能的组合中的其中一种。group_id 一般用于实例分割，若 2 个标注的 group_id 的值为整数且相等，则意味着这 2 个标注实际上是同一个实例的不同部分。flags 是标注层面的属性，一般用于多属性分类任务。

3.5　标注清洗

3.5.1　清洗方法

一种快速筛查错误标注的方法是将所有标注从原图中裁剪下来，下面简称"裁剪下来的图"为"裁剪图"，给所有类别建立一个文件夹，将相同类别的裁剪图放到同一个文件夹中，并用"原图 id.标注 id.jpg"的格式命名图片，再进入每一个文件夹观察。同一文件夹中都是同类的裁剪图，若出现不同类的裁剪图则大概率是一个错误的标注，查看这个裁剪图的文件名里原图 id 的部分，就可以快速定位标注错误的图片是哪一张，定位到错误的标注后手动更改为正确的类别即可。一轮清洗结束后，需要根据修改后的标注重新进行一次上述流程，直到没有发现新的错误为止。

3.5.2　实现细节

人工智能数据集中的标注有成百上千个，人工手动裁剪图片的效率慢、耗时长。本文编写一段程序，根据标签文件的内容自动完成裁剪操作。下面分析"自动裁剪程序"的实现细节。

本文为 cv2.imwrite 写一个包装函数 imwrite，使用".jpg"编码以节省磁盘空间：

```
def imwrite(img: np.ndarray, path):
    flag, arr = cv2.imencode('.jpg', img)
    arr.tofile(path)
```

设置路径信息，anno_dir 是保存标注文件的文件夹，roi_dir 是保存裁剪图的文件夹。

```
from pathlib import Path
anno_dir = Path('images/anno')
roi_dir = Path('annotation_check')
```

遍历标注文件，用 json 解析文件的内容，读取标注文件对应的图片。遍历每一个标注文件的标注信息，取出标注信息的 points，将这些数字转为整型，并按照坐标系大小排序，以确定左上角 xy 和右下角 xy 的值。使用 numpy 数组的切片功能将标注区域裁剪下来，保存到对应的文件夹中。

3.6　数据扩充

深度学习模型需要经过大量数据训练才能做到较好的泛化，训练数据量不足很可能导致过拟合。通常的解决方法是增加训练数据，但采集、标注数据需要消耗大量人力、财力。本文利用图片翻转特性快速生成很多新的图片，并同步更新标注信息，使之与生成的图片对应。

图片翻转可分为水平翻转、垂直翻转和转置翻转，如图 1 所示。翻转后图片中的目标位置改变了，但其类别不变。通过翻转图片来扩充训练数据，让模型达到较好的泛化，提升训练精度。

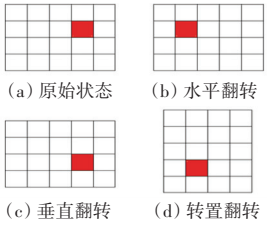

图 1　3 种翻转示例

Fig. 1　Three flipping examples

首先构建一个图片类，可表示不同的翻转状态，为了能用上 Python 的集合类来快速查找已经到达过的状态，将其设计成可哈希的。使用 numpy 数组作为图片容器，使用一个二维列表作为初始化参数，

表示图片的状态。

Image.__hash__假设内部数组是[[1, 2], [3, 4]],先将数组展平,使数组从二维状态变成一维状态的[1, 2, 3, 4],再使用 numpy.logspace 函数生成一个形如[1, 10, 100, 1000],首项为 1,公比为 10 的等比数列,再计算 2 个数组的点积,即可得到哈希值 4321。当内部数组的值域为 0~9 时,通过上述算法所计算出的哈希值是唯一的。

接下来定义 3 个翻转函数,都接收一个 Image 对象作为参数,返回一个经过翻转的新的 Image 对象,使用负步长切片来颠倒某个轴上的数据。函数内容具体如下:

```
def horizontal_flip(img: Image):
    return Image(img.arr[:, ::-1])
def vertical_filp(img: Image):
    return Image(img.arr[::-1])
def diagonal_flip(img: Image):
    return
    Image(img.arr.transpose((1,0)))
```

使用深度优先搜索算法来搜索不同的翻转组合顺序,并记录每一种状态对应的翻转顺序、翻转次数。用集合来记录已访问的图片哈希,以图片哈希为键,以翻转组合顺序、翻转次数为值,使用字典来代替集合。如果遇到从未到达过的状态或是能花费更少的翻转次数到达相同的状态,则更新簿记表;如果遇到已经到达过的状态,并且花费的翻转次数和当前最少翻转次数一样多,则将新的路径添加到当前状态对应的路径列表中。如遇到已到达过的状态,并且花费的翻转次数比当前最少翻转次数还要多,则不更新,并终止递归。

