天津中德应用技术大学

Tianjin Sino-German University of Applied Sciences

本科生毕业设计

基于深度学习的集装箱标识符智能识别系统设计

Design of Intelligent Recognition System of Container Identifier Based on Deep Learning

姓　　名 张坤

学　　院 机械工程学院

专　　业 机械电子工程

指导教师 孟祥懿

职　　称 讲师

完成时间 **2021 年 6 月**

天津中德应用技术大学

本科生毕业设计（论文）的声明

本人郑重声明：所呈交的毕业设计（论文）,是本人在指导教师指导下,进行研究工作所取得的成果。除文中已经注明引用的内容外,本毕业设计（论文）的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本设计（论文）所涉及的研究工作做出贡献的其他个人和集体,均已在文中以明确方式标明。本毕业设计（论文）原创性声明的法律责任由本人承担。

毕业设计（论文）作者签名：

年　月　日

本人声明：该毕业设计（论文）是本人指导学生完成的研究成果,已经审阅过设计（论文）的全部内容,并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

毕业设计（论文）指导教师签名：

年　月　日

摘　要

随着港口自动化的发展,港口集装箱吞吐量不断增加,记录集装箱标识符的速度和准确度就成为管理集装箱的重种之重了。识别集装箱标识符有许多方法,最方便经济的就是使用新兴技术—采用深度学习技术,利用视觉信息来实现低成本的解决方案。针对港口集装箱管理和分类难的问题,本文通过深度学习技术针对集装箱标识符进行定位识别通过处理识别结果针对集装箱标识符进行分类和管理,在一定程度上解决了港口集装箱管理和分类难的问题。集装箱标识符的识别,普遍做法就是先定位出箱号区域然后进行识别。

针对箱号区域特点,本文使用 Faster R-CNN 作为集装箱箱号区域定位网络。引入二次定位,最大程度保留了集装箱箱号区域特征图的空间信息。采用深度残差网络—— ResNet50 作为特征提取网络,使得更深层网络对箱号区域的提取能力显著提升；在合理范围里增加输入图像的分辨率,提高定位精度。

针对箱号字符特点,本文使用了基于 CRNN 的字符识别网络。引入生成的数据集进行训练,加快训练速度,在一定程度上提高准确率；引入自我判断机制,通过判断机制直接除掉不合格的识别,通过识别结果我们知道每个规格集装箱的数量和每个集装箱所有人全部的集装箱信息。为了方便了管理,我们在识别集装箱时,会根据类别的不同实时记录到不同的文本文件中。

最后,本文利用Flask框架实现嵌入式设备与PC端的通讯,将嵌入式设备与集装箱标识符的识别算法相结合,可实现在无人机等设备上搭载,极大的方便了集装箱的动态管理和加强了港口的自动化。

关键词：集装箱箱号；深度学习；目标检测；箱号定位；字符识别

ABSTRACT

With the development of port automation, the port container throughput is increasing, the speed and accuracy of recording container identifier has become the most important part of container management.There are many ways to identify container identifiers, but the most convenient and economical is to use emerging technologies -- deep learning techniques that leverage visual information to implement low-cost solutions.Aiming at the difficult problem of port container management and classification, this paper uses deep learning technology to locate and identify container identifiers and classify and manage container identifiers by processing the identification results, which solves the difficult problem of port container management and classification to a certain extent.The general method of identifying container identifier is to locate the area of container number and then identify it.

According to the characteristics of container number area, Faster R-CNN was used as the container number area location network in this paper.By introducing secondary positioning, the spatial information of feature map of container number area is preserved to the greatest extent.Resnet50, a deep residual network, is used as the feature extraction network, which makes the ability of the deeper network to extract the box number region significantly improved.In a reasonable range to increase the resolution of the input image, improve the positioning accuracy.

In view of the characteristics of box number characters, this paper uses a character recognition network based on CRNN.The generated data set is introduced for training to speed up the training speed and improve the accuracy to a certain extent.The introduction of self-judgment mechanism, through the judgment mechanism directly remove the identification of unqualified, through the identification results we know the number of each specification container and the owner of each container all the container information.In order to facilitate management, when we identify containers, we will record them in different text files in real time according to different categories.

Finally, this paper uses the FLASK framework to realize the communication between embedded devices and PC terminals, and combines the embedded devices with the recognition algorithm of container identifier, which can be carried on the UAV and other devices, greatly facilitating the dynamic management of containers and strengthening the automation of the port.

Key words: container number; Deep learning; Target detection; Box number positioning; character recognition

目　录

第一章　绪论

1.1 研究背景与意义

随着现代科技的发展，天津港希望对现有人工码头进行改造，设计一套智能集装箱码头解决方案，为全球自动化集装箱码头建设树立典范。作为集装箱唯一的身份证，通过集装箱标识获取管理信息尤为重要。主要就要解决集装箱标识符的识别如图 1-1。

集装箱是通过集装箱箱体的标识符进行管理，我们就需要完成集装箱标识符识别，以便于无人车和无人机能够根据识别的结果自动进行装载和归类运输。但是传统集装箱标识符识别系统，在自然环境下，光线条件复杂，难以对集装箱进行正确识别。不能直接将光学字符识别（OCR）未校对获得的结果应用于后续的工作任务中。如果采用人工校对，工作繁琐复杂，校对效果受人为干扰较大。因此，OCR 识别后的文本自动校对问题显得特别重要。本文采用 Faster R-CNN 和 CRNN 两种深度学习技术来设计和开发一套精确的识别和校对程序。

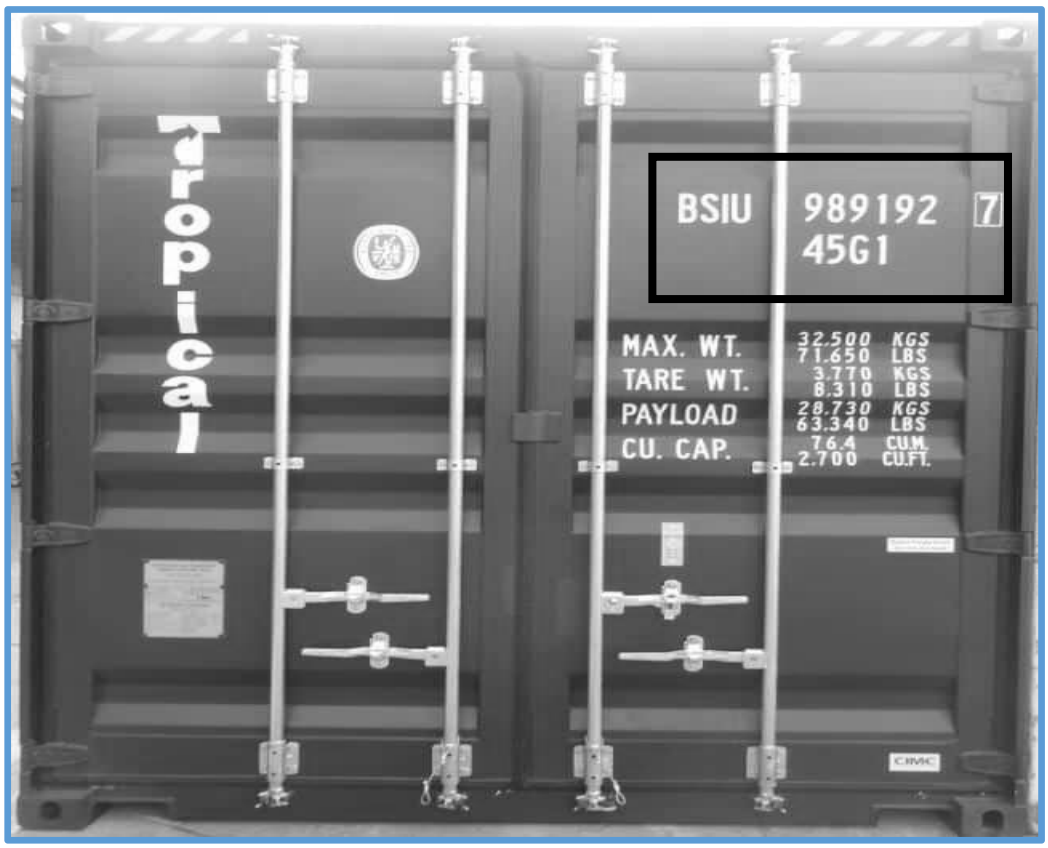


图 1-1　集装箱标识符

1.2 国内外研究现状

1.2.1 目标检测技术的发展与研究现状

近年来,由于深度学习技术的进步,各类的自动识别技术取得了新的突破[1]。基于深度学习的目标检测算法通过卷积池化后能得到深层语义特征。深度学习借助卷积神经网络采取反向传播方法来自主学习从数据中提取特定任务的特征向量,从而显著提高模型的识别性能,就目前而言未来的发展趋势就是研究更好的目标检测算法以提高模型精度。

1.2.2 文本识别技术的发展及研究现状

自然场景中的文本字符识别是一项艰巨的任务。近年来,随着深度学习技术的发展,这一领域取得了很大的进展。自然场景字符识别方法大致可以分为两类:

在早期,主要的自下而上的方法可以识别每个字符,然后将其连接起来。在传统方法中,特征向量由手工特征提取模块提取。识别性能有限。近年来,大多数工作采用自上向下的方法,其中自上向下的识别方法直接从文本实例的图像中识别整个单词或文本行,而不需要对单个字符进行任何预测,文本识别技术的发展也要依靠深度学习算法的进步。

1.2.3 集装箱箱号识别技术研究现状

目前的集装箱识别一般通过集装箱箱号的图像,通过分析图像来识别,不同于车辆的 ETC 功能,不需要在安装射频识别模块。传统 OCR 箱号识别算法一般分为定位和识别,一般情况两部分都是分开处理的。

在识别方面,随着技术的不断进步,国内外有许多优秀算法。Yoon [2]等为了解决字符识别过程中的丢失和粘连问题,提出了同时采集多个集装箱图像视图结合决策级与特征级融合的方法,美中不足的是其单视图识别率不高。

深度学习的发展,部分学者开始引入深度学习算法进行箱号识别。Verma [3]等提出了一种空间变换网络,为了解决集装箱字符在特征提取后产生的畸变,利用图文本检测器进行识别,识别率高达 99.64%,但是对于采集环境要求严格;陈力畅[4]等针对数字识别和字母识别使用不同的卷积神经网络,在优良场景下的识别率达到 98%。

1.3 本文的主要工作

本论文针对集装箱图片上标识符在复杂场景中难以正确分割和识别的问题,利用箱号区域定位网络模型—Faster R-CNN 和箱号字符识别网络模型—CRNN,进行网络训练。

本论文主要内容有:

(1) 对集装箱图片上的标识符区域进行定位。通过数据清洗,数据标注制作集装箱标识符定位数据集,然后通过网络训练得到集装箱标识符定位模型。

(2) 对集装箱图片上的标识符区域进行识别。通过合成数据集进行预训练,通过集装箱标识符定位模型输出的结果进行网络训练。然后根据结果进行分类。

1.第一章—绪论部分:课题通过研究港口集装箱管理和分类的 AI 应用 ,以人工智能算法为载体,程序能够进行识别分析、合理管控,节省人力,提高生产效率,通过数据分析

制定合理的物流计划。通过对课题的整体设计,提升自身的学术水平和职业素养。课题将针对传统识别的局限性问题,使采用 Faster R-CNN 和 CRNN 两种深度学习技术来设计和开发一套精确的识别和校对程序。

2.第二章—基于深度学习的集装箱箱号识别理论与应用：介绍了相关基础知识,利用集装箱基础知识对集装箱进行针对性的处理方案,还介绍了深度学习的有关的基础知识和几种可用到本课题的几种算法。加强了对于人工智能的理解和算法的发展趋势。

3.第三章—集装箱标识符定位模型：介绍了选择 Faster R-CNN 算法以其架构和具体执行情况,还介绍了数据集的制作方法和规范,加强了系统化的人工智能学习能力。还对实验结果进行分析力求最大化的完美解决集装箱标识符定位问题。

4.第四章—集装箱标识符识别模型：介绍了选择 CRNN 算法以其架构和具体执行情况,辅助介绍了下预训练方法和数据增广。还对实验结果进行分析力求最大化的完美解决集装箱标识符识别问题。

5.第五章——集装箱标识符识别整体调试：介绍了港口集装箱管理和分类的 AI 应用的程序运行流程和部分功能块的功能和调试,最后介绍了程序整体运行情况。

第二章　基于深度学习的集装箱箱号识别理论与应用

2.1 集装箱基础知识

2.1.1 集装箱箱号编写规则

集装箱箱号采用 ISO6346 标准。标注集装箱箱号由 11 个字符所组成唯一的识别符，前 4 位大写英文字母组成，前三位大写字母代表的是箱主。第四位大写英文字母代表设备识别码 ，第五至第十位数字为数字，是箱体注册码，是集装箱的身份证。 第十一位 为校验码，通常用方框来区分。它前四个大写英文字母，最后六个数字通过验证规则计算，作为验证记录的箱号正确的依据。箱号的每个字母都有一个运算的对应值，A 到 Z 如表 2-1 所示。箱号的前 10 位代码中，数字的对应值为自身[5]。

