北京理工大學

本科生毕业设计（论文）

无锚点框的全卷积单阶段目标检测方法研究

Research of Anchor-Free Full Convolutional One-Stage Object Detection

学　　院：计算机学院

专　　业：软件工程

学生姓名：陶润洲

学　　号：1120162055

指导教师：赵三元

2020 年 4 月 23 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名：　　日 期：　　年　月　日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：　　日 期：　　年　月　日

指导老师签名：　　日 期：　　年　月　日

无锚点框的全卷积单阶段目标检测方法研究

摘　要

本文目标检测作为计算机视觉和数字图像处理的一个热门方向，广泛应用于机器人导航、智能视频监控、工业检测、航空航天等诸多领域，通过计算机视觉减少对人力资本的消耗，具有重要的现实意义。

本课题主要研究无锚点框的全卷积单阶段目标检测方法，通过消除预定义的锚点框设置，该方法完全避免了与锚点框相关的复杂计算，大大减少了训练内存占用。该方法还避免了所有与锚点框相关并对最终检测性能非常敏感的超参数。由于该方法最后使用非最大抑制对回归框进行处理，该检测器比以前的基于锚点框的单阶段检测器具有更加简单的优点。该方法还提出新分支中心度，用于打压距离目标中心较远位置的一定数量的低质量边界框，降低低质量检测框的权重，提高检测性能。

本文基于 Python 的科学计算软件包和 Pytorch 深度学习框架来搭建全卷积单阶段目标检测网络，并使用 PASCAL VOC 和 COCO 数据集进行模型的训练和评估，最后使用训练好的模型对实物图片进行目标检测。

关键词：目标检测；深度学习；无锚点框

Research of Anchor-Free Full Convolutional One-Stage Object Detection

Abstract

As a popular direction of computer vision and digital image processing, object detection is widely used in robot navigation, intelligent video surveillance, industrial inspection, aerospace and many other fields. It has great practical significance to reduce the consumption of the human capital through computer vision.

This paper mainly studies the full convolutional one-stage object detection without anchor boxes. By eliminating the predefined anchor box settings, this method completely avoids the complex calculations related to anchor boxes and greatly reduces the training memory footprint. This method also avoids all hyperparameters that are related to the anchor box and very sensitive to the final detection performance. Since this method finally uses non-maximum suppression(NMS) to process the regression box, the detector has a simpler advantage than the previous single-stage based on the anchor box. This method also proposes a new branch center-ness, which is used to suppress a certain number of low-quality bounding boxes far away from the target center, reduce the weight of low-quality detection boxes, and improve detection performance.

Based on Python's scientific computing software package and Pytorch deep learning framework, this paper builds a full convolutional one-stage object detection network, and uses the PASCAL VOC and COCO datasets for model training and evaluation. Finally, the trained model is used to object detection for real pictures.

Key Words: Object Detection； Deep Learning； Anchor-Free

目 录

第 1 章 绪论

1.1 课题研究背景意义

当人类最开始发明计算机的时候，就在思考如何才能将计算机变得更加智能，来帮助人类完成一些工作。现如今，人工智能已经成为了一个非常热门的领域，不仅仅应用在众多活跃的研究课题中，并且也在人们生活的方方面面中也有很多实际的应用。目前，这个领域正在以几何倍的速度发展着，并且在未来也将继续健康发展。正因如此，人们也越来越希望可以通过人工智能来处理一些主观的、非规范性的事物，如识别图像等。

对于一些人类觉得处理起来很困难的抽象和形式化的任务，计算机却很擅长处理。在上个世界，计算机就在国际象棋方面崭露头角，战胜了人类棋手。但是直到最近几年，计算机在图像和语音的识别任务中才能达到人类的一般水平。一个人的思维发育是需要海量的外界相关知识，并且其中有很大一部分的知识是主观的，难以用形式化的符号和结构表示出来；如同人类一样，计算机也需要获取相等数量级的知识才能表现的足够智能。在如何让计算机学会这些主观的、非形式化的知识这一问题上，科学家首次提出了层次化的概念，借助人脑的工作方式，使得计算机通过构建简单的模型来学习一些较为复杂的特征。这种学习方式因其最终是构造出了一张“深层次”的图，且层与层之间通过简单的规则进行连接，所以被称为深度学习。

自 2006 年开始，大量的深度神经网络论文被发表，特别是在 2012 年，Hinton 课题组首次参加 ImageNet[3]图像识别大赛，通过构建的 CNN 网络 AlexNet 一举多得冠军，从那之后，神经网络就开始收到广泛的关注。现如今，这项技术已经成功地应用在包括计算机视觉领域在内的多种模式分类问题上。而目标检测则是计算机视觉领域需要解决的基础任务之一，也是视频监控技术的基础任务。同时目标检测作为泛身份识别领域的一个基础算法，对之后的人脸识别、步态识别、人群计数、实例分割等任务有着至关重要的作用。所以研究出高效、准确的目标检测算法具有非常重要的意义。

现如今，几乎所有的最先进的目标检测网络，如 RetinaNet、SSD、YOLOv3 和 Faster R-CNN 都依赖于预定义的锚点框，而本课题研究的无锚点框全卷积单阶段目标检测网络，通过消除预定义的锚点框，避免了大量与锚点框相关的运算和相关的超

参数。相比于目前主流的基于锚点框的单阶段网络，全卷积单阶段网络做到了设计复杂度更低、性能较好，速度较快。鉴于其有效性和高效性，全卷积单阶段网络可以作为目前主流基于锚点框检测器的一个简单而又强大的替代品，甚至可以扩展到许多其他实例级的识别任务中去，对目标检测领域的发展有着较为重大的意义。

1.2 国内外研究现状分析

目标检测是“在哪里有什么”的任务，对于目标检测任务，目标的类别不确定、数量不确定、位置不确定、尺度不确定，根据传统非深度学习方法如 VJ 和 DPM 和早期深度学习方法如 OverFeat[4]，都使用金字塔多尺度和遍历滑窗的方式，逐尺度位置判断该尺度该位置处有没有可以识别的目标，非常笨重且耗时长。

锚点框这个概念首次提出是在 Faster R-CNN[5]论文中，通过预置的一组固定参考框，避免复杂的遍历和计算，直接判断该固定参考框和目标框的关系来进行检测，相较于之前的方法又快又好。但是同时还有以关键点等特征取代锚点框进行目标检测的方法，也能够达到类似的效果。所以根据有无锚点框，可以大致的将目标检测方法分为：基于锚点框目标检测算法和无锚点框目标检测算法。

1.2.1 基于锚点框目标检测算法

目前来说，比较顶尖的目标检测方法几乎都是用了锚点框技术。首先预定义一组不同尺寸不同位置的固定参考框，覆盖几乎所有的位置和尺寸，每个参考框均负责检测与其交并比大于阈值的目标，通过直接判断该参考框中有没有需要检测的目标以及目标框偏离参考框多远，来避免多尺度遍历滑窗，使得检测器检测效果好，并且检测时间短。

在首次提出锚点框概念的 Faster R-CNN 方法中，首先通过 VGG16 骨干网络输出 256 维的特征图，再通过区域提案网络(RPN)对之前输出的特征图进行预测。在 RPN 网络中，特征图先经过一个 3×3 的卷积层，最后进入两个分支。根据预置的由 3 个不同尺度{128, 256, 512}以及 3 个不同比例{1:1, 1:2, 2:1}所构成的 9 个固定的锚点框，两个分支分别会输出 2×9=18 个分类的分数以及 4×9=36 个回归的坐标。再将以上得到的分类器结果和回归结果输入到 RPN 网络的最后一部分 Proposal Layer，对预测的回归边界框进行修正和结果的输出。根据 Faster R-CNN 论文锚点框训练后学习到的平均区域大小（见表 1-1）显示，这 9 个固定的锚点框大约能覆盖到边长 70~768 的目标。

表 1-1 不同锚点框对应平均区域大小

锚点框大小	平均区域大小
128^2, 2:1	188×111
128^2, 1:1	113×114
128^2, 1:2	70×92
256^2, 2:1	416×229
256^2, 1:1	261×284
256^2, 1:2	174×332
512^2, 2:1	768×437
512^2, 1:1	499×501
512^2, 1:2	355×715

最终在使用锚点框之后，Faster R-CNN 在 VOC07 和 VOC12 数据集上的 AP 分别达到了 73.2%，70.4%，在 COCO 数据集上 AP 也达到了 21.9%的高度。这样的识别精度在当时众多算法框架中是非常高的，从这之后锚点框技术也广泛应用在众多的目标检测框架中，也都取得了很优异的效果。

在 2016 年 ECCV 上发表的 SSD[6]框架中也采用了锚点框技术，同样也达到了很好的效果。SDD 框架的作者认为仅仅只靠同一层特征图上的多个锚点框进行回归的话，精度还远远不够，因为有很大可能这一层上所有预置的锚点框和目标框的交并比都很小，这样一来训练误差就会很大。通过对多个层级上的锚点框计算其相关的交并比，去找到与目标框的尺寸、位置最接近的一批锚点框，这样在训练时也能达到最好的准确度。

在 SSD 框架中骨干网络采用与 Faster R-CNN 一样的 VGG16 网络，但是把最后的两层全连接层换成了普通的卷积层，之后再通过多个卷积层，最后得到 6 个不同尺度的特征图。对于这 6 层特征图每层分配 4 或 6 个不同形状的锚点框，对于不同尺度的特征图，采用不同尺度、不同比例的锚点框进行匹配 。根据原文的计算，6 个特征图共计 8732 个锚点框，并且相对于 Faster R-CNN 而言，锚点框设置更加合理，小尺度锚点框多且密，大尺度锚点框少且疏。最终 SSD 框架在 VOC07 和 VOC12 数据集上得到的 AP 分别达到了 76.8%，74.9%，在 COCO 数据集上 AP 也达到了十分优异的 31.2%。

1.2.2 无锚点框目标检测算法

自从 2018 年在 ECCV 上发表的 CornerNet[8]开始，无锚点框的目标检测模型就开始层出不穷，最近更是达到了井喷的状态，宣告无锚点框目标检测算法开始在目标检测领域占有一席之地。