最后,创建初始化状态,启动搜索,搜索完成后,输出每种状态表示的图片、以及所对应的翻转组合。

观察输出结果可以得出结论:应用水平翻转、垂直翻转和转置翻转以及相应的叠加组合可以创造出 8 种不同的图片(如图 2 所示),即通过图片翻转可以扩充出的数据量相当于原先的 8 倍。表 1 是图 1 中所有状态对应的最短路径(注:最短路径可能不止一条)。

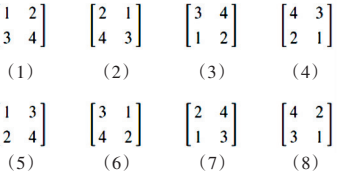

图 2 由 3 种翻转组合出的 8 种情况

Fig. 2 8 scenarios composed of 3 flipping combinations

表 1 训练日志术语解释

Tab. 1 Explanation of training log terms

键名	描述
lr	学习率
eta	预估完成时间
time	5 次迭代中,每次迭代的平均耗时
data_time	5 次迭代中,每次迭代加载数据的平均耗时
memory	最大占用显存率,单位 MB
loss_rpn_cls	Region Proposal Network 关于物体置信度的损失值
loss_rpc_bbox	Region Proposal Network 关于定位框的损失值
loss_cls	ROI Head 关于分类的损失值
loss_bbox	ROI Head 关于定位框的损失值
acc	ROI Head 的定位准确率(参考价值较低)
loss	上述所有损失值的总和

关于标注框更新,以图 1 中水平翻转为例,观察红色色块坐标在水平翻转前后的变化,可看出,原始状态中红色色块距离图片左侧边界有 1 个格子的距离,水平翻转后,红色色块距离图片右侧边界也有 1 个格子的距离;原始状态中红色色块距离图片右侧边界有 3 个格子的距离,水平翻转后红色色块距离图片左侧边界有也有 3 个格子的距离。红色色块在图片纵向上的位置没有改变。根据以上信息,得到关于水平翻转的坐标更新计算公式:

$$x' = width - x \tag{1}$$

同理,关于垂直翻转的坐标更新公式如下:

$$y' = height - y \tag{2}$$

其中, x' 和 y' 是更新后的横坐标与纵坐标, *width* 和 *height* 是图片的宽度与高度。

转置翻转相当于将图片纵方向上的数据以横方向排列,将图片横方向上的数据以纵方向排列,就是将(x,y) 的值赋给(y,x)。明确了图片转置的算法,便得到坐标更新公式:

$$x' = y \tag{3}$$

$$y' = x \tag{4}$$

本文编写 3 个翻转函数,3 个翻转函数带上了 points 参数,新的函数除了能翻转图片,还能更新标注框的坐标。

random_flip 函数从__aug_methods 随机选取一个翻转组合,并按照顺序调用翻转组合中的函数,最后得到经过随机翻转的图片和标注框。

3.7 转换标注格式

MMDetection 目标检测框架提供训练接口,但需要预先将数据集的标注格式转换为 COCO 数据集的格式才能使用。本节先介绍 COCO 数据集的格式,

然后描述数据集格式转换的实现。

3.7.1　COCO 数据集格式

COCO 数据集是一个大规模的计算机视觉数据集，包含目标检测、实例分割、场景描述等多种类型的标注信息，共有 32.8 万张图片（其中有 20 万张是被标注过的），150 万个标注物体，80 个类别[4]。与本次任务相关的是目标检测数据集，故本节只介绍目标检测数据集的标注格式。

COCO 数据集的标注信息可以从关系数据库的视角来描述。目标检测数据集中最重要的 3 张表是：图片表、标注表、类别表。下面以关系模式描述这些表。

图片表（图片 ID，宽，高，文件名）

标注表（标注 ID，图片 ID，类别 ID，面积，x，y，宽，高，群聚标识）

类别表（类别 ID，类别名，超类名）

一张图片中可以有多个标注，图片与标注是一对多的关系。在 COCO 数据集中，一个标注只能属于一个类别，所以标注与类别是一对一的关系。实际上，图片表还包含了许可证 ID、来源链接、COCO 链接、拍摄日期，但这些信息在训练过程中不重要，所以在关系模式中省略。标注表中的 x 和 y 指标注框左上角的 x、y 值，从 0 开始编号。当标注的对象是一大群个体时，群聚标识记为 1，如果标注的对象是单个个体时，群聚标识记为 0。类别表中的超类名指类别所属的大类。本次任务不用到超类名。

