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 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 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 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

House Tile Damage Detection Based on SSD Network Model

Abstract:In recent years, the frequent occurrence of natural disasters, especially damage caused by hailstones on house tiles, has resulted in the United States housing The housing claims industry urgently needs a set of non-damaging, low-cost testing programs for damage to house tiles. With commercial unmanned The development of the aircraft and the appearance of various detection algorithms based on deep learning have made it possible to pass unmanned testing equipment. Machines, combined with target detection algorithms instead of workers to detect house tiles become a reality. The study is combed In recent years, the main target detection algorithm, taking into account the characteristics of tiles and hail, based on SSD: Single Shot The MultiBox Detector algorithm performs tile damage detection. The main improvements include two points: 1.Adopt Focal Loss replaces the loss function used in the original paper. Through experiments, this improvement will improve the accuracy by about 2%;2.Using Soft-NMS instead of NMS, this experiment has improved the accuracy by about 1%. Integrated two Improve the program, the final experimental results not only meet the requirements of the insurance industry and detection accuracy reached 65% mAP, The detection speed reached 80FPS.

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

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，据美国国家海洋和大气管理局 (NOAA) 统计，多次强大的冰雹灾害致使 2017 年成为遭受冰雹灾害最为严重的年份之一。自然灾害尤其是冰雹灾害让美国遭受了超过 3000 亿美元的损失[1]。

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，品种齐全，数量庞大的无人机正迅速的进入市场。较小的无人机低至 10 美元就可以买到。消费类无人机的用途主要是拍摄和娱乐，那些可以执行特定任务的无人机则开始用于商业用途。

机器视觉的发展：谷歌的 AlphaGo 将对深度学习的研究推向高潮，不过深度学习有关理论和实践还处在发展阶段，也都还存在许多有待解决的问题，然而处在 “大数据” 时代，丰富的计算资源使我们能很快地验证新模型、新理论。人工智能时代的开启必然会很大程度的从人们生活的方方面面改变这个世界。

1.1.2　研究意义

针对传统的房屋瓦片损害检测方法存在着周期长、评估主观、高危和对房屋瓦片具有破坏性等缺点，本项研究尝试利用搭载了计算机视觉技术的无人机对房屋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

[1]美国中文网,2018-1-8

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。

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。由于检测到的对象的数量是不定的，所以对象检测的输出长度也是可变的。

1.2.1 研究现状

在深度学习被广泛应用于图像特征提取之前，传统的“目标检测”方法通常是区域选择、提取特征、分类回归三个步骤 [2]，这会导致两个难以解决且至关重要的问题；一个是区域选择的策略 (Region Proposal) 效果不好、时间复杂度特别高；二个是手工设计的特征其鲁棒性较差。随着大数据时代的到来，“目标检测”算法主要形成两种思想。

基于区域预测加分类的目标检测方法，RCNN 系列 (R-CNN [3]、SPPnet [4]、Fast R-CNN [5] 以及 Faster R-CNN [6]) 在目标检测精度方面有较好的效果，因为这一类方法将边框预先回归后才传入网络进行训练，所以检测的精度高。这类方法用了两步进行目标检测所以被人们称为“Two Stage”的方法。除此之外，从简化边框回归的角度而产生的以 YOLO [7] 为代表的系列算法，只做了一次边框回归和打分，被人们称为“One Stage”的方法。检测速度快是这类方法的最大特点，虽然 YOLO 系列

算法能达到实时的效果，但是对尺度小的目标训练非常不充分，检测效果不是很理想。换句话说，YOLO 系列算法对目标的尺度非常敏感，而且缺少对尺度变化大的物体的泛化能力。

消化吸收了 YOLO 和 Faster R-CNN 的优缺点，WeiLiu 等人提出了 Single Shot MultiBox Detector [1] 算法，简称为 SSD。SSD 的改进总体来说有三点: 其一、SSD 整体设计采取了 “One Stage” 的思想，以此提高检测速度。其二、网络中融入了 Faster R-CNN [6] 中的 anchors 思想，以此提高检测的精度。其三、对 feature map 分层提取并依次计算边框回归值和分类的操作，以此可以适应多尺度目标的训练和检测任务，解决了 “One Stage” 系列算法对小目标检测精度不好的问题。SSD 的出现是该时期的集大成者，向大家证明了实时高精度目标检测的任务是可以实现的。

1.2.2 发展难点

对象的数量是不确定的 (Variable Number of Objects)。在设计网络模型时，一般要将数据约定为大小固定的向量。由于图片中对象的数量事先是未知的，所以我们没办法预先约定好输出的维度。这无疑会增加模型的复杂性。

对象检测窗口的调整 (Resizing)。另一个棘手的问题是不同对象其大小也是不同的，意思是不论是大的对象还是小到只有几个像素大小的对象，都希望能有较好的检测效果。利用多尺度的滑动窗口可以解决多尺度对象的问题，但是效率很低。

建模。最后的挑战是一个模型要同时解决两个截然不同的需求——定位和分类。

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，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 、采用 Focal 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：谢辞。

第二章　目标检测相关算法

2.1　目标检测算法概述

目标检测是人工智能领域的基础问题 [8]，纵观其发展历程，2010 年是一个转折点。自 2013 年 Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation [3] 一文的发表，基于卷积神经网络的特征提取代替了传统手工提取特征方法，从此为目标检测领域引入了深度学习这个高效的方法。跟着历史的潮流，本章将简要梳理“目标检测”算法的两种重要思想和扼要地介绍在这些重要思想影响下的那些具有代表性的算法。

概括地说，“目标检测”系列算法宏观上可以划分为两类，一个是以 R-CNN [3] 为代表的“Two-Stage”方法，另一个则是以 YOLO [7] 为代表的”One-Stage”方法。下面分别解释一下什么是“One-Stage”方法和“Two-Stage”方法。

One Stage: 顾名思义，对预测的目标物体直接进行检测，即直接利用位置回归进行目标检测，其特点是简单快速，但是精度要低于“Two Stage”方法，其代表算法是 YOLO [7] 和 SSD [1]。

Two-Stage: 顾名思义，分两步进行目标检测：1、生成可能区域 (Region Proposal) & 利用 CNN 提取特征。2、放入分类器进行分类，与此同时修正位置。这一流派的算法都离不开 Region Proposal，优缺点参半，一方面提高了精度，另一方面却损失了速度，其代表算法是 Faster R-CNN [6]。

总的说来，“One-Stage”和“Two-Stage”方法，都是在同一个衡量标准下寻找速度与精度的平衡点。“One-Stage”的方法倾向快，”Two-Stage”的方法更倾向于准。接下来将分别介绍在两种重要思路影响下的具有代表性的算法。

2.2　R-CNN 系列

R-CNN 系列算法是一个很庞大的算法簇，Ross Girshick 发表 Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation [3] 一文以来衍生出非常多的算法。在此，只介绍与 R-CNN 关系最亲密的算法，他们的发展顺序如图2.1

图 2.1 RCNN 系列算法发展顺序

他们在整个算法簇发展的过程中，更充分地利用 Feature Maps 的信息是系列算法发展的脉络。

2.2.1 R-CNN

RCNN:(Region CNN) [3] 首次将深度学习应用到目标检测算法中。在 PASCAL VOC[2]的目标检测竞赛中，其作者 Ross Girshick 多次带领团队折桂。图3.1这个模型，“目标检测”领域内第一次结合了神经网络的结构图，其深远意义不言而喻。

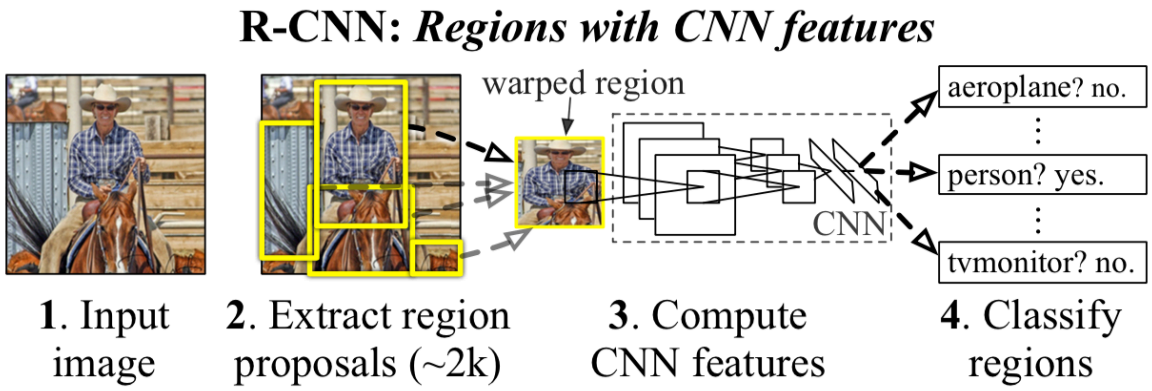

图 2.2 RCNN 算法框架

该算法对目标检测中的两个关键问题提供了重要思路:

解决问题一、速度。 R-CNN 使用启发式搜索方法 (Selective Search [9]) 取代使用滑窗的区域选择算法。传统的区域选择算法，滑动一个窗口就需要检测一次，因此相邻窗口信息重叠高，导致检测速度慢。启发式搜索方法先生成候选区域再检测，大大降低了信息冗余的程度，从而检测速度得到提高。

解决问题二、特征的鲁棒性。 R-CNN 使用卷积神经网络 (CNN) 提取特征取代

[2]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

人工设定的特征如 Haar、HOG 等。传统的手工提取特征鲁棒性差，特征设计复杂。仅限于低层次 (Low-level) 的特征如颜色、纹理等。使用 CNN 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% 提高到 53.7 % 。

