计算机科学与技术学院

毕业设计

论文题目	基于 SSD 网络模型的房屋瓦片损害检测		
学校导师	刘立	职称	教授
企业导师	刘立	职称	教授
学生姓名	李开运	学号	20144330106
专业班级	物联网	班级	14 级 01 班
系主任	毛宇	院长	刘振宇
起止时间	2017 年 6 月 5 日至 2018 年 5 月 22 日		

2018 年 3 月 8 日

目录

基于 SSD 网络模型的房屋瓦片损害检测

摘要：近年来，自然灾害的频发，特别是冰雹对房屋瓦片造成的损害，导致美国房屋理赔行业急需一套无损害、低成本的对房屋瓦片损害的检测方案。随着商用无人机的发展和基于深度学习的各类检测算法的出现，使得通过搭载了检测装置的无人机，结合目标检测算法代替工作人员进行房屋瓦片检测成为现实。本研究通过梳理近年来主要的目标检测算法，考虑到瓦片和冰雹的特征，采用基于 SSD: Single Shot MultiBox Detector [1] 算法进行瓦片损害检测。其主要的改进包括两点： 1 、采用 Focal Loss [2] 代替原论文中采用的损失函数，经实验，这种改进将正确率提高了 2% 左右； 2 、采用 Soft-NMS [3] 代替 NMS，经实验，这种改进将正确率提高了 1% 左右。综合了两种改进方案，最后的实验结果不仅满足了保险行业的要求[1]。且检测精度达到了 65%mAP，检测速度达到了 80FPS。

关键词：目标检测，深度学习，SSD，瓦片损害检测

[1]房屋屋顶每 $10m^2$ 中只要检测出 6 个损害 (Damage)，即鉴定结果为损害严重，会替屋主替换掉这面屋顶瓦片

House Tile Damage Detection Based on SSD Network Model

Abstract:In recent years, the frequent occurrence of natural disasters, especially the damage caused by hailstones to house tiles, has led to American housing claims industry is in urgent need of a non-damaging, low-cost detection scheme for house tile damage. With commercial unmanned.The development of the machine and the emergence of various kinds of detection algorithms based on deep learning have made the detection device unmanned.The machine, combined with the target detection algorithm to replace the staff in the house tile detection to become a reality. This study is sorted out.In recent years, the main target detection algorithm,considering the characteristics of tile and hail, is based on SSD: Single Shot.MultiBox Detector [1]algorithm for tile damage detection. Its main improvements include two points :1.Focal Loss [2]replaced the Loss function used in the original text, which improved the accuracy rate by 2% through experiments.Or so; 2. Using Soft-NMS [3]instead of NMS, the improved accuracy of the experiment was improved by 1% to the left.On the right. Combining two kinds of improvement schemes, the final experiment result not only satisfies the requirement of insurance industry[2]. And detection of chosen It reached 65%mAP and the detection speed reached 80FPS.

Key Words:Object Detection，Deep Learning，SSD，Tile Damage Detection

[2]As long as six Damage is detected in the roof of the house, the identification result is serious, and the roof tile will be replaced by the owner slice

第一章　绪论

1.1　课题背景及研究意义

1.1.1　研究背景

自然灾害频发：根据今日美国报道，美国国家海洋和大气管理局 (NOAA) 周一宣布，由于三次强大的冰雹，使 2017 年成为美国遭受冰雹灾害最为严重的年份之一。冰雹灾害让美国遭受了超过 3000 亿美元的损失。[3]

无人机的商用：从钮扣大小的微型飞行器到可用于执行特殊任务的商用无人机，目前进入市面上的无人机种类和数量都在快速增加。较小的最低 10 美元就可以买到。消费类无人机的用途主要是娱乐和拍摄，大的可以执行任务的无人机则开始用于商业投递。

机器视觉的发展：近几年深度学习发展迅猛，更是由于前段时间的谷歌的 AlphaGo 而轰动一时，国内也开始迎来这一技术的研究热潮，深度学习目前还处于发展阶段，不管是理论方面还是实践方面都还有许多问题待解决，不过由于我们处在了一个“大数据”时代，以及计算资源的大大提升，新模型、新理论的验证周期会大大缩短。人工智能时代的开启必然会很大程度的改变这个世界，无论是从交通，医疗，购物，军事等方面，或许我们正处于最好的年代。

1.1.2　研究意义

针对传统的房屋瓦片检测方法存在着操作复杂、耗时、高危和具有破坏性等缺点, 本项研究尝试利用计算机视觉技术对瓦片进行快速无损检测。本文提出了一种利用基于 SSD(Single Shot MultiBox Detector) [1] 改进算法进行瓦片损害检测的方案，为房屋检测行业提供利用机器视觉处理传统检测问题的高效手段。其意义有三:

1、**无损检测：**传统的房屋瓦片损害检测工作，需要工作人员爬上房屋拍照，将照片带回工作室进行损害鉴定，这种操作无疑会对瓦片带来人为的破坏；与此同时，工作人员因操作不当受伤甚至致死的报道也时有发生，本项研究利用无人机替代工作人员的拍照工作，利用无人机的图像处理模块，实时进行损害的检测，将结果和图片一并送到系统中进行信息的汇总和检测报告的生成。做到的对瓦片和对工作人员的两个“无损”。

2、**缩短检测周期：**在美国，遇到自然灾害致使房屋受损后，参保的家庭会联系保险公司进行理赔，按照现在的处理水平，一栋房屋平均会耗时 7 个星期，对于偏远的地方会更久。采用无人机对受损房屋瓦片进行检测会将这个时长缩短到 1 个星期。大大节约了成本。同时实验也表明有更好的检测效果。

3、**节约人力成本：**经融危机和通货膨胀造成了人力成本的极大提高。传统的损害检测十分依赖工作人员的经验，所以人力成本一直居高不下，采用搭载了检测算法的无人机进行检测，摆脱了对工作人员经验的依赖，将成本从 100 美元降到了 10 美元。

[3]美国中文网,2018-1-8

1.2 研究现状及发展难点

本文主要是利用 SSD 改进算法对房屋瓦片损害进行检测，涉及到目标检测系列算法，对于此系列算法的研究是深度学习方向的研究热点。简单的说，目标检测算法是对物体进行定位 + 分类。目标检测算法与定位算法、分类算法相比，重要的区别是对图片中的对象既要定位又要判断其类别。且对象检测的输出长度是可变的，因为检测到的对象的数量是不定的。

1.2.1 研究现状

在深度学习正式介入之前，传统的“目标检测”方法都是区域选择、提取特征、分类回归三部曲 [4]，这样就有两个难以解决且至关重要的问题；其一是区域选择的策略效果差、时间复杂度高；其二是手工提取的特征鲁棒性较差。大数据时代来临后，“目标检测”算法大家族主要划分为两大派系。

基于” Proposal + Classification” 的 Object Detection 的方法，RCNN 系列 (R-CNN [5]、SPPnet [6]、Fast R-CNN [7] 以及 Faster R-CNN [8]) 取得了非常好的效果，因为这一类方法先预先回归一次边框，然后再进行骨干网络训练，所以精度要高，这类方法被称为“Two Stage”的方法。但也正是由于此，这类方法在速度方面还有待改进。由此，YOLO [9] 应运而生，YOLO 系列只做了一次边框回归和打分，所以相比于 RCNN 系列被称为“One Stage”的方法，这类方法的最大特点就是速度快。但是 YOLO 虽然能达到实时的效果，但是由于只做了一次边框回归并打分，这类方法导致了小目标训练非常不充分，对于小目标的检测效果非常的差。简而言之，YOLO 系列对于目标的尺度比较敏感，而且对于尺度变化较大的物体泛化能力比较差。

针对 YOLO 和 Faster R-CNN 的各自不足与优势，WeiLiu 等人提出了 Single Shot MultiBox Detector，简称为 SSD。SSD 整个网络采取了“One Stage”的思想，以此提高检测速度。并且网络中融入了 Faster R-CNN [8] 中的 anchors 思想，并且做了特征分层提取并依次计算边框回归和分类操作，由此可以适应多种尺度目标的训练和检测任务。SSD 的出现是时期的集大成者，向大家证明了实时高精度目标检测的可行性。

1.2.2 发展难点

可变数量的对象 (Variable Number of Objects)。在训练模型时，通常需要将数据表示为固定大小的向量。但是，由于图片中对象的数量事先不知道，所以我们不知道正确的输出维度。因此需要一些后期处理，这增加了模型的复杂性。

调整对象检测窗口大小 (Resizing)。另一个巨大的挑战是各种可能的对象大小，即在进行分类时，既希望占图片大部分的对象进行分类，又想要找到一些可能只有 12 个像素、或者是原始图像一小部分的小对象。使用不同尺寸的滑动窗口可以解决这个问题，但效率很低。

建模。第三个挑战是同时解决两个问题——定位和分类如何用一个简单的模型解决这两种截然不同的需求。

1.3 研究内容及章节安排

第一章：绪论。本章概要阐述本文主要内容，及研究背景及意义。

第二章：介绍目标检测相关算法。本章按照两条主线系统讲解国内外对于目标检测算法的研究。一条 R-CNN 系列，另一条 YOLO 系列。

第三章：瓦片损害检测算法设计。通过第二章的综述，使我们了解到现有的目标检测相关算法。在第三章吸收了各种算法的优劣，本文对 SSD 算法进行了两点改进：

1 、Loss 函数 2 、Soft-NMS

第四章：瓦片损害检测算法实现。本章展示具体的算法实现，基于 pytorch。

第五章：实验结果及分析。本章结合原始算法与改进算法进行准确率和召回率的对比，并展示该算法对于瓦片损害检测的效果

第六章：总结及展望。

第七章：致谢。

第二章　目标检测相关算法

2.1　目标检测算法概述

目标检测是计算机视觉领域的基础问题，在 2010 年左右其发展就开始停滞不前了。自 2013 年——Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation [5] 的发表，目标检测从原始的传统手工提取特征方法变成了基于卷积神经网络的特征提取，从此一发不可收拾。根着历史的潮流，本章简要地探讨“目标检测”算法的两种思想和这些思想引申出的具有代表性的算法。

“目标检测”算法大家族主要划分为两大派系，一个是以 R-CNN 为代表的 “Two Stage”, 另一个则是以 YOLO 为代表的”One Stage”。下面分别解释一下 “Two Stage” 和 “One Stage”。

Two Stage: 顾名思义，分两步进行目标检测：1、生成可能区域 (Region Proposal) & CNN 提取特征。2、放入分类器分类并修正位置。这一流派的算法都离不开 Region Proposal，即是优点也是缺点，主要代表算法就是 Faster R-CNN。

One Stage: 顾名思义，对预测的目标物体直接进行检测：直接利用回归进行目标检测，其特点简单快速，但是有损精度，主要代表算法是 YOLO 和 SSD。

无论 “Two Stage” 还是 “One Stage”, 他们都是在同一个天平下选取一个平衡点、或者选取一个极端——要么准，要么快。”Two Stage” 的天平主要倾向准，“One Stage” 的天平主要倾向快。但最后大家也找到了自己的平衡，平衡点的有略微的不同。接下来将分别介绍这两个派系的算法。

2.2　Viola-Jones 人脸检测器

2001 年，Viola 和 Jones 发表了经典的 Rapid Object Detection using a Boosted Cascade of Simple Features [10] 和 Robust Real-Time Face Detection [11]，在 AdaBoost 算法的基础上，使用 Haar-like 小波特征和积分图方法进行人脸检测，并对 AdaBoost 训练出的强分类器进行级联。这是人脸检测史上里程碑式的一笔，这个算法被人们称为 Viola-Jones 检测器。

Viola-Jones 人脸检测器基本的思路就是用一个固定大小的窗口在输入图像进行滑动，窗口框定的区域会被送入到分类器，去判断是人脸窗口还是非人脸窗口。滑动的窗口其大小是固定的，但是人脸的大小则多种多样，为了检测不同大小的人脸，还需要把输入图像缩放到不同大小，使得不同大小的人脸能够在某个尺度上和窗口大小相匹配。这种滑动窗口式的做法有一个很明显的问题，就是有太多的位置要去检查，去判断是人脸还是非人脸。

判断是不是人脸，这是两个分类问题，在 2000 年的时候，采用的是 AdaBoost 分类器。进行分类时，分类器的输入用的是 Haar Like 特征[4]。AdaBoost 分类器是一种由多个弱分类器组合而成的强分类器, Viola-Jones 检测器是由多个 AdaBoost 分类器级联组成，这种级联结构的一个重要作用就是加速。