表 2-1　字母与数字运算对应表

A	B	C	D	E	F	G
10	12	13	14	15	16	17
H	I	J	K	L	M	N
18	19	20	21	23	24	25
O	P	Q	R	S	T	U
26	27	28	29	30	31	32
V	W	X	Y	Z		
34	35	36	37	38		

图 2-1 编码 TCLU 5429758 的集装箱

计算方式如下：第 N 位的箱号对应值再分别乘以 2^（N－1）（N＝1,2,3……10）。

例如：如图 2.1 箱号为 TCLU 5429758 的集装箱，

T＝31×2^（1－1）＝31×2^0＝31。

C＝13×2^（2－1）＝13×2^1＝26。

……

将前 10 位的代码值累加，TCLU 5429758 的值 6245，取 11 的模后结果为 8，就是这个箱号校验码。

箱体规格普遍位于箱号后面，由于两者字符相差不大，而且箱体规格富含信息，有了箱体规格这部分信息，可以更好的为管理和分类服务，因此本文也将其归入集装箱箱号，作为识别研究。下文所有集装箱标识符均为集装箱箱号和箱体规格号。

表 2-2　箱体规格号表

尺寸	箱型	95 码	尺寸	箱型	95 码
20 英尺	干货箱	22G1	40 英尺	干货箱	42G1
	干货高箱	25G1		干货高箱	45G1
	挂衣箱	22V1		挂衣箱	42V1
	开顶箱	22U1		开顶箱	42U1
	冷冻箱	22R1		冷冻箱	42R1
	冷高箱	25R1		冷高箱	45R1
	油罐箱	22T1		油罐箱	42T1
	框架箱	22P1		框架箱	42P1
45 英尺	干货箱	L2G1	45 英尺	开顶箱	L2U1
	干货高箱	L5G1		冷冻箱	L2R1
	挂衣箱	L2V1		冷高箱	L5R1
	框架箱	L2P1		油罐箱	L2T1

2.1.2 集装箱特点

1. 图像特点

由于集装箱端门处喷涂集装箱箱号最为常见，本文就以集装箱端门进行研究。在一张集装箱图像上，集装箱箱号区域面积较小，不易定位与识别[6]。为了定位箱号区域和识别，就需要我们分析集装箱图像的特点。

一般来说集装箱端门上具有许多信息，集装箱端门图像的箱号在集中在一块相对较小的平面上，为了方便研究，本文就研究集装箱端门。

图 2-2　二行标识符

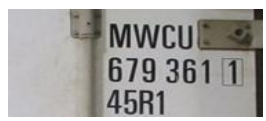


图 2-3　三行标识符

2. 字符特点

集装箱号由四个大写英文字母和七位数字组成。字母 O 和数字 0 的结构结构比较相似，字母 I 和数字 1 的结构比较相似。特别是在一些集装箱的图像中，字母 I 和数字 1 是肉眼无法区分的，只能通过集装箱号所写的规则来判断。

2.2 深度学习概论

2.2.1 神经网络

深度学习是基于机器学习算法。其模型结构借鉴了脑神经系统的传递函数结构。

图 2-4　数据传递结构

上图 2-4 是神经网络的数据传递结构。神经网络模型的训练依赖于大量的样本数据作为输入，这样能使模型学习到足够的特征，避免过拟合。在对神经网络学习的特征进行计算后，通过分类器对数据进行分类，得到分类结果。神经网络模型就是根据输入数据，学习更高级的特征对映信息，并根据输入数据的标注优化网络参数，最后得到特征和目标之间的对应关系。

1. 卷积神经网络

卷积神经网络（CNN）[7]是计算机视觉领域中经常用到的一种网络。CNN 通常由三个基本模块组成：卷积层、池化层和全连接层。

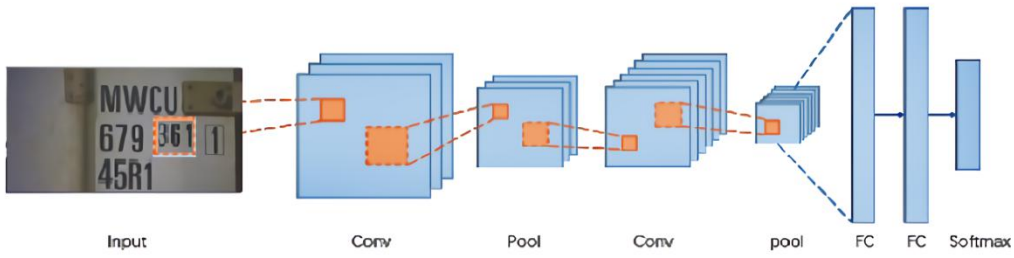


图 2-5　卷积网络结构图

人们对外界的认知一般是从局部到全局，局部图像的像素相关性比较接近，而远处图像的像素相关性比较弱。因此，每个神经元只需要感知局部信息，通过局部信息，就可以获得全局信息[8]。因此用随机卷积核来反向传播到与局部信息类似的卷积核，生成了一个新的特征图。这个特征图包含局部信息的特征。因此卷积核为将图像拆分成对应的特点。用卷积核扫描目标图得出的一个二维图为特征图。然后查看被识别图像有无对应的特征来确认是否为目标物体。由于特征图过多过大，加长了运行时间，我们进行了池化即缩小特征图，然后把得到的最简单的特征图们展开得到一条特征数组（全连接），对数组按目标图的数组权值操作得到一个判断是否为目标的概率数。然后通过概率更改卷积核，使其接近局部信息，这过程叫反向传播。

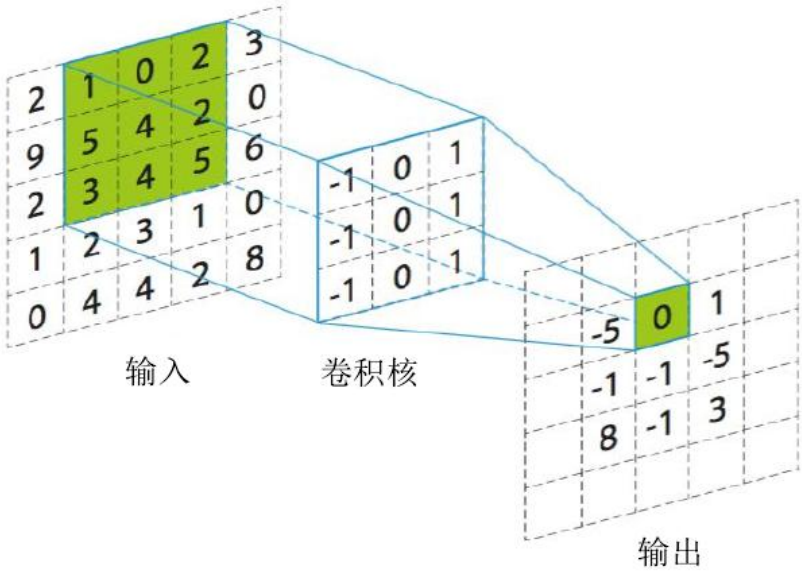


图 2-6　卷积操作

2. 激活函数

如果神经网络简单地通过线性卷积操作配置，则不能形成复杂的表示空间，并且难以提取高水平的意义信息，简而言之就是在反向传播过程通过非线性函数把卷积核拟合为局部信息。

激活函数的函数是提高神经网络模型的非线性。即使堆叠了几个层，那也只是一个矩阵相乘。因此，如果你没有非线性结构，你就完全不是神经网络了。在目标检测中，常使用的激活函数主要是 Sigmoid、ReLU 、tanh 、Softmax。

1．Sigmoid 函数

图像平滑、函数易于求导。

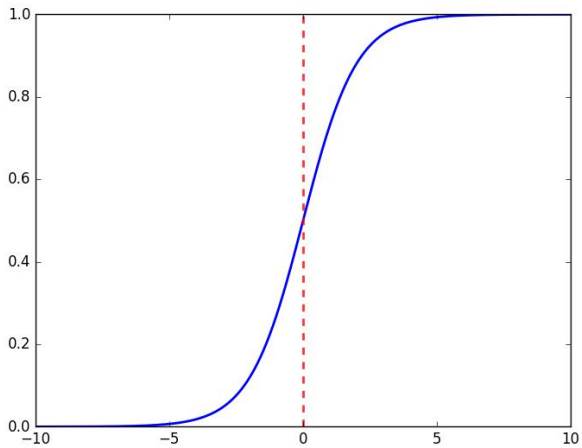
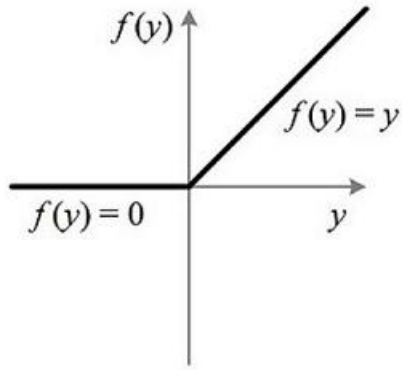


图 2-7　Sigmoid 函数图像

2. ReLU 函数

ReLU 不存在梯度消失的问题,使得模型的收敛速度维持稳定状态。



$$ReLU(x) = \begin{cases} x & if\, x > 0 \\ 0 & if\, x \le 0 \end{cases}$$

图 2-8 ReLU 函数图像

3. tanh 函数

tanh 的输出和输入能够保持非线性单调上升和下降关系。

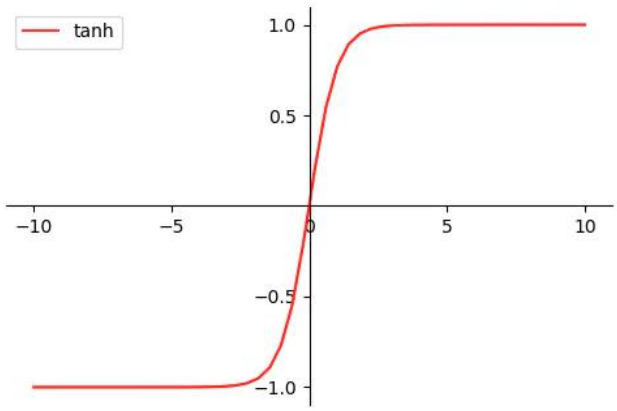


图 2-9 tanh 函数图像

4. Softmax 函数

将多分类的结果以概率的形式展现出来。

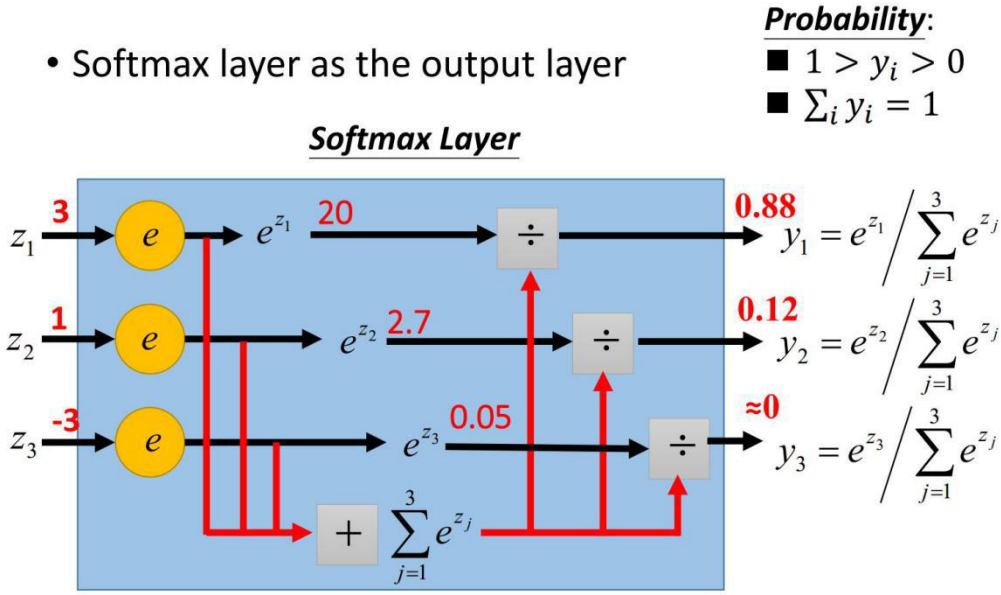


图 2-10　softmax 函数计算方法

2.2.2 特征金字塔网络

在传统深神经网络的训练过程中，特征图的大小通过连续卷积操作逐渐减小。压缩特征图可以减少计算量并改善模型的操作速度。然而，由于特征图的尺寸减小，这种非常抽象的功能在描述小对象时具有一些困难。特征金字塔网络（fpn）[9]。fpn 算法的目的是通过混合高层和低层来融合具有低层和高层意义信息的高分辨率信息，预测每个融合功能而不是融合后的统一预测。为了确保融合不会引起识别错误。

2.2.3 深度残差网络

根据以前的认识，网络水平越深，非线性映射就拟合就越好，特征信息就可以提取更丰富的信息，就可以实现更多的分类类型，网络模型的正确率就更高。但是实际上随着层数的增加，深神经网络也会面临过度拟合和梯度消失的问题。同时，还面临网络退化的问题。He K [10]等为了解决劣化问题引入了深度残差学习框架。深度残差学习框架的主体就是残差块，残差块就是将输出分为两部分，一部分经过正常的卷积网络到输出，另一部分不需要任何操作直接到输出。这样输出始终由两部分组成，能保持原有输入信息，在一定程度上降低了网络退化的风险。