COCO 数据集在发布时使用 JSON 格式存储标注信息，下方的 JSON 伪代码大致描述标注文件的格式。除了标注表中的“x，y，宽，高”被整合到了 bbox 字段中，其它 JSON 字段都与关系模式中的字段一一对应。第四个 JSON 对象组合了图片表、标注表和类别表，可将其视为一个数据库。

```
image{ "id": int, "width": int, "height": int, "file_name": str }
category{ "id": int, "name": str, "supercategory": str }
annotation{
  "id": int, "image_id": int, "category_id": int, "area": float, "iscrowd": 0 or 1
  "bbox": [x, y, width, height], }
  {  "images": [image],
  "annotations": [annotation],
  "categories": [category]}
```

3.7.2　转换代码

明确了 Labelme 标注文件格式和 COCO 数据集格式，然后可以开始编写转换代码。“@ utils.dataset.iter_over_labels”的作用是遍历由参数指定的文件夹里的标注文件，用 json 模块解码文件内容得到标注信息后，用标注信息和标注文件路径作为参数调用由装饰器包裹的函数（也就是 handle）。

为了根据图片 ID 分割训练集/验证集、建立 COCO 格式的类别表和从类别名到类别 ID 的映射，第一次遍历收集图片 ID 和类别信息。遍历完成后，打乱图片 ID 列表，取前 70%的图片用于训练，取后 30%的图片用于验证。

由于 labelme 保存的标注文件没有为标注框编号，所以设置一个变量 label_num 用于存放将要分配给当前标注框的 ID。再次遍历标注文件，之前由于没有从类别名到类别 ID 的映射，无法生成标注，现在准备好了映射，因此，可以生成 COCO 格式的标注信息。由于 labelme 存放的 points 是（左上角 x，左上角 y，右下角 x，右下角 y），与 COCO 格式里的 bbox 不一致，所以转换右下角 x 和右下角 y 为标注的宽和高。最后，将字典保存成 json 格式文件，标注格式的转换就完成。

3.8　超参数配置

在训练开始前，需配置一系列超参数。MMDetection 构建了独特的配置系统，并于 V2 版本加入了模块化和继承性设计，便于进行各种实验[4]。在 mmdetection 文件夹下有一个 configs 文件夹，里面包含了许多预定义的训练配置与模型配置。config/_base_中包含了 4 种最基本的组件类型：数据集（dataset），模型（model），训练计划（schedule），默认运行时（default_runtime），许多模型与方法都可以由这些基本组件构造。

本文使用 Faster RCNN 作为商品目标检测模型。将 configs/faster_rcnn/caffe_fpn_1x_coco.py 复制到 mmdetection 的工作目录，更名为 RetailDetectionConfig.py，以这个文件为基础，自定义一系列训练配置，使之能训练本文的数据集。

3.8.1　配置 Faster RCNN 模型

由于配置文件已继承了_base_/models/faster_rcnn_r50_fpn.py，所以不必写出模型的所有配置信息，只需要列出要修改的键值对，旧的键值对即可被覆盖掉。为了能用 OpenMMLab 提供的预训练权重，需将主干网络 Resnet-50 的风格改为 caffe。并设置 *frozen_stages* = 1，来冻结 Resnet-50 和第一个

ResLayer 部分的权重。继承而来的模型可以预测 80 个类别,本次研究的数据集只有 8 个类别,所以将 roi_head.bbox_head.num_classes 设置为 8。最后,因商品检测场景不会出现物体遮挡的情况,将 test_cfg.rcnn.nms.class_agnostic 设置为 True,在所有交并比大于 0.5 的检测框之间,仅留下置信度最高的检测框。

3.8.2 配置数据输入管道

摄像头拍摄到的图片分辨率只有 640×480,可以将训练数据输入管道的 img_scale 值改为(640,480),同理,测试数据输入管道的 img_scale 也改为(640,480),这样可减少运算量,加快训练速度和预测速度。

