第 5 章

使用深度学习进行目标检测

有些事情不能仅仅因为不像你计划的那样，就认为它们毫无意义。

——托马斯·爱迪生

为了解决特定的问题，在尝试了多种解决方案，经过多次迭代后，最终找到了最佳的解决方案。机器学习和深度学习没有什么不同。在探索阶段，为了提高前一版本算法的性能，需要对算法进行不断的改进和完善。在最后阶段，观察到的模型的性能弱点、缓慢的计算、错误的分类——所有这些都为了建立更好的解决方案铺平道路。

第 3 章和第 4 章分别介绍并创建了将图像分类为两个类别或多个类别的解决方案。但是，大多数图像中只有一个目标，并没有识别出图像中目标的位置，因此只能判别某个目标是否存在于某个图像之中。本章将识别图像中的目标，并通过在目标周围创建边界框确定这个目标的位置。

目前已经存在相当多的目标检测网络模型架构，例如 R-CNN、Fast R-CNN、Faster R-CNN、单阶段多框检测器（Single Shot MultiBox Detector，SSD），和 YOLO（You Only Look Once）等。本章将讨论这些网络架构并创建相应的 Python 解决方案。

本章覆盖的如下主题：

- 目标检测和应用实例；
- R-CNN、Fast R-CNN 和 Faster R-CNN 网络架构；
- SSD；
- YOLO；
- 算法的 Python 实现。

本章有关代码和数据集已经上传到 GitHub 链接 https://github.com/Apress/computer-vision-using-deep-learning/tree/main/Chapter5 中，建议使用 Jupyter Notebook 代码编辑器。对于本章内容，常用计算机的 CPU 就足以执行全部的代码。但是，如果需要的话，也可以使用 Google Colaboratory。如果读者不会设置 Google Colaboratory，可以参考本书附录列出的参考信息。

5.1 目标检测

目标检测是机器学习和深度学习领域中被引用最多且得到普遍公认的一种解决方案，也是一种相当新颖和有趣解决方案。关于目标检测的实际应用案例有很多，因此，研究机构和研究人员正在花费大量的时间和资源来发掘这种能力。顾名思义，目标检测是一种在图像或视频中定位目标的计算机视觉技术，例如可以在直播的视频中进行实时目标检测。当看到一张图片时，可以快速地识别出图片中的目标和它们各自在图像中所处的位置。如果它是一个苹果，一辆汽车或一个人，还可以迅速地对它们的类别进行分类。我们可以从任何角度来确定目标的类别，原因在于我们的大脑已经被训练成能够识别各种目标的方式。即使一个目标的形状变小或者变大，我们也能发现并定位它们。

目标检测的目标是利用机器学习和深度学习复制人类这种智能的决策能力。本章将介绍目标检测、定位和分类的概念并开发相应 Python 代码。

在学习目标检测基础知识之前，首先应该考察目标分类、目标定位和目标检测之间的区别，它们都是为目标检测而建立的概念。

5.1.1 目标分类、目标定位与目标检测

某个吸尘器的外观如图 5-1 所示。基于前面的章节中开发的图像分类解决方案可以将这样的图像分类到“真空吸尘器”类别或“非真空吸尘器”类别，所以很容易可以把第一个图像标记为真空吸尘器。

(a) 目标　　(b) 目标定位

图 5-1　目标检测中的目标识别和定位

目标定位是要找到目标在图像中的位置。因此，对图像中的目标进行定位时，就意味着算法要有双重的职责：对图像中的目标进行分类的同时在目标的周围绘制一个边界框。在图 5-1(a)中，将目标分类为真空吸尘器，即可以对图像中的目标是否为吸尘器进行分类；在图 5-1(b)中，在目标的周围画了一个方框，即给图像中的目标进行了定位。

由于同一幅图像中可以有多个对象，甚至在同一幅图像中有多个不同类别的对象，因此必须扩展解决方案，实现对这些目标的识别，并在它们周围画出边界框。一个被训练用于检测汽车目标的网络模型解决方案就是一个具体的应用案例。在繁忙的道路上会有很多辆汽车，因此训练出来的网络模型应该能够检测出每辆汽车，并在它们的周围正确地绘制出边界框。

图像中的目标检测无疑是一个极好的问题求解案例。下面讨论目标检测的具体应用

案例。

5.1.2　目标检测的应用案例

深度学习已经扩展了许多跨领域和跨组织的功能。目标检测是一个关键的非常强大的问题求解案例和应用场景，正在对我们的商业和个人世界产生巨大的影响。目标检测的主要应用场景如下所述。

(1) 目标检测是支撑自动驾驶系统的一个关键智能技术。目标自动检测算法能够自动检测到汽车、行人、背景、摩托车等目标，以提高汽车在道路上行驶的安全性。

(2) 可以使用目标检测算法自动检测手中的目标。将这个算法应用于公共安全和视频监控的场合，可以使视频监控系统变得更加智能和准确，可以使人群控制系统变得更加复杂并且反应更加敏捷。

(3) 可以使用目标检测算法自动检测购物篮中的目标。零售商可以使用这种目标算法进行商品的自动交易，减少人工干预的过程，加快交易进程。

(4) 可以将目标检测算法用于机械系统和生产线上的产品质量检测。使用目标检测算法检测出产品上可能存在的对产品形成污染的物质。

(5) 在医学领域，还可以通过分析身体部位的扫描图像识别疾病，帮助患者更快地得到有效的治疗。

目标检测目前是一个研究焦点，每天都有崭新的进展。世界各地的研究组织和研究人员正在这个领域掀起一个个巨大的波澜，并创造多个具有开创性的解决方案。

5.2　目标检测方法

可以使用机器学习和深度学习方法进行目标检测。本书主要讨论基于深度学习的目标检测方法，但对于比较好奇的读者，还可以关注下面这些解决方案。

(1) 图像分割使用诸如形状，大小和颜色等简单的属性对象。

(2) 可以使用聚合信道特征(Aggregated Channel Feature，ACF)，它是信道特征的一种变体。ACF 并不对不同位置或范围的矩形进行累加计算，而是直接提取像素特征。