早在 2016 年 CVPR 上发表提出的曾经大火的 YOLO[7]框架，就是最早的无锚点框目标检测模型。YOLO 将物体检测的几个部分统一成一个简单的神经网络。YOLO 模型关注整张图片以及在这张图片中所有物体的信息，所以模型支持使用整张图片的特征去预测每一个边界框，同时也支持对一张图片预测出所有的边界框。YOLO 模型直接对全图进行训练，首先将输入图片分成 S×S 个网格，如果检测目标的中心落在了网格内，那么这个网格负责检测这个目标。每一个网格单元将预测 B 个边界框以及对应边界框的置信分数，以此来进行训练和回归。通过这样无锚点框的全局回归的方式，YOLO 模型在目标检测时，处理速度很快，并且在 VOC07 和 VOC12 数据集上的 AP 也分别达到了 66.4%和 57.9%。虽然效果相比 Faster R-CNN 较差，但是综合检测效果和处理速度而言，在同时期的检测器中还是很优异的。

而 2018 年发表在 ECCV 上的 CornerNet 则是通过关键点预测的方式来进行目标检测。CornerNet 将检测目标的边界框这个任务转换成了检测边界框的左上角角点和右下角角点，然后将其组合起来完成检测任务，完全抛弃了锚点框这一概念。模型首先通过骨干网络沙漏网络(Hourglass Network)提取特征，之后分出两个单独的模块用来得到左上角角点和右下角角点的类别分类，并找到每个目标的一对关键点，以及减少基于坐标回算目标位置时的偏置。通过所得到的这些信息来进行训练，最终 CornerNet 在 COCO 数据集上达到了 42.1%的好成绩，也同时为无锚点框目标检测算法框架打开了一扇大门。

1.3 研究内容和主要工作

本课题的主要任务是实现无锚点框全卷积单阶段目标检测模型。本课题的主要研究内容是：掌握深度学习中常用的数据预处理和数据增强方法，提高检测器训练效果；熟悉主流的目标检测模型网络结构，为搭建全卷积单阶段目标检测网络模型做准备；了解 AP、AR 等模型评估指标计算方式，以便对搭建好的网络模型进行评估。

本文完成工作如下：

1. 对数据集图像进行图像处理，实现数据预处理和数据增强。数据预处理部分主要对彩色图像进行色彩归一化处理；数据增强部分主要对彩色图像进行缩放平移、左右翻转、上下翻转。
2. 搭建全卷积单阶段目标检测网络模型，结合原有的中心度分支增加中心部分采样处理，以提高检测器性能。
3. 对搭建好的网络模型进行训练、评估，并判断中心部分采样处理是否能提高检测器性能；使用训练好的模型检测实物图片，查看检测效果。

1.4 论文组织结构

本文的结构是严格按照此次课题的研究顺序进行安排的。本文详细结构如下：

第 1 章：绪论部分。介绍了深度学习与目标检测的技术发展，以及目标检测的发展对人工智能的重要意义。介绍了几款具有代表性的基于锚点框的目标检测框架与无锚点框目标检测框架，分析了锚点框的作用与意义以及无锚点框模型的处理流程。

第 2 章：主要介绍了与搭建网络模型相关的一些准备工作。首先对搭建网络模型相关的开发平台、所使用的深度学习框架和相关的重要 Python 库进行介绍；然后，介绍了实验的环境以及实验环境的搭建；最后介绍了本次实验所采用的数据集以及制作训练集和测试集的过程。

第 3 章：主要介绍了搭建网络模型的具体流程和相关工作。首先对数据进行增强与预处理；然后依次搭建骨干网络、特征金字塔网络和全卷积单阶段网络的检测头部分；最后实现模型训练、模型评估、数据分析与检测实物等功能。

第 4 章：主要介绍了搭建网络模型之后的实验过程。主要有以下几个部分：CuPy 库对模型训练的加速效果；搭建好的模型分别在处理好的 VOC0712 与 COCO2017 数据集上的检测效果；对训练过程中产生的相关数据进行分析；训练完成的模型对于实物的检测效果。

第 2 章 实验平台搭建与数据集处理

2.1 相关的开发平台与主要技术介绍

2.1.1 开发平台 Anaconda 与 VScode

Anaconda 是针对 Python 进行数据科学研究所专门建立的一组软件包，涵盖了大部分数据科学领域常用的 Python 库，并且为了解决软件环境依赖问题，Anaconda 自带有 Conda 包管理系统。在 Anaconda 的帮助下，数据工作者可以更加简单地处理在不同项目下对软件库甚至是对 Python 版本的不同需求。Anaconda 包含有其可视化图形界面 Anaconda Navigator，专门用来管理软件包和环境，通过图形化界面用户可以在不使用命令行的情况下管理软件包和环境等。目前 Anaconda 也因其良好的用户体验，如今被广泛的应用在人工智能、科学计算、大数据及云计算、金融等领域。

VScode 是微软推出的一款轻量级的代码编辑器，免费、开源而且功能强大，支持 Windows，OS X 和 Linux 系统。VScode 内置有 JavaScript、TypeScript 和 Node.js 支持，而且自带有丰富强大的插件生态系统，可以通过安装插件来支持 C++、C#、Python、PHP 等不同的主流语言。对比于其他代码编辑器，VScode 在打开特别大的文件时不会存在明显卡顿的情况，速度甚至可以和 NotePad++相媲美。并且 VScode 在处理数据文件时，如 json 文件，有相对应的快捷键来格式化文件的排版，使得在处理数据文件时，能够更加舒适的查看数据格式。

VScode 在开发基于 Python 语言的项目时，能够匹配 Anaconda 所创建的虚拟环境，以供用户选择。虽然 VScode 是代码编辑器，但是因为其自带底部终端的特质，在编写 Python 代码时，可以通过其快捷按钮来调用内部的终端，并使用选择配置好的 Python 环境来运行代码，十分方便快捷，易于开发。

2.1.2 深度学习框架 Pytorch

Pytorch 是由 Facebook 开源的神经网络框架，基于 Torch，专门针对 GPU 加速的深度神经网络编程。与 TensorFlow 的静态计算图不同，Pytorch 的计算图是动态的，可以更加实时的更改计算图，使得调试网络更加简单。Pytorch 不仅支持动态神经网络，而且实现了强大的 GPU 加速计算，同时还提供了自动求导功能。Pytorch 在源码的设计方面，不同于 TensorFlow 中封装有大量的全新概念，Pytorch 中只有少量的封装，设计追求简洁；并且在设计中张量、变量和神经网络这三个由低到高的抽象层次

之间紧密结合，可以同时进行修改和操作。正是由于 Pytorch 这种简洁明了的设计方式，其源码的可读性十分高，并且其运行速度也比 TensorFlow 和 Keras 等框架快很多。

Pytorch 在使用时，其内部的 API 也十分易于操作。构建神经网络时可以通过神经网络模块 torch.nn 中的 Sequential 方法来构建一层一层序列化的网络结构模型；然后通过神经网络优化器模块 torch.Optim 中的不同的方法来构建不同的优化器，以此来进行训练；再通过神经网络模块 torch.nn 中的损失函数来计算误差值；最后通过调用变量的.backward()方法计算梯度、反向传播，计算参数更新值，再调用优化器的.step()方法将计算得到的新参数更新到网络模型的中。这样就通过 Pytorch 的 API 实现了一个完整的简单训练模型结构，非常适合新手刚开始使用。

2.1.3 GPU 加速运算库 CuPy

在一般的数据运算时，通过 Python 软件包 NumPy，可以将数据向量化然后通过 CPU 并行运算，但是一般来说 CPU 内核数为 8 个甚至更少，所以这也就限制了 NumPy 库的加速效果。而对于 CuPy 而言，可以将其理解为一个借助 GPU 并行运算的 NumPy 库，因为 GPU 自身的特性是具有多个 CUDA 核心，这样一来 CuPy 就可以实现更好的并行运算加速效果。

同时 CuPy 与 NumPy、Pytorch 之间的数据格式的转换也简单。CuPy 与 NumPy 之间的转换可以直接通过 CuPy 库自带的方法来进行实现，而 CuPy 与 Pytorch 之间则需要通过中间库 dlpack 来进行实现转换，具体代码如下：

CuPy 的 ndarray 类型与 NumPy 的 ndarray 类型的相互转换:

```
import cupy as cp
# cupy -> numpy
numpy_data = cp.asnumpy(cupy_data)
# numpy -> cupy
cupy_data = cp.asarray(numpy_data)
```

CuPy 的 ndarray 类型与 Pytorch 的 tensor 类型的相互转换：

```
from cupy.core.dlpack import toDlpack, fromDlpack
from torch.utils.dlpack import to_dlpack, from_dlpack
# cupy -> tensor
```

```
tensor_data = from_dlpack(toDlpack(cupy_data))
# tensor -> cupy
cupy_data = fromDlpack(to_dlpack(tensor_data))
```

2.2 开发环境搭建

2.2.1 实验环境介绍

1) 软件环境：
   a) 操作系统：Ubuntu 18.04.4 LTS
   b) 开发平台：Anaconda 4.7.10、VScode
   c) 开发语言：Python 3.7.7
   d) 运算平台：CUDA 10.1
   e) 深度学习加速库：CuDNN 7.6.5
   f) 深度学习网络框架：Pytorch 1.4.0
2) 硬件环境：
   a) CPU：Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
   b) GPU：GeForce RTX 2080 Ti
   c) 内存：125GB

2.2.2 实验环境搭建

1) 安装 Anaconda 平台

Anaconda 是一个开源的 Python 发行版本，其中包括了 Conda、Python 等 180 多个科学包及其依赖项。根据本实验的需要，前往官网下载 Anaconda3 4.7.10 安装包，对 Anaconda 进行安装。在安装过程中，将其安装路径加入到环境变量中，以便于使用。安装完成之后，打开终端并输入命令：

```
conda --version
```

若输出其版本，则安装成功。

在 Anaconda 使用过程中，有一些常用的命令，为了方便了解使用，在此罗列，以下为 Anaconda 常用命令表(见表 2-1)：

表 2-1 Anaconda 常用命令表

命令	对应的操作
conda create --name <env> python=x.x	创建 Python 版本为 x.x 的虚拟环境
conda/pip install <packge>	安装软件包
conda/pip uninstall <package>	卸载软件包
conda/pip list	查看安装的所有包
conda activate <env>	激活目标虚拟环境
conda deactivate	退出当前虚拟环境，回到 base 环境
conda env list	查看创建的所有虚拟环境