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、**生成候选区域**: 利用 Selective Search [9] 算法从原始图像中生成约 2k-3k 个候选区域。生成步骤：1、先将原始图像分割成小块区域。2、检索所有的小区域，合并最可能是属于同一个物体的两个区域。重复该操作直到整张图像中只剩下一个位置区域。3、输出所有候选的区域包括合并的区域，共同作为所谓的候选区域。

2、**提取特征**: 利用 2012 年，Hinton 在 Image Net 竞赛上提出的神经网络[3]提取图像中的特征。

3、**判断类别**: 对于每一类别的目标，使用线性 SVM [10] 分类器进行判别。输入为 4096 维特征向量，输出为此类别的置信度。同时原算法使用"Hard Negative Mining" [11] 方法。使正负样本的数量保持平衡。

4、**精修位置**: 目标检测的结果好不好是看检测框与实际位置的重叠面积，故需要对检测的位置进行精修，以此提高检测精度。

2.2.2 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 [4] ，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：

[3]Hinton 在 2012 年提出的 AlexNet 网络

缺点一、算法过程冗余。R-CNN 算法是先生成候选区域，再对该区域进行卷积，其导致的问题有二：其一是对相同区域进行了多次卷积，因为候选区域难免会有大量交叠的区域；其二是对存储空间浪费巨大，因为每个候选区域进行卷积后都需要新的存储空间。何恺明等人针对上述两点不足，交换了生成候选区域与卷积的顺序，通过巧妙地改变，不仅减少存储量而且大幅度地提高了训练速度。

缺点二、图片缩放、裁剪问题。如图2.4所示，对于原始图片无论是缩放 (Warp) 还是裁剪 (Crop)，都会使原始图片的信息很大程度上的丢失从而影响训练效果。从示例图中可以看出，当把小车剪裁成一个车门，人们看到这个裁剪后的图片就难以判断图片的整体目标是一辆车；把一座高塔图片进行缩放后，也会使得机器的识别难度巨大。

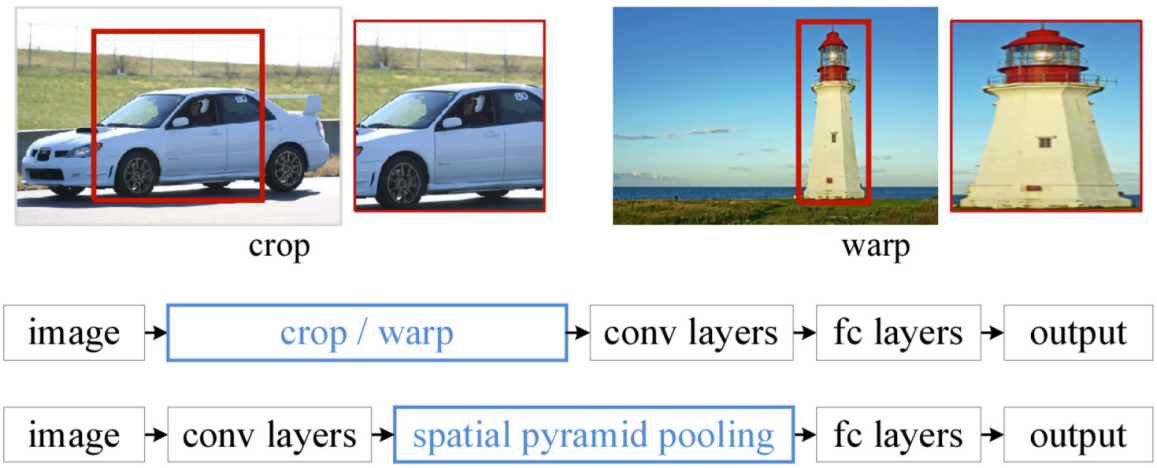

图 2.3 因剪裁和缩放导致视差

问题的关键在于找到一个不对图片进行变形，且依然保留了原图整体信息的方法直接进行学习。最后，何恺明等人找到了问题的根源：全连接层 (FC Layer) 需要固定其输入向量维度，于是只需要定义一个特殊的池化层，使其输出的维度满足全连接层固定输入维度的要求即可。这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，如图2.4，要输入的维度 64*256 ，那么可以这样组合 32*256 + 16*256 + 8*256 + 8*256。

SPP Net 的出现对整个目标检测及其相关领域产生了重要的影响，他不仅减少了计算冗余从而提高了检测速度，更重要的意义在于解禁了输入固定尺寸这一束缚，

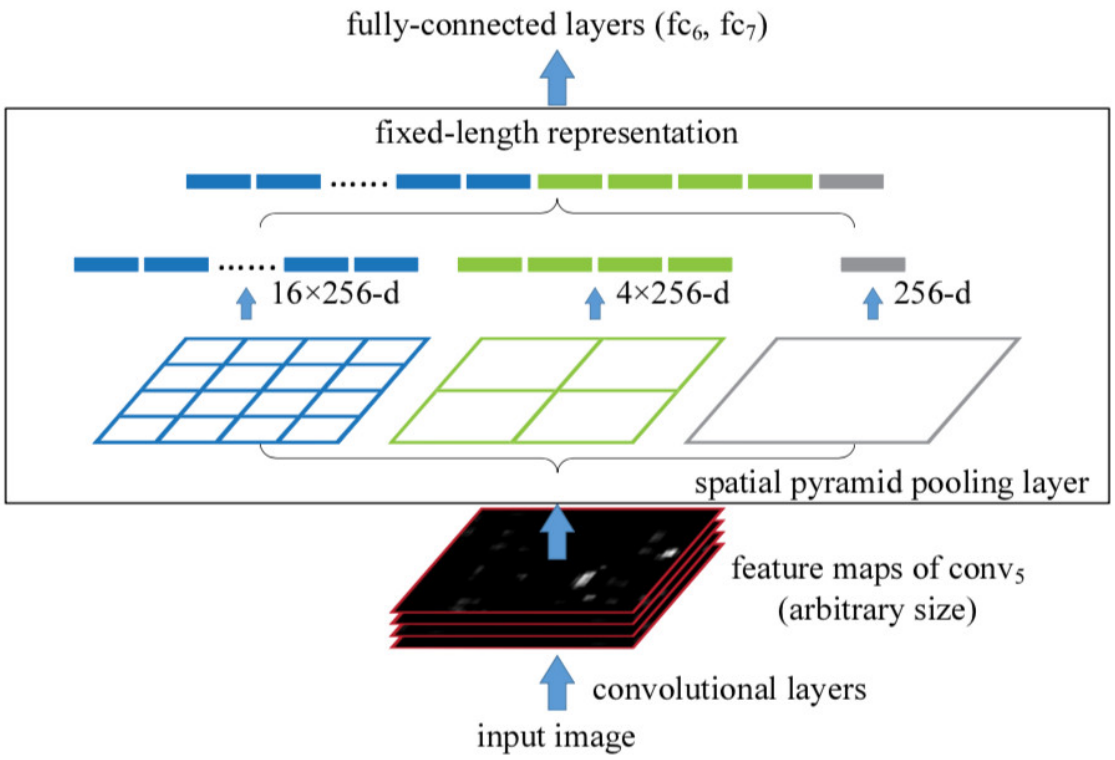

图 2.4 输入维度的组合方式

也因此被后面出现的算法广泛采用。本文基于的 SSD 算法就是更是从中受到启发。

2.2.3 Fast R-CNN

继 2014 年的 RCNN 之后，Ross Girshick 在 2015 年发表了 Fast RCNN [5]，构思巧妙，算法流程更加紧凑，目标检测的速度也大幅提高。Fast R-CNN 引用了 SPP Net 的贡献。概括来说，贡献最大的地方在于用并行结构替换了原算法中的串行结构。Fast RCNN 和 RCNN 相比，同样使用最大规模的网络，测试时间从 47 秒减少为 0.32 秒，训练时间从 84 小时减少为 9.5 小时。在 PASCAL-VOC2007 上的准确率，约在 66%-67% 之间。其算法框架如图2.5

R-CNN 算法是先对候选框区域进行分类，如果检测到目标则对边界框 (Bounding Box) 进行精修、回归。全过程是串联的。能不能改成并联的呢? Ross Girshick 将算法过程改成了并行结构——边分类，边对 Bounding box 进行回归。用一个 Loss 函数将 conf-loss 和 loc-loss 整合到了一起，同时也吸收了 SPP Net 算法的长处，因此其速度和精度都得到了提高。

算法流程:

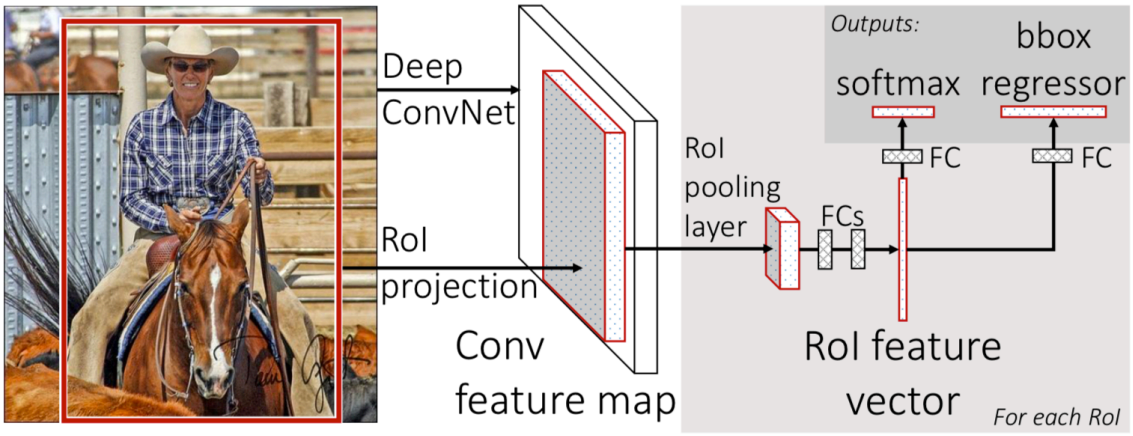

图 2.5 Fast R-CNN 算法框架

> 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 算法针对 RCNN 算法中的三个方面进行了改进:

第一个方面、测试时的速度慢。由于 RCNN 算法中对一张图像内的候选框之间存在大量交叠部分，所以提取特征时的操作冗余，Fast R-CNN 算法将整张归一化后的图像直接送入深度神经网络中。在与后面的网络邻接时，才将候选框的信息加入进来，从而共享了卷积操作，达到了提高速度的目的。

第二个方面、训练时的速度慢。其具体的原因同第一个方面，在训练时候，先将一张归一化后的图像送入深度神经网络中，紧接着将这幅图像上提取出的候选区域传入网络。所以避免了候选区域特征的重复计算。

第三个方面、训练存储空间大。RCNN 算法中需要大量特征作为训练样本用来训练独立的分类器和回归器，Fast R-CNN 不需要额外的存储，将类别判断和位置精修合并到一起，用一个统一的深度神经网络来实现。其网络模型如图2.6

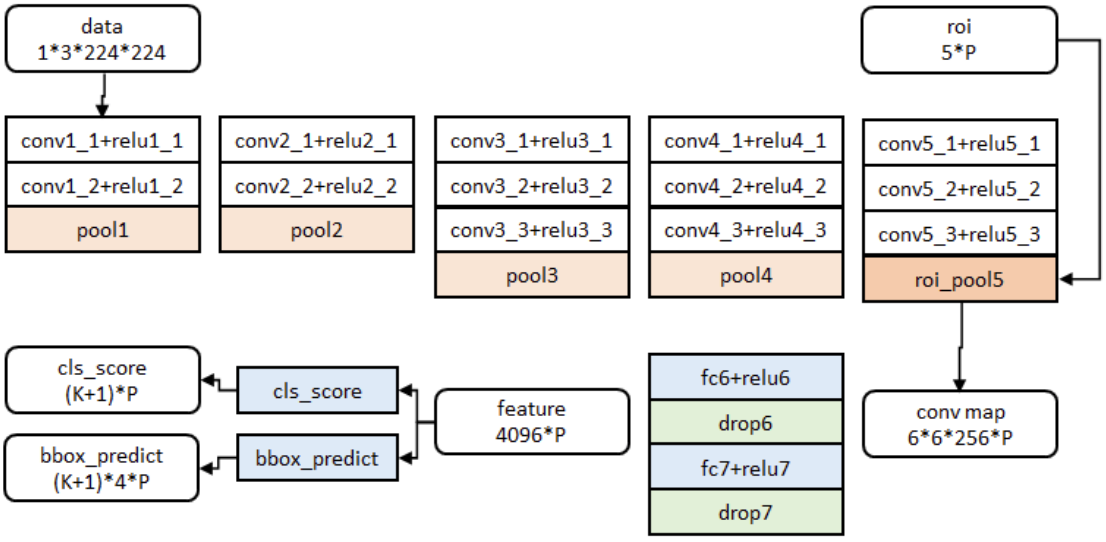

图 2.6 Fast R-CNN 网络模型

损失函数: loss_cls 评估分类损失。由其真实类别 u 对应的概率决定:

$$L_{cls} = -\log p_u \tag{2.1}$$

loss_bbox 评估检测框定位损失。由其真实类别对应的预测位置参数 t^u 和真实位置关于平移缩放 v 的因子来决定:

$$L_{loc} = \sum_{i=1}^{4} g(t_i^u - v_i) \tag{2.2}$$

g 为 Smooth L1 误差，对 outlier 不敏感:

$$g(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{2.3}$$

如果分类为背景则不考虑定位代价，最后总的损失为二者加权和:

$$L = \begin{cases} L_{cls} + \lambda L_{loc} & u \\ L_{cls} & u \end{cases} \tag{2.4}$$

2.2.4 Faster R-CNN

继发表 RCNN，Fast RCNN 两篇力作之后，Ross Girshick 等人又在 2015 年提出了 Faster R-CNN [6] 算法。该算法在 PASCAL VOC 数据集上准确率为 59.9%, 在简单网络目标上的检测速度达到 17FPS；复杂网络目标上的检测准确率 78.8%，速度达到 5FPS。在 Faster R-CNN 算法被提出前，所有的目标检测算法都是基于 Low Level 特征采用启发式搜索算法生产候选区域。这种策略存在着两个缺点：

第一个缺点候选区域具有不确定性。“Two Stage” 系列算法为了解决这个问题生成了大量无效区域造成了大量多余的计算、可是如果减少了候选区域则又会使之漏检；

第二个缺点算法第一步——生成候选区域的算法是在 CPU 上运行的，与在 GPU 上训练的跨结构交互损失了算法效率。

于是乎，针对上述两个缺点，任少卿等人提出利用神经网络代替 Selective Search 算法自己去学习生成候选区域。将这个网络称之为 RPN:Region Proposal Networks。其结构如图2.7。

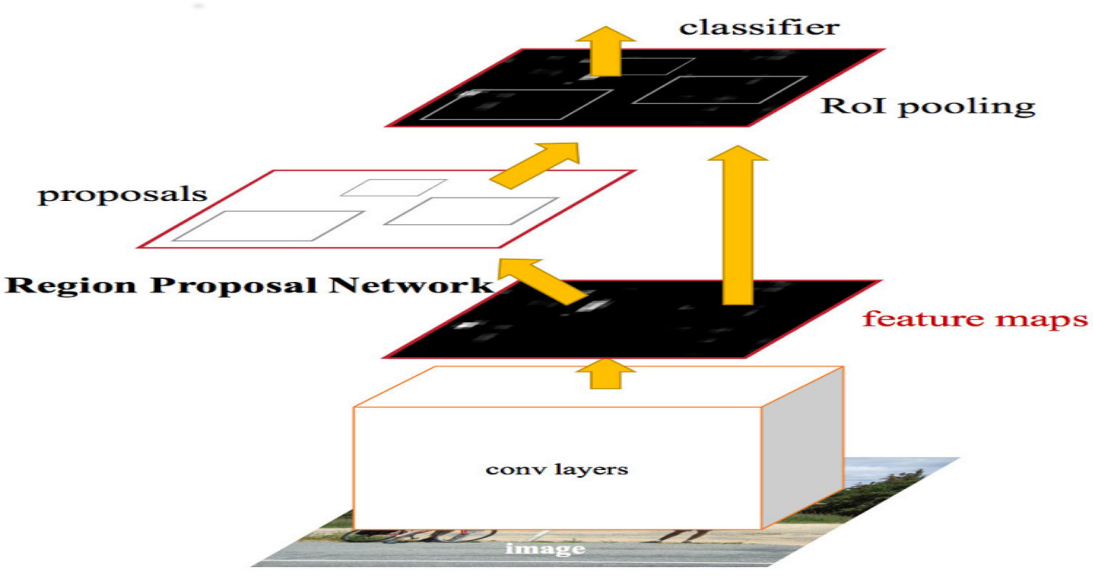

图 2.7 Faster RCNN 算法架构

神经网络可以学到比 Low Leval 特征更加抽象、语义、高层的特征，同时大大提高了候选区域的可靠程度，利用 RPN 生成候选区域的方法一石二鸟，可以从图2.7看

出，将 RPNs 嵌入原有网络使 RPNs 和 RoI Pooling 层共用前面的卷积神经网络，整合原有网络和 RPNs 网络一起参与预测，参数量和预测时间都有在幅度的减少。值得一提的是，在 RPN 中引入了 anchor 的概念。对于 Feature Map 中的每个滑窗位置都会生成 k 个 anchors，然后根据 anchor 与 ground truth 的 IOU 判断 anchor 覆盖的图像区域是前景还是背景，判断的同时回归 Bounding box 的精细位置。

Faster RCNN 是 RCNN 系列算法中的集大成者，将目标检测算法的基本步骤[4]整合到了一个深度网络框架之内。其中所有计算共享，全部在 GPU 中完成，速度和精度都得到了极大的提高。

Faster RCNN 始终围绕着三个问题展开:

> 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**: 如图2.7灰色方框，是对原始特征提取 (用任何分类模型进行提取都可)

2、**候选区域**: 对于图像中的每一个位置，约定 9 个设定的候选窗口：三种不同比例 1:1,1:2,2:1× 三种不同面积 1282,2562,5122。这些候选窗口被称之为 anchors。如图2.8示出 51*39 个 anchor 中心，以及 9 种 anchor。

3、**位置精修和窗口分类**: 分类层 (cls_score) 计算每一个锚点上 9 个 anchors 属于前景和背景的概率；回归层 (bbox_pred) 计算每一个锚点上 9 个 anchors 对应窗口平移缩放的参数。对于每个锚点输出 $9*(4+1)$ 个参数

2.3 YOLO 系列

You Only Look Once: Unified, Real-time Object Detection(YOLO) [7] 是单阶段检测方法的开山之作。正如其名：“你只需要看一次”，它只处理一次图片即可同时