(3) Viola-Jones 算法可以用于人脸检测。具体参考本章最后的推荐阅读论文。

还可以使用 RANSAC (random sample consensus)、基于 Haar 特征的级联分类器、使用 HOG 特征的 SVM 分类等解决方案进行目标检测。用于目标检测的常用深度学习网络架构如下所述。

(1) R-CNN：具有 CNN 特征的区域。它结合了建议区域与 CNN 模型。

(2) Fast R-CNN：快速基于区域的卷积神经网络。

(3) Faster R-CNN：使用基于提议区域目标检测网络来假设目标的位置。

(4) Mask R-CNN：这个网络扩展了 Faster R-CNN，在每个感兴趣的区域上添加了关于分割掩码的预测。

（5）YOLO：使用单个神经网络模型一次性预测图像中目标的边界框和目标所属类别概率。

（6）SSD：使用单个神经网络模型预测图像中目标的位置。

5.3 目标检测的深度学习框架

基于深度学习的目标检测算法和架构由一些组件和概念组成。在深入考察这些网络框架的体系结构之前，首先学习关于目标检测算法的一些重要部件，其中最关键的部件包括面向滑动窗口的目标检测、边界框方法、重叠度（IoU）、非极大性抑制和锚盒。

5.3.1 目标检测的滑窗法

检测图像中目标时，有一个非常简单的思路：首先把图像划分成一些区域或者某些特定的区域，然后逐个对这些区域进行分类。这种目标检测方法就是滑动窗口法，简称为滑窗法。顾名思义，滑窗是一个可以在整个图像中滑动的矩形框。该框具有固定的长度和宽度并使用某个步长在整个图像上移动。

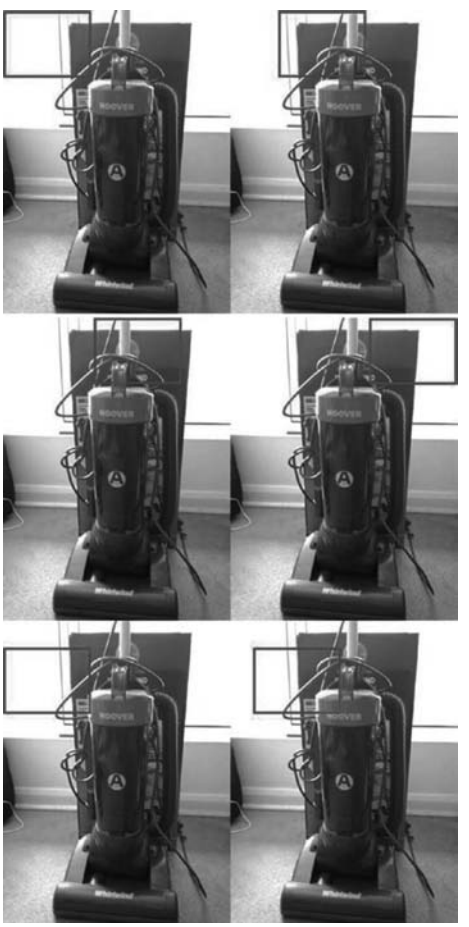


图 5-2 使用滑动窗口法检测和识别某个目标

如图 5-2 所示，在图像中的每个部分使用滑窗。红色方框在整个真空吸尘器的图像上滑动。沿着某一行从左到右的滑动，完成后再进入下一行的滑动。滑窗依次将图像中的不同部分变成观察点。因为这个窗口是滑动的，通常将这种方法称为滑动窗口法。这个过程虽然能够检测目标，却是一个计算昂贵、非常耗时的过程。

对于被滑窗界定为观察点的每个区域，可以将这些区域分类为是否包含感兴趣目标的类别，然后增加滑动窗口的大小并继续这个过程。

滑窗法已经被证明是一种有效的目标检测方法，但它是一种计算非常昂贵的技术，而且实现起来会非常慢，因为需要对图像中的所有区域进行分类。同样，为了实现对图像中目标的定位，需要比较小的窗口大小和比较小的移动步长。

5.3.2　边界框方法

因为滑动窗口法依赖于窗口的大小，输出的边界框不够精确。此时可以使用边界框方法，将整个图像划分为 $x \times x$ 个网格，然后为每个网格定义目标标签。边界框方法可以生成边界框的 x 坐标、y 坐标、高度、宽度和类别概率。如图 5-3 所示的边界框可以提供以下细节信息。

(1) Pc 表示网格单元中包含目标的概率，0：没有目标；1：包含目标。

(2) 如果 Pc 为 1，Bx 是边界框的 x 坐标。

(3) 如果 Pc 为 1，By 是边界框的 y 坐标。

(4) 如果 Pc 为 1，Bh 是边界框的高度。

(5) 如果 Pc 为 1，Bw 是边界框的宽度。

(6) C1 表示该对象属于第 1 个类别的概率。

(7) C2 表示该对象属于第 2 个类别的概率。

图 5-3　边界框方法

注意：类别的数量取决于当前问题是二分类还是多分类。

如果某个目标同时位于多个网格，那么包含关于该目标像素点的所有网格都要负责对该目标的检测。

提示：在实际案例的开发通常使用 19×19 的网格。此外，目标的中点同时位于两个独立网格的可能性较小。

5.3.3　重叠度指标

重叠度(IoU)用于确定关于目标位置的预测值距离实际真相有多近的度量指标。如图 5-4 所示，利用式(5-1)计算重叠度，分子是公共面积，分母是两个面积的并集。

$$\text{IoU} = \text{重叠区域} / \text{所有与目标相关区域的合并} \tag{5-1}$$

IoU 的取值越高越好。IoU 取值越大，就表明与目标相关区域的重叠性越好。因此，对目标位置的预测就越准确，预测效果就越好。如图 5-5 所示，与 IoU 值为 0.85 或 0.90 相比，IoU 值为 0.15 所对应的方框之间重叠度非常少。这就意味着 0.85 或 0.90 的 IoU 比 0.15 的 IoU 更好，接近于 1.0 的 IoU 值比 0.15 的 IoU 值具有更加准确的目标检测效果。因此，可以使用 IoU 这个指标度量和比较关于目标检测的解决方案。