2) 安装 CUDA 运算平台与 CuDNN GPU 加速库

a) 安装 CUDA 运算平台

CUDA Toolkit 是 NVIDIA 公司针对 GPU 编程的基础工具包，也是驱动显卡计算的核心技术工具。首先查看实验环境中 NVIDIA 显卡驱动版本，根据对应版本选择 CUDA 版本，本实验根据环境选用的是 CUDA10.1 版本进行安装。在安装完成之后，打开终端输入命令：

```
nvidia-smi
```

若输出 GPU 显卡详细信息，则安装成功。

b) 安装 CuDNN GPU 加速库

CuDNN 是基于 CUDA 的深度学习 GPU 加速库，有了它才能在 GPU 上进行加速运算。根据实验环境下安装的 CUDA 版本，去对应官网下载所对应的版本，本实验根据环境选用的是 7.6.5 版本。下载完成之后，解压文件，将解压后的文件夹复制到 CUDA 的安装目录下，覆盖对应文件夹即可。在复制完成之后，打开终端输入命令：

```
cat /PATH/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2
```

其中 PATH 为 CUDA 安装目录，若输出对应信息则安装成功。

3) 安装 Pytorch 框架

Pytorch 是神经网络框架，专门用来针对 GPU 加速的深度神经网络编程。因为实验采取 Anaconda 平台，所以采用 Conda 的包管理进行下载，这里选择版本为 1.4.0 的 Pytorch 进行下载安装：

```
conda install pytorch torchvision cudatoolkit=10.1 -c pytorch
```

安装完成之后，进入 Python 环境，输入代码：

```
import torch
```

若未输出报错信息，则安装成功。

再验证 Pytorch 是否能使用 GPU 进行加速运算，输入代码：

```
torch.cuda.is_available()
```

若返回输出 True，则说明 Pytorch 可以成功使用 GPU 进行加速运算。

2.3 数据集介绍与处理

2.3.1 VOC0712 数据集

PASCAL VOC[16]是用于图像识别和分类的一整套标准化的数据集。该数据集中包括有 20 中物体的分类，包括人、动物、交通工具和家具等。在 PASCAL VOC 的训练集中的每张图片在标注文件夹中都能找到相对应的标注文件。标注文件中有每张图片对应的名字、图片大小、需要检测物体的类别以及图片中物体位置的坐标等信息，通过这些标注信息可以将需要识别的物体从图片中提取出来进行训练。

常用的 PASCAL VOC 数据集为 VOC2007 和 VOC2012，在本次实验中将二者结合起来，进行训练与测试。将 VOC2007 数据集中的训练集(5011 张)与 VOC2012 数据集中的训练集(17125 张)进行合并作为实验的训练集，一共 22136 张图片。由于 VOC2012 数据集的测试集标注文件未公开，所以采用 VOC2007 数据集中测试集进行测试与模型评估，一共 4952 张图片。

2.3.2 COCO2017 数据集

COCO[17]数据集起源于微软在 2014 年出资标注的 Microsoft COCO 数据集，主要为目标检测、分割、人体关键点检测、语义分割和字幕生成等任务而设计。COCO 数据集中包含有 80 中物体的分类，数量远大于 PASCAL VOC 数据集，所以检测难度也比 PASCAL VOC 数据集大很多。在 COCO2017 数据集中，数据集与测试集中的图片的标注信息都各自对应一个 json 文件。数据集官方也提供了相对应的 Python 软件包 pycocotools 来方便使用者读取 COCO 数据集的信息。本次实验采用 COCO2017 数据集中的训练集作为实验训练集，一共 118287 张图片。COCO2017 数据集中的 val 测试集作为实验的测试集进行测试与模型评估，一共 5000 张图片。

2.4 本章小结

本章主要介绍了在搭建网络之前，对于实验平台与实验环境的搭建和数据集的构造过程。首先因为本次实验涉及的 Python 软件包较多，为了避免依赖冲突等不必要的问题，采用 Anaconda 平台进行实验；再通过对比 TensorFlow 与 Pytorch 的框架特点和性能等，选用了更加简洁且更易于开发编程的 Pytorch 框架进行编程。在数据集选择方面选用了，当今比较主流的两大数据集——PASCAL VOC 与 COCO。与这两大数据集相对应的 ImageNet 竞赛与 COCO 竞赛也是目标检测领域内的关注热点，所以采用这两个数据集能更好的训练网络并评估检测器性能。

第 3 章 全卷积单阶段目标检测网络的实现

3.1 数据增强及预处理

3.1.1 数据集标注信息提取

不管是 PASCAL VOC 数据集还是 COCO 数据集，其原本的标注信息格式都不便于模型的读入，所以这一部分将数据集原本的标注信息转换为便于模型读入的文本格式。处理数据集标注信息这一部分的功能定义在 annotation/deal_voc.py 与 annotation/deal_coco.py 文件中，下面将分别介绍对于 VOC0712 与 COCO2017 这两个数据集标注信息的提取过程：

1) VOC0712 数据集

对于已经构造好的 VOC0712 数据集，其标注文件位于数据集目录下的 Annotations 下，每一张图片都对应一个标注文件，包含有图片名称、物体类别及对应边界框坐标。标注文件为 xml 格式，通过浏览器打开标注文件，可以观察到在 annotation 节点下的 filename 节点内有对应图片的文件名，size 节点有对应图片的长宽大小，同时还有数个 object 节点，每一个 object 节点都对应一个图中的识别目标，包含目标的类别与目标边界框的坐标。对于 xml 格式的标注文件，本实验使用 Python 自带的 xml 软件包来进行解析，通过遍历树状结构的节点，找到需要读取的信息，将其写入到对应的文件中，方便模型读取训练。

通过依次读取标注文件，提取出所有标注文件的信息，每一个标注文件生成一行文本信息，例如，名为 000001.jpg 的图片所生成的文本信息为：

000001.jpg，12 47 239 194 370，15 7 11 351 497

其中 12 与 15 分别为物体类别对应的标签索引，其后的四个数分别为 X 坐标最小值、Y 坐标最小值、X 坐标最大值、Y 坐标最大值。通过这样的方式生成训练集和测试集的标注文本 trainval.txt 与 test.txt，并且额外生成一个 label.txt 文件用于记录每种物体类别对应的索引信息，如 background 背景类对应索引为 0。

2) COCO2017 数据集

对于 COCO2017 数据集来说，其结构与 VOC0712 不同，标注文件为 json 格式，位于 annotations 目录下。本次实验采用官方给出的 pycocotools 软件包处理 COCO 数据集，通过 pycocotools 软件包中 coco 类中的方法，对 json 类型的标注文件进行解

析。首先通过 coco 类中的 getImgIds()方法来获取所有的图片对应的编号；然后通过 loadImgs()方法可以获取对应图片的信息；再通过 getAnnIds()和 lodaAnns()方法就可以获取得到图片的标注信息。不同于 VOC0712 数据集，COCO2017 数据集给出的坐标形式为：X 坐标最小值、Y 坐标最小值、图片宽度、图片长度，为了与 VOC0712 统一格式，则通过用 X 坐标最小值与图片宽度相加，Y 坐标最小值与图片高度相加得到 X 坐标最大值、Y 坐标最大值。最后与 VOC0712 数据集处理类似，将模型所需要的信息写入到对应的文件中，以方便模型的读取。并且为了便于之后的模型评估，还生成了 coco_table.json 文件用于记录所有可用的测试集图片文件名。

3.1.2 数据增强

数据集中不同图片的形状大小都不相同，为了方便之后的操作计算，本实验选择将 VOC0712 数据集中的图片形状统一固定在 641×641 大小，而 COCO2017 中的图片则固定为 1025×1025 大小，以 VOC 数据集中图片为例，原图如图 3-1a 所示，调整图片尺寸之后如图 3-1b 所示。在此基础上，本实验将对数据集进行数据增强操作，具体功能定义在 dataset/dataset.py 文件中，分别对图片进行了缩放平移、左右翻转、上下翻转和亮度对比度饱和度变换操作：

1) 缩放平移

在 dataset.py 文件中定义了 center_fix()方法，其功能为实现将图片大小通过双线性插值的方式缩小或扩充至目标大小 641×641 或 1025×1025；与之相对应的一个函数 random_fix()，则是对图片进行缩放平移操作，函数在规定好的最小缩放比例（VOC 数据集为 0.6，COCO 数据集为 0.8）到 1.0 之间随机取数，然后对图片按比例进行缩放，并且在缩放之后在保证图片不越界的情况下随机进行上下左右平移。最后根据图片的本身的缩放比例以及上下左右平移距离，相对应的改变图片中物体的坐标框，使模型依旧能通过坐标框找到图片中的对应物体。当原图如图 3-1a 所示时，缩放平移后的图像如图 3-1c 所示。