[4]候选区域生成，特征提取，分类, 位置精修

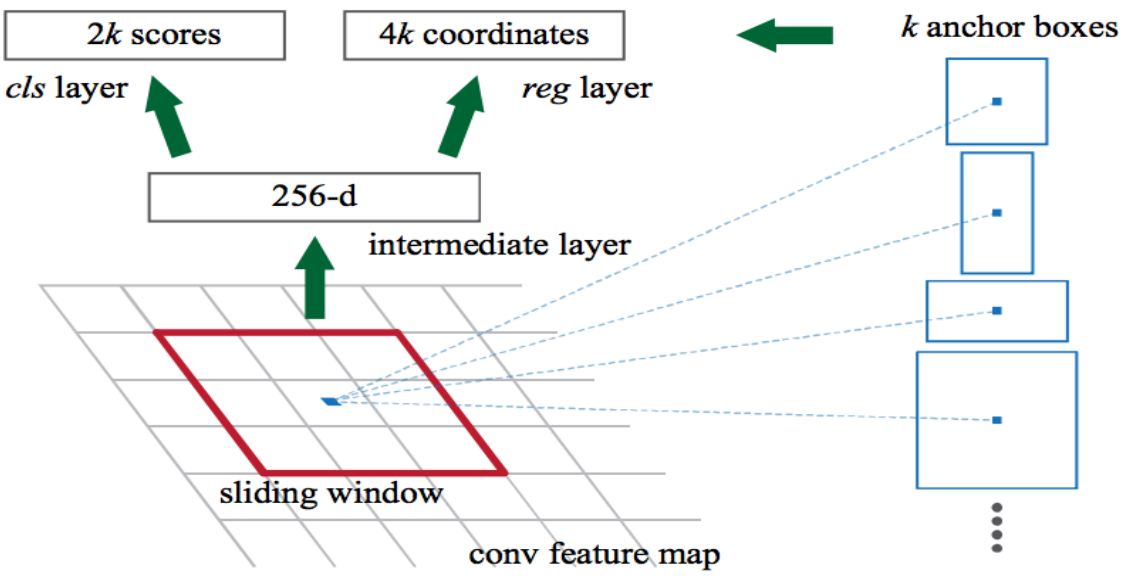

图 2.8 Faster RCNN anchor 示意图

得到目标的位置和其属于什么类别，将目标检测任务描述为一个端到端、统一的回归问题。“One Stage” 最核心的思想可以概括为：首先对全图给出一个大概的范围进行分类，利用了分类器优秀的分类效果，不断迭代直到得出一个精细的位置，

“One Stage” 回归的思想如图2.9从蓝色的框框到红色的框框，这样的检测速度一定很快，但是对小尺度目标的检测也肯定不好。

2.3.1 YOLO

YOLO 是 “One Stage” 方法中的典型算法。首先将原始图片调整到一个固定尺寸，再将调整后的图片划分成一个 SxS 的网格，每个网格负责检测该网格区域内的物体的类别和位置。

更具体的是如下定义：

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \tag{2.5}$$

YOLO 算法基于分而治之的策略，将从卷积后的原始图片中提取到的特征图分为 SxS 块，然后对每一块进行分类，并使用非极大值抑制 (NMS [15]) 算法去除重叠的框，最终得到想要的结果。

图 2.9 YOLO

2.3.2 SSD

YOLO 检测速度非常快，但是问题就在于该算法对小物体的检测效果不好。为解决这个问题，SSD 就在 YOLO 的想法上融合了 Faster R-CNN 的 Anchor 概念，并在不同尺度的卷积层的特征图上做出预测。关于 SSD 算法的细节介绍将在第三章将详细阐述。

2.4 本章小结

回顾本章，从 R-CNN 到 Faster R-CNN，再从 YOLO 到 SSD，不同思想逐渐融合，取长补短，去粗取精。SSD 算吸收了 YOLO 中 “One Stage” 和从 Faster R-CNN 中的 Anchor 思想，并创新地使用 Feature Pyramid 结构。这是本文选取 SSD 作为房屋瓦片损害检测基本算法的重要原因，也是得出一个目标检测相关算法的重要结论——SSD 算法对于速度和精度的权衡要优于本章介绍的其它算法。这两种重要的思路都有其适用的范围，比如说实时快速动作捕捉，“One Stage” 要更好；对于复杂、多物体交叠的情景，“Two Stage” 又要更好。总之没有不好的算法，只有更合适的应用情景。

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、将无人机拍下来的房屋屋顶的图像切割成 300x300 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

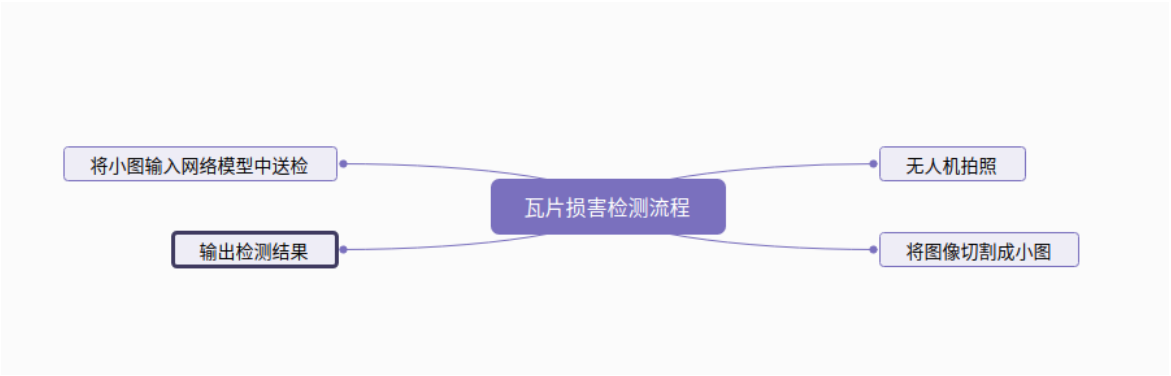

图 3.1 瓦片损害检测流程

3.2　SSD 算法核心思想

SSD 采用了多尺度的特征图进行检测，这不同于 YOLO 算法，其基本架构如图3.2所示。

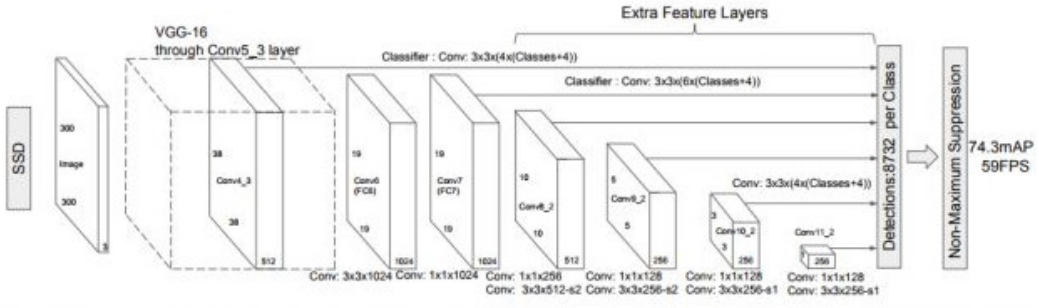

图 3.2 SSD 基本框架

概括地可以将 SSD 算法的核心思想归结为以下三点

1、采用多尺度特征图用于检测

所谓多尺度，即采用大小不同的特征图用于检测，如图3.3所示，一般前面的 CNN

网络其特征图会比较大，后面会采用步长为的卷积或者池化的方式来缩小特征图的大小，

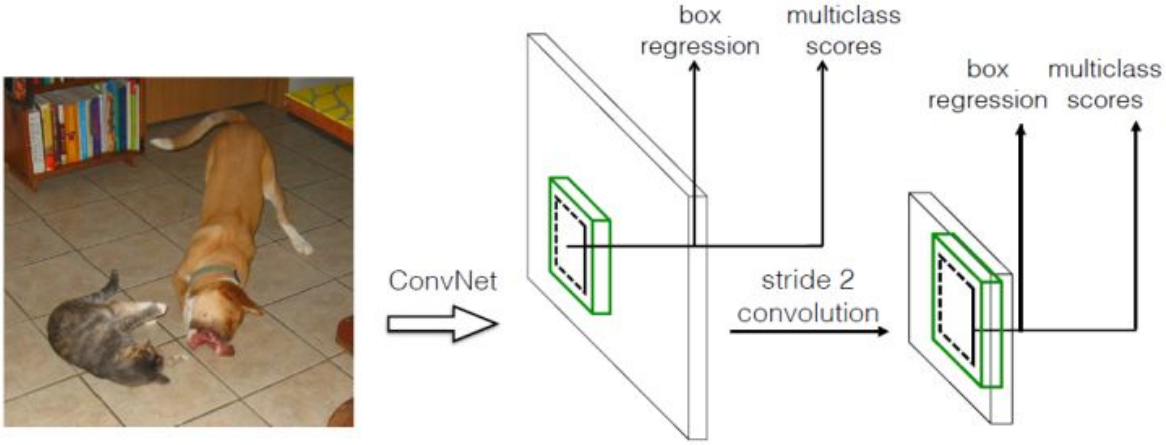

图 3.3 采用多尺度用于检测

不同尺度下得到用来做的特征图都被检测。如图3.4所示，8x8 的特征图可以被划分为更多的单元，但是得到的每个单元的先验框尺度比较小。这样做的优点是相对较小的目标用比较大的特征图来检测，相对较大的目标用小的特征图来检测。