本文中的 FasterR-CNN 的特征提取网络就用了深度残差网络-ResNet50。

2.3 深度学习目标检测模型

2.3.1 Faster R-CNN 网络模型

Faster R-CNN[11]：Fast R-CNN 依赖于外部方法（如 SS 算法）来生成区域建议，但 SS 算法限制了目标检测的精度。有研究表明，CNN 的卷积层在目标定位中表现良好，完全可

以替代 SS 算法生成区域建议。因此 Ren 等人构建了一个高效准确的区域建议网络（RPN）来代替 SS 算法，即提出了 Faster R-CNN。Faster R-CNN 通过 RPN 生成区域建议，并通过 Fast R-CNN 对区域建议进行分类。RPN 和 Fast R-CNN 共享大量的卷积层，从最后一个共享卷积层提取的特征通过独立的分支生成区域建议和分类区域建议。由于 RPN 共享输入图像的卷积特征，可以快速生成区域建议。该算法需要通过两个阶段来完成目标检测。如果数据规模较大，则很难保证系统的实时性。

Faster R-CNN 算法虽然提高了检测精度和速度，被认为是一种真正的基于深度学习的目标检测算法，但在 RPN 网络候选区域的生成过程中使用了不同尺度的锚。如果将其映射到原始图像上，目标的大小可能会变化，感受野可能会不同，从而影响定位的精度。

2.3.2 YOLOv3 网络模型

YOLOv3[12]：YOLOv3 采用 Darknet-53 作为主干网。Darknet-53 具有三个特征：更深；容量大；采用残差结构用于避免梯度消失；下采样采用跨步卷积层来防止有效的信息损失。此外，YOLOV3 还通过学习了特征金字塔网络（FPN），使得网络综合考虑到深层特征和浅层特征，能够更好的检测不同目标。YOLOv3 总体效果很好，可以在保持速度优势的同时提高检测精度，强化小型目标的检测能力。

2.4 深度学习文字识别模型

2.4.1 CRNN 算法

与之前的场景文本识别系统相比，CRNN[13]架构具有 4 个重要特征：

（1）端对端的训练，现有算法的大部分是分开训练和调整；

（2）它能自然处理任意长度序列；

（3）场景的文本识别任务中获得了显著的结果；

（4）在实际应用情形中是实用的，生成更小的模型。

CRNN 对于传统的神经网络模型具有一些明显的优点：

（1）没有详细注释的序列标签，直接学习信息表示；

（2）不需要预处理的程序；

（3）与 RNN 具有一样的特性，生成一系列标签；

（4）没有长度限制，对于具有相同序列的对象，只需要在训练阶段标准化；

（5）场景文本的性能比现有技术更好。

2.4.2 Attention OCR 算法

人类为了在选择性的时间和地点获得信息，将注意力集中在视觉空间的特定部分，将不同的固定点的信息与时间的经过结合起来建立场景的内部表现，并导致未来的眼球运动和决策。集中于场景的几个部分的计算资源可以节省时间，需要较少的“像素”来处

理。根据它的基本作用,在神经科学中人类眼球运动得到了广泛的研究。而低层次的场景属性和自下而上的过程发挥了重要作用,人类固定的位置也显示出强烈的任务特异性[14]。因此外国学者提出了 Recurrent Models of Visual Attention。

该模型是递归神经网络（RNN）,以顺序处理输入,每次处理图像中的不同位置,并逐步组合固定信息以建立场景或环境的动态内部表示。在各步骤中,模型根据过去的信息和任务的要求,选择下一个处理地点。

与 CRNN 算法不同,Attention ocr 算法被用作最终翻译层以提取翻译后的神经网络的特征结果。翻译层网络根据整体图像以 Attention 序列的形式预测字符串序列。基于 Attention 机制,翻译层网络可以有效地将文字检测和语言建模封装到一个模型中,以实现准确的识别。

2.5 本章小结

介绍了集装箱基础知识,利用集装箱基础知识对集装箱进行针对性的处理方案,还介绍了深度学习的有关的基础知识和几种可用到本课题的几种算法。加强了对于人工智能的理解和算法的发展趋势。

第三章　集装箱标识符定位模型

3.1 集装箱标识符定位模型的选型

在完成集装箱标识符识别前，需要在集装箱端面上找到集装箱标识符区域，因此我们的首要目标是将集装箱标识符区域定位出来，由于集装箱标识符区域在集装箱端面上具有特点，我们可以利用深度学习算法将集装箱标识符区域给定位出来，以方便集装箱标识符的识别。

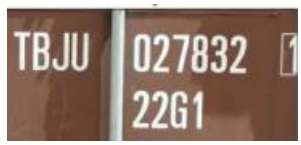


图 3-1　集装箱标识符定位区域

Faster R-CNN 与 YOLOv3 两种算法在不同情景时的速度和准确度方面各自不同。Faster R-CNN 目标检测算法准确率较高[15][16][17]，而且已经被人们集成到目标检测框架中，为了方便学习使用，这也是本文利用 Faster R-CNN 进行集装箱标识符定位的出发点所在。

3.2 集装箱标识符定位模型的架构

我们将箱号区域视为一个整体进行标注，这样就将问题转换为可以用 Faster R-CNN 解决的目标检测问题。由于集装箱箱号区域由字符元素组成的特殊类型对象，这使得集装箱箱号区域定位与其他目标检测有很大区别。我们下面会根据 Faster R-CNN 的基本架构来调整程序参数和分析。

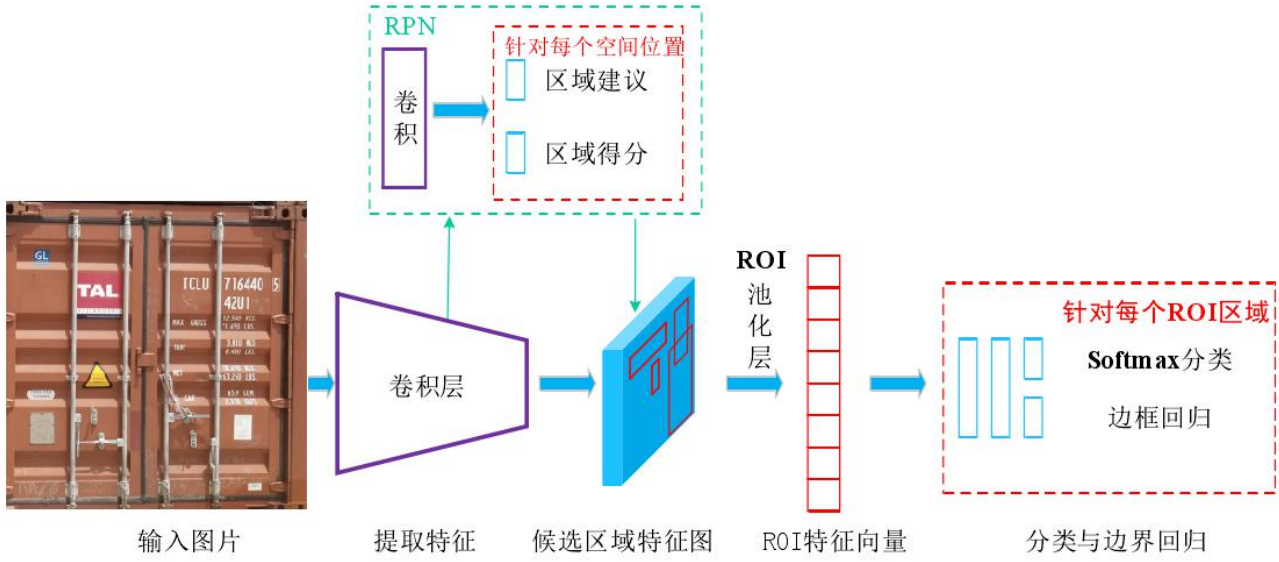
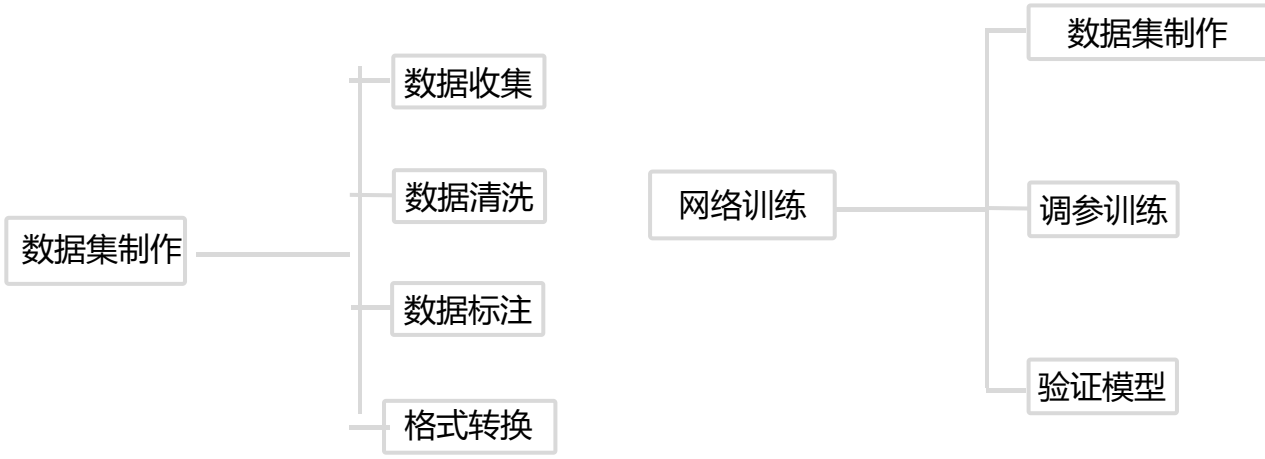


图 3-2　Faster R-CNN 目标检测算法的基本框架

3.3 集装箱标识符定位模型的训练与测试

本小节从集装箱定位数据集的制作入手,随后进行了模型训练和测试,最后进行了误差分析和模型对比总结。



a)数据集制作流程　　b)网络训练流程

图 3-3　流程图

定位模型训练系统软件为：

表 3-1　定位模型训练系统软件环境

序号	软件	序号	软件
1	Ubuntu16.04	5	torchvision:0.6.0
2	torch：1.5.0	6	Python3.6
3	mmdetection2.0	7	cudnn：7.6.5
4	cuda：10.2.89	8	mmcv：0.5.5

集装箱定位数据集图片标注软件为：lablelImg

定位模型训练系统硬件环境：

表 3-2　定位模型训练系统硬件环境

序号	硬件组件
1	英特尔处理器
2	GTX 1050 显卡
3	16 GB 内存（运存）

3.3.1 集装箱标识符定位数据集的制作

一个正常可用的数据集的诞生要经过数据收集,数据清洗,数据标注这 3 个步骤,在数据过少时还可采用数据增强,形成新的数据集。（在数据集的数据过少时,我们通常使用

数据增强——通过对图像进行变换,扩大训练数据集。因为网络模型的准确度与数据集的数据量有密不可分的联系。常见的数据增强方法为：1、随机裁剪和水平翻转；2、颜色调整。）本文的数据来源于天津新工科比赛数据集,因此不需要进行数据收集了。

1.集装箱标识符定位数据集数据清洗

通过一定的方法（如图 3-3 我们将 174. jpg 进行删除处理）将我们收集到的数据进行处理,使其符合我们的使用或者不影响后续使用。通过手动检查等数据分析方式,检

测分析数据中,存在的错误并在此基础上,根据数据源的数量以及缺失或者冗余情况,决定数据转换和清洗步骤。

图 3-3　数据清洗（174. jpg 为清洗掉的图）

2.集装箱定位数据集图像标注

针对本文的标注而言就是将标注框更加贴合集装箱标注集中区域。机器学习算法的训练效果,需要依赖高质量的数据集,如果训练中所使用的标注数据集,存在大量噪声,将会导致机器学习训练不充分,无法获得规律,这样在训练效果验证时会出现目标偏离,无法识别的情况。因此我们在进行标注时一定要细心标注。

在本文的标注中标注软件采用 lablelImg 软件进行人工手动标注。标签如表 3-3 所示。

表 3-3　集装箱定位数据集标签对应区域表

序号	标签	区域
1	Label 1	定位的整体区域
2	Label 2	定位区域的第一行
3	Label 3	定位区域的第二行
4	Label 4	定位区域的第三行

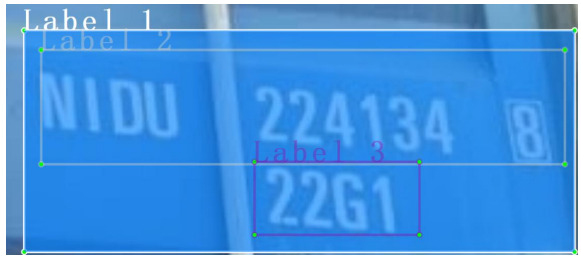


图 3-4　3 个标签

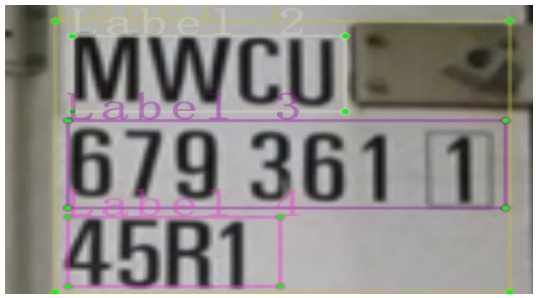


图 3-5　4 个标签

3. 集装箱定位数据集格式转换

因为标注软件生成的集装箱数据集格式与集装箱数据模型训练环境不匹配，需要进行数据集格式转换，即模型训练 Pytorch 环境中目标检测工具箱 mmdetection 需要 voc 格式数据集。