[4]Haar 特征分为三类：边缘特征、线性特征、中心特征和对角线特征，组合成特征模板

2.3 可变形部件模型 (DPM)

Viola-Jones 人脸检测器之后,在 2009 年出现了另外一个重要的方法 DPM:Deformable Part Model [12]，即可变形部件模型。就人脸检测而言，人脸可以大致看成是一种刚体，通常情况下不会有非常大的形变，比方说嘴巴变到鼻子的位置上去。但是对于其它物体，例如人体，人可以把胳膊抬起来，可以把腿翘上去，这会使得人体有非常多非常大的非刚性变换，而 DPM 通过对部件进行建模就能够更好地处理这种变换。刚开始的时候大家也试图去尝试用类似于 Haar 特征 +AdaBoost 分类器这样的做法来检测行人，但是发现效果不是很好，到 2009 年之后，有了 DPM 去建模不同的部件，比如说人有头有胳膊有膝盖，然后同时基于局部的部件和整体去做分类，这样效果就好了很多。DPM 相对比较复杂，检测速度比较慢，但是其在人脸检测还有行人和车的检测等任务上还是取得了一定的效果。后来出现了一些加速 DPM 的方法，试图提高其检测速度。DPM 引入了对部件的建模，本身是一个很好的方法，但是其被深度学习的光芒给盖过去了，深度学习在检测精度上带来了非常大的提升，所以研究 DPM 的一些人也快速转到深度学习上去了。

2.4 R-CNN 系列

R-CNN 其实是一个很大的家族，Ross Girshick 发表 Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation [5] 以来桃李满天下。在此，我们只探讨 R-CNN 直系亲属，他们的发展顺序如图2.1:

图 2.1: RCNN 系列算法发展顺序

他们在整个家族进化的过程中，一致暗埋了一条主线：充分利用 Feature Maps 的价值。

2.4.1 R-CNN

RCNN:(Region CNN) [5] 可以说是利用深度学习进行目标检测的开山之作。作者 Ross Girshick 多次在 PASCAL VOC[5]的目标检测竞赛中折桂，2010 年更带领团队获得终身成就奖。图3.1这个模型，利用卷积神经网络来做 “目标检测”，其深远意义不言而喻。

该算法解决了目标检测中的两个关键问题:

解决问题一、速度。传统的区域选择使用滑窗，每滑一个窗口检测一次，相邻窗

[5]计算机视觉里面很大一块是在做物体的识别、检测还有分类 (object recognition, detection and classification)。几乎在每一个应用领域都需要用到这三项功能，所以能否顺利的完成这三个功能，对检验一个算法的正确性和效率来说是至关重要的。所以每一个算法的设计者都会运用自己搜集到的场景图片对算法进行训练和检测，这个过程就逐渐的形成了数据集

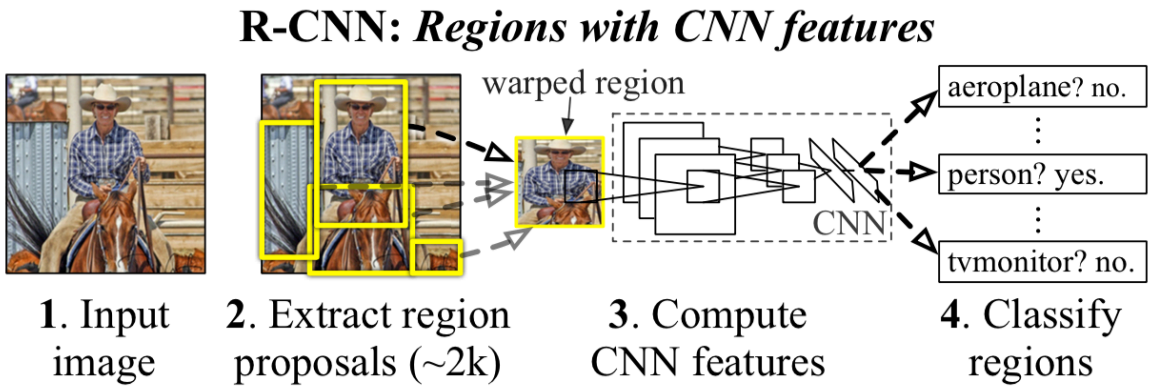

图 2.2: RCNN 算法框架

口信息重叠高，检测速度慢。R-CNN 使用一个启发式方法 (Selective Search [13])，先生成候选区域再检测，降低信息冗余程度，从而提高检测速度。

解决问题二、训练集。经典的目标检测算法在区域中提取人工设定的特征 (Haar,HOG)。传统的手工提取特征鲁棒性差，限于如颜色、纹理等低层次 (Low level) 的特征。使用 CNN(卷积神经网络) 提取特征，可以提取更高层面的抽象特征，从而提高特征的鲁棒性。

该方法将 PASCAL VOC 上的检测率从 35.1% 提升到 53.7 % ，提高了几个量级。

算法流程:

> 一张图像生成 1K 至 2K 个候选区域
> 对每个候选区域，使用深度网络提取特征
> 特征送入每一类的 SVM 分类器，判别是否属于该类
> 使用回归器精细修正候选位置

1、**候选区域生成**: 使用 Selective Search 方法从一张图像生成约 2000-3000 个候选区域。基本思路如下：1、使用一种分割手段，将图像分割成小区域。2、查看现有小区域，合并可能性最高的两个区域。重复直到整张图像合并成一个区域位置。3、输出所有曾经存在过的区域，所谓候选区域。

2、**特征提取**: 借鉴 Hinton 2012 年在 Image Net 上的分类网络[6]，提取特征。

3、**类别判断**: 对每一类目标，使用一个线性 SVM 二类分类器进行判别。输入为深度网络输出的 4096 维特征，输出是否属于此类。由于负样本很多，使用”Hard Negative Mining”[7]方法。

[6]Hinton 在 2012 年提出的 AlexNet 网络
[7]选择正负样本的方法

4、**位置精修**: 目标检测问题的衡量标准是重叠面积：许多看似准确的检测结果，往往因为候选框不够准确，重叠面积很小。故需要一个位置精修步骤。

2.4.2 SPP Net

R-CNN 提出后的一年，以何恺明、任少卿为首的团队发表了 SPP Net:Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition [6] ，这才是真正摸到了卷积神经网络的脉络。尽管 R-CNN 效果不错，但是他还有两个缺点：

缺点一、算力冗余。先生成候选区域，再对区域进行卷积，这里有两个问题：其一是候选区域会有一定程度的重叠，对相同区域进行重复卷积；其二是每个区域进行新的卷积需要新的存储空间。何恺明等人意识到这个可以优化，于是把先生成候选区域再卷积，变成了先卷积后生成区域。通过“简单地”改变顺序，不仅减少存储量而且加快了训练速度。

缺点二、图片缩放。无论是剪裁 (Crop) 还是缩放 (Warp)，在很大程度上会丢失图片原有的信息导致训练效果不好，如图2.4所示。直观的理解，把车剪裁成一个门，人看到这个门也不好判断整体是一辆车；把一座高塔缩放成一个胖胖的塔，使得机器的识别难度巨大。

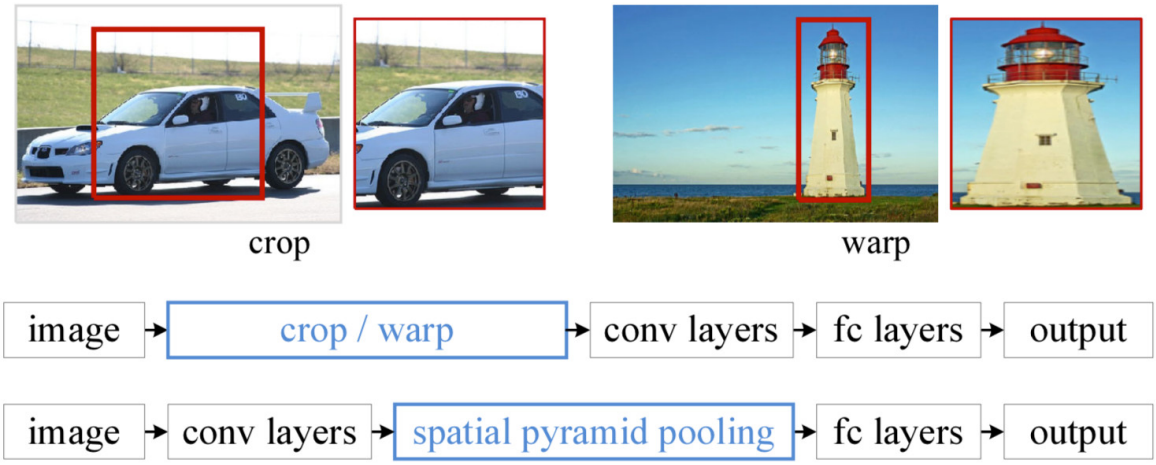

图 2.3: 因剪裁和缩放导致视差

何恺明等人发现了这个问题，于是思索有什么办法能不对图片进行变形，且保留原图整体信息直接进行学习。最后，他们发现问题的根源是全连接层 (FC Layer) 需要确定输入维度，于是他们在输入全连接层前定义一个特殊的池化层，将输入的任意尺度 Feature Maps 组合成特定维度的输出，这个组合可以是不同大小的拼凑，如图2.4，我们要输入的维度 64*256 ，那么我们可以这样组合 32*256+16*256+8*256+8*256。

SPP Net 的出现打破了常规，不仅减少了计算冗余从而提高了检测速度，更重要的是打破了固定尺寸输入这一束缚，被后面出现的算法广泛采用，其意义重大。本文基于的 SSD 算法就是更是从中受到启发。

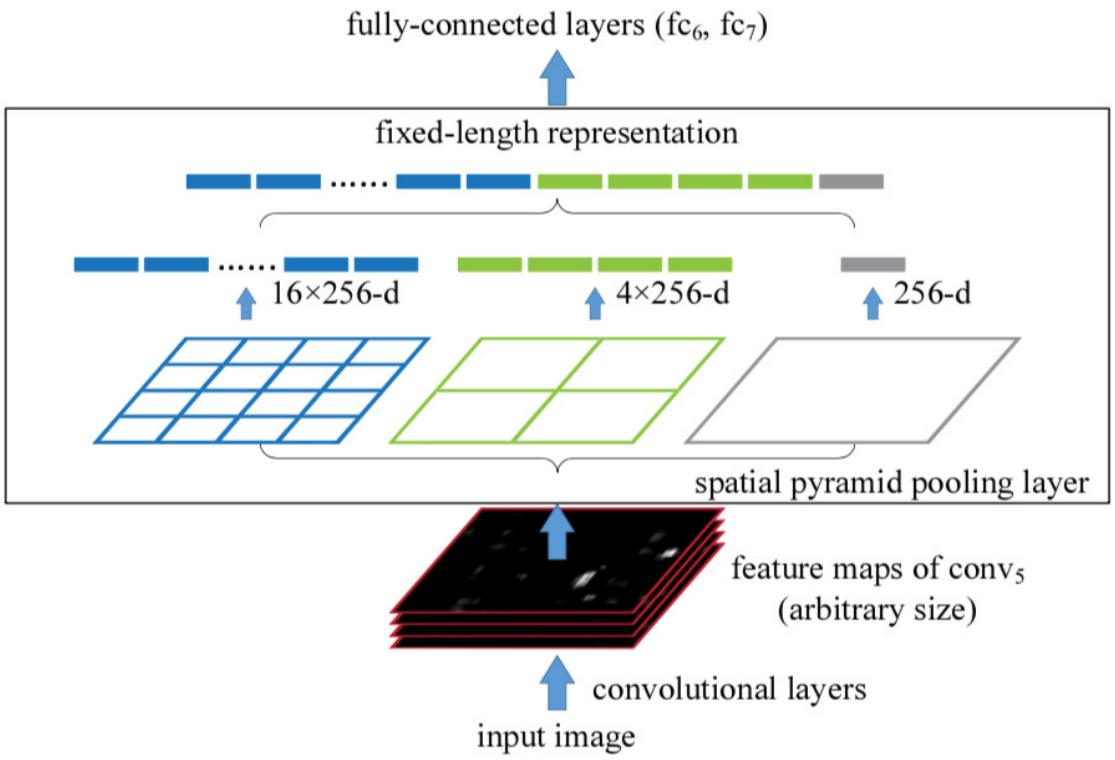

图 2.4: 输入维度的组合方式

2.4.3 Fast R-CNN

继 2013 年的 RCNN 之后，Ross Girshick 在 15 年推出 Fast RCNN [7]，构思精巧，流程更为紧凑，大幅提升了目标检测的速度。在这篇论文中，引用了 SPP Net 的贡献。纵观全文，最大的贡献就是将原来的串行结构改成并行结构。同样使用最大规模的网络，Fast RCNN 和 RCNN 相比，训练时间从 84 小时减少为 9.5 小时，测试时间从 47 秒减少为 0.32 秒。在 PASCAL VOC 2007 上的准确率相差无几，约在 66%-67% 之间。其算法框架如图2.5