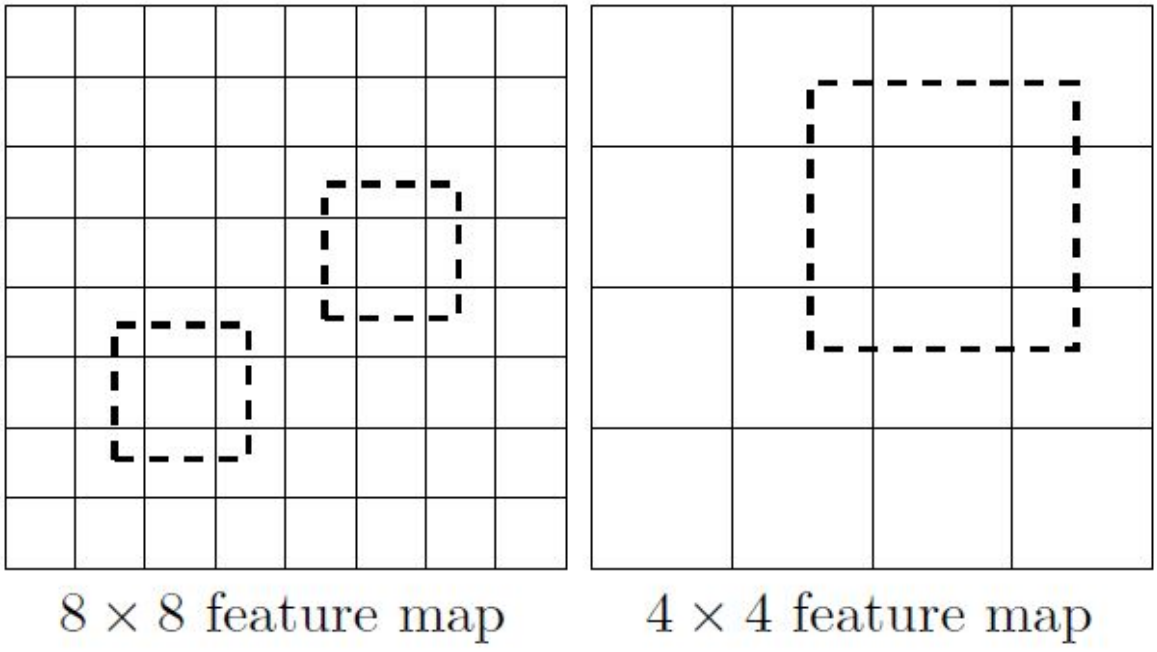

图 3.4 8x8 与 4x4 的特征图

2、采用卷积进行检测

SSD 算法直接采用卷积对不同的特征图进行提取检测结果。采用 $3 \times 3 \times p$ 这样

比较小的卷积核就可以得到形状为 $m \times n \times p$ 的特征图的检测值。

3、**设置先验框**

SSD 算法引用了 Faster R-CNN 中 anchor 的思想，每个单元设置长宽比、大小不同的先验框一般情况下，每个单元会设置多个先验框，其尺度和长宽比存在差异，，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。如图3.5所示，图片中狗和猫分别采用最适合它们形状的先验框来进行训练，可以看到每个单元使用了 4 个不同的先验框，后面会详细讲解训练过程中的先验框匹配原则。

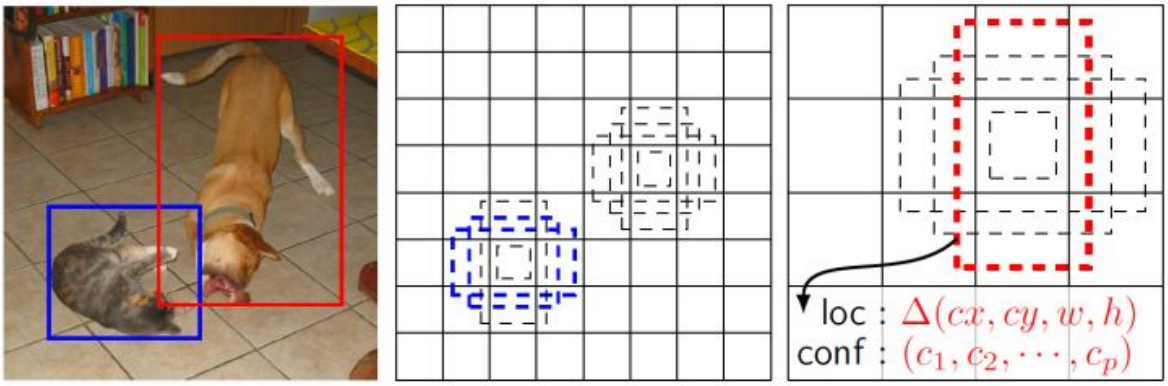

图 3.5 SSD 算法中的先验框

3.3 SSD **模型结构**

“特征分层提取，并依次对其边框进行回归和分类” 是 SSD 主体网络的设计思想。SSD 算法理论上对不同尺度的目标都有很好的适应性，不同层次的 feature map 表示了不同 layer 的语义信息，low layer 的 feature map 能代表 low leayer 的语义信息 (含有能提高语义分割质量的细节信息)，low layer 的图像适合小尺度目标的学习。high layer 的 feature map 能代表 high layer 语义信息，起到光滑边缘的作用，适合对大尺度的目标进行学习。

对不同尺度的目标进行学习的全过程共分成 6 个阶段，每个阶段都能得到一个特征图，然后对该特征图里的目标进行边框回归和分类。

第一阶段：VGG16 [12] 网络的前 5 层卷积网络；

第二、三阶段：VGG16 中的 fc6 和 fc7 两个全连接层转化为两个卷积层 Conv6

和 Conv7；

第四至六阶段：Conv8、Conv9、Conv10 和 Conv11 四层网络；

如图3.2所示就是 SSD 的网络结构。在每个学习阶段，网络包含了多个以小卷积组成的卷积层。

3.4 损失函数

3.4.1 SSD 中采用的损失函数

联合 Loss Function:SSD 网络对于每个阶段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

$$L(x,c,l,g)=\frac{1}{N}(L_{conf}(x,c)+\alpha L_{loc}(x,l,g) \tag{3.1}$$

其中：

1、L_{conf} 表示分类误差，采用多分类的 softmax loss

2、L_{loc} 表示回归误差，采用 Smooth L1 loss

3、α 取 1

回归 loss,smoothL1:

$$L_{loc}(x,l,g)=\sum_{i\in Pos}^{N}\sum_{m\in\{cx,cy,w,h\}}x_{ij}^{k}smooth_{L1}(l_i^m-g_j^m) \tag{3.2}$$

$$g_j^{cx}=\left(g_j^{cx}-d_i^{cx}\right)/d_i^w \quad g_j^{cy}=(g_j^{cy}-d_i^{cy})/d_i^h \tag{3.3}$$

$$g_j^w=\log(\frac{g_j^w}{d_i^w}) \quad g_j^h=\log(\frac{g_j^h}{d_i^h}) \tag{3.4}$$

分类误差，采用 Softmax Loss：

$$L_{conf}(x,c)=-\sum_{i\in Pos}^{N}x_{ij}^{p}\log(c_i^0) \quad where \quad c_i^p=\frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \tag{3.5}$$

3.4.2 改进损失函数-Focal Loss

提出 Focal Loss 的初衷是希望既有”One-Stage Detector“的速度，也有”Two Stage Detector“的准确率。

1、为什么”One Stage Detector” **的准确率低？**

其作者认为主要原因是：不同类别的训练样本数量的不均衡。在目标检测领域，一张原始图像可能生成非常多的预测区域，但是包含目标的图片数量占比非常低，这就带来了类别不均衡。那么不同类别的训练样本数量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. [13]

负样本数量占比过高，使得对总的 loss 的影响过大，因此使得模型不会按所希望的方向进行优化。这也不是新鲜的问题了，先前也有存在一些算法来处理不同类别的训练样本数量不均衡的问题，比如 OHEM(Online Hard Example Mining) [14]，OHEM 算法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 [15]) is then applied, and a minibatch is constructed with the highest-loss examples。[14] 可以看出 OHEM 算法加大了对错分类样本的权重，不过 OHEM 算法忽略了对容易分类的样本的权重处理。Focal Loss 是从标准交叉熵损失基础上改进所得。这个函数可以通过调整易分类样本的权重，使得模型在训练时对难分类的样本更专注。

标准交叉熵损失函数

没有经过加权的分类损失是各个训练样本交叉熵的和，如下图所示:

$$CE(p,y) = \begin{cases} -\log(p) & if y = 1 \\ -\log(1-p) & otherwise \end{cases} \tag{3.6}$$

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，用 p_t 表示样本属于 true class 的概率。所以上式可以写成

$$CE(p, y) = CE(p_t) = -\log(p_t) \tag{3.7}$$

平衡交叉熵

知道了"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：

$$CE(p_t) = -\alpha \log(p_t) \tag{3.8}$$

Focal-Loss **定义**

通过上述分析，此文作者解决了 easy example 与 hard example 数量不均衡的问题，Focal Loss 相当于是对各个样本根据网络模型预测该样本属于真实类别的概率设置各自的权重，如果网络对该样本属于真实类别的预测后的概率很大，那么这个样本就属于 easy example，相反，对这个网络来说该样本就属于 hard example。所以降低绝大部分简单样本的权重，相对增大复杂样本的权重就能训练出一个有效的分类器。Focal Loss 的形式如下：

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{3.9}$$

参数 $\gamma > 0$。当参数 $\gamma = 0$ 的时候，就是普通的交叉熵，作者经过实验发现当 $\gamma = 2$ 时效果最好。从公式中可以发现，当 γ 是一个定值时，比如说等于 2，简单样本 ($p_t = 0.9$) 的 loss 要比标准的交叉熵计算出的 loss 小 100 多倍，当 $p_t = 0.968$ 时，要小 1000 多倍，对于困难样本 ($p_t < 0.5$)，loss 最多小了 4 倍。这样的话困难样本的

权重相对就提升了很多。

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{3.10}$$

3.4.3 **改进** NMS, **采用** Soft-NMS

本设计中使用到了 NMS-非极大值抑制 [15] 进行后处理。NMS 通常的做法是将检测框按得分排序，然后保留得分最高的框，同时删除与该框重叠面积大于一定比例的其它框。这种贪心式方法存在如图所示的问题：