经过集装箱数据集标注工作以后，对应生成了每张标注图片对应的 xml 文件，文件中包含其中包含图片的大小，框选的坐标和框选部分的标签。

```
<annotation>
        <folder>nn</folder>
        <filename>7.jpg</filename>
        <path>C:/001/nn/7.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>630</width>
                <height>630</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>Label 1</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>367</xmin>
                        <ymin>203</ymin>
                        <xmax>594</xmax>
                        <ymax>294</ymax>
                </bndbox>
```

图 3-6　xml 文件文件详情（节选）

制作 voc 数据集时，需要建立 3 个文件夹，名称分别为 Annotations、JPEGImages

、ImageSets, 存入相应的文件, 以下为 linux 环境文件存储格式：

data

|-- Annotations

|存放 xml 文件, 与 JPEGImages 图片一一对应（xml 文件为标注生成的文件, 其中包含图片的大小, 框选的坐标和框选部分的标签）

|-- **JPEGImages**

|所有数据集图片

|-- **ImageSets**

|存放数据集分成的文本文件, 每一行包含图片的名称

3.3.2 网络模型训练与测试

模型训练需要对集装箱数据集进一步的处理, 在训练时, 在程序中将数据集按照 8:2 的比例随机分别为训练集、验证集。训练集和验证集用于模型建立。我们还需要准备与数据集质量相同的照片作为测试照片, 测试照片于模型的结果检测。

1. 用集装箱定位数据集进行模型训练

通常大家会采用预训练的方法,加快训练速度。预训练模型采用 mmdetection 自带的预训练模型。

根据 Linear Scaling Rule,预设初始值 lr=0.025,初始学习率为 0.0025,权重衰减因子为 0.0001,程序为：optimizer=dict(type='SGD',lr=0.0025,momentum=0.9,weight_decay=0.0001)

防止网络过拟合,使用 Step 均匀分步策略改变学习率。训练迭代次数为标准的 12 次 epoch。

```
lr_config = dict(
    policy='step',                          # 优化策略
    warmup='linear',                        # 初始的学习率增加的策略，linear为线性增加
    warmup_iters=500,                       # 在初始的500次迭代中学习率逐渐增加
    warmup_ratio=1.0 / 3,                   # 起始的学习率
    step=[8, 11])                           # 在第8和11个epoch时降低学习率
checkpoint_config = dict(interval=1)        # 每1个epoch存储一次模型
```

图 3-8　程序节选

2. 用验证集测试模型定位识别的结果

在正确的识别集装箱标识符区域后, 我们需要将该部分进行处理, 以方便后面的识别, 由于识别后网络将集装箱标识符区域的坐标告知我们, 我们可以对集装箱标识符区域进行裁剪处理, 处理要求

1. 由于部分图片是翻转了 270 度,我们在识别文字时需要文字处于水平状态。我们利用集装箱标识符区域的从横比不同,有目标的把翻转 270 度图片的裁剪部分给翻转回水平状态 。

2. 由于集装箱标识符区的字符为 2~3 行,我们需要将标签 2,标签 3,标签识别的坐标进行变量保存操作,在本实验中就以打印的形式进行表达。

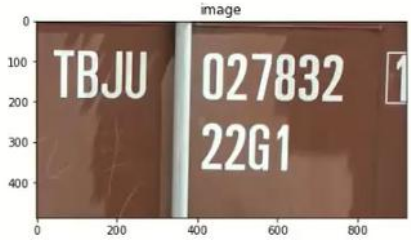


图 3-9 标签一的实验结果展示

3.4 集装箱标识符定位模型评估

经过 800 张集装箱图片训练集和 200 张集装箱图片验证集进行模型参数训练,测试集 100 图片进行模型验证,得到识别正确率 91%。

模型识别准确率与归一化图片大小,数据集训练时的杂乱程度,训练数据的数量等因素有关。

例如：归一化图片大小与模型识别率有关。

本文采用了一个新的数据集,（其中包含 300 张照片,主要可分为以下几类：生成图像 40 张、真实容易识别图像 140 张、白天光照过强图像 45 张、箱面被污损的图像 25 张、箱号倾斜图像 50 张。）将这个数据集按照 8:1:1 的形式进行分为训练集,验证集,测试集。

首先将上述训练集和测试集集装箱图像归一化处理,缩放到不同分辨率后输入训练网络,用训练得到的网络模型对测试集进行测试,从而选择最优的输入图像尺寸。由于定位精度直接影响集装箱标识符的识别效果,本文就采用定位精度来作为评判标准。其中定位精度是指箱号区域与预测框重合程度大于 0.6 时的图像个数与测试集总数的比值。

表 3-4 归一化图片分辨率实验表

分辨率	256*256	320*320	384*384	448*448	512*512
定位精度（%）	94.6	96.6	97.8	97.9	98.2

分析表 3-4 可知,分辨率越高定位精度越高,由于硬件的限制,我们只能把实验进行到 512*512 了,因此就选择 512*512 作为输入图片的分辨率。

3.5 本章小结

介绍了如何选择 Faster R-CNN 算法以其架构和具体执行情况,还介绍了数据集的制作方法和规范,加强了系统化的人工智能学习能力。还对实验结果进行分析力求最大化的完美解决集装箱标识符定位问题。由于本次实验采取的 FasterR-CNN 算法采用了 resnet50 作为特征提取网络,其准确率就高于原始的 VGG16 特征提取网络。又因为标签 2,标签 3 等标签的辅助定位,避免了单一标签定位精度不高的缺点,在代码层面上利用辅助定位标签来扩大和降低标签一的范围。以上的实验都基于辅助标签的存在。辅助标签还有个重要的作用,为 CRNN 识别提供了很好的数据环境。

第四章　集装箱标识符识别模型

4.1 集装箱标识符识别模型的选型

一般来说识别算法有两种设计方案, 一种是端到端识别, 就是用算法直接字符进行识别, 第二种是分割字符, 将字符进行分类, 通过每个字符的分类结果连接得到结果。第二种方案的标注工作量巨大。因此本文采用第一种方案 。

CRNN OCR 和 attention OCR。其实这两大方法在其特征学习阶段都采用了 CNN+RNN 的网络结构主要区别在怎么将网络学习到的序列特征信息转化为最终的识别结果[18][19]。由于 CRNN 的使用方便, 计算量小, 在当前硬件环境下采用 CRNN 更加合理。

4.2 集装箱标识符识别模型的架构

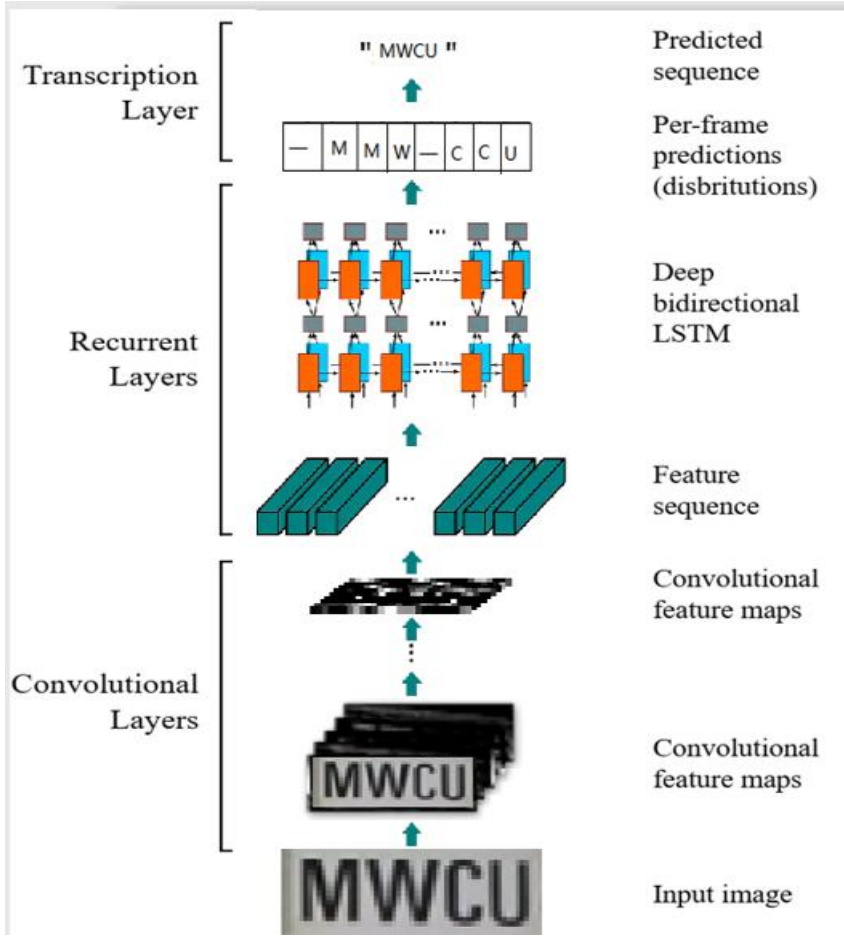


图 4-1　CRNN 架构

整个 CRNN 网络结构[13]包含三部分：

（1）卷积层

网络底部的这些卷积层用于提取特征序列。在图像被输入到这些卷积层之后,不断提取特征信息。在最后一层中,网络形成一个特征向量,这些特征被输入到循环层以用于进一步处理。为了获得深层特征,如字母和数字,CRNN 网络用特征序列表示深层特征。

（2）循环层

从卷积层提取的特征序列被输出到由双向深度循环网络组成的循环层。一个特征序列预测为一个标签。与先前的单个字符识别算法相比,RNN 可以在识别序列中添加字符之间的比较和联系时,使用序列之间的关系来识别序列,识别结果大大提高。为了避免 RNN 的长期依赖性,就设计出来了一个特殊类型的 RNN 网络——短期记忆网络（LSTM）。

LSTM 有 4 个基本部分。在图 4-2 中示出了 LSTM 的基本单位结构。主要包括单元和三个阈门单位,即输入门、输出门和遗忘门。单元模块用于保存输入 X_T 和预先保存的状态信息 H_T-1。LSTM 具有从通过阈值获得添加或删除信息的能力。阈门是选择要通过的信息。当选择性地丢弃存储有被遗忘的阈门的信息时,输入/输出门可以长期存储上下文信息。

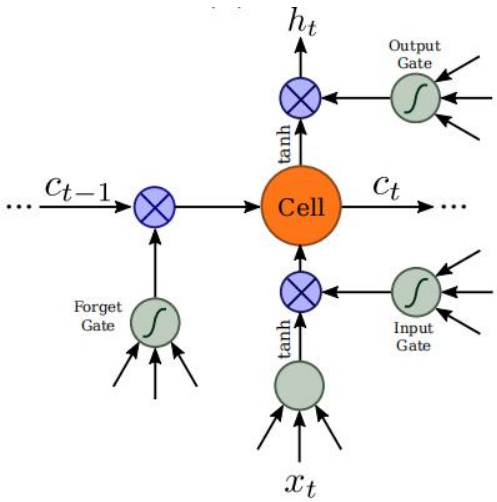


图 4-2 LSTM 基本单元结构

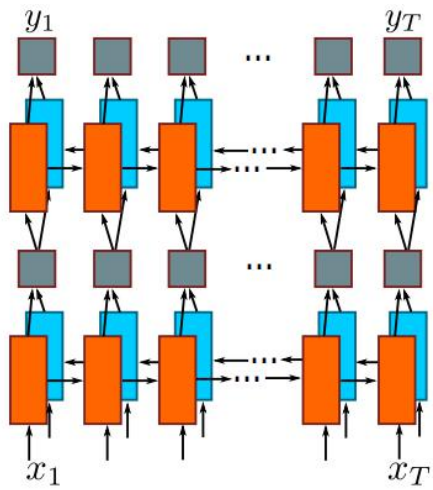


图 4-3 双向 LSTM 深度结构

虽然 LSTM 神经元与过去的信息有关,在基于图像序列的任务中,LSTM 神经元输出不仅与之前的输入有关,还与后续输入相关。通过结合这些信息并补充前者和后者信息,可以提高综合识别率。因此,本文中使用的双向 LSTM,用于前向相关和后向相关。同时,可以形成深度结构来提取深度特征。

（3）转录层

转录就是在预测中找到概率最大的标签序列,转录有两种方式,一种在有词典的情况下,预测首先根据词典进行概率判断,然后根据标签回归。一种是没有词典,直接根据标签进行回归。本文采用有词典的转录。转录层是在 LSTM 网络后连接上一个 CTC 模型,CTC 模型将 LSTM 网络预测的特征序列的结果进行链接处理输出。CTC 模型解决输入数据与