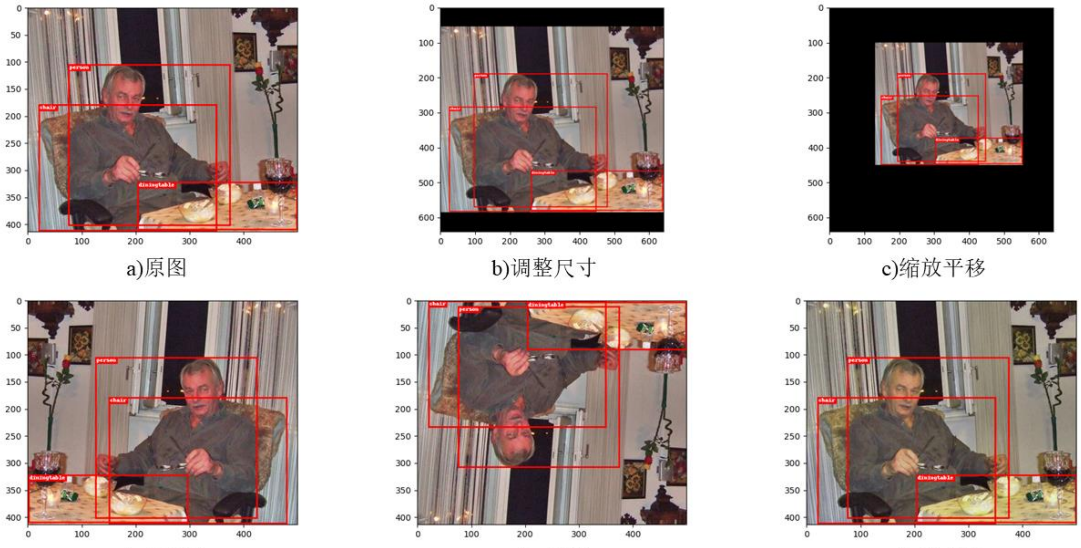
2) 左右翻转与上下翻转

左右翻转与上下翻转的功能实现在 dataset.py 的 filp()方法中。通过调用 Python 的标准图像处理库 PIL 中 Image 类的 transpose()方法，将数据集中的图片进左右翻转或者上下翻转。与缩放平移操作一样，在对图片进行变换之后，同时也需要对检测物体使用的坐标框进行转换，使坐标框依旧能与图中物体对应上。当原图如图 3-1a 所

示时，左右翻转与上下翻转之后的图像则分别如图 3-1d 与图 3-1e 所示。

3) 亮度对比度饱和度变换

通过 torchvision.transforms 中的 ColorJitter 类来对图片进行亮度、对比度和饱和度的变换。ColorJitter 类中有 4 个可输入的参数——brightness、contrast、saturation 和 hue，分别对应图片的亮度、对比度、饱和度和色相。在本次实验中将这 4 个参数均设为 0.1，则对应的亮度因子、对比度因子与饱和度因子均会在 0.9~1.1 之间选取，色相因子从-0.1~0.1 之间随机选取。这样就可以通过 ColorJitter 类调整图片的色彩属性，达到数据增强的目的。并且因为仅仅是对图片色彩进行调整，所以无需对图片中的坐标框进行调整。当原图如图 3-1a 所示时，进行色彩变换之后的图像如图 3-1f 所示。



a)原图　b)调整尺寸　c)缩放平移

d)左右翻转　e)上下翻转　f)色彩变换

图 3-1 图片展示

在训练过程中，对数据进行增强时，通过 random 库中的 random()方法，首先将 25%的图片进行左右翻转，25%的图片进行上下翻转；在此基础之上对 30%的图片进行亮度对比度饱和度的变换操作；最后对 50%的图片进行随机缩放平移操作，以达到数据增强的目的。

3.1.3 数据预处理

对数据进行预处理操作，就是为了使得数据可以更好的适应模型进行训练。在本

次实验中，仅采用数据归一化的操作对数据进行预处理。使用 torchvision.transfroms 中的 Normalize 类对数据进行归一化处理。根据从 ImageNet 训练集中抽样计算的 RGB 的均值和方差，将本次实验中 Normalize 类中的 mean 参数设置为[0.485, 0.456, 0.406]，将 std 参数设置为[0.229, 0.224, 0.225]。在对数据进行数据增强之后，使用数据归一化操作对数据进行预处理，使数据更好的适应模型，并且在训练时加快训练网络的收敛性。以图 3-1a 中图像为例，进行数据预处理之后的图像如图 3-2 所示。

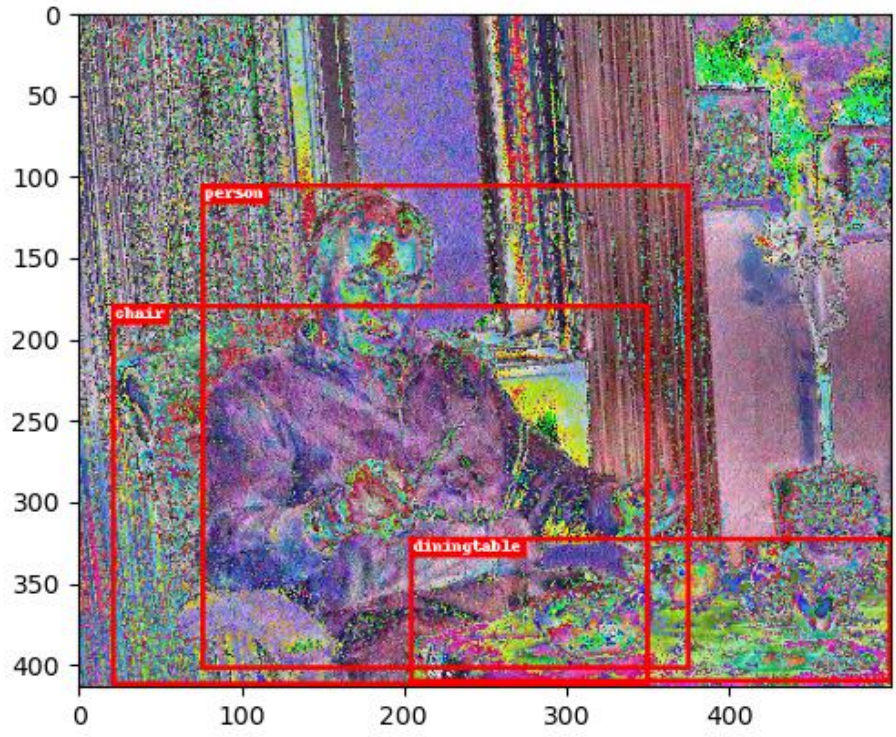


图 3-2 数据预处理展示图

3.2 骨干网络搭建

骨干网络是指卷积神经网络采用的主干架构，其本质是通过一些系列的卷积层来对图像进行卷积提取出图像的特征图。模型后续则利用骨干网络所提取的特征来进行运算和卷积，最后通过网络的检测头来得出最后的检测的结果。所以选取一个好的骨干网络对于目标检测网络的性能是有很大的提升的，本次实验则采用了 ResNet[9] 网络作为模型的骨干网络来获取图像的特征图。

3.2.1 骨干网络 ResNet 介绍

目前主流的骨干网络基本上都是 CNN 特征提取分类网络。而在早期学者们普遍认为深度学习网络越深越复杂，参数越多，则网络所学习到的特征也就越多，表达能力更强，那自然识别分类效果也就越好。所以从最开始的 AlexNet 的 7 层到后面的 VGG 网络的 16 层甚至 19 层，再到后来 GoogleNet 发展到 22 层。学者们一直在探寻着究竟深度神经网络发展到多深时，还可以持续的提高网络的学习能力，提高分类的准确性。但是一味的加深神经网络并不能给网络的分类性能带来很大的提升，甚至在一定程度上会导致网络收敛变得更加缓慢，从而导致准确率变低。就如 VGG 网络来说，当其层数达到 19 层之后，再对网络进行卷积层的增加就会导致 CNN 分类网络性能变差。所以因此学者们不再将目光一味的投向加深神经网络上，而是通过一些其他的方法来使 CNN 特征提取分类网络分类效果增强。

对于 CNN 特征提取分类网络而言，网络深度越深，层数越多反而会带来深度神经网络的退化。正是基于这样的事实，ResNet 的作者提出了一种假设：对于一个浅层的深度神经网络，在不断地往该网络上叠加新层时，若是这些新叠加的层什么特征也不进行学习，仅就是单纯的复制下原本浅层网络的所有特征，那么对于叠加完之后的网络而言，其分类的性能是不会下降的，也就是说新加的层就仅仅只是恒等映射，不会起到任何减弱分类性能的作用，那这样一来便不会出现深度神经网络退化现象。

正是基于这样的假设 ResNet 作者提出利用残差学习来解决退化问题。对于一个浅层的网络而言当输入 x 时，网络所得到的特征为 H(x)，但是当我们希望网络学习到网络的残差 F(x)=H(x)-x 时，这时网络本质所学习到的特征其实为 H(x)=F(x)+x。这样一来，即使是最极端的情况——当学习到的残差为 0 时，这时候所增加的层也仅仅只是做了恒等映射，并不会使得深度神经网络出现退化现象。但是实际上残差几乎是不可能为 0 的，即新叠加的网络层至少是能学习到一点新的特征的，这样一来，使用 ResNet 网络，即使不断地叠加新层也能做到让网络的性能有所提升，至少不会出现退化现象。

正是因为 RestNet 网络可以不断叠加的特性，目前的 RestNet 网络有多种不同层数的变体形式，比如 ResNet-18、ResNet-34、ResNet-50、ResNet-101 与 ResNet-152。这些 ResNet 的多种的变体形式，其主要的区别还是在于叠加块的结构以及叠加块的数量，对于 ResNet-18、ResNet-34 这类比较浅层的网络，叠加块采用 BasicBlock 基础块的结构进行叠加，没有一些复杂的层次结构；但是对于 ResNet-50、ResNet-101 与 ResNet-152 这类层数较多，较为复杂的网络而言，叠加块采用 Bottleneck 瓶颈层

进行叠加深入，Bottleneck 的网络结构也相较复杂，所以效果也会更好。而对于最终提取的特征图结果而言，一些浅层的 ResNet 网络，其最终得到的特征图深度也会远远小于深层的 ResNet 网络。本次实验通过权衡，采用了性能较为适中，内存占用也相对合适的 ResNet-50 网络作为最终的骨干网络，用于提取图像中的特征。

3.2.2 骨干网络 ResNet-50 搭建

在搭建 ResNet-50 骨干网络时，最重要的是对于叠加块的的理解与实现。ResNet-50 网络分为 6 个部分，除了网络一开始使用 7×7 卷积核卷积的卷积层与最后的全连接层之外，剩下的 4 个部分都是由不同数量的 Bottleneck 瓶颈层叠加而构成。所以在搭建骨干网络时，首先实现了最为关键的 Bottleneck 类，再进行整体网络结构的搭建。

1) Bottleneck 类的实现

在 model/ResNet50.py 中定义了 Bottleneck 类。对于 ResNet-50 而言，其瓶颈层由主分支与 shortcut 分支构成。主分支是由两个 1×1 的卷积层中间加一个 3×3 的卷积层所构成；而 shortcut 分支所输出的特征矩阵则需要与主分支所输出的特征矩阵相同才行，所以根据主分支的输出不同 shortcut 分支结构也不相同。