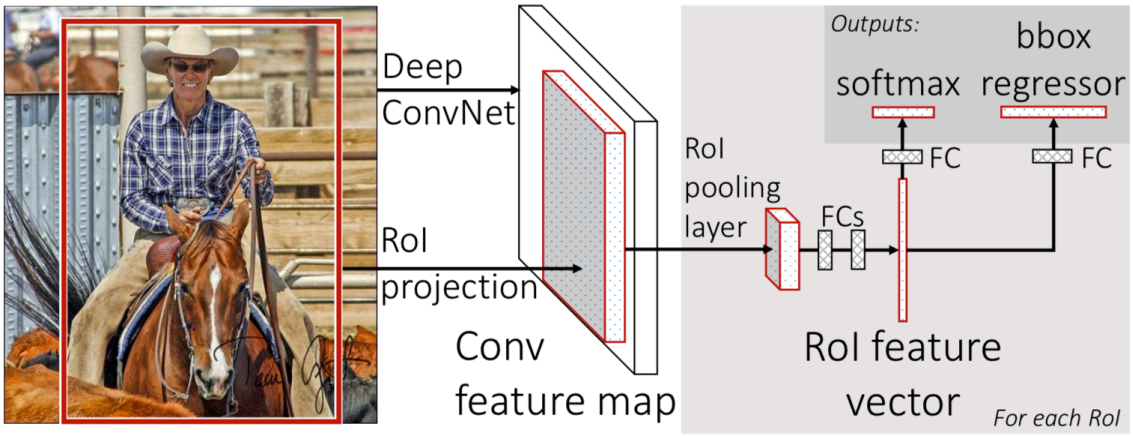

图 2.5: Fast R-CNN 算法框架

原来的 R-CNN 是先对候选框区域进行分类，判断有没有物体，如果有则对

Bounding Box 进行精修、回归。这是一个串联式的任务，那么势必没有并联的快，所以 Ross Girshick 就将原有结构改成并行——在分类的同时，对 Bbox 进行回归。这一改变将 Bbox 和 Clf 的 loss 结合起来变成一个 Loss 一起训练，并吸纳了 SPP Net 的优点，最终不仅加快了预测的速度，而且提高了精度。

算法流程:

> 在图像中确定 1000-2000 个候选框
> 对于每个候选框内图像块，使用深度网络提取特征
> 对候选框中提取出的特征，使用分类器判别是否属于一个特定类
> 对于属于某一特征的候选框，用回归器进一步调整位置

Fast RCNN 方法解决了 RCNN 方法三个问题:

问题一、测试时速度慢。RCNN 一张图像内候选框之间大量重叠，提取特征操作冗余，本文将整张图像归一化后直接送入深度网络。在邻接时，才加入候选框信息，在末尾的少数几层处理每个候选框。

问题二、训练时速度慢。原因同问题一，在训练时，本文先将一张图像送入网络，紧接着送入从这幅图像上提取出的候选区域。这些候选区域的前几层特征不需要再重复计算。

问题三、训练所需空间大。RCNN 中独立的分类器和加归器需要大量特征作为训练样，本文把类别判断和位置精调统一用深度网络实现，不再需要额外存储。其网络模型如图2.6

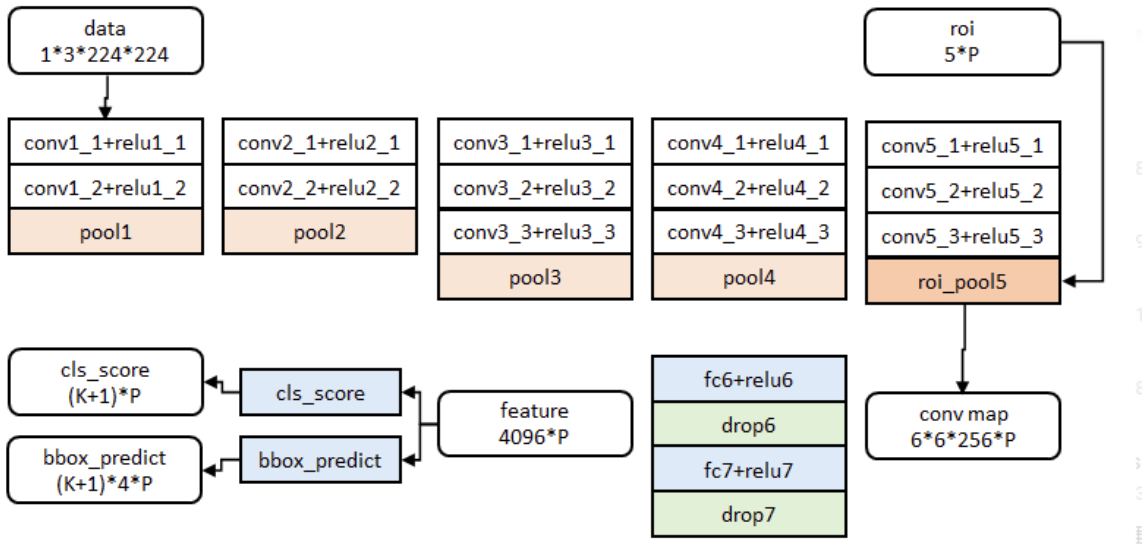

图 2.6: Fast R-CNN 网络模型

损失函数: loss_cls 层评估分类代价。由真实分类 u 对应的概率决定:

$$L_{cls} = -\log p_u$$

loss_bbox 评估检测框定位代价。比较真实分类对应的预测参数 t^u 和真实平移绽放为 v 的差别:

$$L_{loc} = \sum_{i=1}^{4} g(t_i^u - v_i)$$

g 为 Smooth L1 误差，对 outlier 不敏感:

$$g(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{1}$$

总代价为两者加权和，如果分类为背景则不考虑定位代价:

$$L = \begin{cases} L_{cls} + \lambda L_{loc} & u \\ L_{cls} & u \end{cases} \tag{2}$$

2.4.4 Faster R-CNN

继 RCNN, Fast RCNN 之后，目标检测界的领军人物 Ross Girshick 团队在 2015 年的又一力作。简单网络目标检测速度达到 17FPS，在 PASCAL VOC 上准确率为 59.9%；复杂网络达到 5FPS，准确率 78.8%。在 Faster R-CNN 前，我们生产候选区域都是用的一系列启发式算法，基于 Low Level 特征生成区域。这样就有两个问题:

第一个问题每次生成区域具有不确定性，而 “Two Stage” 算法正是依靠生成区域的不确定性——生成大量无效区域则会造成算力的浪费、少生成区域则会漏检；

第二个问题是生成候选区域的算法是在 CPU 上运行的，而我们的训练在 GPU 上面，跨结构交互必定会有损效率。

那么怎么解决这两个问题呢? 于是乎，任少卿等人提出了一个 RPN:Region Proposal Networks 的概念，利用神经网络自己学习去生成候选区域。其结构如图2.7。

这种生成方法同时解决了上述的两个问题，神经网络可以学到更加高层、语义、抽象的特征，生成的候选区域的可靠程度大大提高；可以从上图看出 RPNs 和 RoI Pooling 共用前面的卷积神经网络——将 RPNs 嵌入原有网络，原有网络和 RPNs 一起预测，大大地减少了参数量和预测时间。在 RPN 中引入了 anchor 的概念 Feature Map 中每个滑窗位置都会生成 k 个 anchors，然后判断 anchor 覆盖的图像是前景还是背景，同时回归 Bbox 的精细位置，预测的 Bbox 更加精确。

从 RCNN 到 Fast RCNN，再到 Faster RCNN，目标检测的四个基本步骤 (候选区域生成，特征提取，分类，位置精修) 终于被统一到一个深度网络框架之内。所有计算没有重复，完全在 GPU 中完成，大大提高了运行速度。

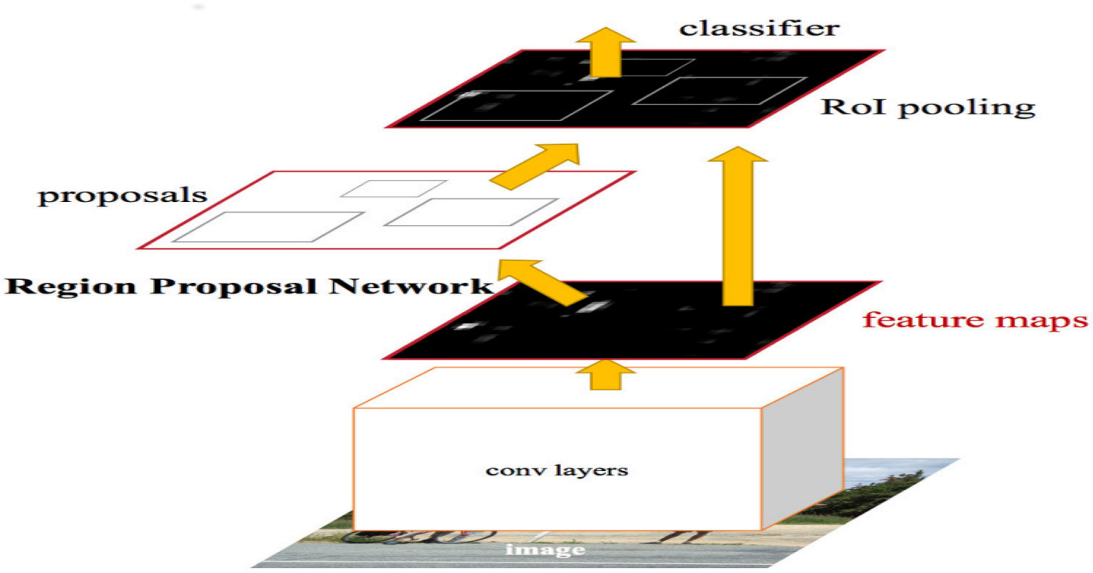

图 2.7: Faster RCNN 算法架构

Faster RCNN 着重解决了三个问题:

> 如何设计区域生成网络
> 如何训练区域生成网络
> 如何让区域生成网络和 Faster RCNN 网络共享特征提取网络

区域生成网络 (RPN)

1、**特征提取**: 原始特征提取 (上图灰色方框) 包含若干层 conv+relu，直接套用 ImageNet 上常见的分类网络即可。本文试验了两种网络：5 层的 ZF[8]，16 层的 VGG-16，具体结构不再赘述。额外添加一个 conv+relu 层，输出 51*39*256 维特征 (feature)。

2、**候选区域**: 特征可以看做一个尺度 51*39 的 256 通道图像，对于该图像的每一个位置，考虑 9 个可能的候选窗口：三种面积 1282,2562,5122× 三种比例 1:1,1:2,2:1。这些候选窗口称为 anchors。下图示出 51*39 个 anchor 中心，以及 9 种 anchor 示例2.8。

3、**窗口分类和位置精修**: 分类层 (cls_score) 输出每一个位置上，9 个 anchor 属于前景和背景的概率；窗口回归层 (bbox_pred) 输出每一个位置上，9 个 anchor 对应窗口应该平移缩放的参数。对于每一个位置来说，分类层从 256 维特征中输出属于前景和背景的概率；窗口回归层从 256 维特征中输出 4 个平移缩放参数。