标签的对齐问题，即每个字符在不同情况下的空间大小不同，在网络处理时，就会出现一个标准空间出现一个字符或者几个标准空间出现一个字符。

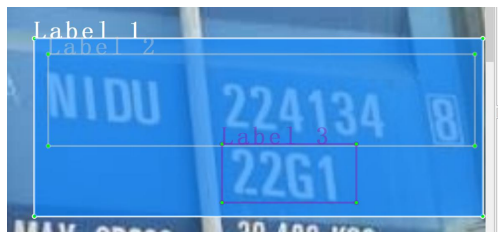
4.3 集装箱标识符识别模型的网络模型训练与测试

4.3.1 集装箱标识符识别数据集的制作

本次进行了两次训练,第一次利用生成的数据集进行预训练。第二次利用真实的数据集进行模型优化训练。

第一次预训练,采用生成的集装箱标识符识别数据集进行训练,如图 4-4,图 a 生成的数据,就是模仿图 b 进行生成的。其中的类似图 b 的 label 2 与 label 3 的原始坐标均是固定值,随着图片随机拉伸的变量不同而不同,但总体偏差不大,因此直接使用原始坐标为 label 2 与 label 3 的坐标。

a）



b）

图 4-4　生成的数据

1、随机拉伸，为了满足现实数据的各种拍摄角度。

2、颜色调整，由于现实中的集装箱颜色丰富，需要增加生成数据的颜色种类。

3、现实环境复杂，因此增加噪声效果。

真实的集装箱标识符识别数据集的不需要额外制作，直接利用第三章训练的模型，直接提取集装箱标识符定位模型识别的结果，直接输入集装箱标识符识别网络进行训练由于本次训练的标签信息都以图片名的形式保存下来了，所以不需要额外标注。即真实的集装箱标识符识别数据集是集装箱图片加上以自身图片相对应的标识符为图片名组成的数据集，不需要其他数据集格式。

a)

b)

图 4-5　标签命名图片

4.3.2 网络模型训练与测试

1. 集装箱标识符识别数据集进行模型训练

在利用生成的数据集进行预训练后,在将真实数据以 9:1 的比例随机分为训练集和验证集,其中训练集 3600 张,验证集是 400 张。由于在第三章已经使用了辅助定位标签,本文把有 2 个辅助定位标签的图像同一标准化为 320*（32*2）,有 3 个辅助定位标签的图像同一标准化为 320*（32*3）。以具有 2 个辅助定位标签的图像为例,利用现辅助定位坐标=原辅助定位坐标*（标准化后的大小/原图片的大小）,将一张图分割为 2 个部分。将 2 部分都标准为 320*32,分离标签分开送入网络,在将结果合并输出。

网络的训练使用随机梯度下降算法来进行训练,通过反向传播纠正误差。本章所采用的 CRNN 具体网络结构参数如表 4-1 所示,该网络共有五层卷积,两个 LSTM 网络的级联。本表以一个辅助标签为例。

表 4-1　CRNN 具体网络结构参数

类型参数	参数
双向 LSTM	256
双向 LSTM	256
卷积 5	512*20
最大下采样 4	512*20*2
卷积 4	512*20*4
最大下采样 3	256*40*4
卷积 3	256*80*8
最大下采样 2	128*80*8
卷积 2	128*160*16
最大下采样 1	64*160*16
卷积 1	64*320*32
输入	320*32

2. 用验证集测试模型识别的结果

本次采用 1000 张真实照片进行模型测试。生成了文本文本存放识别结果,如图 4-6 所示,本次识别准确率为 73.3%,通过猜测定位模型的精度在一定程度上可能影响了本次识别的准确率。

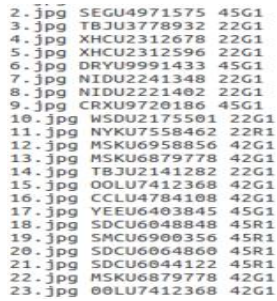


图 4-6　输出结果

4.4 集装箱标识符识别模型网络模型评估

与其他 ocr 方法的对比:

为了方便对比,本次对比采用生成的 10000 张数据进行对比。

表 4-2　OCR 合成数据对比

OCR 方法	识别准确率（%）
传统 OCR	90.8
Attention ocr	98.7
CRNN ocr	97.6

为了引入实际应用,还采取了自然条件下采取 1000 的数据进行对比。

表 4-3　OCR 真实数据对比

OCR 方法	识别准确率（%）
传统 OCR	30.7
Attention ocr	72.6
CRNN ocr	73.3

通过分析可知,传统 OCR 在自然环境下的效果不好,深度学习在自然环境下还具有良好的识效果。对比 Attention ocr 与 CRNN 发现两种方法都是比较接近的。

根据模型复杂度与预测误差的关系图,本文还针对 CRNN 的特征提取网络的进行网络参数调整,保证其符合最佳输出。

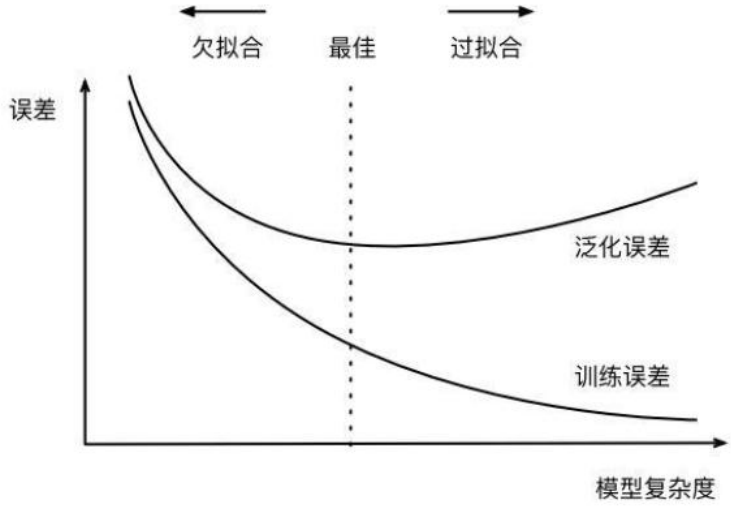


图 4-7　模型复杂度与误差的关系图

利用预训练模型快速训练。由于具有集装箱标识符编码规则,所以我们可以根据改规则进行判断识别结果。参照第二章集装箱标识符编码规则计算设计计算程序。由于集装箱所有人比较固定可以提前锁定表示集装箱所有人的字符,进行字符比对,防止示集装箱所有人的字符错误。

根据返回的识别结果,本文将箱体规格一致的,集装箱所有人一致的,进行分类。可以很清楚的了解,每个规格集装箱的数量和每个集装箱所有人全部的集装箱信息。为了方便了管理,我们在识别集装箱时,根据类别实时记录到不同的文本文件中。

4.5 本章小结

介绍了如何选择 CRNN 算法以其架构和具体执行情况,辅助介绍了下预训练方法和数据增广。还对实验结果进行分析力求最大化的完美解决集装箱标识符识别问题。

第五章　集装箱标识符智能识别系统的设计与调试

5.1 集装箱标识符智能识别系统的设计

本系统实现在嵌入式端直接能通过可视化界面显示识别结果。利用嵌入式设备采集集装箱实时传递数字化的视频帧到PC端,用PC端上的模型进行识别集装箱标识符然后将处理得到的数据传回到嵌入式设备。其运行流程如图 5-1 所示。

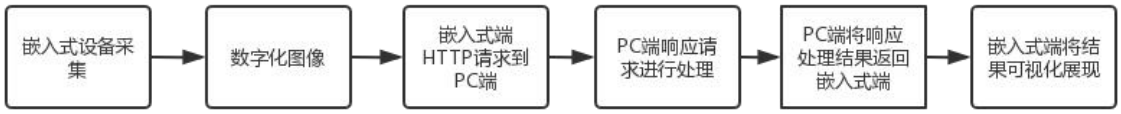


图 5-1　系统运行流程

其中嵌入式设备采用的是500万像素免驱的USB摄像头加上基于liunx系统的嵌入式板-树莓派 4B。数字化图像采用的是 python 中的 opencv—python 库来处理图像。PC 端采用的是 python 编写的 web 框架 Flask 进行 HTTP 响应处理。PC 端的识别采用的是集装箱标识符定位模型与识别模型结合的方式进行处理。嵌入设备通过读取 PC 端的响应结果进行结果分析,然后利用python中的opencv—python库来讲定位区域和识别结果展现在可视化窗口上。

其中嵌入式设备的处理较为简单,能正常使用 USB 摄像头,搭建 python 环境和能正常使用上网功能即可。其中 PC 端 Flask 框架实用了其 HTTP 方法中的 POST 方法,系统的重点还在 PC 端的集装箱标识符的识别处理。

5.2 集装箱标识符识别处理与调试

识别处理的程序的整体运行如图 5-2 所示,针对整体,我们首先将目标检测识别出来的集装箱标识符区域进行预处理,根据标签一坐标直接裁剪,并将其他标签与标签一的相对位子记录,通过经验来说集装箱标识符区域水平状态都是横轴的长度大于纵轴的长度判断该区域是否水平,需要通过 python 中的 image 库来处理,使其旋转到水平状态。然后直接将该区域进行标准化处理,送入 OCR 识别网络中。在送入 OCR 识别网络之前,会将标签二区域和标签三区域分割出来,一同送入 OCR 识别网络。在 OCR 识别网络中,会将标签二区域和标签三区域中的识别结果合并输出。

在得到输出结果后,利用集装箱号判断程序,判断集装箱号是否符合标准,识别结果是否正确,识别的集装箱箱体规格号会跟集装箱箱体规格号表进行对比,判断集装箱箱体规格号是否符合标准,识别结果是否正确。在集装箱号和集装箱箱体规格号都符合标准后,就会进行分类处理。

（1）根据集装箱号前 3 位将同归属人的集装箱进行归类,输出文本文件。

（2）根据集装箱箱体规格号将同一规格的集装箱进行归类,输出文本文件。

（3）在同归属人的集装箱中同一规格的集装箱进行归类,输出文本文件。

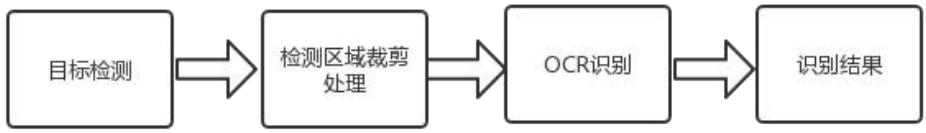


图 5-2　程序运行流程

程序第 4 步的循环识别图片中包含图表的全部功能。程序框架如表 5-1：

表 5-1　程序框架

程序运行	主体程序
1	集装箱标识符定位模型的初始化
2	集装箱标识符识别模型的初始化
3	读取待识别文件
4	循环识别图片
5	分类汇总文件生成

5.2.1 集装箱标识符识别处理功能块的调试

1. 定位模型和识别模型的链接调试

定位模型和识别模型通过 python 的 opencv 库,将图片数据化,通过数据进行传递。集中的关键问题就是定位模型在没有定位出来定位区时,后面的程序没有接受到数据就会报错,为了避免批量识别时的程序停止报错问题,就增加了个判断,只有在定位模型在定位出来定位区的情况才会继续运行,不然就会重新读取新照片直到定位出定位区,定位模型的输出,输出为定位坐标和标签。

2. 定位部分的调试

数据输入是读取图片文件夹中的图片。一般有二种情况。

没有经过处理的图片,即图片名字杂乱没有规律。在读取图片时就采取遍历的方法读取。

事先处理照片,批量改名,从 1.jpg 开始顺序增加。这种方法方便我们最后的人工检查识别结果。

在调用训练好的网络模型时,输入的是通过 python 的 opencv 库数据化的图片。输出的是几个标签的名称和坐标(Xmin,Xmax,Ymin,Ymax)。

调试的重点就是网络输入部分和输出部分,输入要保证能读取到图片,即图片没有问题,能正常打开。输出要保证输出的锚框为概率最大的锚框,不能提取错误,将小概率的锚框给输出[20]。

3. 识别部分的调试

识别部分的数据输入就是通过定位部分,裁剪下的照片,同样是数据化的照片。由于该照片不是一个整体的输入,是在裁剪标签一的基础上裁剪标签二,标签三。再将标签二,标签三的照片同时输入网络,网络输出是两个识别结果。我们要在网络输出后将结果连接成一个结果。即箱号+‘ ’+箱体规格号。三个标签输入网络的情况,输出的是三个结果,也是连接成一个结果。

调试重点也是保证输入和输出。输入就是保证图片为水平,能读取到定位模型发送过来的标签坐标,即在没有输入的情况下识别部分不运行,防止批量识别时,因为没输入导致程序报错停止运行,影响后面的识别。输出就是考虑到输入的不确定性,因为有输入两个标签和输入三个标签的情况。要根据情况利用 if 判断,通过不同的方式将结果进行连接。

4. 分类功能的调试

针对分类功能,首先保证识别的结果符合规律,是正确的输出,利用判断识别程序进行判断。

判断识别程序包括箱号校验程序和集装箱箱体规格号对比程序组成。判断之后利用分类程序进行分类。分类程序由箱号前四个字母比对程序和集装箱箱体规格号对比程序组成。在保存文本文件时,利用分类程序采取识别一条记录一条到文本文件中,即文本文件在识别过程中实时更新。