当主分支的输入与输出的特征矩阵形状不同时，shortcut 分支则使用一个 1×1 的卷积层并选取适当的步长，来对输入的特征图进行卷积，使其与主分支的输出矩阵相同，其结构如图 3-3 所示。

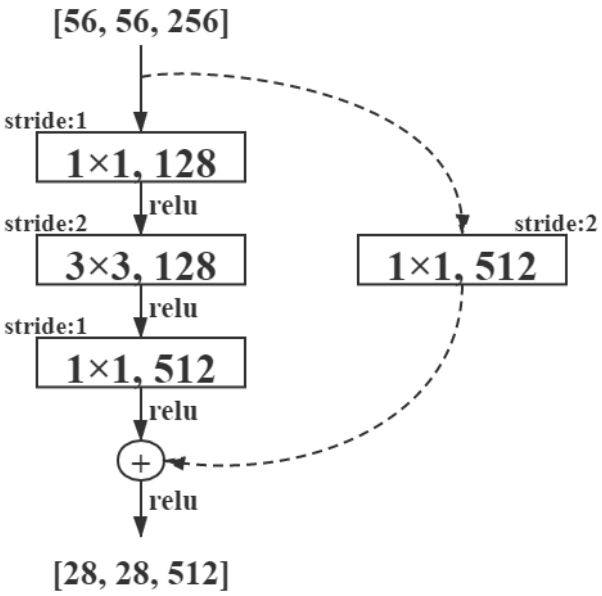


图 3-3 主分支输入输出矩阵形状不同

而当主分支的输入输出的特征矩阵形状相同时，此时 shortcut 分支本质上的作用则为恒等映射，不对输入的特征矩阵进行任何操作，其结构如图 3-4 所示。

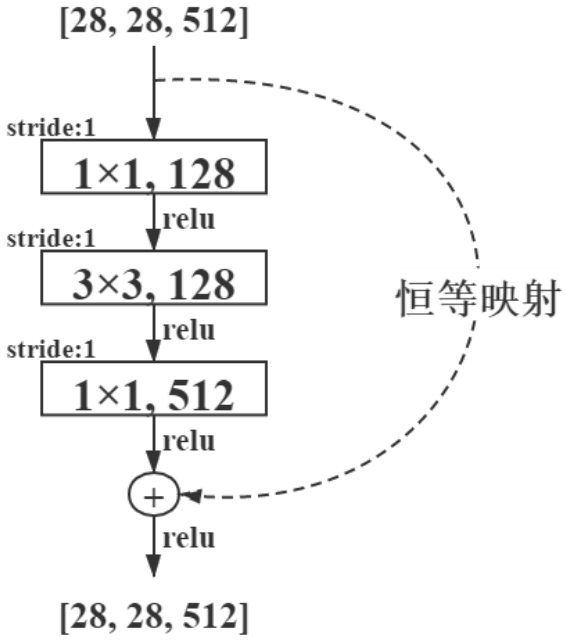


图 3-4 主分支输入输出矩阵形状相同

此外，在瓶颈层中每个卷积层下面都接了一个 BN(Batch Normalization)层[14]用于规范化数据，这样做的好处在于，可以加速网络的收敛，防止模型过拟合以及降低网络模型对于初始化权重的敏感程度。通过 BN 层可以使得模型效果更加理想，而且因为 BN 层的特性，其中的一个超参数可以替代掉卷积层中的 bias 偏置值，所以在使用 BN 层时，可以将卷积层中的 bias 设置为 False，这样对于网络模型是完全没有影响的，并且还减少了超参数的数量。

2) ResNet-50 整体网络的搭建

根据上文所述，ResNet-50 网络一共由六部分所组成，分别为一层卷积核为 7×7 的卷积层、四个不同数量堆叠的瓶颈层和一个全连接层，并且在第一个瓶颈层之前还添加了一个池化层。与瓶颈层中一样，在 ResNet-50 主体网络中，每一个卷积层之后都将跟随一个 BN 层，用于数据规范化，加快网络的收敛，提高模型的检测性能。

对于这四个不同数量堆叠的瓶颈层，每一部分的输出特征图的深度都为瓶颈层设定深度的 4 倍，且这四个瓶颈层所设定的深度分别为 64、128、256 和 512，所以最终 ResNet-50 网络输出的深度为 2048。而对于本次实验来说，因为 ResNet-50 骨干网络所输出的特征图是要交予特征金字塔网络来进行处理的，所以在本次实验中，将

不实现最后一部分全连接层，并且将这四个瓶颈层的后三个所输出的特征图传入网络的下一部分特征金字塔网络，以便于识别不同尺度大小的检测目标。

3.3 特征金字塔网络搭建

此部分为模型网络的中间部分，承接 ResNet-50 骨干网络和全卷积单阶段网络检测头。将骨干网络所输出的三个形状不同、深度不同的特征图，进行处理，然后将输出的用于检测不同尺寸目标的一组特征图交给网络检测头进行最后的处理。

3.3.1 特征金字塔网络介绍

在特征金字塔网络提出来之前，目标检测领域大部分的算法框架都只采用经过卷积之后的最顶层的特征图进行预测，从这些框架的表现上来看，检测效果还行，但是也都没有达到很高的准确率。在整个深度神经网络中，经过卷积的较为低层的特征图中目标的位置很准确，但是特征语义信息较少；而在与之相反的高层的特征图中特征语义信息十分丰富，但是相对低层的来说目标的位置信息比较粗略。正是基于这样的事实，Tsung-Yi Lin、Piotr Dollar 等人提出了多尺度特征金字塔网络结构[10]，原本一般的网络只在最顶层特征图上进行的预测操作，而特征金字塔网络则在不同层次的特征图上独立的进行预测。这样一来，针对图中不同尺度大小的检测目标就可以用不同层次的特征图进行预测。根据作者所做出的实验，在应用了特征金字塔网络进行特征提取之后，可以显著的提高 5~8%模型的召回率，并且在模型的 AP 方面也能广泛的提高 2~3%，可以说是对网络模型的性能有了极大的提升。

3.3.2 特征金字塔网络搭建

在本次实验中，将从骨干网络所输出得三个不同形状大小、深度得特征图经过横向连接、卷积等操作之后，生成一组五个不同形状大小的特征图用于对不同尺度的目标进行检测。在本实验中的特征金字塔网络结构如图 3-5 所示。

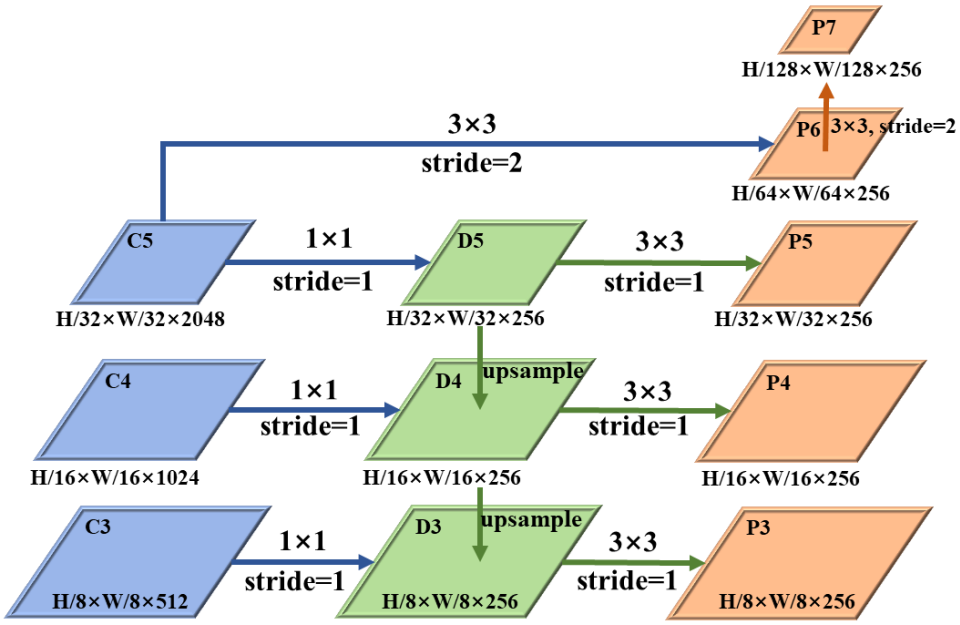


图 3-5 特征金字塔网络结构

因为由 ResNet-50 网络所输出的三张特征图 C3、C4、C5 深度各不相同，所以首先通过卷积核为 1×1，步长为 1 的卷积层将这三张特征图的深度统一变为 256。因为对于深度神经网络来说，较高层次的特征图其中所包含的特征语义信息较为丰富，所以在进行横向连接之后，可以对高层次的特征图进行上采样，然后与低层次的特征图相结合以获取语义信息更加丰富，目标位置更加明确的特征图，即 D3、D4、D5。再由 D3、D4、D5 这三个特征图进行进一步的卷积操作，此时选用卷积核为 3×3 的卷积层进行卷积，以获得最后用于预测的特征图 P3、P4、P5。然后为了保留更多的高层特征语义信息，对 ResNet-50 网络所输出的 C5 进行卷积核为 3×3，步长为 2 的卷积操作，得到 P6；最后再对 P6 进行同样的卷积核为 3×3，步长为 2 的卷积操作，获得最顶层 P7。

至此，需要用于预测的五张特征图都已经准备就绪了，通过这五张形状大小不同的特征图可以对应的检测实物图中不同尺寸大小的目标物体，使得检测器的效果更加理想。

3.4 全卷积单阶段网络检测头搭建

这一部分是整个全卷积单阶段网络最后的预测部分，同时也是整个网络中最核心的一个部分。在这一部分中，模型将前面网络根据图片所提取出来的特征图，进行

预测处理，形成了三个分支——分类分支、中心度分支和检测框回归分支。然后再根据这三个分支所预测得到的预测值与对应的目标值得到相应的误差值，最后通过误差值进行梯度求导、反向传播以及计算更新网络的超参数，以提高模型的检测性能。下面将这一部分细分为三小节进行介绍。