IoU 指标不仅可以衡量和比较各种解决方案的性能，而且更容易区分有用的边界框和不那么有用的边界框。事实上，IoU 指标是一个具有广泛用途的重要概念。使用 IoU 指标，可以比较和对比所有可能解决方案的可接受性，并从中选择最好的一个解决方案。

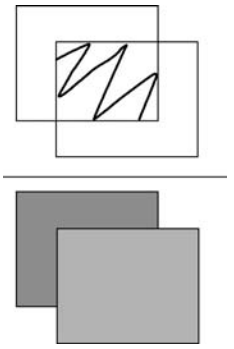


图 5-4　使用 IoU 度量目标检测方法性能

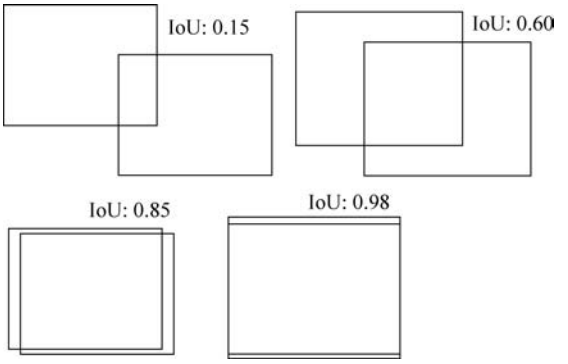


图 5-5　不同位置重叠块的 IoU 值

5.3.4　非极大性抑制

在试图检测图像中某个目标的时候，这个目标通常与多个网格相关联，如图 5-6 所示，一个目标可以跨越多个网格，哪个网格包含目标的效果最好，就选择哪个网格作为最终的目标检测结果。如果某个网格以最高的概率包含这个目标，那么这个网格的中心位置显然就是关于该目标位置的最终预测结果。

图 5-6　确定最终的目标检测结果

通常按下列几个步骤完成这个过程。

(1) 得到所有网格各自的概率。

(2) 设置概率阈值和 IoU 阈值。

(3) 丢弃概率值低于该阈值的网格。

(4) 选择概率较大网格作为边界框。

(5) 计算剩余边界框的 IoU。

(6) 丢弃低于 IoU 阈值边界框。

使用非最大抑制，可以丢弃大多数低于一定的阈值的边界框，保留重要的和有意义的信息，去除具有更多噪声的信息。

提示：通常将阈值设置为 0.5。建议使用不同的值进行迭代计算，并分析比较不同的阈值设置所产生的效果差异。

5.3.5　锚盒

我们不仅希望使用深度学习技术完成图像中的目标检测，还需要一种快速准确的方法来获取目标的位置和大小。锚盒就是对于目标检测非常有用的一个概念。锚盒用于捕获要检测对象的大小和长宽比。通常需要根据被检测对象的大小来预设锚盒的大小(高度和宽

度)。图 5-7 给出了关于锚盒的图示,可以使用锚盒对图像区域进行平铺填充,神经网络将为每个锚盒输出唯一的一组预测值。

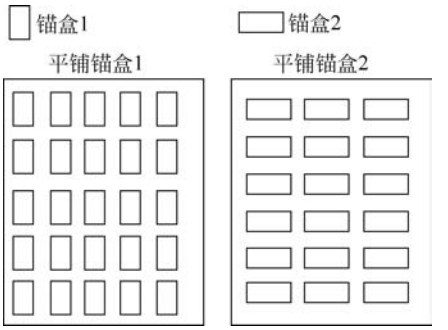


图 5-7　锚盒用于捕获目标的大小和长宽比

在目标检测的过程中,将每个锚盒平铺在图像区域,神经网络将为每个锚盒输出唯一的一组预测值。输出值中包括锚盒的概率值、IoU 指标、背景和偏移量。可以根据预测结果对锚盒做进一步精细化处理,使用多种不同尺寸的锚盒检测不同尺寸的目标。因此,锚盒可用于检测不同大小的目标,甚至可以使用锚盒检测多个或重叠的对象。与滑动窗口相比,这无疑是一个很大的改进。因为可以对整个图像进行一次性处理,所以可以实现更快的实时目标检测。这种网络模型不能预测边界框,只能给出各个平铺锚盒的概率值和改进值。

了解目标检测算法的若干关键部件后,下面讨论用于目标检测的深度学习网络模型架构。

5.4　深度学习网络架构

深度学习技术有助于解决目标检测问题,可以在图像、视频甚至直播视频流中检测感兴趣的对象。本章将创建一个实时视频流解决方案。

滑动窗口方法存在一些问题。对象在图像中可以有不同的位置,可以有不同的纵横比或大小。一个目标可能覆盖整个区域;另外,在某些地方,它只覆盖很小的百分比。图像中可能有多个目标。这些对象可以处于不同的角度或维度,或者一个对象可以位于多个网格中。而且,有些用例需要实时预测,且结果拥有非常多的区域,因此需要巨大的计算能力和计算时间。传统的图像分析和检测方法在这种情况下没有多大帮助。因此,需要基于深度学习的解决方案解决和开发鲁棒的目标检测解决方案。基于深度学习的解决方案能够更好地训练模型,获得更好的结果。