[8]介绍一下这个网络

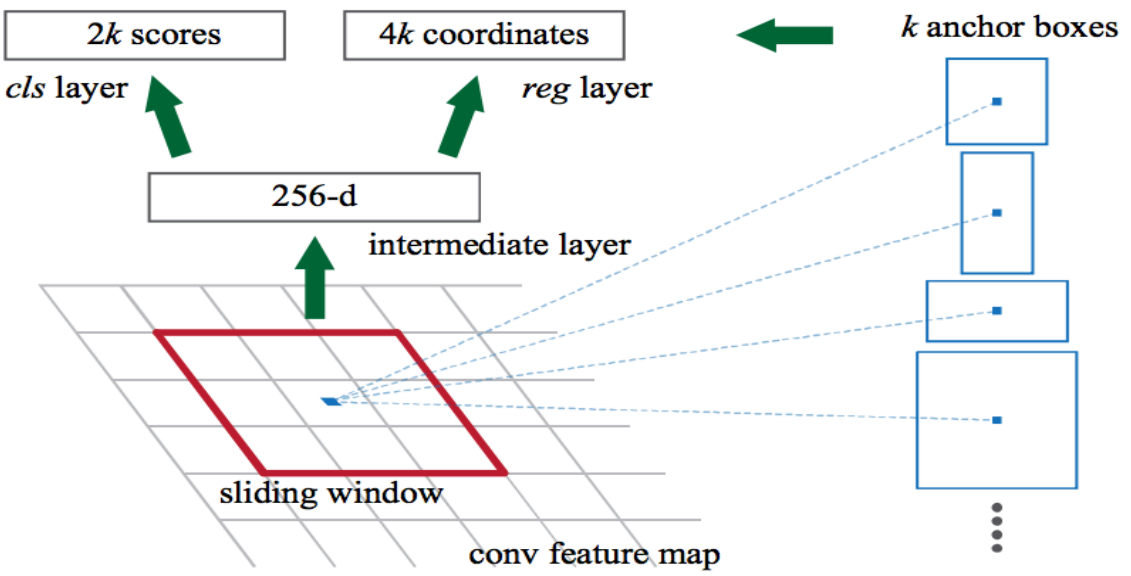

图 2.8: Faster RCNN anchor 示意图

2.5 YOLO 系列

You Only Look Once: Unified, Real-time Object Detection(YOLO) [9] 是单阶段方法的开山之作。它将检测任务表述成一个统一的、端到端的回归问题，并且以只处理一次图片同时到位置和分类而得名。“One Stage” 的想法就比较简单，给定一张图像，使用回归的方式输出这个目标的边框和类别。“One Stage” 最核心的还是利用了分类器优秀的分类效果，首先给出一个大致的范围 (最开始就是全图) 进行分类，然后不断迭代这个范围直到一个精细的位置，

如图2.9从蓝色的框框到红色的框框。这就是 “One Stage” 回归的思想，这样做的优点就是快，但是会有许多漏检。

2.5.1 YOLO

YOLO 就是使用回归这种做法的典型算法。首先将图片 Resize 到固定尺寸，然后通过一套卷积神经网络，最后接上 FC(全连接层) 直接输出结果，这就他们整个网络的基本结构。更具体地做法，是将输入图片划分成一个 SxS 的网格，每个网格负责检测网格里面的物体是什么物体，并输出 Bbox Info 和置信度。这里的置信度指的是该网格内含有什么物体和预测这个物体的准确度。

更具体的是如下定义:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

这个想法其实就是一个简单的分而治之想法，将图片卷积后提取的特征图分为 SxS 块，然后利用优秀的分类模型对每一块进行分类，将每个网格处理完使用非极大值抑制 (NMS) 的算法去除重叠的框，最后得到我们的结果。

图 2.9: YOLO

2.5.2 SSD

YOLO 这样做的确非常快，但是问题就在于该算法对小物体的检测效果不好。

所以 SSD 就在 YOLO 的主意上添加了 Faster R-CNN 的 Anchor 概念，并融合不同卷积层的特征做出预测。关于 SSD 算法的细节介绍将在第三章将详细阐述。

2.6 本章小结

回顾过去，从 YOLO 到 SSD，人们兼收并蓄把不同思想融合起来。SSD 将 YOLO 和 Anchor 思想融合起来，并创新使用 Feature Pyramid 结构。但是 Resize 输入，必定会损失许多的信息和一定的精度，这也许是“One Stage”快的原因。两类算法都有其适用的范围，比如说实时快速动作捕捉，“One Stage”更胜一筹；复杂、多物体重叠，“Two Stage”当仁不让。没有不好的算法，只有不合适的使用场景。

第三章　瓦片损害检测算法设计

3.1　瓦片损害检测流程

1、无人机拍照

2、将图像切割成 224x224 像素大小的图片

3、将小图输入网络模型中进行检测

4、输出检测结果

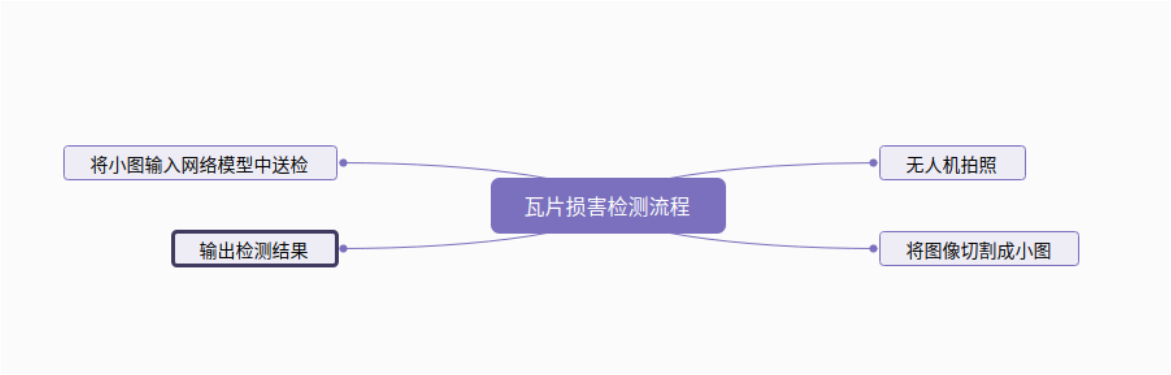

图 3.1: 瓦片损害检测流程

3.2　SSD 算法核心思想

SSD 和 YOLO 一样都是采用一个 CNN 网络来进行检测，但是却采用了多尺度的特征图，其基本架构如图3.2所示。

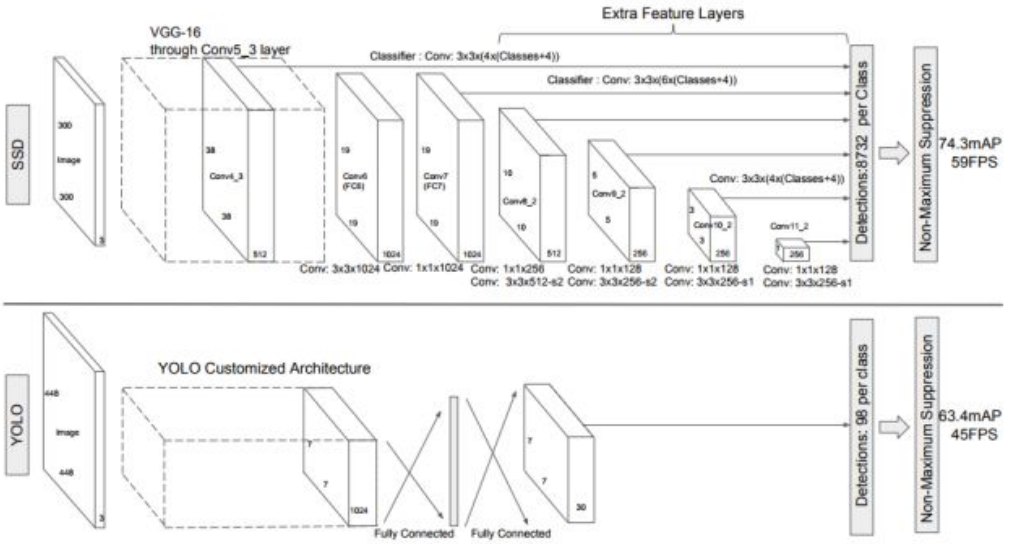

图 3.2: SSD 基本框架

下面将 SSD 核心设计思想总结为以下三点:

1、**采用多尺度特征图用于检测**

所谓多尺度采用大小不同的特征图，CNN 网络一般前面的特征图比较大，后面会逐渐采用 stride=2 的卷积或者 pool 来降低特征图大小，如图3.3所示

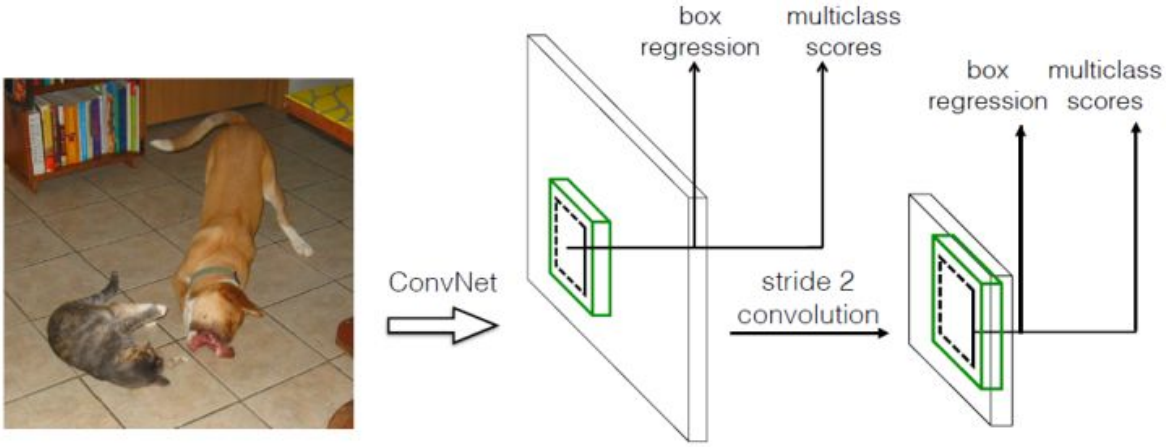

图 3.3: 采用多尺度用于检测

不同尺度下的特征图都用来做检测。这样做的好处是比较大的特征图来用来检测相对较小的目标，而小的特征图负责检测大目标，如图3.4所示，8x8 的特征图可以划分更多的单元，但是其每个单元的先验框尺度比较小。

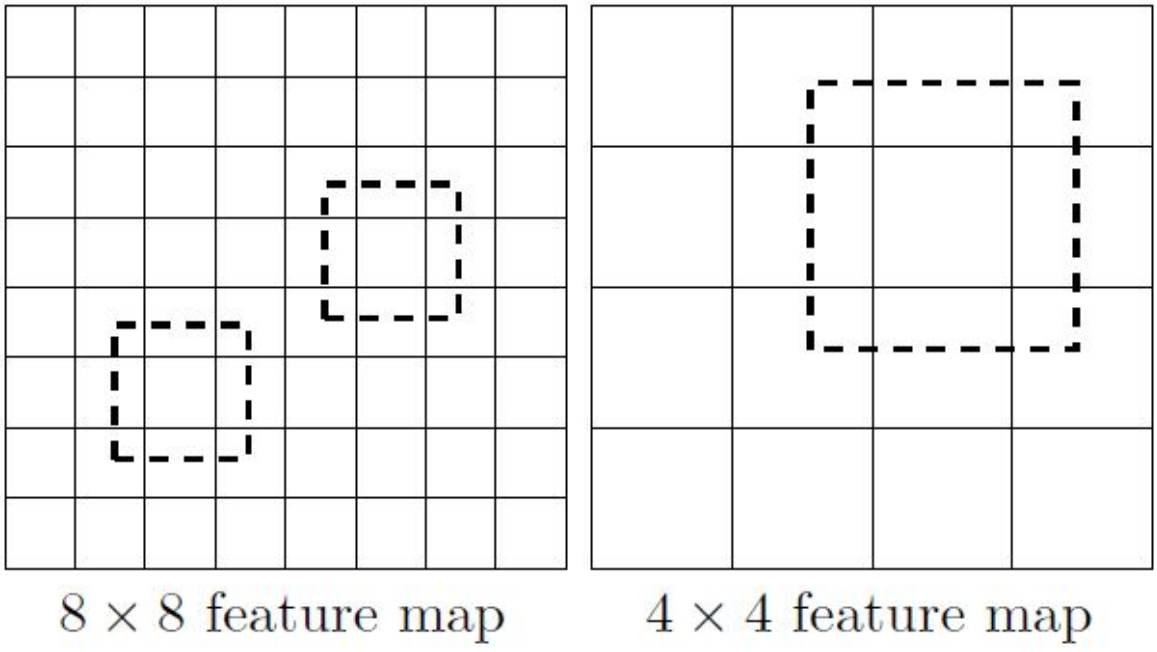

图 3.4: 8x8 与 4x4 的特征图

2、**采用卷积进行检测**

与 YOLO 最后采用全连接层不同，SSD 直接采用卷积对不同的特征图来进行提取检测结果。对于形状为 $m \times n \times p$ 的特征图，只需要采用 $3 \times 3 \times p$ 这样比较小的卷积核得到检测值。