3.4.1 检测头全卷积处理

通过之前的 ResNet-50 骨干网络和 FPN 特征金字塔网络，模型会得到一组五个形状大小不同，深度相同的特征图，分别用于检测不同尺度大小的目标物体。形状较大的特征图因为其卷积的步长较小，预测目标的位置信息较为准确，所以用于检测尺度较小的物体；而相反形状较小的特征图其卷积步长较大，且其中特征语义信息更为丰富，所以用于检测尺度较大的物体。通过这样的预测方式，可以使得预测的结果更加合理，使得模型的检测性能更加优异。

在全卷积单阶段目标检测模型网络的检测头中，有三个分支，分别为目标分类分支，用于检测目标的类别，获得相对应的预测分数；中心度分支，用于衡量目标像素点与目标框中心的距离是否过大；以及检测框回归分支，用于得出对应检测目标在原图中的坐标。其中目标分类分支和中心度分支并行，除了最后得出预测值的一层之外，其余卷积层共用，如图 3-6 所示。

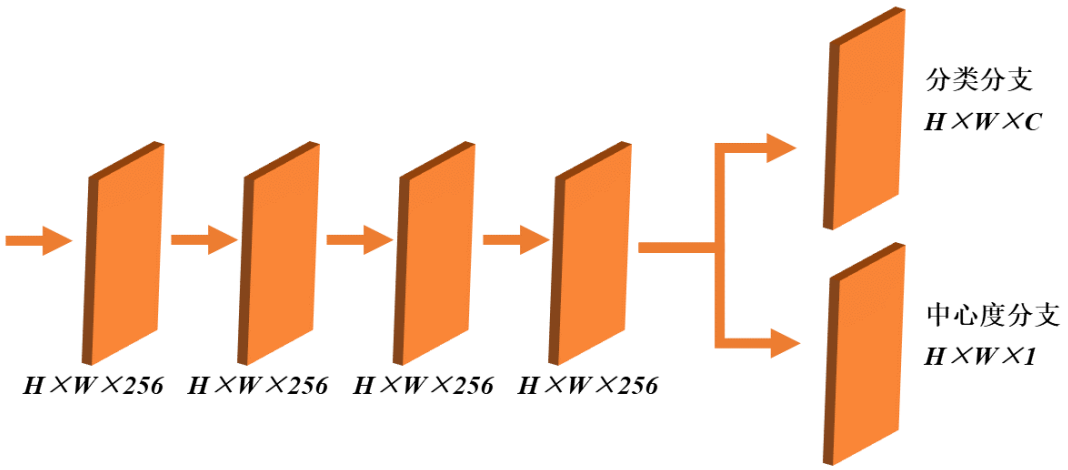


图 3-6 目标分类分支和中心度分支

由图 3-6 中可以看出，在获得特征图之后，将特征图通过四层卷积核为 3×3，步长为 1 的卷积层，然后分别将得到的特征矩阵送入预测分类的卷积层与预测中心

度的卷积层中，得到两个预测的结果矩阵，形状分别为 H×W×C 与 H×W×1，其中 C 代表网络模型一共对 C 种目标物体进行检测。通过分类分支所得到形状为 H×W×C 的预测结果，代表每个像素点中对于这 C 种不同的目标进行检测的分数，分数越高则代表该点更偏向于这类目标。而因为全卷积单阶段目标检测方式是通过逐像素点进行预测的，所以会有很多偏离目标框中心点的像素点参与进卷积预测，这样一来就会导致目标分类时有许多干扰因素致使预测结果不够准确。所以全卷积单阶段目标检测方法为了避免这种情况，在分类分支中额外并行了一个中心度分支，用于预测该像素点到其对应目标物体中心的距离。设定该点到目标框左边界、上边界、右边界和下边界的距离为 l，t，r，b，则中心度 center-ness 如式(3-1)所定义：

$$centerness = \sqrt{\frac{\min(l,r)}{\max(l,r)} \times \frac{\min(t,b)}{\max(t,b)}} \tag{3-1}$$

通过中心度分支所得到的预测值，将用于降低低质量检测边界框的权重，将对应的中心度预测值与目标分类的预测分数相乘，这样就可以使得这些远离目标框中心的像素点所预测出来的分数变低，那么在目标框中心附近的像素点的得分就会相对较高，这样就能使得模型的性能更加优良。

根据中心度分支的作用原理，本次实验还额外采取了中心部分采样的策略，与中心度分支相结合，希望使检测器效果更加优良。对于中心度分支而言，其作用是降低低质量检测框的权重，使得其分类得分较低。但是中心部分采样则是直接从根源上避免这些像素点离目标框中心较远的，低质量的检测边界框。通过计算目标像素点与目标框中心的欧几里得距离，当这个距离大于模型所预设的阈值时，则不将该点考虑在预测范围内，即使目标点位于目标框内，也当作背景像素点进行处理。而对于不同层次的特征图，其对应的阈值也因为其检测目标的尺寸大小不同而不同。对于低层次的特征图，用于检测尺寸较小的目标，则其对应的阈值也偏小；相反，对于高层次的特征图，用于检测尺寸较大的目标，其对应的阈值也就偏大。而在本次实验中，模型对于由低到高五张大小不同的特征图，所预设的阈值分别为 6，12，24，48，96，192。这样通过中心部分采样的策略，从根源上杜绝了这些低质量的检测边界框，加上与分类分支并行的中心度分支，就能使检测器达到更好的检测效果了。

除了分类分支与中心度分支之外，剩下的回归分支与这两个分支相互独立，但是其网络结构与前两个分支相似，如图 3-7 所示。

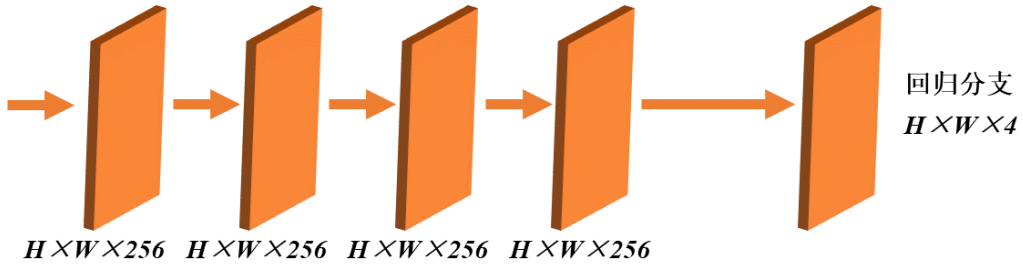


图 3-7 回归分支

由图 3-7 可以看出，先将所得到的用于预测的特征图通过四层卷积核为 3×3，步长为 1 的卷积层，然后再通过使用一个深度为 4 的卷积核，得到回归检测框的预测结果，形状为 H×W×4，其中第三维度代表了回归检测框的位置信息，分别为中心点的横纵坐标以及检测框的长宽大小。又因为在检测头最后会将这一组特征图所检测得到的结果进行合并操作，但是由于不同的特征层是用于回归不同尺度大小的目标物体，直接这样进行合并是不合理的，所以在得到这些位置信息之后，通过模型预置的一组固定的尺度参数，与位置信息相乘，才得到了模型可用于合并的位置信息。而对于原图而言，这些位置信息都过于抽象，所以在得到这些位置信息之后，先把这四个位置信息转换为回归检测框左上角和右下角两个角点的横纵坐标，再将位置信息基于原图的坐标进行映射，得到符合原图尺度大小的坐标，用于与之后的目标检测框一同进行损失值的计算。

在通过了分类分支、中心度分支以及回归分支之后，模型将得到的分类预测分数、中心度预测分数以及回归检测框坐标进行合并，在不同特征层之间共享检测头信息，这样可以使得检测头中的超参数更加高效，还能够提高检测器的性能。至此，所有需要通过网络所预测得到的结果都已经获取并且处理完毕，之后将对目标检测框进行相关的处理。

3.4.2 目标框特征层次分配

在上一部分中，网络模型已经通过检测头得到了预测的结果，在这一部分中将叙述对目标检测框的相关处理，并详细描述不同大小的目标框对不同特征层进行分配的操作。

对于预测的结果来说，其是由五个特征图的预测结果合并而成的，所以对于目标

框而言也需要对于这五个特征图进行对应的分配。在模型中预置了一组固定的尺度大小范围，由小到大分别为：0~64、64~128、128~256、256~512、512~1024。对于每一个特征图而言，其都只会检测规定尺度范围的目标物体，使得检测更加合理，检测效果更加良好。

在分配不同尺度的目标框时，首先将特征图上的点映射到原图上再进行判断，如果该点映射回原图时，不在任何的目标框内，则分类结果置 0，表示为背景类；若在目标框内，则计算出该点到目标边界框的上下左右边界的距离，若是距离的最大值不在规定的范围内，则仍然判定该点为背景类，分类结果置 0；若尺度大小合适，再根据中心部分采样策略，进一步判断该点到目标框中心的欧几里得距离，若是超出了预置的阈值则归为背景类，分类结果置 0；而当满足上述所有条件时，即可将该点置为目标框对应类别的编号，并且记录其对应的目标框的位置信息，以及根据式(3-1)计算出对应的中心度值。这样就完成了目标框对于不同特征层次的分配，最后将得到的五个已经分配好目标框的图，进行合并，生成与预测结果形状一样的目标矩阵，用于损失值的计算。

因为在分配目标边界框时，需要大规模的进行循环遍历，这样仅仅只通过 NumPy 库的 CPU 并行计算，耗时仍然很长，大大影响了训练的效率，所以采用 GPU 加速运算库 CuPy 进行在 GPU 上的并行计算，这样就能使得在进行分配时，耗时大大减少，提高训练的效率，使网络模型在训练过程中更加轻巧快速。

3.4.3 损失函数构建

这一部分使用之前网络所得到的预测值和经过处理的目标值，通过对应的损失函数，计算出对应的损失值，再进行梯度求导和反向传播。因为检测头的三个分支所得到的预测值的形状和产生的作用效都不同，所以针对不同的分支，实验中采用了不同的损失函数来计算损失值。

分类损失值L_{cls}通过 Focal Loss[11]损失函数进行计算训练。因为在进行单阶段目标检测过程中，负样本的数量会远远大于正样本的数量，这样会导致正负样本比例严重失衡，进而影响训练的效果。而 Focal Loss 损失函数可以降低简单的负样本在训练过程中所占的权重，所以采用 Focal Loss 损失函数来计算分类损失值效果最佳。