5.4.1　基于区域的 CNN

拥有大量的备选区域是目标检测算法的一项严峻挑战。Ross Girshick 等提出了基于区域的 CNN(Region-based CNN,R-CNN)解决对大量备选区域的选择问题。R-CNN 是一种基于备选区域的 CNN 网络架构。该解决方案建议使用选择性搜索方法,仅从图像中提取 2000 个区域,而不是对大量的备选区域进行分类。这些被提取出来的区域称为“建议区域”。R-CNN 的基本架构如图 5-8 所示。

图 5-8 给出了基于 R-CNN 的目标检测基本过程,具体如下。

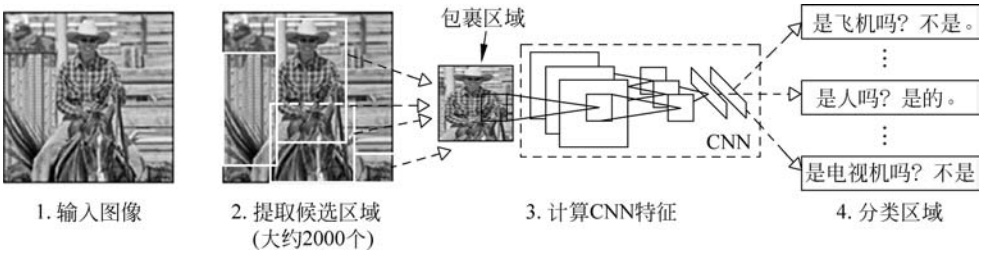


图 5-8　R-CNN 执行过程(来源：https://arxiv.org/pdf/1311.2524.pdf)

(1) 输入图像数据，如图 5-8 中步骤 1 所示。

(2) 获得感兴趣的建议区域，如图 5-8 中步骤 2 所示。这里有 2000 个建议区域。可以通过下列步骤获得建议区域：

- 首先为图像创建初始的区域划分。
- 然后由这些区域划分块生成不同的候选区域。
- 使用贪心算法迭代地将相似的候选区域合并成更大的候选区域。
- 将最后生成的候选区域作为建议区域输出这些建议区域。

(3) 根据 CNN 模型对所有 2000 个建议区域进行重构。

(4) 使用 CNN 考察每个建议区域，获得每个建议区域的特征。

(5) 使用支持向量机分析提取出来的关于建议区域特征信息，对建议区域是否包含目标的类别进行分类。

(6) 使用关于边界框回归分析给出关于目标边界框的预测。这意味着最终实现了对图像中目标类别的预测。如图 5-8 最后一步所示，给出了被预测图像区域是飞机、人还是电视显示器的预测判别。

R-CNN 使用上述过程实现对图像中目标的自动检测。这无疑是一个具有创新性的网络架构，它通过提出一种具有影响力的建议区域来实现目标检测。但是 R-CNN 也面临以下挑战。

(1) R-CNN 模型实现了 3 种算法(CNN 用于特征提取，SVM 用于目标分类，关于边界框回归分析用于获得关于目标的边界框)。这使得 R-CNN 方案的模型训练过程非常缓慢。

(2) R-CNN 模型使用 CNN 实现对每个建议区域的特征信息提取。每个图像中建议区域的数量是 2000 个。这意味着如果有 1000 张图，那么就需要提取 1000×2000 个建议区域的特征信息，显然会降低目标检测速度。

(3) 基于上述分析，对一个图像的目标检测通常需要 40～50s 的时间。对于庞大的图像数据集来说，这显然是一个问题。

(4) 另外，选择性搜索算法比较固定，不能对该算法做太多的改进。

由于 R-CNN 的目标检测速度比较慢，而且对于巨大的图像数据集实现起来相当困难。因此，R-CNN 的作者 Ross Girshick 提出了 Fast R-CNN 来克服这些问题。

5.4.2　Fast R-CNN

在 R-CNN 模型中，由于需要分别为每幅图像提取 2000 个建议区域，因此模型训练或模型测试在计算复杂度上是一个挑战。为了解决这个问题，Ross Girshick 等提出了 Fast R-CNN 网络模型，只需对每张图运行一次 CNN 计算就可以获得所有 2000 个建议区域的特征信息，而不是对每张图执行 2000 次 CNN 计算。图 5-9 给出了 Fast R-CNN 的网络架构。