为进行数据扩充,在训练数据输入管道中插入 3 个 RandomFlip,将对应的 direction 分别设置为 diagonal、horizontal、vertical,并将 *flip_ratio* 设置为 0.5,表示有 50%的概率执行翻转。由于输入管道的元素比较多,这里省略一些操作。虽然没有展示出来,但由于 mmdetection 配置系统里的列表不像字典有继承性,新定义的 pipeline 会覆盖 base 文件里的数据,所以在实际使用时不能省略。

3.8.3 配置数据集信息

pipeline 定义了如何操作输入数据,data 定义了数据来源。将 dataset_type 定义为 RetailDataset,目前 MMDetection 的注册表中没有这个数据集类型,这里自定义,所以另外实现这个数据集类型,并将其加入注册表。data.samples_per_gpu 定义每个 GPU 上的 batch size。data.train、data.val、data.test 的结构是相同的,仅值不同,可分别用于训练、验证、测试,这里只介绍 data.train,val 和 test 同理。

将 data. train. type 设置为 dataset _ type,MMDetection 在解析配置文件时,会从注册表中搜索名叫“RetailDataset”的类,并将 ann_file、img_prefix、pipeline 作为参数传给 RetailDataset 的构造函数。ann_file 是输出的 COCO 格式的标注文件,data.train 使用 train.json,data.test、data.val 可以共用 val.json。img_prefix 是保存图片的文件夹,pipeline 是数据输入管道。

3.8.4 配置训练计划

根据多次调参观察得到的结果,设置优化器的类型为随机梯度下降优化器(SGD),学习率[5]初始值为 0.02,动量为 0.9,权重衰减率为 0.000 1。

runner 是 MMDetection 的训练核心,设置 *max_epochs* = 12 意味着最多训练 12 个周期。

optimizer = dict(type =′ SGD′, *lr* = 0. 02, *momentum* = 0.9, *weight_decay* = 0.000 1)

runner = dict (type = ´ EpochBasedRunner ´, *max_epochs* = 12)

lr_config 是关于学习率的配置,包含学习率预热和学习率衰减两个部分。关于学习率预热,采用基于迭代次数的线性式预热,预热比值为 0.001,这样的设定使得在训练初期的学习率被设置为 0.001×0.02,每迭代一次,学习率就会增加一个固定的数值,在第 100 次迭代时,学习率会恢复到初始值 0.02。关于学习率衰减,采用基于周期的阶梯式衰减,在第 8 和第 11 周期结束时,学习率会设置为原来的 0.1 倍,这意味着在第 9 到第 11 周期的学习率为0.002,第 12 周期的学习率为0.000 2,图 3 绘制了每一次迭代的学习率大小。具体设置内容为:

lr_config = dict(
warmup=´linear´, *warmup_iters* = 100,
warmup_ratio = 0.001,policy=´step´,
gamma = 0.1 *step* = [8,11])

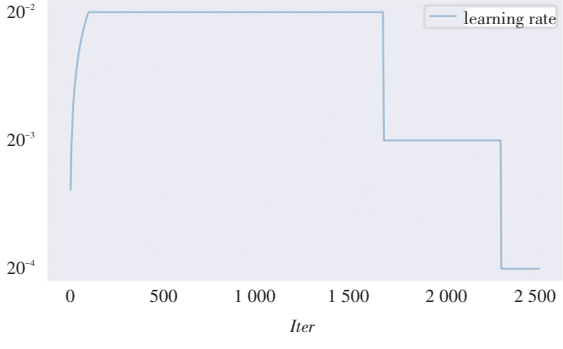

图 3 学习率日志

Fig. 3 Learning rate log

3.8.5 配置训练运行时

配置日志每隔 5 次迭代打印一次,设置 load_from 为预训练权重文件路径。

log_config = *dict*(*interval* = 5)

load_from= ´faster_rcnn_r50_caffe_fpn_mstrain_3x_coco _bbox _mAP - 0. 398 _20200504 _163323 - 30042637.pth´

3.9 训练

保存上述配置文件,打开终端,切换到 MMDetection 根目录,运行预定义的训练脚本 tools/train.py,将配置文件路径作为参数传入,即可开始训练。

程序会输出环境信息和展开后的配置信息,然后打印训练日志。每隔 5 次迭代会打印一条这样的

信息：

2022-05-05 15:28:21,694 - mmdet - INFO - Epoch [1][5/210]　lr: 8.192e-04, eta: 1:06:42, time: 1.591, data_time: 0.460, memory: 1390, loss_rpn_cls: 0.018 7, loss_rpn_bbox: 0.007 2, loss_cls: 2.054 1, acc: 31.582 0, loss_bbox: 0.335 8, loss: 2.415 8