中心度损失值L_{cen}通过二进制交叉熵损失函数进行计算训练，该损失函数会在模型效果较差时，训练速度变快，在模型效果较好时，训练速度变慢，这样就可以使中

心度损失值逐渐稳定的达到一个最优值。

回归损失值L_{reg}则通过经典的 IoU Loss[12]损失函数进行计算训练，IoU 可以很好的反映出预测边界框对于目标框的检测效果，在回归过程中，IoU 就是判断预测边界框和目标框距离最为直接的标准。所以采用 IoU Loss 损失函数来进行回归的训练，可以达到一个比较好的效果。

在计算完这三个分支的损失值之后，最后通过式(3-2)来计算出网络的最终损失值L，来进行梯度求导和反向传播。

$$L = \frac{1}{N_{pos}}(L_{cls} + L_{cen} + L_{reg}) \tag{3-2}$$

至此整个全卷积单阶段网络模型结构就已经全部搭建完毕了，接下来将介绍一些通过网络模型实现的功能——训练、评估、数据分析和实物检测。

3.5 训练模型、评估模型、数据分析与实物检测功能实现

3.5.1 训练模型功能实现

在 train.py 文件中实现了训练模型这一功能。在对网络模型进行训练时，为了保证每次训练时网络的初始化结构都一样，首先对网络模型使用随机初始化种子进行初始化；然后在训练过程中，对于骨干网络使用官方所提供的预训练模型进行迁移学习，将骨干网络 ResNet-50 中所有的 BN 层以及四个堆叠层中的第一个堆叠层进行冻结来训练网络。训练时的优化器选用 SGD 随机梯度下降优化器进行优化，初始学习率设置为 0.01，每一批次训练数量 VOC0712 数据集设为 16，COCO2017 数据集设为 8，训练总轮次 VOC0712 与 COCO2017 数据集分别设置为 30 轮和 24 轮，并且为了避免网络模型在训练时不收敛，在总训练次数的 65%和 90%时，学习率分别都降低 10 倍，权重衰减和动量分别设置为 0.0001 和 0.9。

并且还采用了 warm-up 策略进行训练，在初始训练时，选用较小的学习率，然后随着训练次数的增强，逐渐提高学习率，直到学习率达到 0.01 再稳定学习率进行训练。通过 warm-up 策略训练，可以尽量避免在训练初期出现过拟合现象，并且有利于保持模型深层的稳定性，使网络可以更好的收敛。

为了能够保证训练中断之后，还能完全恢复之前的训练环境继续训练，在每一轮次训练结束之后，会将网络模型的所有超参数、优化器中所有的超参数以及训练轮次、训练次数都保持进 net.pkl 文件，方便之后恢复训练或者使用训练好的网络进行

检测时使用。

3.5.2 评估模型功能实现

在进行模型评估的过程中，当得到了网络模型所计算出来的预测值时，通过对分类分支得到的分类分数进行从高到低的排序，然后通过 NMS 非极大值抑制的方法，去除掉多个重叠区域过大的检测框，最终得到网络模型所预测得到的边界框。利用这些边界框与目标边界框进行交并比值的计算，当与目标框交并比大于所设定的阈值时，则判定该预测框成功预测出了一个正例，为 TP，否则判定预测错误，为 FP。通过这样来计算出模型的准确率和召回率，然后进一步的计算出模型的 AP 值，来衡量检测器的检测效果是否优良。

对于 COCO2017 数据集，则可以通过使用官方提供的 pycocotools 库来进行模型的评估，计算出相关的 AP 与 AR，更加准确的来评估检测器的检测效果和性能。

3.5.3 数据分析功能实现

在训练过程中，每训练一个轮次，都将对模型进行评估，生成对应的 AP 值。并且在训练中，每一次的训练都有会有一个对应的损失值，来衡量网络的收敛情况。在每一论训练结束之后，网络都会通过 NumPy 库的保存方法将这一轮次的损失值和 AP 值叠加保存至 log.npy 文件中。这样在训练结束之后，log.npy 文件中就保存了整个训练过程中所有轮次的损失值与 AP 值，最后通过 matplotlib 数据绘画库来画出随着训练次数的增加，损失值与 AP 值变化的曲线。通过这样对数据进行可视化，可以更加直观的了解到训练过程是否合理，网络是否逐渐开始收敛，有便于对网络结构以及参数的调试。

3.5.4 实物检测功能实现

实物检测功能与评估模型功能的类似，先是将要检测的原图通过训练好的网络模型，来得出相应的预测值；再通过对分类得分进行从高到低的排序，并以此进行非极大值抑制去除掉重叠部分较多的检测框，得到网络模型最终预测的检测框和对应的类别。并且将最终检测得到检测框的对应分类得分与预置对阈值进行比较，当得分低于阈值时，则判定该检测框可信度较低，不予以考虑；当得分高于阈值时，就通过 Python 的标准图像处理库 PIL 在原图中画出对应的检测边界框，并标明对应的类别。再将所得到的图片进行保存，这样就实现了实物检测的功能。

3.5 本章小结

本章主要介绍了整个全卷积单阶段目标检测网络模型的搭建过程，并对其中的关键结构和操作进行了详细的描述；还对通过搭建好的网络模型实现的一些功能进行了相关的介绍。首先对于骨干网络 ResNet 进行了介绍，阐明了其作为骨干网络进行特征提取的优势，然后介绍了搭建 ResNet-50 网络的过程；之后对于骨干网络后面的特征金字塔网络进行相关的介绍以及说明了搭建的过程；然后介绍了网络模型的核心部检测头的搭建，并且对其中关键性的处理，如目标框对不同特征层次的分配以及损失函数的构建进行了详细的介绍。最后在搭建好整个网络模型之后，对基于模型所实现的一些功能也进行了相关的介绍。

第 4 章 基于全卷积单阶段目标检测网络的实验

4.1 CuPy 库加速效果

整个全卷积单阶段目标检测网络中，在最后的核心部分有一个目标框分配到不同的特征层的操作。因为全卷积单阶段目标检测方法是逐像素点进行预测的，所以在进行这一步操作时，需要逐点的遍历整个图层，这就会导致每一次的训练时间大大加长，会严重影响到模型训练时的效率。本次实验通过使用 CuPy GPU 加速运算库来加速这一部分的运算操作，通过 GPU 并行计算，大大降低了每一次训练的时间，提高了模型训练的效率。

实验通过随机抽取的十个小批次训练，每批次训练数量为 16，对比其在进行分配目标框操作时使用 NumPy 库与 CuPy 库所消耗的时间，来观察 CuPy 库通过 GPU 并行计算所带来的加速效果，实验结果如表 4-1 所示。

表 4-1 NumPy 库与 CuPy 库训练耗时

	NumPy 库(ms)	CuPy 库(ms)
1	18581	534
2	12638	534
3	21152	530
4	14783	543
5	19160	550
6	22752	543
7	23192	545
8	36955	551
9	9165	547
10	47002	553
平均耗时	22538	543
标准差	10829.53	7.51

由表 4-1 中数据可知，使用 NumPy 库在 CPU 上并行运算的平均耗时为使用

CuPy 库在 GPU 上进行并行运算平均耗时的 41 倍左右，并且根据表中所计算得到的方差可以看出，使用 NumPy 训练库时，每一次的训练耗时极其不稳定，标准差高达 10829.53，而当使用 CuPy 库时，标准差为 7.51，远远比使用 NumPy 库进行训练时稳定。 所以在本次实验中使用 CuPy 库替换掉 NumPy 库来进行运算操作，是十分有必要的。

4.2 数据集训练结果

将已经搭建好的网络模型，分别使用已经构建好的 VOC0712 与 COCO2017 数据集进行训练，并且在每一轮次训练之后，都进行模型的评估，得到相关的模型评估指标。本次实验采用 IoU 阈值分别为 0.5 与 0.75 的 AP 值以及 IoU 阈值从 0.5 到 0.95 所有 AP 的平均值来衡量检测器的性能。并且在本次实验中，在基础的全卷积单阶段目标检测方法上，增加中心部分采样策略，所以在对模型进行评估的同时也将观察中心部分采样对于检测器性能是否有提升。

在模型训练的过程中，使用 VOC0712 数据集依据是否采用中心部分采样策略进行了两次训练，并使用模型的评估功能对检测器的性能进行了评估，结果如表 4-2 所示。

表 4-2 VOC0712 数据集训练结果

	AP	AP_{50}	AP_{75}
无中心部分采样	53.33	77.36	57.86
有中心部分采样	55.04	78.45	60.27

根据表 4-2 所示的测试结果，可以看出不使用中心部分采样策略时，在 VOC012 数据集上进行训练和测试，最终的检测得到的平均的 AP 值可以达到 53.33%，当 IoU 阈值分别为 0.5 与 0.75 时，AP 也分别为 77.36%与 57.86%。在添加中心部分采样策略之后，平均 AP 值提高了 1.71%，到达了 55.04%，而阈值分别为 0.5 与 0.75 的 AP 值也提高了 1.09%与 2.41%，达到了 78.45%与 60.27%。

同时模型还使用了 COCO2017 数据集进行训练，与 VOC0712 类似，根据有无中心部分采样策略训练了两次，并且使用官方提供了 pycocotools 软件包进行模型的评估，最后的结果如表 4-3 与表 4-4 所示。

表 4-3 COCO 数据集无中心部分采样训练结果

	IoU	area	maxDets	result
Average Precision (AP)	0.50:0.95	all	100	0.364
Average Precision (AP)	0.50	all	100	0.539
Average Precision (AP)	0.75	all	100	0.394
Average Precision (AP)	0.50:0.95	small	100	0.200
Average Precision (AP)	0.50:0.95	medium	100	0.396
Average Precision (AP)	0.50:0.95	large	100	0.477
Average Recall (AR)	0.50:0.95	all	1	0.297
Average Recall (AR)	0.50:0.95	all	10	0.473
Average Recall (AR)	0.50:0.95	all	100	0.509
Average Recall (AR)	0.50:0.95	small	100	0.311
Average Recall (AR)	0.50:0.95	medium	100	0.553
Average Recall (AR)	0.50:0.95	large	100	0.622