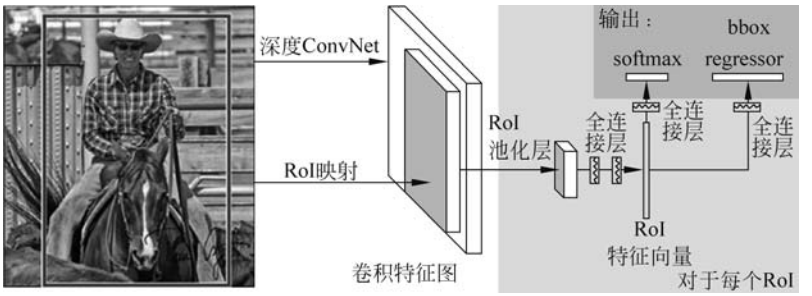


图 5-9　Fast R-CNN 模型执行过程

（来源：https://arxiv.org/pdf/1504.08083.pdf，并经研究人员许可在此发布）

获得建议区域后，对建议区域进行池化层处理，重塑所有输入信息，然后使用全连接层对其进行评估，最后使用 Softmax 层进行分类。

除了少数一些变化外，Fast R-CNN 的处理流程与 R-CNN 类似。

（1）图像输入如图 5-9 所示。

（2）图像传递给卷积网络，由卷积网络输出各个建议区域。

（3）对建议区域进行池化层处理，通过对卷积输入数据重塑得到新的建议区域。因此，通过建议区域池化层处理，使得所有建议区域的大小都相同。

（4）将每个区域传递到全连接层。

（5）使用 Softmax 层进行分类，并使用边界框回归器获得目标边界框的坐标。

与 R-CNN 模型相比，Fast R-CNN 具有下面几个优势。

（1）Fast R-CNN 不需要每次向 CNN 提供 2000 个提案区域，比 R-CNN 更快。

（2）Fast R-CNN 对每张图像只使用一次卷积操作，而不是像 R-CNN 那样使用 3 次卷积操作（提取特征、分类和生成目标边界框）。不需要存储特性映射，可以节省磁盘空间。

（3）Softmax 层通常比 SVM 层具有更好的准确性和更快的执行时间。

Fast R-CNN 显著减少了模型训练时间，也被证明具有更好的模型准确性。然而，Fast R-CNN 仍然使用选择性搜索方法获取建议区域，这使得它的模型性能没有得到显著的提高。因此，对于一个大型数据集，Fast R-CNN 预测速度仍然不够快。

5.4.3 Faster R-CNN

为了克服 R-CNN 和 Fast R-CNN 模型训练速度慢的问题，Shaoqing Ran 等提出了 Faster R-CNN 模型。Faster R-CNN 的改进思路是使用区域提议网络(Region Proposal Network，RPN)取代缓慢耗时的选择性搜索算法。Faster R-CNN 的基本架构如图 5-10 所示，该模型主要包括两个模块，第一个模块用于产生建议区域的深度全卷积网络，第二个模块利用使用区域进行目标检测的 Fast R-CNN 检测器，两者构成一个统一的目标检测网络架构。

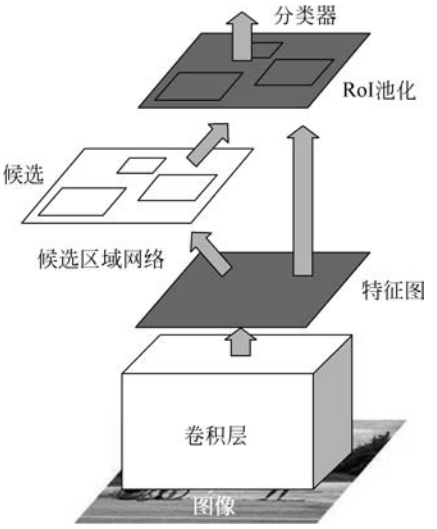


图 5-10 Faster R-CNN

(来源：https://papers.nips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf)

Faster R-CNN 的工作方式如下所述。

(1) 首先将一张图像输入 CNN，如图 5-10 所示。

(2) 由 CNN 生成的特征图后，交给 RPN 进行处理。RPN 的工作原理参见图 5-11，要点如下：RPN 接收由 CNN 模型在最后一步生成的特征图；RPN 使用滑窗基于特征图生成 k 个锚盒；生成的锚盒具有不同的形状和大小；RPN 预测锚盒是否为包含目标的锚盒；通过对边界盒的回归分析调整锚盒；RPN 并没有给出目标的类别，获得可能包含目标的建议区域和相应的得分。

(3) 对建议区域使用池化层进行处理，使得所有的建议区域具有相同的尺寸。

(4) 将建议区域提供给带有 Softmax 和线性回归分析功能的全连接层。

(5) 输出关于目标类别及其边界框的预测结果。

Faster R-CNN 能够巧妙地将带全连接层的深度卷积网络和 Fast R-CNN 的 RPN 方法结合起来，形成一体化的统一目标检测解决方案。

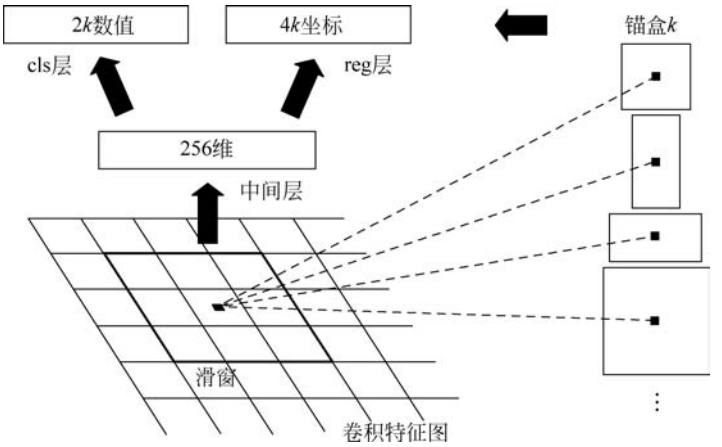


图 5-11　Faster R-CNN 中使用提议区域网络

虽然 Faster R-CNN 在性能上比 R-CNN 和 Fast R-CNN 都有一定的提高，但是该算法并不能同时分析图像的所有部分。相反，图像中的每个部分都被安排在某个序列中进行分析。因此，Faster R-CNN 仍然需要对单个图像进行大量的遍历来识别所有的目标。此外，对于按序列进行串行工作的大多数系统来说，其性能大多取决于前面各个步骤的性能。