“INFO”表示日志级别，Epoch [1][5/210]表示这条日志属于第 1 个周期的第 5 次迭代，总共有 210 次迭代。后面紧接着许多键值对，表 1 是对每一个键的解释。

每一个周期结束时会保存一次模型权重，并在验证集上进行一次测试，输出模型在验证集上的精度和召回率，格式如下：

Average Precision　(AP) @[*IoU* = 0.50:0.95 | area = all | *maxDets* = 100] = 0.936

Average Recall　(AR) @[*IoU* = 0.50:0.95 | area = all | *maxDets* = 100] = 0.953

所有模型权重、标准输出日志、json 格式日志、训练配置文件都会保存在 work_dirs/RetailDetectionConfig 文件夹中。

4　实验结果分析

4.1　评价指标

由于目标检测任务不仅输出了目标的类别，还输出了目标的位置，故无法使用普通的分类指标来衡量目标检测模型。

虽然评价指标很多，但都是基于以下 4 种基础数据，其中，*True*、*False* 分别表示检测结果的正确与否，*Positive* 和 *Negative* 分别表示检测结果判定存在物体、还是不存在物体。

True Positive：模型检测到物体，事实上的确存在物体。

True Negative：模型没有检测到物体，事实上的确没有物体。

False Positive：模型检测到物体，但事实上没有物体。

False Negative：模型没有检测到物体，但事实上存在物体。

检测结果究竟属于 *True* 还是 *False* 与 *IoU* (Intersection over Union)阈值有关。其中的“交”指的是预测框和标注框的相交的面积，“并”指的是预测框和标注框合并的面积。当检测框与标注框的 *IoU* 超过设定的阈值，比如说 0.75，就可以认为这个预测框属于 *True*；当 *IoU* 低于这个阈值，则认为预测结果是 *False*。*IoU* 阈值的高低取决于对检测框精确程度的严格与否，如希望模型输出的检测框要能严密贴合标注框，则应把 *IoU* 阈值设大。如检测框的精确程度要求不那么严格，或是标注框本身并不精确，则可将 *IoU* 阈值设定得小一些。

检测结果究竟属于 *Positive*、还是 *Negative* 和置信度阈值有关系。当检测框的置信度大于某个阈值，比如 0.5，则认为检测结果是 *Positive*，当置信度低于这个阈值，则认为检测结果是 *Negative*。

接下来介绍目标检测评价指标。

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{5}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{6}$$

值得一提的是，*Precision* 和 *Recall* 必须联立起来看。因为置信度阈值和 *IoU* 阈值会同时影响 *TP*、*FP*、*FN* 的数量。当把置信度阈值调得太高时，模型只会输出置信度非常高的检测框，此时的检测框数量是非常少，但精准的，*Precision* 可以达到一个很高的程度，而 *Recall* 则不理想。当把置信度阈值调得太低时，模型会输出非常多检测框，此时 *False Negative* 的数量会变少，*Recall* 的值很好，但 *Precision* 的值不理想。在部署模型时设置一个置信度阈值，这个阈值会同时影响 *Precision* 和 *Recall* 的值，根据需求的不同仔细考量，在 *Precision* 和 *Recall* 之间作取舍。*Precision Recal curve* 是一个帮助决策的工具，图 4 是一个实际的例子。

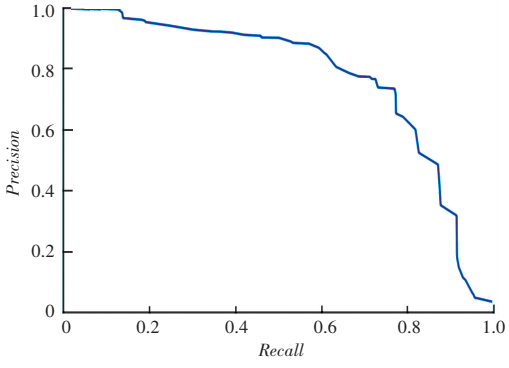

图 4　查准率-召回率曲线

Fig. 4　*Precision-Recall* curve

Precision - Recall 曲线能够帮助决策，但却是一条线，不是一个值，要衡量模型预测多个类别的性能，就要同时观察多张图，不够方便。 这里，给出的公式具体如下：

$$AveragePrecision = \int_0^1 Precision(Recall)\,\mathrm{d}(Recall) \tag{7}$$

Average Precision 能够有助于同时比较多个类别的性能,但有多少个类别,就会有多少个数字,如果要同时比较多个模型的性能,就要同时比较许多个数字,这也不方便,所以人们又发明了 mean Average Precision,简称 *mAP*,其中的"mean"是指在类别上的平均。