表 4-4 COCO 数据集有中心部分采样训练结果

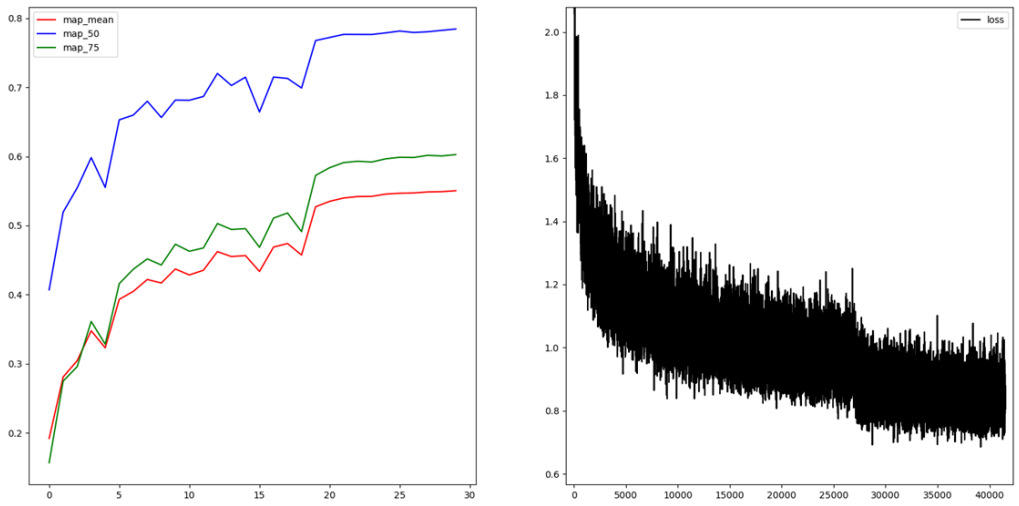
	IoU	area	maxDets	result
Average Precision (AP)	0.50:0.95	all	100	0.355
Average Precision (AP)	0.50	all	100	0.530
Average Precision (AP)	0.75	all	100	0.378
Average Precision (AP)	0.50:0.95	small	100	0.190
Average Precision (AP)	0.50:0.95	medium	100	0.390
Average Precision (AP)	0.50:0.95	large	100	0.469
Average Recall (AR)	0.50:0.95	all	1	0.293
Average Recall (AR)	0.50:0.95	all	10	0.457
Average Recall (AR)	0.50:0.95	all	100	0.489
Average Recall (AR)	0.50:0.95	small	100	0.283
Average Recall (AR)	0.50:0.95	medium	100	0.537
Average Recall (AR)	0.50:0.95	large	100	0.613

从表 4-3 与表 4-4 的结果来看，当使用中心部分采样策略进行训练时，可以将平均的 AP 值从无中心部分采样的 35.5%提升到 36.4%，提高了 0.9%，而 IoU 阈值为 0.5 与 0.75 的 AP 值也从 53%与 37.8%提升到了 53.9%与 39.4%，分别提高了 0.9%与 1.6%。并且根据 pycocotools 软件包对模型评估所得到的其他评估指标来看，在使用中心部分采样策略之后，这些指标数值都有所提高。

综合模型在 VOC0712 与 COCO2017 数据集上的训练结果来看，实验所搭建出来模型的检测效果较好，并且在实验中所采用中心部分采样策略，也对检测器的检测效果有着很明显的提升。

4.3 数据分析

在训练过程中，模型会对每一轮的训练结果以及其中每次训练的损失值进行记录并保存，通过模型的数据分析功能进行这些相关数值的可视化，通过可视化的结果可以看出 AP 值随着训练轮次的变化曲线以及损失值随着训练次数的变化曲线。以在 VOC0712 数据集上训练且有中心部分采样的结果为例，其可视化结果如图 4-1 所示。



a)AP变化曲线　　b)损失值变化曲线

图 4-1 模型训练数据分析

根据图 4-1a 中的化曲线图以及图 4-1b 中的损失值变化曲线可以看出，每当 AP

值与损失值在训练时趋于稳定时，通过降低学习率大小，可以使得模型的进一步收敛，检测效果更加优良。

4.4 实物检测

在模型训练结束之后，会得到一个保存有已经训练好的网络中所有超参数的文件，模型通过加载这个保存文件，可以对实物进行实时的检测。对实物图片进行检测完毕之后，会在原图的基础上，框出所有检测得分高于阈值的目标物体，并且在检测框左上角写出检测得分。

在 COCO2017 数据集的测试集中，选取一张图片，如图 4-2a 所示，通过已经在 COCO2017 数据集上训练好的网络参数对其进行检测，最终的检测结果如图 4-2b 所示。

a)原图　　b)检测图

图 4-2 实物检测结果

对比图 4-2 中的原图与检测图，可以看出通过训练好的模型能成功的检测出图片的目标物体，检测效果较好，能够准确的框出原图中目标物体的位置并标出相应的得分。

4.5 本章小结

本章主要介绍了在搭建完全卷积单阶段目标检测网络模型之后，所做的一系列

实验。首先在目标框进行特征层分配时，通过 CuPy 库进行 GPU 并行运算，加快训练过程，并与使用 NumPy 库进行运算时的效果进行对比；然后介绍了模型分别在 VOC0712 与 COCO2017 数据集上训练的效果，以及中心部分采样策略对于模型检测性能的提升效果；再通过对于每一个训练轮次所保存的数据进行可视化，生成 AP 与损失值变化曲线，来观察模型的训练过程是否合理；最后通过训练好的模型对图片进行实物检测，并观察模型对于实物图片的检测效果。

结　论

伴随着科研领域对深度学习越来越深刻的研究与探索，人工智能技术正在飞速地发展。而目标检测作为人工智能领域中比较重要的一环，自然也成为了众多科研人员所研究的重点，从最开始的 R-CNN，到利用锚点框进行目标检测，再到现在层出不穷的各类网络模型，目标检测的检测效果越来越精确，并且随着无锚点框概念的提出，检测速度也越来越快。本次实验通过利用深度学习框架 Pytorch 对无锚点框全卷积单阶段目标检测网络模型进行搭建，使用大量标准数据集图片对模型进行训练，以获得检测性能较好的网络模型。

相较其他基于锚点框进行目标检测的网络模型，全卷积单阶段目标检测方法通过逐像素点预测的方式解决目标检测问题，摒弃了锚点框以及所有与锚点框相关的计算和超参数，大大减轻了网络模型的负担，使得模型在进行检测时十分高效。为了避免因逐像素点进行预测时所产生的大量的低质量边界框，还在网络的检测头部分增加了中心度分支，用于降低低质量边界框的权重，使网络模型的检测效果更加理想。并且，根据中心度分支的作用原理，在网络模型中额外增加了中心部分采样策略，通过欧几里得距离来直接摒弃一些会产生低质量边界框的像素点，这样也使得模型的检测效果在原来的基础上更进一步。

在整个搭建目标检测模型的过程中，更加深入的了解了目标检测整体的发展历史以及现今主要的研究趋势。同时也对深度学习框架 Pytorch 有了更进一步的认识与掌握，学习了 Python 中用于数据科学研究的一些基础软件包，如 NumPy、MatplotLib 等，并且为了提高模型的训练效率，还学习了用于 GPU 并行运算的软件包 CuPy。同时，还掌握了在深度学习中几种常用的数据增强的方法，相互结合应用在了实验中，使训练效果更加理想。

虽然无锚点框的全卷积单阶段目标检测方法与同等级的检测器相比达到了非常良好的检测性能，但是其自身还是有很多的缺陷与不足的地方。在实验中可以看出，通过中心部分采样策略可以提高模型的检测性能，但是这也同时说明了网络所新增加的中心度分支并未起到一个很好的作用，无法将低质量与高质量的检测框合理的区分开。如果可以对中心度分支进行一个更加合理的改进，我相信检测器的效果会大大提升。同时，在数据的预处理与模型的一些参数上选取更合适的方法与数值，也可

以使模型的检测效果更加良好。即便如此，全卷积单阶段目标检测方法的提出，因其检测的有效性与高效性，对整个目标检测领域也有着十分积极的意义，值得我们去更进一步的学习与探究。

参考文献

[1] Tian Z, Shen C, Chen H, et al. Fcos: Fully convolutional one-stage object detection[C]//Proceedings of the IEEE International Conference on Computer Vision. 2019: 9627-9636.

[2] 郭济民. 基于深度神经网络的物体识别方法研究及实现[D]. 电子科技大学, 2018.

[3] Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International journal of computer vision, 2015, 115(3): 211-252.

[4] Sermanet P, Eigen D, Zhang X, et al. Overfeat: Integrated recognition, localization and detection using convolutional networks[J]. arXiv preprint arXiv:1312.6229, 2013.

[5] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[C]//Advances in neural information processing systems. 2015: 91-99.

[6] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//European conference on computer vision. Springer, Cham, 2016: 21-37.

[7] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[8] Law H, Deng J. Cornernet: Detecting objects as paired keypoints[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 734-750.

[9] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[10] Lin T Y, Dollár P, Girshick R, et al. Feature pyramid networks for object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 2117-2125.

[11] Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2980-2988.

[12] Yu J, Jiang Y, Wang Z, et al. Unitbox: An advanced object detection network[C]//Proceedings of the 24th ACM international conference on Multimedia. 2016: 516-520.

[13] Rezatofighi H, Tsoi N, Gwak J Y, et al. Generalized intersection over union: A metric and a loss for bounding box regression[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 658-666.

[14] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[15] Wu Y, He K. Group normalization[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 3-19.

[16] Everingham M, Eslami S M A, Van Gool L, et al. The pascal visual object classes challenge: A retrospective[J]. International journal of computer vision, 2015, 111(1): 98-136.

[17] Lin T Y, Maire M, Belongie S, et al. Microsoft coco: Common objects in context[C]//European conference on computer vision. Springer, Cham, 2014: 740-755.

[18] Huang L, Yang Y, Deng Y, et al. Densebox: Unifying landmark localization with end to end object detection[J]. arXiv preprint arXiv:1509.04874, 2015.

致　谢

值此论文完成之际，首先向我的导