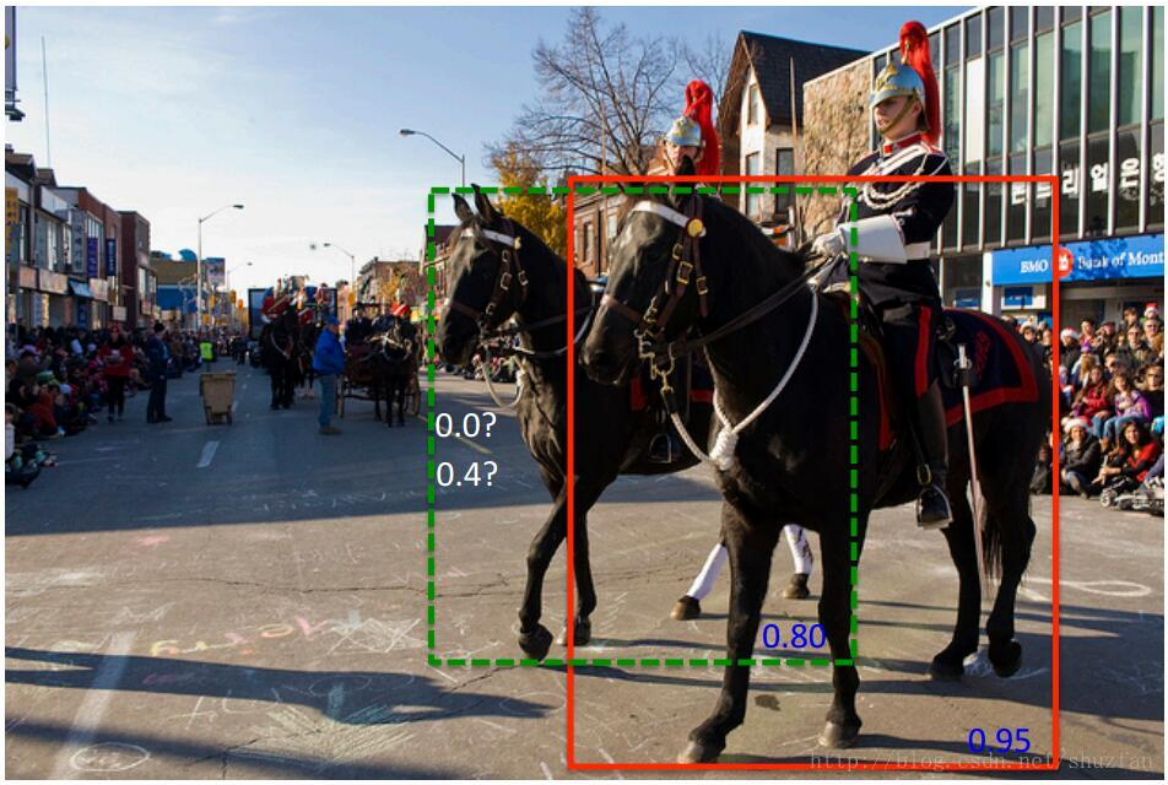

图 3.6 非极大值抑制

红色框和绿色框是当前的检测结果，二者的得分分别是 0.95 和 0.80。如果按照传统的 NMS 进行处理，首先选中得分最高的红色框，然后绿色框就会因为与之重叠面积过大而被删掉。另一方面，NMS 的阈值也不太容易确定，设小了会出现下图的情况（绿色框因为和红色框重叠面积较大而被删掉），设置过高又容易增大误检。

Soft-NMS 采用的策略是用降低置信度来取代直接删除掉 IOU 大于阈值的框。

原来的 NMS 可以描述如下：

$$s_i = \begin{cases} s_i & iou(M, b_i) < N_t, \\ 0 & otherwise \end{cases} \tag{3.11}$$

现在将其替换成如下形式：

$$s_i = \begin{cases} s_i & iou(M, b_i) < N_t \\ s_i(1 - iou(M, b_i)) & otherwise \end{cases} \tag{3.12}$$

3.5 SSD 模型训练

相比于基于 “Region Proposal “方法，SSD 需要精确的将 Ground Truth 与输出结果相对应。这样才能做到检测准确率的提高。

匹配策略

Default boxes 生成器

Hard Negative Mining

Data Augmentation

---

1. **匹配策略**

此处的匹配指的是 Ground Truth 与 Default Box 的匹配。SSD 采取的方法与 Faster R-CNN 中的做法相似。操作分成两步：第一步是根据 Ground Truth 与 Default Box 最大的交叠 (Overlap) 将二者进行匹配，找到 IOU 最大的区域作为正样本；第二步是与 Overlap 大于某个阈值的 Ground truth 进行匹配。

2. Default Boxes **生成器**

3. Hard Negative Mining 3.4.2 节探讨了 “One Stage” 方法检测精度低的原因，经过上述匹配策略只有少量的正样本，会得到大量的负样本，从而导致了正负样本数量不平衡问题。经过试验表明，正负样本的不均衡会导致检测正确率低下 [13]。因此在

训练过程中采用了 Hard Negative Mining [11] 的策略，根据所有的 box 的 Confidence Loss 的排序结果，将正负样本的比例控制在 1:3 以下，实验结果表明，这样做后能提高 4% 左右的准确度。

4. Data Augmentation

对不同尺度目标的输入数据进行了增强处理，增加了模型的鲁棒性。

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，分析了“One Stage”方法准确率不如“Two Stage”方法的原因，并对该算法进行了两点改进

1. 将损失函数替换为 focal 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**: 为了方便使用自己的数据集进行深度学习训练，需要对图片进行标注。本文使用 labelImg，这是一款被广泛使用的图片标注工具。

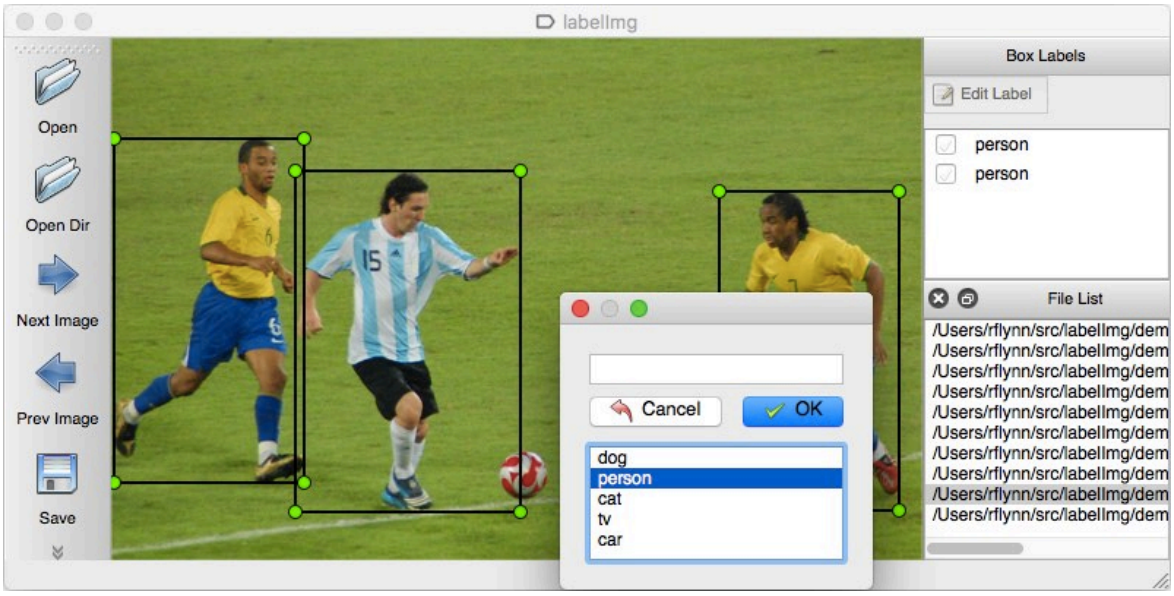

图 4.1 labelImg 界面

4.1.2　数据集

VOC **数据集介绍**:PASCAL VOC 为图像分类、目标检测等任务的需要提供了标准的数据库。系统的目标检测任务只需要使用 JPEGImage、Annotations、ImageSets 三个文件夹。将受损的瓦片图片放入 JPEGImage 文件夹，将对应的保存为.xml 文件的标注信息放入 Annotations 文件夹。

其目录结构为:

- JPEGImages: 将受损瓦片的图片放入此文件夹，图片大小为 224*224 像素;

- Annotations: 将 JPEGImages 文件夹中对应图片的标注信息放入此文件夹，其文件格式为 xml;

- ImageSet: 此处将训练、测试、验证等图片的信息放到此处，方便程序的读写。

4.1.3 Pytorch 介绍

PyTorch 是深度学习框架，因为非常易于使用，所以被非常多的研究人员所喜爱。PyTorch 支持动态计算图非常有吸引力。

4.1.4 OpenCV 介绍

OpenCV 是一个开源的计算机视觉库，他实现了图像处理和计算机视觉方面的非常多的通用算法。底层由 C/C++ 语言写成，所以运行速度快，跨平台支持挺好。

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)
```

输入的图片其大小为 300*300 大小的像素，利用 OpenCV 函数库提供的 resize 函数对图片进行大小的设定。

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

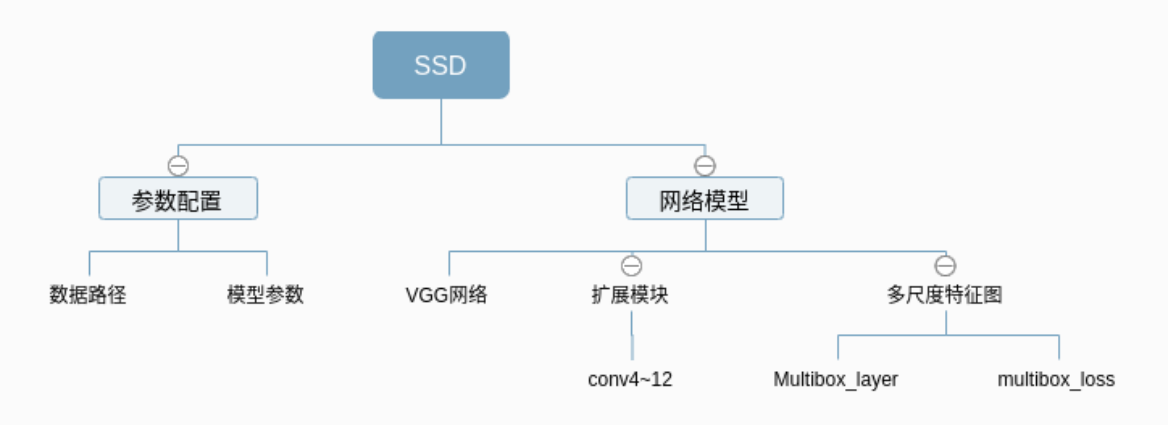

图 4.2 SSD 模型设计框架

如图4.2显示了 SSD 网络模型实现代码的主要元素。其包括 SSD 类的参数：阶

段 (训练/测试)、类别 (前景/损害)、配置参数 (数据集的路径/图片的大小的设定...), 还有前向传播的 forword() 方法，以及构成 SSD 网络的三个基础网络——VGG16、Extra、Multibox。

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)
```