3、**设置先验框**

在 YOLO 中，每个单元预测多个边界框，但是其都是相对这个单元本身 (正方块)，但是真实目标的形状是多变的，YOLO 需要在训练过程中自适应目标的形状。而 SSD 借鉴了 Faster R-CNN 中 anchor 的理念，每个单元设置尺度或者长宽比不同的先验框，预测的边界框 (bounding boxes) 是以这些先验框为基准的，在一定程度上减少训练难度。一般情况下，每个单元会设置多个先验框，其尺度和长宽比存在差异，如图3.5所示，可以看到每个单元使用了 4 个不同的先验框，图片中猫和狗分别采用最适合它们形状的先验框来进行训练，后面会详细讲解训练过程中的先验框匹配原则。

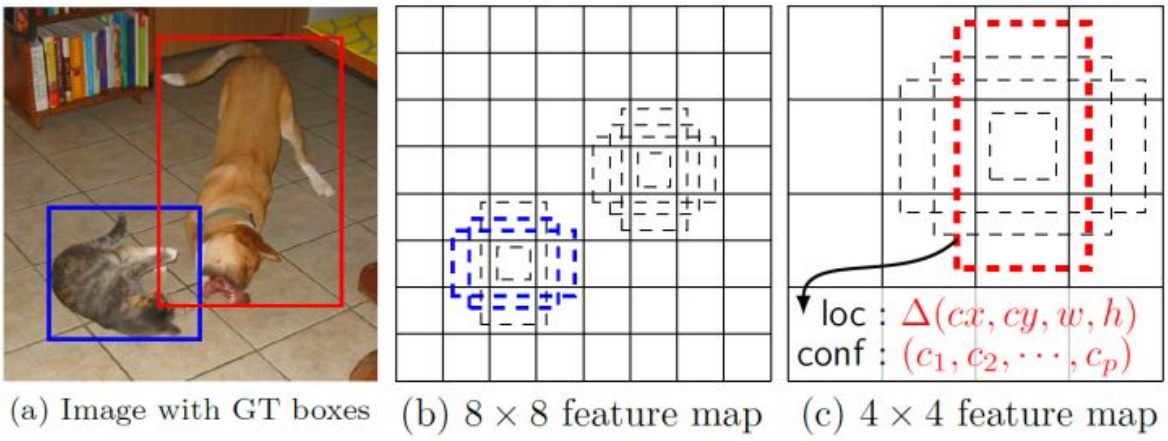

图 3.5: SSD 算法中的先验框

3.3 SSD 模型结构

SSD 网络主体设计的思想是特征分层提取，并依次进行边框回归和分类。因为不同层次的特征图能代表不同层次的语义信息，低层次的特征图能代表低层语义信息 (含有更多的细节)，能提高语义分割质量，适合小尺度目标的学习。高层次的特征图能代表高层语义信息，能光滑分割结果，适合对大尺度的目标进行深入学习。所以该算法提出的 SSD 的网络理论上能适合不同尺度的目标检测。

所以 SSD 网络中分为了 6 个 stage，每个 stage 能学习到一个特征图，然后进行边框回归和分类。SSD 网络以 VGG16 的前 5 层卷积网络作为第 1 个 stage，然后将 VGG16 中的 fc6 和 fc7 两个全连接层转化为两个卷积层 Conv6 和 Conv7 作为网络的第 2、第 3 个 stage。接着在此基础上，SSD 网络继续增加了 Conv8、Conv9、Conv10 和 Conv11 四层网络，用来提取更高层次的语义信息。如图3.2所示就是 SSD 的网络结构。在每个 stage 操作中，网络包含了多个卷积层操作，每个卷积层操作基本上都是小卷积。

骨干网络：SSD 前面的骨干网络选用的 VGG16 的基础网络结构，如上图所示，虚线框内的是 VGG16 的前 5 层网络。然后后面的 Conv6 和 Conv7 是将 VGG16 的后两层全连接层网络 (fc6, fc7) 转换而来。另外：在此基础上，SSD 网络继续增加了 Conv8 和 Conv9、Conv10 和 Conv11 四层网络。如图3.2，立方体的长高表示特征图的大小，厚度表示是 channel。

3.4 损失函数

3.4.1 SSD 中采用的损失函数

联合 LOSS FUNCTION:SSD 网络对于每个 Stage 输出的特征图都进行边框回归和分类操作。SSD 网络中作者设计了一个联合损失函数

$$L(x,c,l,g)=\frac{1}{N}(L_{conf}(x,c)+\alpha L_{loc}(x,l,g)$$

其中:

1、L_{conf} 代表的是分类误差，采用的是多分类的 softmax loss

2、L_{loc} 代表的是回归误差，采用的是 Smooth L1 loss

3、α 取 1

回归 loss,smoothL1:

$$L_{loc}(x,l,g)=\sum_{i\in Pos}^{N}\sum_{m\in\{cx,cy,w,h\}}x_{ij}^{k}smooth_{L1}(l_i^m-g_j^m)$$

$$g_j^{cx}=\left(g_j^{cx}-d_i^{cx}\right)/d_i^w$$

3.4.2 改进损失函数-Focal Loss

作者提出 Focal Loss 的出发点也是希望 One-Stage Detector 可以达到 Two Stage Detector 的准确率，同时不影响原有的速度。

先分析 One Stage Detector 的准确率不如 Two Stage Detector 的原因，作者认为原因是：样本的类别不均衡导致的。我们知道在 object detection 领域，一张图像可能生成成千上万的 candidate locations，但是其中只有很少一部分是包含 object 的，这就带来了类别不均衡。那么类别不均衡会带来什么后果呢？引用原文讲的两个后果:

(1) training is inefficient as most locations are easy negatives that contribute no useful learning signal;

(2) en masse, the easy negatives can overwhelm training and lead to degenerate models. [2]

什么意思呢？负样本数量太大，占总的 loss 的大部分，而且多是容易分类的，因此使得模型的优化方向并不是我们所希望的那样。其实先前也有一些算法来处理类别不均衡的问题，比如 OHEM(Online Hard Example Mining) [14]，OHEM 的主要思想可以用原文的一句话概括：In OHEM each example is scored by its loss, non-maximum suppression (NMS) is then applied, and a minibatch is constructed with the highest-loss examples。[14]OHEM 算法虽然增加了错分类样本的权重，但是 OHEM 算法忽略了容易分类的样本。因此针对类别不均衡问题，Focal Loss 这个损失函数是在标准交叉熵损失基础上修改得到的。这个函数可以通过减少易分类样本的权重，

使得模型在训练时更专注于难分类的样本。

标准交叉熵损失函数

原来的分类 loss 是各个训练样本交叉熵的直接求和，也就是各个样本的权重是一样的。如下图所示:

$$CE(p,y)=\begin{cases}-\log(p) & if y=1\\ -\log(1-p) & otherwise\end{cases} \tag{3}$$

其中:

CE 表示 cross entropy,p 表示预测样本属于 1 的概率，y 表示 label，y 的取值为 +1,-1，这里仅仅以二分类为例，多分类以此类推。为了表示简便，我们用 p_t 表示样本属于 true class 的概率。所以上式可以写成

$$CE(p,y)=CE(p_t)=-\log(p_t)$$

平衡交叉熵

既然"one stage detector" 在训练的时候正负样本的数量差距很大，那么一种常见的做法就是给正负样本加上权重，负样本出现的频次多，那么就降低负样本的权重，正样本数量少，就相对提高正样本的权重，如下式所示:

$$CE(p_t)=-\alpha\log(p_t)$$

Focal-Loss **定义**

作者实际上解决的是 easy example 和 hard example 不均衡的问题，这个和训练时候正负样本不均衡是两码事，因为正负样本里面都会有简单的样本和容易分错的样本。作者提出的 focal loss，相当于是对各个样本加上了各自的权重，这个权重是和网络预测该样本属于 true class 的概率相关的，显然，如果网络预测的该样本属于 true class 的概率很大，那么这个样本对网络来说就属于 easy(well-classified) example。如果网络预测的该样本属于 true class 的概率很小，那么这个样本对网络来说就属于 hard example。为了训练一个有效的 classification part，显然应该降低绝大部分 easy example 的权重，相对增大 hard example 的权重。作者提出的 focal loss 如下式所示:

$$FL(p_t)=-(1-p_t)^{\gamma}\log(p_t)$$

参数 γ 大于 0。当参数 $\gamma=0$ 的时候，就是普通的交叉熵，作者的实验中发现 $\gamma=2$ 效果最高。可以看到，当 一定的时候，比如等于 2，一样 easy example($p_t=0.9$) 的 loss 要比标准的交叉熵 loss 小 100+ 倍，当 $p_t=0.968$ 时，要小 1000+ 倍，但是对于 hard example($p_t<0.5$)，loss 最多小了 4 倍。这样的话 hard example 的权重相对

就提升了很多。实际实验中，作者和 3.2 一样，对正负样本又做了一个 reweighting

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t)$$

3.5 SSD 模型训练

训练 SSD 和基于 region proposal 方法的最大区别就是：SSD 需要精确的将 ground truth 映射到输出结果上。这样才能提高检测的准确率。文中主要采取了以下几个技巧来提高检测的准确度。

匹配策略 Default boxes 生成器 Hard Negative Mining Data Augmentation

1. **匹配策略**

这里的匹配是指的 ground truth 和 Default box 的匹配。这里采取的方法与 Faster R-CNN 中的方法类似。主要是分为两步：第一步是根据最大的 overlap 将 ground truth 和 default box 进行匹配 (根据 ground truth 找到 default box 中 IOU 最大的作为正样本)，第二步是将 default boxes 与 overlap 大于某个阈值 (目标检测中通常选取 0.5，Faster R-CNN 中选取的是 0.7) 的 ground truth 进行匹配。

2. Default Boxes **生成器**

3. Hard Negative Mining 经过匹配策略会得到大量的负样本，只有少量的正样本。这样导致了正负样本不平衡，经过试验表明，正负样本的不均衡是导致检测正确率低下的一个重要原因。所以在训练过程中采用了 Hard Negative Mining 的策略，根据 Confidence Loss 对所有的 box 进行排序，使得正负样本的比例控制在 1:3 之内，经过作者实验，这样做能提高 4% 左右的准确度。

4. Data Augmentation

为了模型更加鲁棒，需要使用不同尺度目标的输入，作者对数据进行了增强处理。

1、使用整张图像作为输入

2、使用 IOU 和目标物体为 0.1、0.3、0.5、0.7 和 0.9 的 patch，这些 patch 在原图的大小的 [0.1, 1] 之间，相应的宽高比在 [1/2, 2] 之间。

3、随机采取一个 patch

3.6 本章小结

本章对 SSD 算法进行了详细描述，并对该算法进行了两点改进

1. 将损失函数替换为 focus loss

2. 将 NMS 替换为 Soft-NMS

第四章　瓦片损害检测算法实现

4.1　图像预处理

4.1.1　标注工具

labelImg **介绍**: 图片标注主要是用来创建自己的数据集，方便进行深度学习训练。本文使用 labelImg

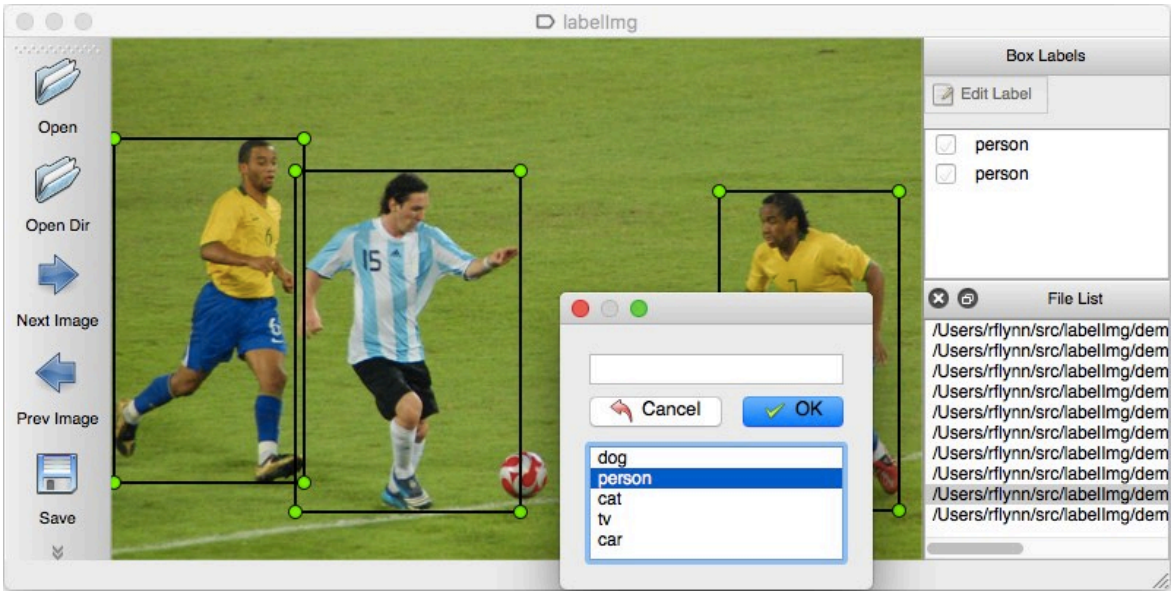

图 4.1: labelImg 界面

4.1.2　数据集

VOC **数据集介绍**:PASCAL VOC 为图像识别和分类提供了一整套标准化的优秀的数据集。对于目标检测任务，我们只需要使用 JPEGImage、Annotations、ImageSets 三个文件夹。