5.2.2 集装箱标识符识别整体调试结果

本次采用 1000 张真实照片进行模型测试。生成了文本文本存放识别结果,如图 5-3 所示,本次识别准确率为 73.3% ,通过猜测定位模型的精度在一定程度上可能影响了本次识别的准确率。

```
1.jpg TEMU3988040 22G1
2.jpg SEGU4971575 45G1
3.jpg TBJU3778932 22G1
4.jpg XHCU2312678 22G1
5.jpg XHCU2312596 22G1
6.jpg DRYU9991433 45G1
7.jpg NIDU2241348 22G1
8.jpg NIDU2221402 22G1
9.jpg CRXU9720186 45G1
10.jpg WSDU2175501 22G1
11.jpg NYKU7558462 22R1
12.jpg MSKU6958856 42G1
13.jpg MSKU6879778 42G1
14.jpg TBJU2141282 22G1
15.jpg OOLU7412368 42G1
16.jpg CCLU4784108 42G1
17.jpg YEEU6403845 45G1
18.jpg SDCU6048848 45R1
19.jpg SMCU6900356 45R1
20.jpg SDCU6064860 45R1
21.jpg SDCU6044122 45R1
22.jpg MSKU6879778 42G1
23.jpg 00LU7412368 42G1
24.jpg TDTU3219619 22G1
25.jpg TBJU0472390 22G1
26.jpg MWLU2706512 22G1
27.jpg MWCU6791080 45R1
```

图 5-3　调试结果

第六章　总结与展望

6.1 论文工作总结

本文主要利用深度学习中的目标检测和 OCR 识别集成为集装箱标识符识别系统。本文首先用目标检测技术,通过集装箱标识符定位解决 OCR 识别的前置任务；最后基于 CRNN 的序列字符识别技术实现集装箱标识符识别,其目标输入为集装箱标识符分割的几部分图像,输出为集装箱标识符对应的字符序列。下面为本文的工作总结。

1.论文在分析集装箱图像特点的基础上,采用了整体定位集装箱标识符区域,辅助使用分割定位,方便识别集装箱标识符。在实践中发现整体定位集装箱标识符区域的定位精度影响后面的集装箱标识符识别,于是提出来辅助分割定位,提高集装箱标识符区域的定位精度。

2.提出了在集装箱标识符识别后增加判断,以满足集装箱标识符的准确率,并在识别后自动生成文本文件方便管理和记录。

3.在训练上采取了,用生成数据集进行预训练,在一定程度上加快了神经网络的训练。

4.研究深度学习和使用 Faster R-CNN+CRNN 网络,配置 Faster R-CNN,使其能在 GPU 性能不足的情况下工作。

6.2 论文工作展望

虽然在本文的集装箱标识符数据集上基于 Faster R-CNN+CRNN 的方法其效果相对较好,但是还存在一些其他的限制,因此本文的研究还存在一定不足。

1.在本文中,为了便于基于深层学习的集装箱标识符识别系统的开发和实现,在个人笔记本电脑上实现,基本完成了模型的训练和测试。实现算法框架语言是 Python 语言。可以安装到嵌入设备上进行识别。此外,使用服务器来训练所有模型算法,缩短训练时间,提高模型的精度

2.在本文中,为了缩短基于深层学习的集装箱标识符识别系统的开发周期,使用两组算法模型将实现过程分成定位和识别,并进行融合。为了提高模型的运行速度,后面的工作需要考虑使用一个卷积型神经网络的定位识别算法,为了提高模型的精度,鲁棒性,实时性,可以自动生成训练数据。

3.由于深度学习是通过从大量数据中寻找规则。在基于深层学习的集装箱标识符识别系统的开发中,除了实际数据外,还需要合成数据集。以便数据集达到平衡,训练出更好的模型。

在本文中,不考虑集装箱标识符数据集的合成。数据集合的生成结果可能是好的,但是与实际场景中设定的数据不相等。或者后期改进采用大量真实应用场景下的数据,以提高系统的定位准确率和识别准确率。

参考文献

[1] 张索非,冯烨,吴晓富.基于深度卷积神经网络的目标检测算法进展[J].南京邮电大学学报(自然科学版),2019,39(05):72-80.

[2] Yoon Y, Ban K D, Yoon H, et al. Automatic container code recognition from multiple views[J].ETRI Journal,2016,38(04):767-775.

[3] Verma A, Sharma M, Hebbalaguppe R, et al. Automatic container code recognition via spatial transformer networks and connected component region proposals. 2016 15th IEEE International Conference on Machine Learning and Applications(ICMLA)[C]. IEEE,2016:728-733.

[4] 陈力畅,李宇波.基于双卷积神经网络的铁路集装箱号 OCR[J].计算机时代,2019(06):1-4.

[5] 黄深广,翁茂楠,史俞,刘清.基于计算机视觉的集装箱箱号识别[J].港口装卸,2018(01):1-4.

[6] 丁嵩冰,屠舒华,戴越.集装箱箱号校验规则及其应用[J].集装箱化,2006(02):11-13.

[7] Lecun Y, Bottou L,BengioY,et al.Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

[8] 袁梅,全太锋,黄俊,黄洋,胡嘉豪.基于卷积神经网络的烟雾检测[J].重庆邮电大学学报(自然科学版),2020,32(04):620-629.

[9] Lin T Y, Dollár P, Girshick R, et al. Feature Pyramid Networks for Object Detection [C]. IEEE Conference on Computer Vision and Pattern Recognition, 2017:1-9.

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

[11] Ren S, HeK M, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,2015, 39(6):1137-1149.

[12] 陈正斌,叶东毅,朱彩霞,等.基于改进 YOLOv3 的目标识别方法[J].计算机系统应用,2020,29(01):49-58.

[13] 赵龙,李飞,王伟峰.基于 PSENet 和 CRNN 的身份证识别[J].现代计算机,2020(34):78-82.

[14] 陈榕,任崇广,王智远,曲志坚,王海鹏.基于注意力机制的 CRNN 文本分类算法[J].计算机工程与设计,2019,40(11):3151-3157.

[15] 崔循.基于深度学习的集装箱箱号自动识别[D].长春理工大学,2020.

[16] 周凯龙.基于深度学习的图像识别应用研究[D].北京工业大学,2016.

[17] 孟琪.基于深度学习的集装箱标识符识别[D].山东科技大学,2018.

[18] 刘易.基于神经网络的集装箱箱号识别系统[D].上海:上海交通大学,2018.

[19] 张索非,冯烨,吴晓富.基于深度卷积神经网络的目标检测算法进展[J].南京邮电大学学报(自然科学版),2019,39(05):72-80.

[20] 李奥.利用深度学习进行目标检测[J].中国设备工程,2018,409(23):165-167.

致　谢

不知不觉已然毕业了,从大一上学期不知所谓的玩闹,到大一下学期的不甘现状——通过智能制造学院的工程实践创新社接触 PCB,嵌入式等相关技术。奈何只学会了皮毛,大三通过嵌入式做了个浅显的嵌入式翻译器和简单的智能家居。在此感谢一路成长的社团伙伴们,我们一起学会了合作,学会了项目流程图的编写,感谢社团的指导老师刘艺柱老师,教会了我们做 DIY 的一些思路和纲领性的思想。

大三几个浅显的 DIY 接触到了人工智能。机遇巧合下得知我的指导老师孟祥懿老师在人工智能方向有研究,通过孟老师去了华大科技实习学习了和了解人工智能目标识别的相关知识。在此要感谢公司同事的帮助和父母的赞同以及无私的帮助。

在实习期间利用学习到的相关知识,参加了华为的集装箱识别比赛,本次毕业论文就利用的本次比赛的成果进行编写。我想在这里感谢孟祥懿老师,孟老师传授的各种经验给了我很多启发,引导了我的成长。

说到性格上的变化,就必须感谢教液压与气动系统装调的邹炳燕老师,邹老师的教育使我的急性子慢了一点,以及教机械设计基础的孙学娟老师,孙老师的教育使我充满阳光,相信自己能行。

其次,感谢我的班主任张培老师,就像邻家的大姐姐一样给了我们许多的帮助,不断的引导我们进行思考和进行一些 DIY,加强了我动手能力和思考能力。还要感谢生活和学习一路上的伙伴。

针对本次论文的选题,写作思路,架构上的完善,都离不开指导老师孟老师的帮助,一有问题孟老师都会在百忙之中抽出时间对我进行指导不顾劳累,提出了许多建设性的建议,使我不断的完善论文。在此再一次感谢孟老师的指导。我的论文作品不是很成熟,还有很多缺点。希望在以后的工作学习中能够记住本次论文的各种解决问题的方式,不轻言放弃。

大学四年生活已经结束,希望在以后的工作学习中能够记住母校的校训“崇实,求精,致良知”—脚踏实地的做事,注重细节和创新,能做到知行合一。

最后,我要感谢在百忙之中抽出时间来参与我的论文评审的所有老师。

附　录

附录一 中文译文及外文资料

中文译文:

Faster R-CNN：面向实时目标检测的区域提议网络

3 Faster R-CNN

我们的目标检测系统称为 Faster R-CNN,由两个模块组成。第一个模块是提出区域的深度全卷积网络,第二个模块是使用提出区域的 FasterR-CNN 检测器[2]。整个系统是一个单一的、统一的目标检测网络（图 2）。RPN 模块使用最近流行的带有“attention”机制的神经网络术语,告诉 FasterR-CNN 模块去哪里寻找。在第 3.1 节中,我们介绍了区域建议网络的设计和特性。在第 3.2 节中,我们开发了用于训练两个具有共享特性的模块的算法。

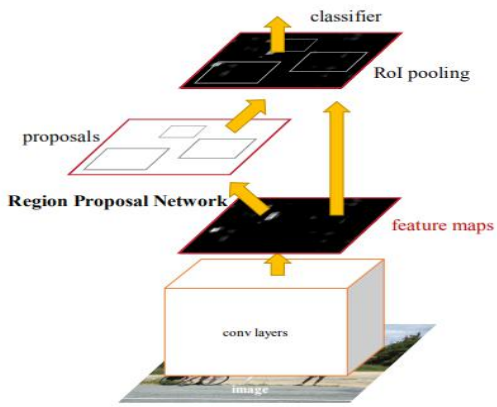


图 2　R-CNN 是一个单一的、统一的目标检测网络。RPN 模块是这个统一网络的“关注点”。

3.1 区域提案网络

区域建议网络（RPN）将图像（任何大小）作为输入,并输出一组矩形对象建议,每个建议都有一个对象性得分。我们用一个完全卷积网络来模拟这个过程,我们将在本节中对其进行介绍。因为我们的最终目标是与 Fast R-CNN 目标检测网络共享计算[2],我们假设两个网络共享一组共同的卷积层。在我们的实验中,我们研究了 Zeiler 和 Fergus 模型[32]（ZF）,具有 5 个可共享的卷积层,以及 Simonyan 和 Zisserman 模型[3]（VGG-16）,具有 13 个可共享的卷积层。

为了生成建议区域,我们在最后一个共享卷积层输出的卷积特征图上滑动一个小网络。这个小网络将 n×n 的窗口作为输入输入卷积特征图 。每次滑动窗口被映射到一个低维特征（对于 ZF 为 256-d,对于 VGG 为 512-d,含有 ReLU[33]）。这个特性被输入到两个同级完全连接的层中——一个回归层（reg）和一个分类层（cls）。我们在本文中使用 n=3,

有效接受领域输入图像较大（ZF 和 VGG 分别为 171 和 228 像素）。图 3（左）中的单个位置显示了这个小型网络。请注意,由于小型网络以滑动窗口方式运行,因此完全连接的层在所有空间位置上共享。该体系结构自然地由一个 n×n 卷积层和两个同级 1×1 卷积层（分别用于 reg 和 cls）实现。

3.1.1 锚

在每个滑动窗口位置,我们同时预测多个建议区域,其中每个位置的最大可能建议数表示为 K。因此,reg 层有 4k 输出编码 k 个框的坐标,cls 层输出 2k 分数,用于估计每个提议的目标或非目标概率。这 k 个提案是-相对于 k 个参考框而确定的,我们称之为锚. 锚定位于所讨论的滑动窗口的中心,并与比例和纵横比相关联（图 3,左侧）。默认情况下,我们使用 3 个比例和 3 个纵横比,在每个滑动位置产生 k=9 个锚点。对于尺寸为 w×H（通常为 2400）的卷积特征图,总共有 W*H*K 个锚。