5.4.4　YOLO 算法

YOLO 算法可以实现实时的目标检测效果。之前讨论的算法使用图像中的特定区域来定位图像中的目标。这些算法每次看到的只是图像中的某个部分，而不是完整的图像。YOLO 则使用单个 CNN 模型预测目标的边界框和属于某个类别的概率。YOLO 目标检测算法由 Joseph Redmon 等在 2016 年提出。如图 5-12 所示，YOLO 将图像划分为一种网络形式（用 S 表示），使用每个网格单元预测目标的边界框（用 B 表示），然后 YOLO 对每个边界框进行运算，生成用于评判边界框优劣的置信度分数，并且预测出目标的属于某个类别的概率。最后，选取带有上述类别概率和置信度分数的边界框，并使用这些边界框实现对图像中目标的定位。

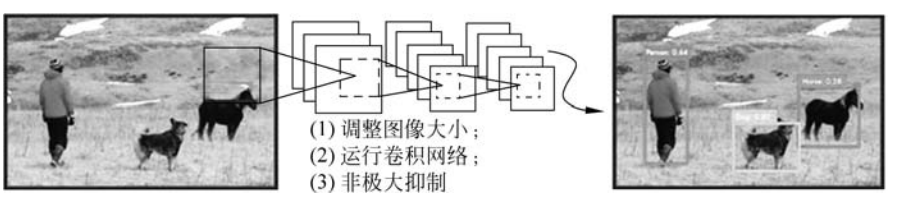


图 5-12　YOLO 算法过程（来源：https://arxiv.org/pdf/1506.02640v5.pdf）

1. YOLO 的显著性特征

YOLO 具有以下显著性特征。

(1) YOLO 将输入图像划分为 $S\times S$ 的网格，每个网格只负责预测某个对象。如果某个对象的中心落在某个网格单元中，那么该网格单元就负责实现对该目标的检测。

(2) 对于每个网格单元，都有一个预测边界框(B)。每个边界框有 5 个属性：x 坐标、y 坐标、宽度、高度和置信度分数，即它有(x，y，w，h)和一个分数。这个置信分数值表示的是关于方框中包含一个目标置信度。它也反映了边界框的准确度。

(3) 网格单元的宽度 w 和高度 h 被归一化为图像的宽度和高度。坐标 x 和 y 表示相对于网格单元边界的中心。

(4) 置信被定义为概率(目标)乘以 IoU 指标。如果网格中没有目标，那么置信度为零。否则，置信度就等于被预测的方框和真实数据之间的 IoU。

(5) 每个网格单元被预测为 C 类的条件概率—— Pr(Classi | Object)。这些概率取决于包含目标的网格单元。对每个网格单元只预测一组关于类别的概率，而不考虑边界框 B 的数量。

(6) 在模型测试环节，将类别条件概率和与将目标类别归属的模型预测概率相乘，得到关于每个边界框包含某个特定类别目标的置信度分数：

$$\Pr(\text{Class}_i \mid \text{Object})\times\Pr(\text{Object})\times\text{IoU}_{\text{pred}}^{\text{truth}}=\Pr(\text{Class}_i)\times\text{IoU}_{\text{pred}}^{\text{truth}} \tag{5-2}$$

下面讨论如何在 YOLO 中计算损失函数。在详细研究整个模型的体系结构之前，理解关于损失函数的计算公式是一件非常重要的事情。

2. YOLO 中的损失函数

YOLO 算法首先对每个单元格的多个边界框进行预测，然后根据真值数据选择具有最大 IoU 的边界框。为了计算损失，YOLO 算法面向模型输出的误差平方和进行最优化计算，因为误差平方和比较容易进行最优化计算。

损失函数的计算公式如式(5-3)所示，包括局部损失、置信度损失和分类损失这几项。首先给出损失函数的完整表达式，然后再详细介绍其中的每一项。

$$\begin{aligned}
&\lambda_{\text{coord}}\sum_{i=0}^{s^2}\sum_{j=0}^{s}1_i^{\text{obj}}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]\\
&+\lambda_{\text{coord}}\sum_{i=0}^{s^2}\sum_{j=0}^{s}1_{ij}^{\text{obj}}\left[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{n_i}-\sqrt{\hat{n}_i})^2\right] \quad \Big\}\ \text{目标位置损失函数}\\
&+\sum_{i=0}^{s^2}\sum_{j=0}^{s}1_{ij}^{\text{obj}}(c_i-\hat{c}_i)^2 \longrightarrow \text{框内检测出物体的置信度损失}\\
&+\lambda_{\text{coord}}\sum_{i=0}^{s^2}\sum_{j=0}^{s}1_{ij}^{\text{nobj}}(c_i-\hat{c}_i)^2 \longrightarrow \text{框内未检测出物体的置信度损失}\\
&+\sum_{i=0}^{s^2}1_i^{\text{obj}}\sum_{\text{coord}}(p_i(c)-\hat{p}_i(c))^1 \longrightarrow \text{目标分类损失函数}
\end{aligned} \tag{5-3}$$

式(5-3)表示的损失函数计算式中包含定位损失、置信度损失和分类损失，其中 1_i^{obj} 表示目标出现在单元格 i 中，1_i^{obj} 表示网格单元 i 中的第 j 个边界框的预测器对该预测“负责”。

(1) 定位损失用于度量预测边界框的误差，就是度量边界框的位置和大小误差。在式(5-3)中，前两项表示局部损失。如果单元格 i 中的第 j 个边界框负责检测对象，则 1_i^{obj} 为

1；否则 1_i^{obj} 为 0。λ_{coord} 负责增加位于边界框的坐标损失的权重，λ_{coord} 的默认值为 5。

（2）置信度损失表示在边界框中检测到目标时的损失。这是计算公式中第二个损失项，即为置信度损失计算公式中的前一项。其中，$\hat{C}_i$ 是网格单元 i 中的第 j 个边界框的置信度；如果网格单元 i 中的第 j 个边界框检测目标，则 $1_i^{obj}=1$；否则 $1_i^{obj}=0$。