其目录结构

- JPEGImage: 文件夹包含了 PASCAL VOC 所提供的所有的图片信息，包括了训练图片和测试图片。

- Annotations: 文件夹中存放的是 xml 格式的标签文件，每一个 xml 文件都对应于 JPEGImages 文件夹中的一张图片。

- ImageSet: 存放的是每一种类型的 challenge 对应的图像数据。

4.1.3　Pytorch **介绍**

PyTorch 是一个比较新的深度学习框架，正在研究人员中迅速普及。和深度学习框架 Chainer 类似，PyTorch 支持动态计算图，这个功能使 PyTorch 对使用文本与时间序列的研究者和工程师很有吸引力。TensorFlow 解决了质量控制和包装的问题。它提供了一种 Theano 风格的编程模式，所以它是一种非常底层的深度学习框架。由于 TensorFlow 本身是非常底层的框架，所以许多基于 TensorFlow 的前端框架应运而生，比如 TF-slim 和 Keras。这样的框架目前有 10 到 15 个，仅 Google 就

可能有四五个。

Torch 的思想和 Theano 一直略有不同。TensorFlow 是一个更好的 Theano 风格的框架，并且我们在 Torch 中注入了强制的思想，这意味着你可以立即运行你的计算。调试应该是平稳顺利的，用户在调试过程中不应当遇到难题，无论使用 Python 调试器还是像 GDB 或其他类似的东西。

4.1.4 OpenCV 介绍

OpenCV 是 Intel© 开源的计算机视觉库。它由一系列 C 函数和少量 C++ 类构成 ，实现了图像处理和计算机视觉方面的很多通用算法。

4.1.5 将图片切割成 300px 大小

Listing 1 图像切割

```
1: img = img.resize(self.img_size, self.img_size)
```

4.2 网络模型实现

4.2.1 模型设计

1、VGG16:SSD 网络的基础网络，使用前 16 层作为特征提取层

2、Conv{ 4-12 }: 额外添加的层，生成 feature map

3、将对 conv4 的处理抽象成一个类名为 L2Norm2d

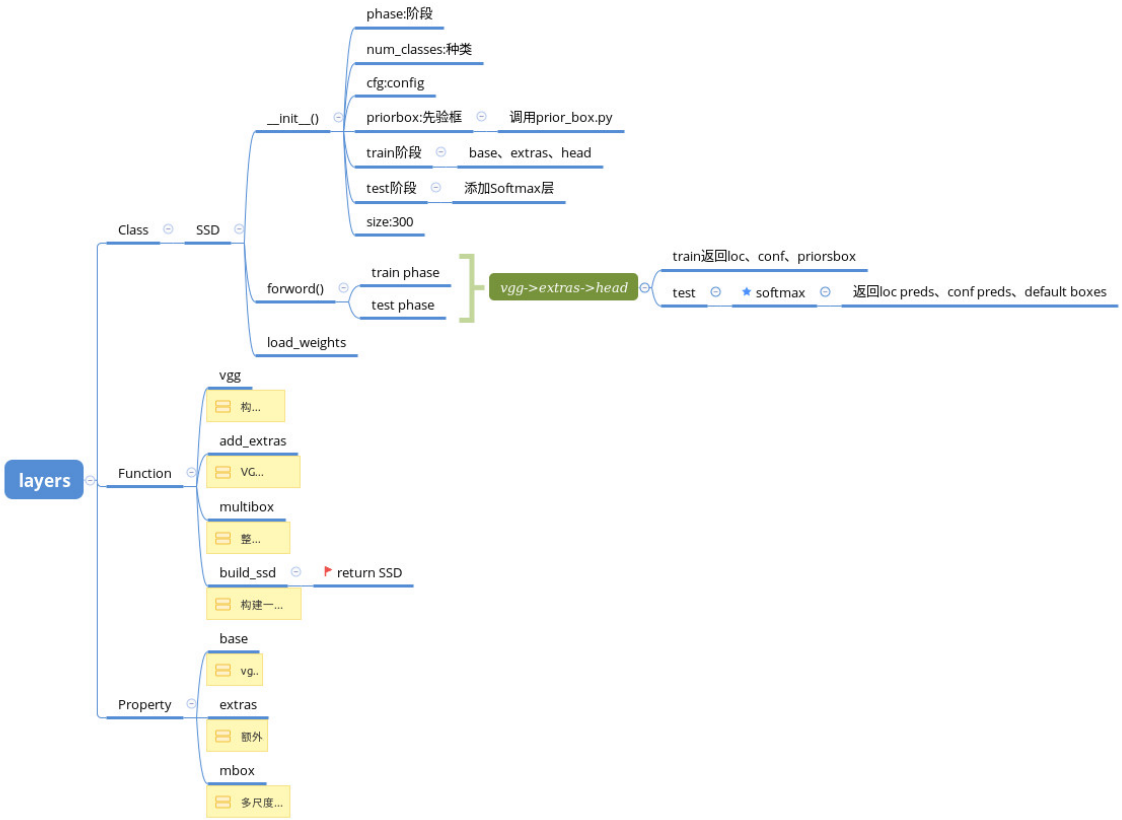

图 4.2: SSD 模型设计框架

4.2.2 L2Norm2d

Listing 2 L2Norm2d

```
class L2Norm2d(nn.Module):
        def __init__(self, scale):
                super(L2Norm2d, self).__init__()
                self.scale = scale

        def forward(self, x, dim = 1):
                return  self.scale * x *    \
                                x.pow(2).sum(dim).  \
                                clamp(min=1e-12).       \
                                rsqrt().expand_as(x)
```

4.2.3 SSD Layer

Listing 3 SSD layer

```
class SSD300(nn.Module):
        def __init__(self):
                super(SSD300, self).__init__()

        #model
        VGG16
        self.conv5(1~3)
        self.conv6
        self.conv7
        self.conv(8~12)(1~2)
```

4.2.4 Multibox Layer

1、Multibox Layer 作为生成 Anchor 的网络

2、loc_loss、conf_loss 的计算

Listing 4 Multibox layer

```
class MultiBoxLayer(nn.Module):
        num_classes = 2
        num_anchors = [4, 6, 6, 6, 4, 4]
        in_planes = [512, 1024, 512, 256, 256, 256]

        for i n range(len(self.in_planes)):
                self.loc_layers.apend(
                        nn.Conv2d(self.in_planes[i],
                        self.num_anchors[i] * 4,
                        kernel_size = 3, padding = 1
```

```
11:                         )
12:                 )
13:                 self.conf_layers.apend(
14:                         nn.Conv2d(self.in_planes[i],
15:                         self.num_anchors[i] * 21,
16:                         kernel_size = 3, padding = 1
17:                         )
18:                 )
```

4.3 损失函数

4.3.1 loc_loss

Listing 5 location 损失

```
1:          loc_loss = SmoothL1Loss(pos_loc_preds,
2:                                                  pos_loc_targets)
```

4.3.2 conf_loss

Listing 6 confidence 损失

```
1:          conf_loss = CrossEntropyLoss(pos_conf_preds,
2:                                                  pos_conf_targets
3:                                      ) + CrossEntropyLoss(
4:                                                  neg_conf_preds,
5:                                                  neg_conf_targets)
```

4.4 模型训练/测试

4.4.1 训练策略

Listing 7 训练方法

```
1: for batch_idx, (images, loc_targets,conf_targets)
2:          in enumerate(trainloader):
3:          optimizer.zero_grad()
4:          loc_preds, conf_preds = net(images)
5:          loss = criterion(loc_preds,
6:                          conf_preds,
7:                          conf_targets)
8:          loss.backward()
9:          optimizer.step()
```

第五章　实验结果与分析

5.1　改进 SSD 算法的实验结果

图 5.1: 检测结果

5.2　改进 SSD 对瓦片损害检测的准确率实验

表 1: 改进 SSD 算法的准确率实验结果

网络模型	mAP	fps
SSD	75%	60
改进 SSD	60%	50

5.3　实验结果分析

从瓦片损害的检测实验过程中发现，其结果是低于原算法在 PACSL VOC 数据库中的实验结果的，其具体原因和以下几点:

1、瓦片损害的形态不定，且与距离受损害的时间的长短有关——时间越长损害处的颜色越深，不利于标注从而导致降低了算法的识别精度。

2、数据量不够导致了降低了识别精度，由于没有现成的房屋屋顶瓦片数据，且一般家庭住房不愿意无人机拍照。

3、相当数量的房屋屋顶年代久远，以至于瓦片自身已经风化，为检测任务添加了不确定性

5.4 算法改进建议

随着目标识别相关算法的成熟，现在的识别精度已经超过了 80%，房屋瓦片损害检测只能充分利用其自身的特点，手工设计一些特征，从而提高算法的识别精度。与此同时，数据量是一个急需解决的问题，数据量一大，对于一些干扰就能有效的去除。

第六章　总结及展望

目前业界出现的目标检测算法有一下几种：

1、传统的目标检测算法：Cascade + Haar / SVM + HOG / DPM ；

2、候选窗 + 深度学习分类：通过提取候选区域，并对相应区域进行以深度学习方法为主的分类的方案，如：RCNN / SPP-net/ Fast-RCNN / Faster-RCNN / R-FCN 系列方法；

3、基于深度学习的回归方法：YOLO / SSD / DenseBox 等方法；

4、结合 RNN 算法的 RRC detection；结合 DPM 的 Deformable CNN 等方法；

基于深度学习方法的几个可能的方向：

1、从原始图像、低层的 feature map 层，以及高层的语义层获取更多的信息，从而得到对目标 bounding box 的更准确的估计；

2、对 bounding box 的估计可以结合图片的一些由粗到细（coarse-to-fine）的分割信息；

3、对 bounding box 的估计需要引入更多的局部的 content 的信息；

4、目标检测数据集的标注难度非常大，如何把其他如 classfication 领域学习到的知识用于检测当中，甚至是将 classification 的数据和检测数据集做 co-training（如 YOLO9000）的方式，可以从数据层面获得更多的信息；

5、更好的启发式的学习方式，人在识别物体的时候，第一次可能只是知道这是一个单独的物体，也知道 bounding box，但是不知道类别；当人类通过其他渠道学习到类别的时候，下一次就能够识别了；目标检测也是如此，我们不可能标注所有的物体的类别，但是如何将这种快速学习的机制引入，也是一个问题；

6、RRC，deformable cnn 中卷积和其他的很好的图片的操作、机器学习的思想的结合未来也有很大的空间；

7、语意信息和分割的结合，可能能够为目标检测提供更多的有用的信息；

8、场景信息也会为目标检测提供更多信息；比如天空不会出现汽车等等。

谢辞

白驹过隙，美好的大学生活匆匆而过，当毕业论文写到这章节的时候，我满脑子的关于大学这四年的回忆，生命中有太多太多给予我帮助和陪伴我成长的人需要感谢。赋予我生命的爸爸妈妈给予我的关爱、计算机学院最负责任的老师——汪姐、308 室友、物联网的同学们.....。在论文撰写之前我对深度学习相关知识还了解的不多，感谢刘立教授的悉心教导，以及对聚蜂智能科技有限公司的冉建大哥以及金华清同学的帮助表示感谢！

参考文献

[1] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

[2] tsung-yi lin, priya goyal, ross girshick, kaiming he, and piotr dollár. focal loss for dense object detection. arxiv preprint arxiv:1708.02002, 2017.