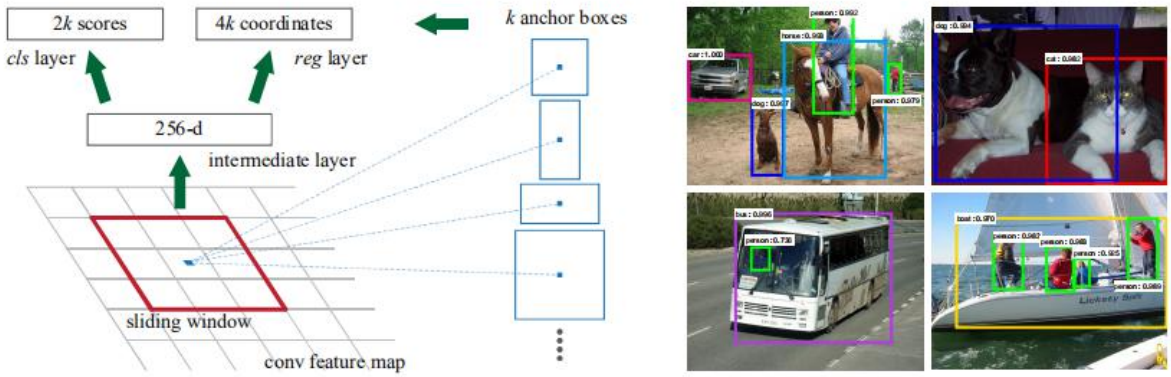


图 3:左:区域提议网络(RPN)。右:在 PASCALVOC 2007 测试中使用 RPN 提案的检测示例。我们的检测对象在一个广泛的尺度和长宽比。

平移不变锚

我们使用的方法一个重要特性就是-它是平移不变的,无论是在锚和计算相对于锚的建议区域的函数方面。如果一个人翻译图像中的一个对象,那么翻译相同对象就应该能够预测在任何位置上的对象建议区域。我们的方法 5 保证了这种平移不变性。作为比较,MultiBox 方法[27]使用 k-means 生成 800 个锚,这些锚不是平移不变的。因此,MultiBox 不能保证在翻译对象时生成相同的建议区域。

平移不变特性还减小了模型的大小。MultiBox 有一个（4+1）×800 维的全连通输出层,而我们的方法有一个（4+2）×9 维的卷积输出层,在 k=9 个锚的情况下。结果,我们的输出层有 2.8×10^4 个参数（VGG-16 为 512×（4+2）×9）,比 MultiBox 的输出层有 6.1 $\times10^6$ 个参数少了两个数量级.对于 MultiBox 中的 GoogleNet[34],为(1536×(4 +1)×800))。如果考虑到特征投影层,我们的建议层的参数仍然比 MultiBox6 少一个数量级。我们希望我们的方法在小数据集（如 pascalvoc）上具有较小的过度拟合风险。

外文资料：

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

1 Introduction

Recent advances in object detection are driven by the success of region proposal methods (*e.g.*, [4]) and region-based convolutional neural networks (R-CNNs) [5]. Although region-based CNNs were computationally expensive as originally developed in [5], their cost has been drastically reduced thanks to sharing convolutions across proposals [1], [2]. The latest incarnation, Fast R-CNN [2], achieves near real-time rates using very deep networks [3], *when ignoring the time spent on region proposals*. Now, proposals are the test-time computational bottleneck in state-of-the-art detection systems.

Region proposal methods typically rely on inexpensive features and economical inference schemes. Selective Search [4], one of the most popular methods, greedily merges superpixels based on engineered low-level features. Yet when compared to efficient detection networks [2], Selective Search is an order of magnitude slower, at 2 seconds per image in a CPU implementation. EdgeBoxes [6] currently provides the best tradeoff between proposal quality and speed, at 0.2 seconds per image. Nevertheless, the region proposal step still consumes as much running time as the detection network.

- *S. Ren is with University of Science and Technology of China, Hefei, China. This work was done when S. Ren was an intern at Microsoft Research. Email: sqren@mail.ustc.edu.cn*
- *K. He and J. Sun are with Visual Computing Group, Microsoft Research. E-mail: {kahe,jiansun}@microsoft.com*
- *R. Girshick is with Facebook AI Research. The majority of this work was done when R. Girshick was with Microsoft Research. E-mail: rbg@fb.com*

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to re-implement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses important opportunities for sharing computation.

In this paper, we show that an algorithmic change—computing proposals with a deep convolutional neural network—leads to an elegant and effective solution where proposal computation is nearly cost-free given the detection network's computation. To this end, we introduce novel *Region Proposal Networks* (RPNs) that share convolutional layers with state-of-the-art object detection networks [1], [2]. By sharing convolutions at test-time, the marginal cost for computing proposals is small (*e.g.*, 10ms per image).

Our observation is that the convolutional feature maps used by region-based detectors, like Fast R-CNN, can also be used for generating region proposals. On top of these convolutional features, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. The RPN is thus a kind of fully convolutional network (FCN) [7] and can be trained end-to-end specifically for the task for generating detection proposals.

RPNs are designed to efficiently predict region proposals with a wide range of scales and aspect ratios. In contrast to prevalent methods [8], [9], [1], [2] that use

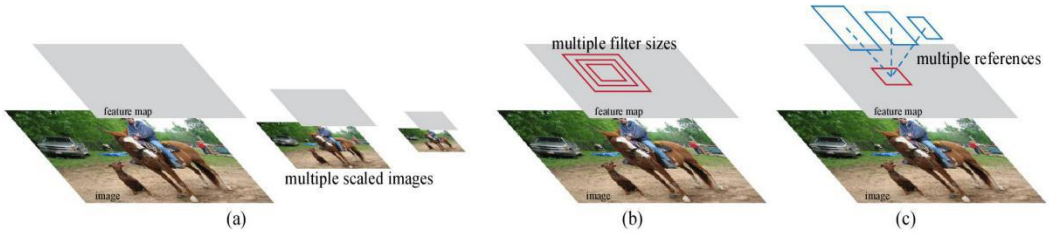


Figure 1: Different schemes for addressing multiple scales and sizes. (a) Pyramids of images and feature maps are built, and the classifier is run at all scales. (b) Pyramids of filters with multiple scales/sizes are run on the feature map. (c) We use pyramids of reference boxes in the regression functions.

pyramids of images (Figure 1, a) or pyramids of filters (Figure 1, b), we introduce novel "anchor" boxes that serve as references at multiple scales and aspect ratios. Our scheme can be thought of as a pyramid of regression references (Figure 1, c), which avoids enumerating images or filters of multiple scales or aspect ratios. This model performs well when trained and tested using single-scale images and thus benefits running speed.

To unify RPNs with Fast R-CNN [2] object detection networks, we propose a training scheme that alternates between fine-tuning for the region proposal task and then fine-tuning for object detection, while keeping the proposals fixed. This scheme converges quickly and produces a unified network with convolutional features that are shared between both tasks.[1]

We comprehensively evaluate our method on the PASCAL VOC detection benchmarks [11] where RPNs with Fast R-CNNs produce detection accuracy better than the strong baseline of Selective Search with Fast R-CNNs. Meanwhile, our method waives nearly all computational burdens of Selective Search at test-time—the effective running time for proposals is just 10 milliseconds. Using the expensive very deep models of [3], our detection method still has a frame rate of 5fps (*including all steps*) on a GPU, and thus is a practical object detection system in terms of both speed and accuracy. We also report results on the MS COCO dataset [12] and investigate the improvements on PASCAL VOC using the COCO data. Code has been made publicly available at `https://github.com/shaoqingren/faster_rcnn` (in MATLAB) and `https://github.com/rbgirshick/py-faster-rcnn` (in Python).

A preliminary version of this manuscript was published previously [10]. Since then, the frameworks of RPN and Faster R-CNN have been adopted and generalized to other methods, such as 3D object detection [13], part-based detection [14], instance segmentation [15], and image captioning [16]. Our fast and effective object detection system has also been built in commercial systems such as at Pinterests [17], with user engagement improvements reported.

In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the basis of several 1st-place entries [18] in the tracks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. RPNs completely learn to propose regions from data, and thus can easily benefit from deeper and more expressive features (such as the 101-layer residual nets adopted in [18]). Faster R-CNN and RPN are also used by several other leading entries in these competitions[2]. These results suggest that our method is not only a cost-efficient solution for practical usage, but also an effective way of improving object detection accuracy.

2 Related Work

Object Proposals. There is a large literature on object proposal methods. Comprehensive surveys and comparisons of object proposal methods can be found in [19], [20], [21]. Widely used object proposal methods include those based on grouping super-pixels (*e.g.*, Selective Search [4], CPMC [22], MCG [23]) and those based on sliding windows (*e.g.*, objectness in windows [24], EdgeBoxes [6]). Object proposal methods were adopted as external modules independent of the detectors (*e.g.*, Selective Search [4] object detectors, R-CNN [5], and Fast R-CNN [2]).

Deep Networks for Object Detection. The R-CNN method [5] trains CNNs end-to-end to classify the proposal regions into object categories or background. R-CNN mainly plays as a classifier, and it does not predict object bounds (except for refining by bounding box regression). Its accuracy depends on the performance of the region proposal module (see comparisons in [20]). Several papers have proposed ways of using deep networks for predicting object bounding boxes [25], [9], [26], [27]. In the OverFeat method [9], a fully-connected layer is trained to predict the box coordinates for the localization task that assumes a single object. The fully-connected layer is then turned

1. Since the publication of the conference version of this paper [10], we have also found that RPNs can be trained jointly with Fast R-CNN networks leading to less training time.

2. http://image-net.org/challenges/LSVRC/2015/results

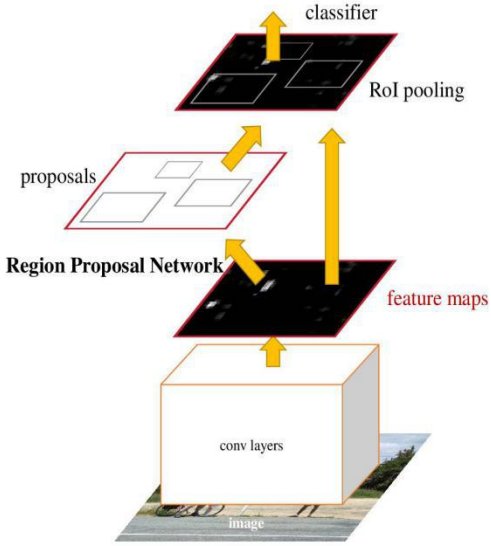


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

into a convolutional layer for detecting multiple class-specific objects. The MultiBox methods [26], [27] generate region proposals from a network whose last fully-connected layer simultaneously predicts multiple class-agnostic boxes, generalizing the "single-box" fashion of OverFeat. These class-agnostic boxes are used as proposals for R-CNN [5]. The MultiBox proposal network is applied on a single image crop or multiple large image crops (*e.g.*, 224×224), in contrast to our fully convolutional scheme. MultiBox does not share features between the proposal and detection networks. We discuss OverFeat and MultiBox in more depth later in context with our method. Concurrent with our work, the DeepMask method [28] is developed for learning segmentation proposals.

Shared computation of convolutions [9], [1], [29], [7], [2] has been attracting increasing attention for efficient, yet accurate, visual recognition. The OverFeat paper [9] computes convolutional features from an image pyramid for classification, localization, and detection. Adaptively-sized pooling (SPP) [1] on shared convolutional feature maps is developed for efficient region-based object detection [1], [30] and semantic segmentation [29]. Fast R-CNN [2] enables end-to-end detector training on shared convolutional features and shows compelling accuracy and speed.

3 Faster R-CNN

Our object detection system, called Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector [2] that uses the proposed regions. The entire system is a single, unified network for object detection (Figure 2). Using the recently popular terminology of neural networks with 'attention' [31] mechanisms, the RPN module tells the Fast R-CNN module where to look. In Section 3.1 we introduce the designs and properties of the network for region proposal. In Section 3.2 we develop algorithms for training both modules with features shared.

3.1 Region Proposal Networks

A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score.[3] We model this process with a fully convolutional network [7], which we describe in this section. Because our ultimate goal is to share computation with a Fast R-CNN object detection network [2], we assume that both nets share a common set of convolutional layers. In our experiments, we investigate the Zeiler and Fergus model [32] (ZF), which has 5 shareable convolutional layers and the Simonyan and Zisserman model [3] (VGG-16), which has 13 shareable convolutional layers.

To generate region proposals, we slide a small network over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an $n \times n$ spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature (256-d for ZF and 512-d for VGG, with ReLU [33] following). This feature is fed into two sibling fully-connected layers—a box-regression layer (*reg*) and a box-classification layer (*cls*). We use $n = 3$ in this paper, noting that the effective receptive field on the input image is large (171 and 228 pixels for ZF and VGG, respectively). This mini-network is illustrated at a single position in Figure 3 (left). Note that because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. This architecture is naturally implemented with an $n \times n$ convolutional layer followed by two sibling 1×1 convolutional layers (for *reg* and *cls*, respectively).

3.1.1 Anchors

At each sliding-window location, we simultaneously predict multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. So the *reg* layer has $4k$ outputs encoding the coordinates of k boxes, and the *cls* layer outputs $2k$ scores that estimate probability of object or not object for each proposal[4]. The k proposals are parameterized *relative* to k reference boxes, which we call

3. "Region" is a generic term and in this paper we only consider *rectangular* regions, as is common for many methods (*e.g.*, [27], [4], [6]). "Objectness" measures membership to a set of object classes *vs.* background.

4. For simplicity we implement the *cls* layer as a two-class softmax layer. Alternatively, one may use logistic regression to produce k scores.