基础网络部分特征图分辨率高，原图中信息更完整，感受野较小，可以用来检测图像中的小目标，这也是 SSD 相对于 YOLO 检测小目标的优势所在。增加对基础网络 conv4_3 的特征图的检测可以使 mAP 提升 4%。L2Norm2d 类为对 conv4_3 进行处理的代码。

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)
```

```
8:          self.conv6
9:          self.conv7
10:         self.conv(8~12)(1~2)
```

上述代码是对 SSD 网络模型的实现，其主要的结构包括了 VGG16、Extra(SSD 添加的几层，其作用是产生不同尺度的特征图)、Multibox。

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

```
1: class MultiBoxLayer(nn.Module):
2:          num_classes = 2
3:          num_anchors = [4, 6, 6, 6, 4, 4]
4:          in_planes = [512, 1024, 512, 256, 256, 256]
5:
6:          for i n range(len(self.in_planes)):
7:                  self.loc_layers.apend(
8:                          nn.Conv2d(self.in_planes[i],
9:                          self.num_anchors[i] * 4,
10:                         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:                 )
```

Multibox Layer 是将不同层产生的特征图进行预测其位置偏移和类别的运算的

层。其中 Multibox loss 是对损失函数的实现，采用了全网络结构，也就是说所有的计算都在一个网络中进行。

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()
```

通常不同的学习策略，产生的训练效果也有所不同，本设计采用原作者的训练策略。即：

1、参数初始化——清零

2、用 DataLoader 接口产生一批数据，将其传入网络中进行计算　3、前向传播

4、自动求导，并更新参数

5、跳到第 2 步

第五章　实验结果与分析

5.1　训练时的损失敛散情况

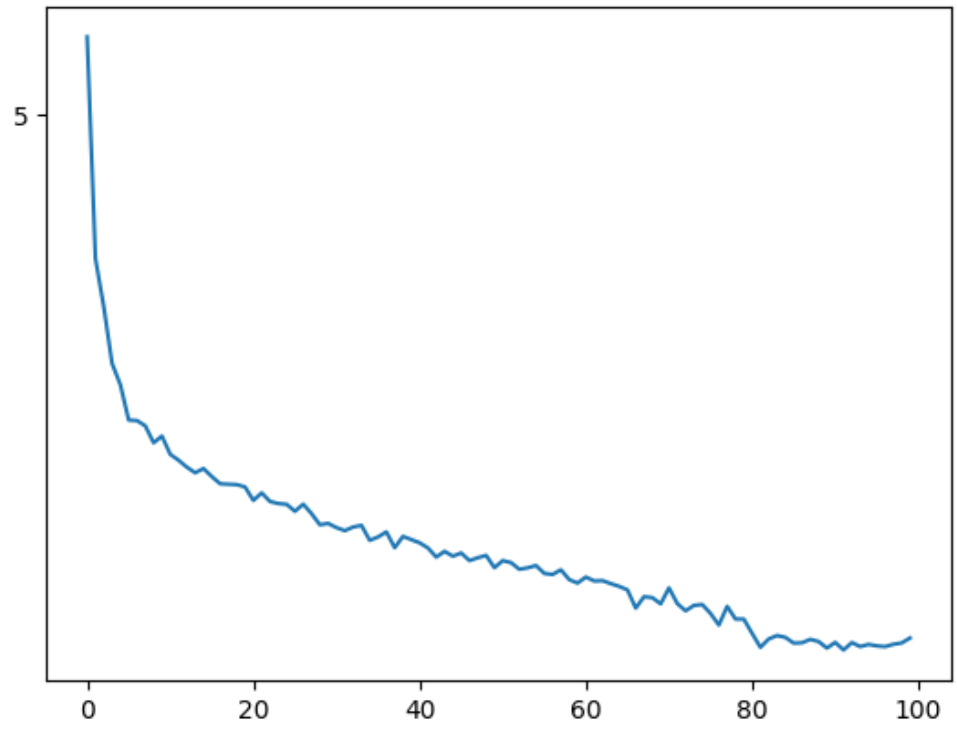

图 5.1 训练时的损失敛散情况

本算法对 8000 张大小为 224*224 像素的图片进行了 120000 次迭代，如图5.1反映的是训练过程中的损失收敛的情况。横坐标中的 1 个单位表示的是 1200 次迭代，纵坐标的 1 个单位表示的是每 1200 次迭代的损失函数的平均值。

5.2　改进 SSD 算法的实验结果

从图5.2中可以看到形状如斑点的黑色区域就是瓦片被损害的区域，有大部分被检测出来了，但也有小部分没有检测出来，或者是被误检了。

5.3　改进 SSD 对瓦片损害检测的准确率实验

mAP：Mean Average Precision　平均精度均值。

FPS：Frames Per Second 帧率

图 5.2 检测结果

表 1 改进 SSD 算法的准确率及速度实验结果

网络模型	mAP	FPS
SSD	60%	90
SSD+focal-loss	62%	87
SSD+soft-nms	64%	85
SSD+soft-nms+focal-loss	65%	80

5.4 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

图 5.3 冰雹对瓦片造成的损害

5.5 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

目标检测算法出现了以下几种：

1、传统的目标检测算法：Cascade [16] + Haar [17]、SVM [10] + 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 [18] 系列方法；

3、基于深度学习的回归方法：YOLO、SSD、DenseBox [19] 等方法；

4、结合 RNN 算法的 RRC Detection；结合 DPM 的 Deformable CNN [20] 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、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；

6、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；

7、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。

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] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. nature, 521(7553):436, 2015.

[9] 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.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine learning, 20(3):273–297, 1995.

[11] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9):1627–1645, 2010.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

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

[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.

[15] 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.

[16] 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.

[17] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–I. IEEE, 2002.

[18] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In Advances in neural information processing systems, pages 379–387, 2016.

[19] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. arXiv preprint arXiv:1509.04874, 2015.

[20] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. CoRR, abs/1703.06211, 1(2):3, 2017.

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，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——汪琳霞老师、308 室友、物联网的同学们以及其它关心照顾我的人。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！

附录

源代码

datagen.py

Listing 8 datagen.py

```
1: '''Load image/class/box from a annotation file.
2:
3: The annotation file is organized as:
4: image_name #obj xmin ymin xmax ymax class_index ..
5: '''
6: from __future__ import print_function
7:
8: import os
9: import sys
10: import os.path
11:
12: import random
13: import numpy as np
14:
15: import torch
16: import torch.utils.data as data
17: import torchvision.transforms as transforms
18:
19: from encoder import DataEncoder
20: from PIL import Image, ImageOps
21:
22:
23: class ListDataset(data.Dataset):
24: img_size = 300
25:
```

```
26: def __init__(self, root, list_file, train, transform):
27: '''
28: Args:
29: root: (str) ditectory to images.
30: list_file: (str) path to index file.
31: train: (boolean) train or test.
32: transform: ([transforms]) image transforms.
33: '''
34: self.root = root
35: self.train = train
36: 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).
23:         rsqrt().expand_as(x)
24:
25:
26: class SSD300(nn.Module):
27: input_size = 300
28:
29: def __init__(self):
30: super(SSD300, self).__init__()
31:
32: # model
33: self.base = self.VGG16()
34: self.norm4 = L2Norm2d(20)
```