（3）如果没有检测到目标，则使用置信度损失计算公式中的后面一项。与前一项的定义类似，1_i^{noobj} 是 $1_i^{obj}=1$ 的补数；$\hat{C}_i$ 是网格单元 i 中的第 j 个边界框的置信度；λ_{noobj} 负责减少位于边界框的坐标损失的权重。

（4）最后一项表示分类损失。如果一个目标确实被检测到，那么对于每个网格单元，它的分类损失是关于每个类别概率的平方误差。如果网格单元 i 中出现检测目标，则 $1_i^{obj}=1$；否则 $1_i^{obj}=0$；$\hat{p}_i(c)$ 表示网格单元 i 中的 c 类的条件类概率。

（5）最后得到的总的损失是所有这些部分项的总和。深度学习解决方案的目标是最小化这种损失值。

3. YOLO 的网络架构

YOLO 的网络架构设计如图 5-13 所示，构建该网络的灵感来自 GoogLeNet。该网络有 24 个卷积层，然后是 2 个全连接层。与 GoogLeNet 使用的 Inception 模块不同，YOLO 使用的 1×1 还原层，后面跟着 3×3 的卷积层。YOLO 可能会检测关于同一个目标的多个副本，因此 YOLO 使用了非极大性抑制方法删除重复的低置信度边界框。

在图 5-14 中提供了一个包含 13×13（$S=13$）网格的图。图中总共有 169 个网格，每个网格可以预测 5 个边界框，因此，总共有 169×5 = 845 个边界框。当使用 30%或更大阈值时，可以得到如图 5-14 所示的 3 个边界框。

因此，YOLO 只对图像查看一次，的确是一种很聪明的目标检测方法。YOLO 是一种非常快速的实时处理算法，具体参见扩展阅读[5]。YOLO 的特点如下所述（引自拓展阅读[5]）。

（1）YOLO 非常简单。

（2）YOLO 的运行速度非常快。因为把目标检测作为一个回归问题，不需要使用复杂的管道。在模型测试的时候，只是简单地在一个新的图像上运行神经网络，实现对图像中目标检测的预测。基本网络运行速度是 45 帧/秒，在 Titan X GPU 上没有进行批处理，快速版本的运行速度可以超过 150 帧/秒。这意味着可以以小于 25ms 的延迟实时处理流媒体视频。此外，YOLO 的平均准确度是其他实时目标检测系统的两倍以上。

（3）YOLO 在做目标检测预测时，会考虑整个图像的信息。这一点与基于滑动窗口和建议区域的技术有着很大的不同，YOLO 在训练和测试期间查看的是整个图像，可以隐式地实现对目标类别及其外观上下文信息的有效编码。

（4）YOLO 学习到的是关于目标的一种泛化表示形式。在使用自然图像进行模型训练，使用人工绘制图像进行测试的场合，YOLO 的模型性能远远优于 DPM 和 R-CNN 等顶级目标检测方法。YOLO 是一种具有高度普适性的模型，在应用于新域或遇到意外输入时不太可能出现故障。

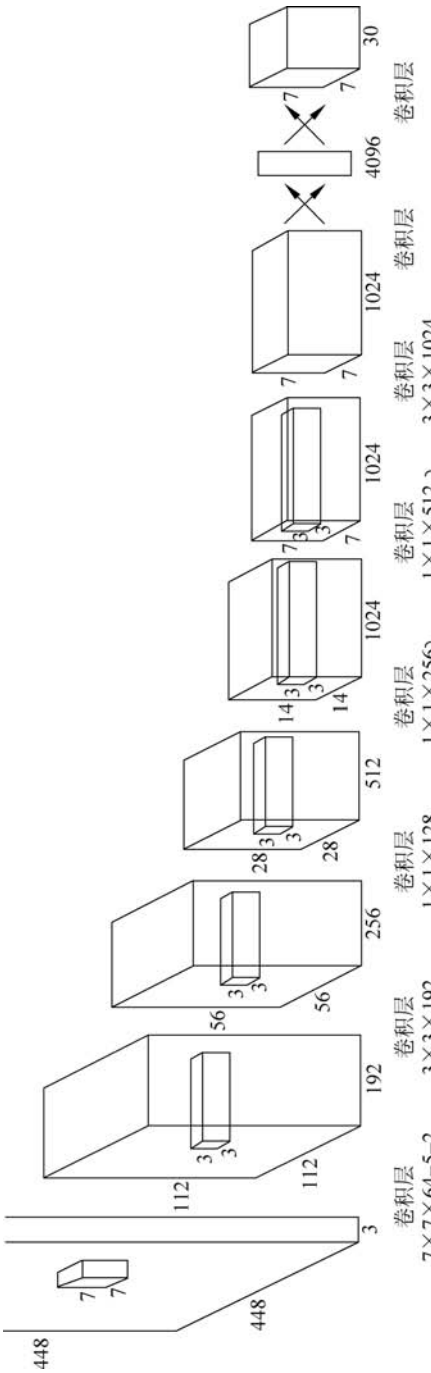


图 5-13 YOLO 网络架构(图片来源：https://arxiv.org/pdf/1506.02640v5.pdf)