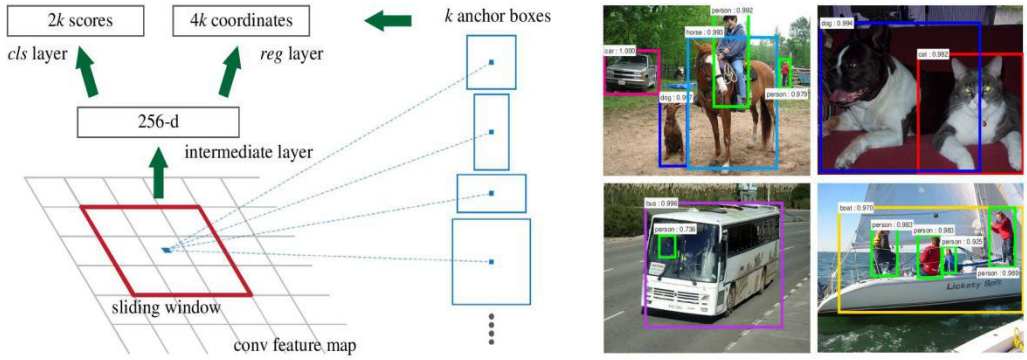


Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

anchors. An anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio (Figure 3, left). By default we use 3 scales and 3 aspect ratios, yielding $k = 9$ anchors at each sliding position. For a convolutional feature map of a size $W \times H$ (typically ~2,400), there are WHk anchors in total.

Translation-Invariant Anchors

An important property of our approach is that it is *translation invariant*, both in terms of the anchors and the functions that compute proposals relative to the anchors. If one translates an object in an image, the proposal should translate and the same function should be able to predict the proposal in either location. This translation-invariant property is guaranteed by our method[5]. As a comparison, the MultiBox method [27] uses k-means to generate 800 anchors, which are *not* translation invariant. So MultiBox does not guarantee that the same proposal is generated if an object is translated.

The translation-invariant property also reduces the model size. MultiBox has a $(4+1) \times 800$-dimensional fully-connected output layer, whereas our method has a $(4+2) \times 9$-dimensional convolutional output layer in the case of $k = 9$ anchors. As a result, our output layer has 2.8×10^4 parameters ($512 \times (4+2) \times 9$ for VGG-16), two orders of magnitude fewer than MultiBox's output layer that has 6.1×10^6 parameters ($1536 \times (4+1) \times 800$ for GoogleNet [34] in MultiBox [27]). If considering the feature projection layers, our proposal layers still have an order of magnitude fewer parameters than MultiBox[6]. We expect our method to have less risk of overfitting on small datasets, like PASCAL VOC.

5. As is the case of FCNs [7], our network is translation invariant up to the network's total stride.

6. Considering the feature projection layers, our proposal layers' parameter count is $3 \times 3 \times 512 \times 512 + 512 \times 6 \times 9 = 2.4 \times 10^6$; MultiBox's proposal layers' parameter count is $7 \times 7 \times (64+96+64+64) \times 1536 + 1536 \times 5 \times 800 = 27 \times 10^6$.

Multi-Scale Anchors as Regression References

Our design of anchors presents a novel scheme for addressing multiple scales (and aspect ratios). As shown in Figure 1, there have been two popular ways for multi-scale predictions. The first way is based on image/feature pyramids, *e.g.*, in DPM [8] and CNN-based methods [9], [1], [2]. The images are resized at multiple scales, and feature maps (HOG [8] or deep convolutional features [9], [1], [2]) are computed for each scale (Figure 1(a)). This way is often useful but is time-consuming. The second way is to use sliding windows of multiple scales (and/or aspect ratios) on the feature maps. For example, in DPM [8], models of different aspect ratios are trained separately using different filter sizes (such as 5×7 and 7×5). If this way is used to address multiple scales, it can be thought of as a "pyramid of filters" (Figure 1(b)). The second way is usually adopted jointly with the first way [8].

As a comparison, our anchor-based method is built on *a pyramid of anchors*, which is more cost-efficient. Our method classifies and regresses bounding boxes with reference to anchor boxes of multiple scales and aspect ratios. It only relies on images and feature maps of a single scale, and uses filters (sliding windows on the feature map) of a single size. We show by experiments the effects of this scheme for addressing multiple scales and sizes (Table 8).

Because of this multi-scale design based on anchors, we can simply use the convolutional features computed on a single-scale image, as is also done by the Fast R-CNN detector [2]. The design of multi-scale anchors is a key component for sharing features without extra cost for addressing scales.

3.1.2 Loss Function

For training RPNs, we assign a binary class label (of being an object or not) to each anchor. We assign a positive label to two kinds of anchors: (i) the anchor/anchors with the highest Intersection-over-Union (IoU) overlap with a ground-truth box, *or* (ii) an anchor that has an IoU overlap higher than 0.7 with

附录二 主要程序

```
import argparse
import cv2 as cv
import torch
import numpy as np
import torch.nn.functional as F
import torch.nn as nn
import os
from time import time
from flask import Flask, request
import base64
from PIL import Image
import PIL.Image
from mmdet.apis import init_detector,
inference_detector
import mmcv
config_file =
'other/object/faster_rcnn_r50_fpn_1x.py'
checkpoint_file = 'model/object.pth'
model1 = init_detector(config_file, checkpoint_file,
device='cuda:0')
#PICS_PATH = "../data/train"
def show(img, result, class_names):
    assert isinstance(class_names, (tuple, list))
    img = mmcv.imread(img)
    img = img.copy()
    if isinstance(result, tuple):
        bbox_result, segm_result = result
    else:
        bbox_result, segm_result = result, None
    bboxes = np.vstack(bbox_result)
    # draw bounding boxes
    labels = [
        np.full(bbox.shape[0], i, dtype=np.int32)
        for i, bbox in enumerate(bbox_result)
    ]
    labels = np.concatenate(labels)
    #print (bboxes)
    #print(labels)
    return bboxes, labels
def get_bboxes(bboxes, labels, class_names,
score_thr=0):
    assert bboxes.ndim == 2
    assert labels.ndim == 1
    assert bboxes.shape[0] == labels.shape[0]
    assert bboxes.shape[1] == 4 or bboxes.shape[1]
== 5

    if score_thr > 0:
        assert bboxes.shape[1] == 5
        scores = bboxes[:, -1]
        inds = scores > score_thr
        bboxes = bboxes[inds, :]
        labels = labels[inds]

    dict_list = []
    for bbox, label in zip(bboxes, labels):
        bbox_int = bbox.astype(np.int32)
        label_text = class_names[
            label] if class_names is not None
else 'cls {}'.format(label)
        label_text1 = class_names[
            label] if class_names is not None
else 'cls {}'.format(label)
        if len(bbox) > 4:
            label_text +=
'|{:.02f}'.format(bbox[-1])
        obj_dict = {}
        obj_dict['similarity'] = label_text
        obj_dict['name'] = label_text1
        obj_dict['xmin'] = str(bbox_int[0])
        obj_dict['ymax'] = str(bbox_int[1])
        obj_dict['xmax'] = str(bbox_int[2])
        obj_dict['ymin'] = str(bbox_int[3])
        dict_list.append(obj_dict)
    #print('{:.02f}'.format(bbox[0]))
    #print(bbox_int)
    #print(dict_list)
    return
dict_list[0]['ymin'],dict_list[0]['ymax'],dict_list[0]
['xmin'],dict_list[0]['xmax']

CHARS = ["0", "1", "2", "3", "4", "5", "6",
"7", "8", "9", "A", "B",
         "C", "D", "E", "F", "G", "H", "J",
```

```
"K", "L", "M", "N", "P",
                "Q", "R", "S", "T", "U", "V",
"W", "X", "Y", "Z", "\r","I"," ", "O",
"-","\t" ]
ap = argparse.ArgumentParser()
ap.add_argument("path",
                        help="path to test image")
args = vars(ap.parse_args())
PICS_PATH = args["path"]

def parseOutput(output):
    label = ""
    last_char = ''
    last_is_char = -1   # 上一个char 是'-'时等于0,不是'-'时等于 1,初始值-1
    for i in range(output.shape[0]):
        latter = CHARS[output[i]]
        if latter == "-":
            last_is_char = 0
        else:
            if i > 0 and latter == last_char and last_is_char == 1:
                continue
            label += latter
            last_char = latter
            last_is_char = 1
    return label


class FeatureMap(torch.nn.Module):
    def __init__(self, batch):
        super(FeatureMap, self).__init__()
        self.batch = batch

    def forward(self, x):
        x = torch.split(x, 2, dim=3)
        tl = []
        for i in range(len(x)):
            tmp = x[i].reshape(self.batch, 512)
            tl.append(tmp)
        out = torch.stack(tl, dim=1)
        return out

class Net(torch.nn.Module):
    def __init__(self, batch, device, num_layers):
        super(Net, self).__init__()
        self.batch = batch
        self.device = device
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 32, 3, padding=1)
        # an affine operation: y = Wx + b
        self.num_layers = num_layers
        self.gru1 = nn.GRU(512, 128, num_layers=self.num_layers, bidirectional=True, dropout=0.3)
        self.fm = FeatureMap(self.batch)
        self.fc = nn.Linear(256, 40)
        #2*10    4*20 8*40 16*80 32*160

    def forward(self, x):
        x = F.leaky_relu(self.conv1(x))
        x = F.leaky_relu(self.conv2(x))
        x = F.max_pool2d(x, (2, 2))
        x = F.leaky_relu(self.conv3(x))
        x = F.leaky_relu(self.conv4(x))
        x = F.max_pool2d(x, (2, 2))
        x = self.fm(x)
        x = x.permute(1, 0, 2)
        x, h = self.gru1(x)
        x = self.fc(x)
        x = F.log_softmax(x, dim=2)
        return x
def characterConversion(str):
        basic_character = "0123456789A?BCDEFGHIJK?LMNOPQRSTU?VWXYZ"
        for i in range(0, len(basic_character)):
                if str == basic_character[i]:
                        return i
def CheckCartonNo(containerId): #判断合理性
        sum = 0
        containerId_list = list(containerId)
```

```
        for i in range(0, 10):

            sum +=
characterConversion(containerId[i]) * (2 ** i)

        print(sum)
        check_digit = sum % 11 % 10
        print(check_digit)

        if int(check_digit) ==
int(containerId_list[-5]):
            return 0 #通过
        else:
            return 1
def pan(output_label):          #处理数据
    w1=output_label
    output_label=output_label
    while(len(output_label)>15):


            ww = [0, 0, 0, 0, 0]

            w = output_label

            i = 0
            n = 0
            n1 = -1
            n2 = 0
            n3 = 1
            z = 0
            if (len(output_label) >15):
                if
(len(output_label) >= 16 &
c(output_label[4])>=10 ): # -#
                    for i in range(6):

                        if
(c(output_label[i]) == c(output_label[i + 1])):

output_label = output_label[0:i] + output_label[i +
1:]

                if
(len(output_label)>=17): # -2two
                    for i in
range(len(output_label)-3):
                        if
(c(output_label[i:i+1]) == c(output_label[i+2:i+3])):

output_label = output_label[0:i+2] +
output_label[i+4:]
                if
(len(output_label) >= 16):
                    for i in
range(len(output_label)-1):
                        if
(c(output_label[i])==c(output_label[i+1])):    # -
two
                            n
+=1

n1 = i

print(1000)

print(n)

print(output_label)
                if (n == 1 & n1>4):
                    output_label =
output_label[0:ww[n - 1]] + output_label[ww[n - 1]
+ 1:]

                if (n >= 1):
                    while (n >= 1):
                        n2 = n

output_label = output_label[0:ww[n2 - 1]] +
output_label[ww[n2 - 1] + 1:]
                        n -= 1

                if
(len(output_label) >= 16):
                    output_label =
output_label[0:4]+output_label[4:11]+output_label[
-4:]
            if (len(output_label) ==15):
```

```
if(CheckCartonNo(output_label)==0):
                                file_handle =
open('r.txt', mode='a')

file_handle.write(output_label + "\n")
                                file_handle.close()
                        if (len(output_label) < 15):
                                file_handle =
open('jia.txt', mode='a')

file_handle.write(output_label + "\n")
                                file_handle.close()
        return output_label

def main():
        pics = os.listdir(PICS_PATH)      #读取文件
        use_cuda = torch.cuda.is_available()
        device = torch.device("cuda" if use_cuda else
"cpu")
        model = Net(1, device, 2).to(device)

model.load_state_dict(torch.load("model/ocr.pt"))
        model.eval()   #初始化识别模型

        right_count = 0
        for i in range(len(pics)):
                label = pics[i][:]
                img = PICS_PATH + "/" + label
                result = inference_detector(model1, img)
                class_names = model1.CLASSES
                bboxes, labels = show(img, result,
class_names)
                y1, y2, x1, x2 = get_bboxes(bboxes,
labels, class_names, score_thr=0)
                img = cv.imread(img)
                img = img[int(y2):int(y1), int(x1):int(x2)]
                img = cv.resize(img, (320, 32))
                r, g, b = cv.split(img)
                numpy_array = np.array([r, g, b])
                img_tensor =
torch.from_numpy(numpy_array)
                img_tensor = img_tensor.float()
                img_tensor /= 256
                img_tensor = img_tensor.reshape([1, 3,
32, 320])
                img_tensor = img_tensor.to(device)

                t1 = time()
                output = model(img_tensor).cpu()
                output = torch.squeeze(output)
                values, indexs = output.max(1)
                t2 = time()
                output_label = parseOutput(indexs)
                if output_label == label:
                        right_count += 1
                print("label is " + label + " ,network
predict is " + output_label +" cost is
"+str(t2-t1)+"s")
                file_handle = open('result.txt', mode='a')
                file_handle.write(label +' '+ output_label
+ "\n")
                file_handle.close()


if __name__ == '__main__':
        main()
```