[3] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms —improving object detection with one line of code. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 5562–5570. IEEE, 2017.

[4] 黄凯奇, 任伟强, 谭铁牛, et al. 图像物体分类与检测算法综述. 计算机学报, 37(6):1225–1240, 2014.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 580–587, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In european conference on computer vision, pages 346–361. Springer, 2014.

[7] Ross Girshick. Fast r-cnn. arXiv preprint arXiv:1504.08083, 2015.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 779–788, 2016.

[10] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 1, pages I–I. IEEE, 2001.

[11] Paul Viola and Michael J Jones. Robust real-time face detection. International journal of computer vision, 57(2):137–154, 2004.

[12] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008.

[13] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. International journal of computer vision, 104(2):154–171, 2013.

[14] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 761–769, 2016.

附录

源代码

datagen.py

Listing 8 datagen.py

```
'''Load image/class/box from a annotation file.

The annotation file is organized as:
image_name #obj xmin ymin xmax ymax class_index ..
'''
from __future__ import print_function

import os
import sys
import os.path

import random
import numpy as np

import torch
import torch.utils.data as data
import torchvision.transforms as transforms

from encoder import DataEncoder
from PIL import Image, ImageOps


class ListDataset(data.Dataset):
img_size = 300

def __init__(self, root, list_file, train, transform):
'''
Args:
root: (str) ditectory to images.
list_file: (str) path to index file.
train: (boolean) train or test.
transform: ([transforms]) image transforms.
'''
self.root = root
self.train = train
self.transform = transform
```

```
37:
38: self.fnames = []
39: self.boxes = []
40: self.labels = []
41:
42: self.data_encoder = DataEncoder()
43:
44: with open(list_file) as f:
45: lines = f.readlines()
46: self.num_samples = len(lines)
47:
48: for line in lines:
49: splited = line.strip().split()
50: self.fnames.append(splited[0])
51:
52: num_objs = int(splited[1])
53: box = []
54: label = []
55: for i in range(num_objs):
56: xmin = splited[2+5*i]
57: ymin = splited[3+5*i]
58: xmax = splited[4+5*i]
59: ymax = splited[5+5*i]
60: c = splited[6+5*i]
61: box.append([float(xmin),float(ymin),float(xmax),float(ymax)])
62: label.append(int(c))
63: self.boxes.append(torch.Tensor(box))
64: self.labels.append(torch.LongTensor(label))
65:
66: def __getitem__(self, idx):
67: '''Load a image, and encode its bbox locations and class labels.
68:
69: Args:
70: idx: (int) image index.
71:
72: Returns:
73: img: (tensor) image tensor.
74: loc_target: (tensor) location targets, sized [8732,4].
75: conf_target: (tensor) label targets, sized [8732,].
76: '''
77: # Load image and bbox locations.
78: fname = self.fnames[idx]
```

```
79: img = Image.open(os.path.join(self.root, fname))
80: boxes = self.boxes[idx].clone()
81: labels = self.labels[idx]
82:
83: # Data augmentation while training.
84: if self.train:
85: img, boxes = self.random_flip(img, boxes)
86: img, boxes, labels = self.random_crop(img, boxes, labels)
87:
88: # Scale bbox locaitons to [0,1].
89: w,h = img.size
90: boxes /= torch.Tensor([w,h,w,h]).expand_as(boxes)
91:
92: img = img.resize((self.img_size,self.img_size))
93: img = self.transform(img)
94:
95: # Encode loc & conf targets.
96: loc_target, conf_target = self.data_encoder.encode(boxes, labels)
97: return img, loc_target, conf_target
98:
99: def random_flip(self, img, boxes):
100: '''Randomly flip the image and adjust the bbox locations.
101:
102: For bbox (xmin, ymin, xmax, ymax), the flipped bbox is:
103: (w-xmax, ymin, w-xmin, ymax).
104:
105: Args:
106: img: (PIL.Image) image.
107: boxes: (tensor) bbox locations, sized [#obj, 4].
108:
109: Returns:
110: img: (PIL.Image) randomly flipped image.
111: boxes: (tensor) randomly flipped bbox locations, sized [#obj, 4].
112: '''
113: if random.random() < 0.5:
114: img = img.transpose(Image.FLIP_LEFT_RIGHT)
115: w = img.width
116: xmin = w - boxes[:,2]
117: xmax = w - boxes[:,0]
118: boxes[:,0] = xmin
119: boxes[:,2] = xmax
120: return img, boxes
```

```
121:
122: def random_crop(self, img, boxes, labels):
123: '''Randomly crop the image and adjust the bbox locations.
124:
125: For more details, see 'Chapter2.2: Data augmentation' of the paper.
126:
127: Args:
128: img: (PIL.Image) image.
129: boxes: (tensor) bbox locations, sized [#obj, 4].
130: labels: (tensor) bbox labels, sized [#obj,].
131:
132: Returns:
133: img: (PIL.Image) cropped image.
134: selected_boxes: (tensor) selected bbox locations.
135: labels: (tensor) selected bbox labels.
136: '''
137: imw, imh = img.size
138: while True:
139: min_iou = random.choice([None, 0.1, 0.3, 0.5, 0.7, 0.9])
140: if min_iou is None:
141: return img, boxes, labels
142:
143: for _ in range(100):
144: w = random.randrange(int(0.1*imw), imw)
145: h = random.randrange(int(0.1*imh), imh)
146:
147: if h > 2*w or w > 2*h:
148: continue
149:
150: x = random.randrange(imw - w)
151: y = random.randrange(imh - h)
152: roi = torch.Tensor([[x, y, x+w, y+h]])
153:
154: center = (boxes[:,:2] + boxes[:,2:]) / 2  # [N,2]
155: roi2 = roi.expand(len(center), 4)  # [N,4]
156: mask = (center > roi2[:,:2]) & (center < roi2[:,2:])  # [N,2]
157: mask = mask[:,0] & mask[:,1]  #[N,]
158: if not mask.any():
159: continue
160:
161: selected_boxes = boxes.index_select(0, mask.nonzero().squeeze(1))
162:
```

```
163: iou = self.data_encoder.iou(selected_boxes, roi)
164: if iou.min() < min_iou:
165: continue
166:
167: img = img.crop((x, y, x+w, y+h))
168: selected_boxes[:,0].add_(-x).clamp_(min=0, max=w)
169: selected_boxes[:,1].add_(-y).clamp_(min=0, max=h)
170: selected_boxes[:,2].add_(-x).clamp_(min=0, max=w)
171: selected_boxes[:,3].add_(-y).clamp_(min=0, max=h)
172: return img, selected_boxes, labels[mask]
173:
174: def __len__(self):
175: return self.num_samples
```

ssd.py

```
 1: import math
 2: import itertools
 3:
 4: import torch
 5: import torch.nn as nn
 6: import torch.nn.functional as F
 7: import torch.nn.init as init
 8:
 9: from torch.autograd import Variable
10:
11: from multibox_layer import MultiBoxLayer
12:
13:
14: class L2Norm2d(nn.Module):
15: '''L2Norm layer across all channels.'''
16: def __init__(self, scale):
17: super(L2Norm2d, self).__init__()
18: self.scale = scale
19:
20: def forward(self, x, dim=1):
21: '''out = scale * x / sqrt(\sum x_i^2)'''
22: return self.scale * x * x.pow(2).sum(dim).clamp(min=1e-12).rsqrt().expand_as(
23:
24:
25: class SSD300(nn.Module):
26: input_size = 300
27:
```

```
28: def __init__(self):
29: super(SSD300, self).__init__()
30:
31: # model
32: self.base = self.VGG16()
33: self.norm4 = L2Norm2d(20)
34:
35: self.conv5_1 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1)
36: self.conv5_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1)
37: self.conv5_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1)
38:
39: self.conv6 = nn.Conv2d(512, 1024, kernel_size=3, padding=6, dilation=6)
40:
41: self.conv7 = nn.Conv2d(1024, 1024, kernel_size=1)
42:
43: self.conv8_1 = nn.Conv2d(1024, 256, kernel_size=1)
44: self.conv8_2 = nn.Conv2d(256, 512, kernel_size=3, padding=1, stride=2)
45:
46: self.conv9_1 = nn.Conv2d(512, 128, kernel_size=1)
47: self.conv9_2 = nn.Conv2d(128, 256, kernel_size=3, padding=1, stride=2)
48:
49: self.conv10_1 = nn.Conv2d(256, 128, kernel_size=1)
50: self.conv10_2 = nn.Conv2d(128, 256, kernel_size=3)
51:
52: self.conv11_1 = nn.Conv2d(256, 128, kernel_size=1)
53: self.conv11_2 = nn.Conv2d(128, 256, kernel_size=3)
54:
55: # multibox layer
56: self.multibox = MultiBoxLayer()
57:
58: def forward(self, x):
59: hs = []
60: h = self.base(x)
61: hs.append(self.norm4(h))  # conv4_3
62:
63: h = F.max_pool2d(h, kernel_size=2, stride=2, ceil_mode=True)
64:
65: h = F.relu(self.conv5_1(h))
66: h = F.relu(self.conv5_2(h))
67: h = F.relu(self.conv5_3(h))
68: h = F.max_pool2d(h, kernel_size=3, padding=1, stride=1, ceil_mode=True)
69:
```

```
70: h = F.relu(self.conv6(h))
71: h = F.relu(self.conv7(h))
72: hs.append(h)  # conv7
73:
74: h = F.relu(self.conv8_1(h))
75: h = F.relu(self.conv8_2(h))
76: hs.append(h)  # conv8_2
77:
78: h = F.relu(self.conv9_1(h))
79: h = F.relu(self.conv9_2(h))
80: hs.append(h)  # conv9_2
81:
82: h = F.relu(self.conv10_1(h))
83: h = F.relu(self.conv10_2(h))
84: hs.append(h)  # conv10_2
85:
86: h = F.relu(self.conv11_1(h))
87: h = F.relu(self.conv11_2(h))
88: hs.append(h)  # conv11_2
89:
90: loc_preds, conf_preds = self.multibox(hs)
91: return loc_preds, conf_preds
92:
93: def VGG16(self):
94: '''VGG16 layers.'''
95: cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512]
96: layers = []
97: in_channels = 3
98: for x in cfg:
99: if x == 'M':
100: layers += [nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)]
101: else:
102: layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
103: nn.ReLU(True)]
104: in_channels = x
105: return nn.Sequential(*layers)
```

multibox_layer.py

```
1: from __future__ import print_function
2:
3: import math
4:
```

```
 5: import torch
 6: import torch.nn as nn
 7: import torch.nn.init as init
 8: import torch.nn.functional as F
 9:
10: from torch.autograd import Variable
11:
12:
13: class MultiBoxLayer(nn.Module):
14: num_classes = 21
15: num_anchors = [4,6,6,6,4,4]
16: in_planes = [512,1024,512,256,256,256]
17:
18: def __init__(self):
19: super(MultiBoxLayer, self).__init__()
20:
21: self.loc_layers = nn.ModuleList()
22: self.conf_layers = nn.ModuleList()
23: for i in range(len(self.in_planes)):
24: self.loc_layers.append(nn.Conv2d(self.in_planes[i], self.num_anchors[i]*4, ke
25: self.conf_layers.append(nn.Conv2d(self.in_planes[i], self.num_anchors[i]*21,
26:
27: def forward(self, xs):
28: '''
29: Args:
30: xs: (list) of tensor containing intermediate layer outputs.
31:
32: Returns:
33: loc_preds: (tensor) predicted locations, sized [N,8732,4].
34: conf_preds: (tensor) predicted class confidences, sized [N,8732,21].
35: '''
36: y_locs = []
37: y_confs = []
38: for i,x in enumerate(xs):
39: y_loc = self.loc_layers[i](x)
40: N = y_loc.size(0)
41: y_loc = y_loc.permute(0,2,3,1).contiguous()
42: y_loc = y_loc.view(N,-1,4)
43: y_locs.append(y_loc)
44:
45: y_conf = self.conf_layers[i](x)
46: y_conf = y_conf.permute(0,2,3,1).contiguous()
```

```
47: y_conf = y_conf.view(N,-1,21)
48: y_confs.append(y_conf)
49:
50: loc_preds = torch.cat(y_locs, 1)
51: conf_preds = torch.cat(y_confs, 1)
52: return loc_preds, conf_preds
```