4.2 绘制 *mAP* 折线图

打开终端,切换到 MMDetection 根目录,执行以下命令,绘制 *mAP* 折线图和训练 *loss* 折线图:

python tools/analysis_tools/analyze_logs.pyplot_curve work_dirs/RetailDetectionConfig/202204025_152749.log.json --keys bbox_mAP --legend mAP

python tools/analysis_tools/analyze_logs.pyplot_curve work_dirs/RetailDetectionConfig/202204025_152749.log.json --keys loss

mAP 折线图如图 5 所示。观察图 5 可得出,本文模型在第 1 周期结束时就达到很高的准确率,在第 2 周期结束便达到性能瓶颈。第 2 到第 8 周期的 *mAP* 都维持在 0.83 左右。在第 9 周期结束时,*mAP* 跃上 0.922,这是由于学习率在第 9 周期衰减所导致的。第 9 到第 11 周期的 *mAP* 都维持在 0.925 左右,第 12 周期学习率最后一次衰减,*mAP* 迎来最后一次提升,到达 0.936,训练结束。

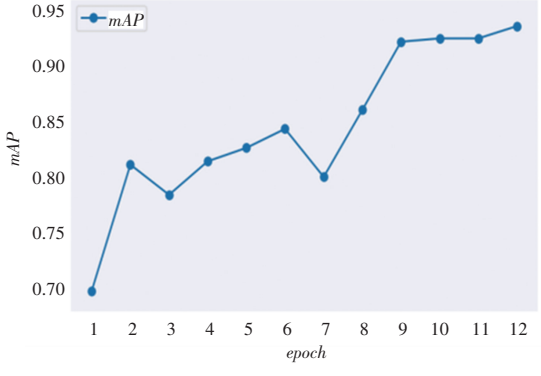

图 5 *mAP* 折线图

Fig. 5 *mAP* line chart

训练过程 *loss* 折线图如图 6 所示。从图 6 可得出本文模型学习率对稳定性、精度的影响。根据日志中的信息可得知每一个周期有 210 次迭代,学习率的第一次衰减在第 8 周期结束时,正处在第 8×210=1 680 次迭代,观察横坐标1 680处的曲线,可以看到 *loss* 出现了骤降,并且在以后的迭代中,震荡幅度没有之前那么大了。

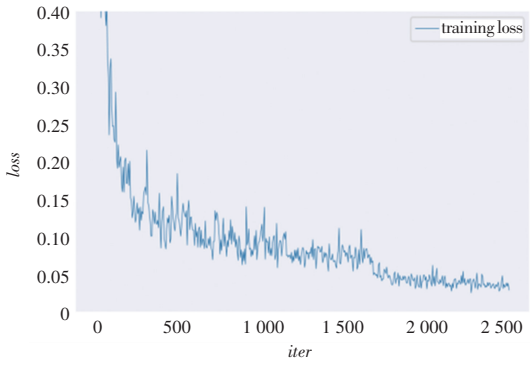

图 6 训练过程 *loss* 折线图

Fig. 6 Line chart of *loss* in training process

5 结束语

训练 *loss* 折线图如图 6 所示。本文将基于深度学习的目标检测算法引入到无人零售领域中,针对零售商品构建了对应数据集,提出了一种快速清洗目标检测数据集的方法,并讨论了基于图片翻转的数据扩充方法,训练了目标检测网络 Faster RCNN,使其能定位、识别零售商品,分析了目标检测常用度量方法,并通过在训练好的模型上进行验证,结果表明,商品识别速度快、整体性能良好。

参考文献

[1] 刘莹,王晓宇,徐卓飞,等. 基于卷积神经网络的商品图像识别[J]. 数字印刷,2020(06):33-40.

[2] 曾祥云. 人工智能推动无人零售崛起[J]. 企业管理,2017(11):94-96.

[3] CHEN Kai, WANG Jiaqi, PANG Jiangmiao, et al. MMDetection: Open MMLab detection toolbox and benchmark[J]. arXiv preprint arXiv:1906.07155, 2019.

[4] LIN T Y, MAIRE M, BELONGIE S, et al. Microsoft COCO: Common objects in context[C]//European Conference on Computer Vision. Cham: Springer, 2014: 740-755.

[5] HE Kaiming, ZHANG Xiangyu, REN Shaoqing, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 770-778.