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

```
66: h = F.relu(self.conv5_1(h))
67: h = F.relu(self.conv5_2(h))
68: h = F.relu(self.conv5_3(h))
69: h = F.max_pool2d(h, kernel_size=3, padding=1, stride=1, ceil_mode=True)
70:
71: h = F.relu(self.conv6(h))
72: h = F.relu(self.conv7(h))
73: hs.append(h)  # conv7
74:
75: h = F.relu(self.conv8_1(h))
76: h = F.relu(self.conv8_2(h))
77: hs.append(h)  # conv8_2
78:
79: h = F.relu(self.conv9_1(h))
80: h = F.relu(self.conv9_2(h))
81: hs.append(h)  # conv9_2
82:
83: h = F.relu(self.conv10_1(h))
84: h = F.relu(self.conv10_2(h))
85: hs.append(h)  # conv10_2
86:
87: h = F.relu(self.conv11_1(h))
88: h = F.relu(self.conv11_2(h))
89: hs.append(h)  # conv11_2
90:
91: loc_preds, conf_preds = self.multibox(hs)
92: return loc_preds, conf_preds
93:
94: def VGG16(self):
95: '''VGG16 layers.'''
96: cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512]
```

```
97: layers = []
98: in_channels = 3
99: for x in cfg:
100: if x == 'M':
101: layers += [nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)]
102: else:
103: layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
104: nn.ReLU(True)]
105: in_channels = x
106: 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],
25:  self.num_anchors[i]*4, kernel_size=3, padding=1))
26: self.conf_layers.append(nn.Conv2d(self.in_planes[i],
27:  self.num_anchors[i]*21, kernel_size=3, padding=1))
28:
29: def forward(self, xs):
30: '''
31: Args:
32: xs: (list) of tensor containing intermediate layer outputs.
33:
34: Returns:
35: loc_preds: (tensor) predicted locations, sized [N,8732,4].
36: conf_preds: (tensor) predicted class confidences,
37:  sized [N,8732,21].
38: '''
39: y_locs = []
40: y_confs = []
41: for i,x in enumerate(xs):
42: y_loc = self.loc_layers[i](x)
43: N = y_loc.size(0)
44: y_loc = y_loc.permute(0,2,3,1).contiguous()
45: y_loc = y_loc.view(N,-1,4)
46: y_locs.append(y_loc)
47:
48: y_conf = self.conf_layers[i](x)
49: y_conf = y_conf.permute(0,2,3,1).contiguous()
50: y_conf = y_conf.view(N,-1,21)
```

```
51: y_confs.append(y_conf)
52:
53: loc_preds = torch.cat(y_locs, 1)
54: conf_preds = torch.cat(y_confs, 1)
55: 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
42: number as postive indices.
43:
44: Args:
45: conf_loss: (tensor) cross entroy loss
46: between conf_preds and conf_targets, sized [N*8732,].
47: pos: (tensor) positive(matched) box indices, sized [N,8732].
48:
49: Return:
50: (tensor) negative indices, sized [N,8732].
51: '''
52: batch_size, num_boxes = pos.size()
53:
54: conf_loss[pos] = 0  # set pos boxes = 0,
55: the rest are neg conf_loss
```

```
56: conf_loss = conf_loss.view(batch_size, -1)  # [N,8732]
57:
58: _,idx = conf_loss.sort(1, descending=True)  # sort by neg conf_loss
59: _,rank = idx.sort(1)  # [N,8732]
60:
61: num_pos = pos.long().sum(1)  # [N,1]
62: num_neg = torch.clamp(3*num_pos, max=num_boxes-1)  # [N,1]
63:
64: neg = rank < num_neg.expand_as(rank)  # [N,8732]
65: return neg
66:
67: def forward(self, loc_preds, loc_targets, conf_preds, conf_targets):
68: '''Compute loss between (loc_preds, loc_targets) and (conf_preds, conf_target
69:
70: Args:
71: loc_preds: (tensor) predicted locations, sized [batch_size, 8732, 4].
72: loc_targets: (tensor) encoded target locations, sized [batch_size, 8732, 4].
73: conf_preds: (tensor) predicted class confidences, sized [batch_size, 8732, nu
74: conf_targets: (tensor) encoded target classes, sized [batch_size, 8732].
75:
76: loss:
77: (tensor) loss = SmoothL1Loss(loc_preds, loc_targets) + CrossEntropyLoss(conf_
78: '''
79: batch_size, num_boxes, _ = loc_preds.size()
80:
81: pos = conf_targets>0  # [N,8732], pos means the box matched.
82: num_matched_boxes = pos.data.long().sum()
83: if num_matched_boxes == 0:
84: return Variable(torch.Tensor([0]))
85:
86: ##################################################################
```

```
87: # loc_loss = SmoothL1Loss(pos_loc_preds, pos_loc_targets)
88: ########################################################################
89: pos_mask = pos.unsqueeze(2).expand_as(loc_preds)      # [N,8732,4]
90: pos_loc_preds = loc_preds[pos_mask].view(-1,4)       # [#pos,4]
91: pos_loc_targets = loc_targets[pos_mask].view(-1,4)   # [#pos,4]
92: loc_loss = F.smooth_l1_loss(pos_loc_preds, pos_loc_targets, size_average=Fals
93:
94: ########################################################################
95: # conf_loss = CrossEntropyLoss(pos_conf_preds, pos_conf_targets)
96: #           + CrossEntropyLoss(neg_conf_preds, neg_conf_targets)
97: ########################################################################
98: conf_loss = self.cross_entropy_loss(conf_preds.view(
99: -1,self.num_classes),
100: conf_targets.view(-1))  # [N*8732,]
101: neg = self.hard_negative_mining(conf_loss, pos)      # [N,8732]
102:
103: pos_mask = pos.unsqueeze(2).expand_as(conf_preds)  # [N,8732,21]
104: neg_mask = neg.unsqueeze(2).expand_as(conf_preds)  # [N,8732,21]
105: mask = (pos_mask+neg_mask).gt(0)
106:
107: pos_and_neg = (pos+neg).gt(0)
108: preds = conf_preds[mask].view(-1,self.num_classes)  # [#pos+#neg,21]
109: targets = conf_targets[pos_and_neg]                 # [#pos+#neg,]
110: conf_loss = F.cross_entropy(preds, targets, size_average=False)
111:
112: loc_loss /= num_matched_boxes
113: conf_loss /= num_matched_boxes
114: print('%f␣%f' % (loc_loss.data[0], conf_loss.data[0]), end='␣')
115: 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
40: two set of boxes, each box is [x1,y1,x2,y2].
41:
42: Args:
43: box1: (tensor) bounding boxes, sized [N,4].
44: box2: (tensor) bounding boxes, sized [M,4].
45:
46: Return:
47: (tensor) iou, sized [N,M].
48: '''
49: N = box1.size(0)
50: M = box2.size(0)
51:
52: lt = torch.max(
53: box1[:,:2].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
54: box2[:,:2].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
55: )
56:
57: rb = torch.min(
58: box1[:,2:].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
59: box2[:,2:].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
60: )
```

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

```
92: '''
93: default_boxes = self.default_boxes
94: num_default_boxes = default_boxes.size(0)
95: num_objs = boxes.size(0)
96:
97: iou = self.iou(  # [#obj,8732]
98: boxes,
99: torch.cat([default_boxes[:,:2] - default_boxes[:,2:]/2,
100: default_boxes[:,:2] + default_boxes[:,2:]/2], 1)
101: )
102:
103: iou, max_idx = iou.max(0)  # [1,8732]
104: max_idx.squeeze_(0)        # [8732,]
105: iou.squeeze_(0)            # [8732,]
106:
107: boxes = boxes[max_idx]     # [8732,4]
108: variances = [0.1, 0.2]
109: cxcy = (boxes[:,:2] + boxes[:,2:])/2 - default_boxes[:,:2]  # [8732,2]
110: cxcy /= variances[0] * default_boxes[:,2:]
111: wh = (boxes[:,2:] - boxes[:,:2]) / default_boxes[:,2:]      # [8732,2]
112: wh = torch.log(wh) / variances[1]
113: loc = torch.cat([cxcy, wh], 1)  # [8732,4]
114:
115: conf = 1 + classes[max_idx]   # [8732,], background class = 0
116: conf[iou<threshold] = 0       # background
117: return loc, conf
118:
119: def nms(self, bboxes, scores, threshold=0.5, mode='union'):
120: '''Non maximum suppression.
121:
122: Args:
```

```
123: bboxes: (tensor) bounding boxes, sized [N,4].
124: scores: (tensor) bbox scores, sized [N,].
125: threshold: (float) overlap threshold.
126: mode: (str) 'union' or 'min'.
127:
128: Returns:
129: keep: (tensor) selected indices.
130:
131: Ref:
132: https://github.com/rbgirshick/py-faster-rcnn/
133: blob/master/lib/nms/py_cpu_nms.py
134: '''
135: x1 = bboxes[:,0]
136: y1 = bboxes[:,1]
137: x2 = bboxes[:,2]
138: y2 = bboxes[:,3]
139:
140: areas = (x2-x1) * (y2-y1)
141: _, order = scores.sort(0, descending=True)
142:
143: keep = []
144: while order.numel() > 0:
145: i = order[0]
146: keep.append(i)
147:
148: if order.numel() == 1:
149: break
150:
151: xx1 = x1[order[1:]].clamp(min=x1[i])
152: yy1 = y1[order[1:]].clamp(min=y1[i])
153: xx2 = x2[order[1:]].clamp(max=x2[i])
```

```
154: yy2 = y2[order[1:]].clamp(max=y2[i])
155:
156: w = (xx2-xx1).clamp(min=0)
157: h = (yy2-yy1).clamp(min=0)
158: inter = w*h
159:
160: if mode == 'union':
161: ovr = inter / (areas[i] + areas[order[1:]] - inter)
162: elif mode == 'min':
163: ovr = inter / areas[order[1:]].clamp(max=areas[i])
164: else:
165: raise TypeError('Unknown nms mode: %s.' % mode)
166:
167: ids = (ovr<=threshold).nonzero().squeeze()
168: if ids.numel() == 0:
169: break
170: order = order[ids+1]
171: return torch.LongTensor(keep)
172:
173: def decode(self, loc, conf):
174: '''Transform predicted loc/conf back to real bbox
175: locations and class labels.
176:
177: Args:
178: loc: (tensor) predicted loc, sized [8732,4].
179: conf: (tensor) predicted conf, sized [8732,21].
180:
181: Returns:
182: boxes: (tensor) bbox locations, sized [#obj, 4].
183: labels: (tensor) class labels, sized [#obj,1].
184: '''
```

```
185: variances = [0.1, 0.2]
186: wh = torch.exp(loc[:,2:]*variances[1]) * self.default_boxes[:,2:]
187: cxcy = loc[:,:2] * variances[0] *
188:  self.default_boxes[:,2:] + self.default_boxes[:,:2]
189: boxes = torch.cat([cxcy-wh/2, cxcy+wh/2], 1)  # [8732,4]
190:
191: max_conf, labels = conf.max(1)  # [8732,1]
192: ids = labels.squeeze(1).nonzero().squeeze(1)  # [#boxes,]
193:
194: keep = self.nms(boxes[ids], max_conf[ids].squeeze(1))
195: return boxes[ids][keep], labels[ids][keep], max_conf[ids][keep]
```