multibox_loss.py

```
 1: from __future__ import print_function
 2:
 3: import math
 4:
 5: import torch
 6: import torch.nn as nn
 7: import torch.nn.init as init
 8: import torch.nn.functional as F
 9:
10: from torch.autograd import Variable
11:
12:
13: class MultiBoxLoss(nn.Module):
14: num_classes = 21
15:
16: def __init__(self):
17: super(MultiBoxLoss, self).__init__()
18:
19: def cross_entropy_loss(self, x, y):
20: '''Cross entropy loss w/o averaging across all samples.
21:
22: Args:
23: x: (tensor) sized [N,D].
24: y: (tensor) sized [N,].
25:
26: Return:
27: (tensor) cross entroy loss, sized [N,].
28: '''
29: xmax = x.data.max()
30: log_sum_exp = torch.log(torch.sum(torch.exp(x-xmax), 1)) + xmax
31: return log_sum_exp - x.gather(1, y.view(-1,1))
32:
33: def test_cross_entropy_loss(self):
34: a = Variable(torch.randn(10,4))
```

```
35: b = Variable(torch.ones(10).long())
36: loss = self.cross_entropy_loss(a,b)
37: print(loss.mean())
38: print(F.cross_entropy(a,b))
39:
40: def hard_negative_mining(self, conf_loss, pos):
41: '''Return negative indices that is 3x the number as postive indices.
42:
43: Args:
44: conf_loss: (tensor) cross entroy loss between conf_preds and conf_targets, siz
45: pos: (tensor) positive(matched) box indices, sized [N,8732].
46:
47: Return:
48: (tensor) negative indices, sized [N,8732].
49: '''
50: batch_size, num_boxes = pos.size()
51:
52: conf_loss[pos] = 0  # set pos boxes = 0, the rest are neg conf_loss
53: conf_loss = conf_loss.view(batch_size, -1)  # [N,8732]
54:
55: _,idx = conf_loss.sort(1, descending=True)  # sort by neg conf_loss
56: _,rank = idx.sort(1)  # [N,8732]
57:
58: num_pos = pos.long().sum(1)  # [N,1]
59: num_neg = torch.clamp(3*num_pos, max=num_boxes-1)  # [N,1]
60:
61: neg = rank < num_neg.expand_as(rank)  # [N,8732]
62: return neg
63:
64: def forward(self, loc_preds, loc_targets, conf_preds, conf_targets):
65: '''Compute loss between (loc_preds, loc_targets) and (conf_preds, conf_target
66:
67: Args:
68: loc_preds: (tensor) predicted locations, sized [batch_size, 8732, 4].
69: loc_targets: (tensor) encoded target locations, sized [batch_size, 8732, 4].
70: conf_preds: (tensor) predicted class confidences, sized [batch_size, 8732, nu
71: conf_targets: (tensor) encoded target classes, sized [batch_size, 8732].
72:
73: loss:
74: (tensor) loss = SmoothL1Loss(loc_preds, loc_targets) + CrossEntropyLoss(conf_
75: '''
76: batch_size, num_boxes, _ = loc_preds.size()
```

```
77:
78: pos = conf_targets>0  # [N,8732], pos means the box matched.
79: num_matched_boxes = pos.data.long().sum()
80: if num_matched_boxes == 0:
81: return Variable(torch.Tensor([0]))
82:
83: ##############################################################
84: # loc_loss = SmoothL1Loss(pos_loc_preds, pos_loc_targets)
85: ##############################################################
86: pos_mask = pos.unsqueeze(2).expand_as(loc_preds)     # [N,8732,4]
87: pos_loc_preds = loc_preds[pos_mask].view(-1,4)       # [#pos,4]
88: pos_loc_targets = loc_targets[pos_mask].view(-1,4)  # [#pos,4]
89: loc_loss = F.smooth_l1_loss(pos_loc_preds, pos_loc_targets, size_average=Fals
90:
91: ##############################################################
92: # conf_loss = CrossEntropyLoss(pos_conf_preds, pos_conf_targets)
93: #           + CrossEntropyLoss(neg_conf_preds, neg_conf_targets)
94: ##############################################################
95: conf_loss = self.cross_entropy_loss(conf_preds.view(-1,self.num_classes), \
96: conf_targets.view(-1))  # [N*8732,]
97: neg = self.hard_negative_mining(conf_loss, pos)     # [N,8732]
98:
99: pos_mask = pos.unsqueeze(2).expand_as(conf_preds)  # [N,8732,21]
100: neg_mask = neg.unsqueeze(2).expand_as(conf_preds)  # [N,8732,21]
101: mask = (pos_mask+neg_mask).gt(0)
102:
103: pos_and_neg = (pos+neg).gt(0)
104: preds = conf_preds[mask].view(-1,self.num_classes)  # [#pos+#neg,21]
105: targets = conf_targets[pos_and_neg]                 # [#pos+#neg,]
106: conf_loss = F.cross_entropy(preds, targets, size_average=False)
107:
108: loc_loss /= num_matched_boxes
109: conf_loss /= num_matched_boxes
110: print('%f␣%f' % (loc_loss.data[0], conf_loss.data[0]), end='␣')
111: return loc_loss + conf_loss
```

encoder.py

```
1: '''Encode target locations and labels.'''
2: import torch
3:
4: import math
5: import itertools
```

```
 6:
 7: class DataEncoder:
 8: def __init__(self):
 9: '''Compute default box sizes with scale and aspect transform.'''
10: scale = 300.
11: steps = [s / scale for s in (8, 16, 32, 64, 100, 300)]
12: sizes = [s / scale for s in (30, 60, 111, 162, 213, 264, 315)]
13: aspect_ratios = ((2,), (2,3), (2,3), (2,3), (2,), (2,))
14: feature_map_sizes = (38, 19, 10, 5, 3, 1)
15:
16: num_layers = len(feature_map_sizes)
17:
18: boxes = []
19: for i in range(num_layers):
20: fmsize = feature_map_sizes[i]
21: for h,w in itertools.product(range(fmsize), repeat=2):
22: cx = (w + 0.5)*steps[i]
23: cy = (h + 0.5)*steps[i]
24:
25: s = sizes[i]
26: boxes.append((cx, cy, s, s))
27:
28: s = math.sqrt(sizes[i] * sizes[i+1])
29: boxes.append((cx, cy, s, s))
30:
31: s = sizes[i]
32: for ar in aspect_ratios[i]:
33: boxes.append((cx, cy, s * math.sqrt(ar), s / math.sqrt(ar)))
34: boxes.append((cx, cy, s / math.sqrt(ar), s * math.sqrt(ar)))
35:
36: self.default_boxes = torch.Tensor(boxes)
37:
38: def iou(self, box1, box2):
39: '''Compute the intersection over union of two set of boxes, each box is [x1,y
40:
41: Args:
42: box1: (tensor) bounding boxes, sized [N,4].
43: box2: (tensor) bounding boxes, sized [M,4].
44:
45: Return:
46: (tensor) iou, sized [N,M].
47: '''
```

```
48: N = box1.size(0)
49: M = box2.size(0)
50:
51: lt = torch.max(
52: box1[:,:2].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
53: box2[:,:2].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
54: )
55:
56: rb = torch.min(
57: box1[:,2:].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
58: box2[:,2:].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
59: )
60:
61: wh = rb - lt  # [N,M,2]
62: wh[wh<0] = 0  # clip at 0
63: inter = wh[:,:,0] * wh[:,:,1]  # [N,M]
64:
65: area1 = (box1[:,2]-box1[:,0]) * (box1[:,3]-box1[:,1])  # [N,]
66: area2 = (box2[:,2]-box2[:,0]) * (box2[:,3]-box2[:,1])  # [M,]
67: area1 = area1.unsqueeze(1).expand_as(inter)  # [N,] -> [N,1] -> [N,M]
68: area2 = area2.unsqueeze(0).expand_as(inter)  # [M,] -> [1,M] -> [N,M]
69:
70: iou = inter / (area1 + area2 - inter)
71: return iou
72:
73: def encode(self, boxes, classes, threshold=0.5):
74: '''Transform target bounding boxes and class labels to SSD boxes and classes.
75:
76: Match each object box to all the default boxes, pick the ones with the
77: Jaccard-Index > 0.5:
78: Jaccard(A,B) = AB / (A+B-AB)
79:
80: Args:
81: boxes: (tensor) object bounding boxes (xmin,ymin,xmax,ymax) of a image, sized
82: classes: (tensor) object class labels of a image, sized [#obj,].
83: threshold: (float) Jaccard index threshold
84:
85: Returns:
86: boxes: (tensor) bounding boxes, sized [#obj, 8732, 4].
87: classes: (tensor) class labels, sized [8732,]
88: '''
89: default_boxes = self.default_boxes
```

```
num_default_boxes = default_boxes.size(0)
num_objs = boxes.size(0)

iou = self.iou(  # [#obj,8732]
boxes,
torch.cat([default_boxes[:,:2] - default_boxes[:,2:]/2,
default_boxes[:,:2] + default_boxes[:,2:]/2], 1)
)

iou, max_idx = iou.max(0)  # [1,8732]
max_idx.squeeze_(0)        # [8732,]
iou.squeeze_(0)            # [8732,]

boxes = boxes[max_idx]     # [8732,4]
variances = [0.1, 0.2]
cxcy = (boxes[:,:2] + boxes[:,2:])/2 - default_boxes[:,:2]  # [8732,2]
cxcy /= variances[0] * default_boxes[:,2:]
wh = (boxes[:,2:] - boxes[:,:2]) / default_boxes[:,2:]      # [8732,2]
wh = torch.log(wh) / variances[1]
loc = torch.cat([cxcy, wh], 1)  # [8732,4]

conf = 1 + classes[max_idx]   # [8732,], background class = 0
conf[iou<threshold] = 0       # background
return loc, conf

def nms(self, bboxes, scores, threshold=0.5, mode='union'):
'''Non maximum suppression.

Args:
bboxes: (tensor) bounding boxes, sized [N,4].
scores: (tensor) bbox scores, sized [N,].
threshold: (float) overlap threshold.
mode: (str) 'union' or 'min'.

Returns:
keep: (tensor) selected indices.

Ref:
https://github.com/rbgirshick/py-faster-rcnn/blob/master/lib/nms/py_cpu_nms.p
'''
x1 = bboxes[:,0]
y1 = bboxes[:,1]
```

```
132: x2 = bboxes[:,2]
133: y2 = bboxes[:,3]
134:
135: areas = (x2-x1) * (y2-y1)
136: _, order = scores.sort(0, descending=True)
137:
138: keep = []
139: while order.numel() > 0:
140: i = order[0]
141: keep.append(i)
142:
143: if order.numel() == 1:
144: break
145:
146: xx1 = x1[order[1:]].clamp(min=x1[i])
147: yy1 = y1[order[1:]].clamp(min=y1[i])
148: xx2 = x2[order[1:]].clamp(max=x2[i])
149: yy2 = y2[order[1:]].clamp(max=y2[i])
150:
151: w = (xx2-xx1).clamp(min=0)
152: h = (yy2-yy1).clamp(min=0)
153: inter = w*h
154:
155: if mode == 'union':
156: ovr = inter / (areas[i] + areas[order[1:]] - inter)
157: elif mode == 'min':
158: ovr = inter / areas[order[1:]].clamp(max=areas[i])
159: else:
160: raise TypeError('Unknown␣nms␣mode:␣%s.' % mode)
161:
162: ids = (ovr<=threshold).nonzero().squeeze()
163: if ids.numel() == 0:
164: break
165: order = order[ids+1]
166: return torch.LongTensor(keep)
167:
168: def decode(self, loc, conf):
169: '''Transform predicted loc/conf back to real bbox locations and class labels.
170:
171: Args:
172: loc: (tensor) predicted loc, sized [8732,4].
173: conf: (tensor) predicted conf, sized [8732,21].
```

```
174:
175: Returns:
176: boxes: (tensor) bbox locations, sized [#obj, 4].
177: labels: (tensor) class labels, sized [#obj,1].
178: '''
179: variances = [0.1, 0.2]
180: wh = torch.exp(loc[:,2:]*variances[1]) * self.default_boxes[:,2:]
181: cxcy = loc[:,:2] * variances[0] * self.default_boxes[:,2:] + self.default_box
182: boxes = torch.cat([cxcy-wh/2, cxcy+wh/2], 1)  # [8732,4]
183:
184: max_conf, labels = conf.max(1)  # [8732,1]
185: ids = labels.squeeze(1).nonzero().squeeze(1)  # [#boxes,]
186:
187: keep = self.nms(boxes[ids], max_conf[ids].squeeze(1))
188: return boxes[ids][keep], labels[ids][keep], max_conf[ids][keep]
```