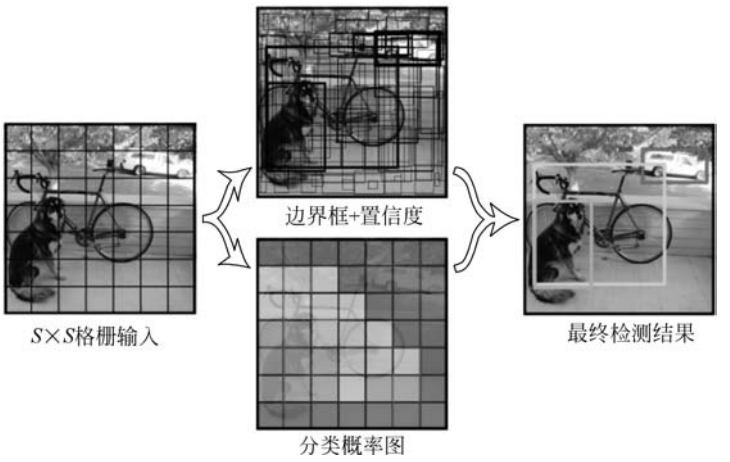


图 5-14　YOLO 算法(图片来源：https://arxiv.org/pdf/1506.02640v5.pdf)

YOLO 也会面临一些挑战。它的定位误差比较大。此外，因为每个网格单元只能预测两个边界框，输出只能有一个类别，所以 YOLO 只能预测有限数量的临近对象。YOLO 还存在记忆能力低下的问题。不过，在 YOLO 的下一个版本中 YOLO v2 和 YOLO v3 中，这些问题都得到了很好的解决。有兴趣的读者可以通过官方网站 https://pjreddie.com/darknet/yolo/获得更加深入的相关知识。

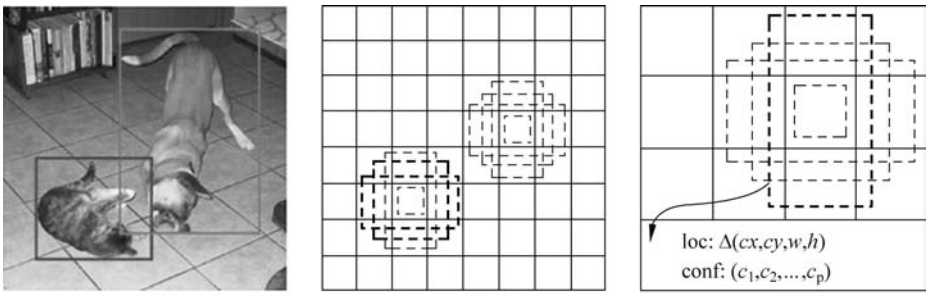
YOLO 是一种应用最为广泛的目标检测解决方案。它的独特之处在于模型的简单和快速。

5.4.5　单阶段多框检测器

到目前为止，已经讨论了 R-CNN、Fast R-CNN、Faster R-CNN 和 YOLO。为了克服网络模型在目标进行实时检测所面临的检测速度慢的问题，C. Szegedy 等在 2016 年 11 月提出了 SSD 网络。

SSD 使用第 4 章讨论的 VGG16 架构，在 VGG16 架构上做了一些修改。通过使用 SSD，只需经过单次处理就可以从一幅图像中检测到多个目标。由于 SSD 只使用一个单一的前向传递计算，就可以同时实现对目标的定位和分离，因此它是一种单阶段目标检测方法。基于 RPN 的解决方案(例如 R-CNN、Fast R-CNN 等模型)则需要两个阶段实现对目标的检测，第一个阶段获得建议区域，第二个对每个建议区域进行目标检测。因此，SSD 要比基于 RPN 的方法要快得多。Szegedy 等给 SSD 命名中的多框 multibox，又是什么含义呢？检测器这个单词的意义是显而易见的。

如图 5-15 所示，一个带有真值 (GT)的原始图像，进行了卷积计算获得 8×8 大小的特征图，并且获得了多个具有不同大小和位置的边界框。所以 SSD 的处理过程就是图像经过一系列的卷积计算，可以获得一个大小为 $m \times n$ 的特征图层，这个特征图层具有 p 个通道。



(a) 带GT框的图像　(b) 8×8特征图　(c) 4×4特征图

图 5-15　SSD 模型的处理过程(图片来源：https://arxiv.org/pdf/1512.02325.pdf)

对于每个位置，可以得到 k 个可能的具有不同大小和长宽比的边界框。对其中的每一个边界框中的计算 c 类得分和相对于原始默认边界框的 4 个偏移量，并最终获得输出 $(c+4)\times k\times m\times n$。

SSD 采用平滑 L_1 范数来计算位置损失。它可能没有 L_1 那么精确，但仍然是相当准确的一种计算方法。

作为与 YOLO 模型进行对比的形式，图 5-16 给出了 SSD 模型完整的网络架构。

在 SSD 中，不同层次的特征图通过使用 3×3 卷积层来提高准确度。通过分析前述网络结构，对于目标检测的第一层(conv4_3)，它的空间尺寸为 38×38，这是一个相当小的尺寸，使得较小尺寸目标的预测准确度较低。对于同样的 conv4_3，可以使用前述公式进行计算并输出结果。对于 conv4_3，它的输出将是 $38\times38\times4\times(c+4)$，其中 c 是要预测的类别的数量。

SSD 使用 L_{conf}(loss - confidence loss)和 L_{ioc}(localization loss) 这两个损失函数实现对模型的优化计算。L_{conf} 是关于类别预测的损失，L_{ioc} 表示真值和被预测边界框之间的不匹配程度。这两种损失的数学公式已在前面的文章中给出，关于这些公式的推导超出了本书的范围。

SSD 中还有一些其他重要处理环节。

(1) 通过翻转、裁剪和颜色失真实现对样本数据的增强以提高模型的准确性。对每个训练样本进行如下随机抽样：使用原始图像；使用 IoU 为 0.1、0.3、0.5、0.7 或 0.9 为图像打上补丁；随机地给图像打上一个补丁；将使用的采样补丁宽高比限制在 0.5～2，并且每个采样补丁的大小是原始大小的[0.1,1]倍。然后将每个图像样本调整为固定大小并进行水平翻转。也可以将照片变形处理用于实现图像增强。

(2) SSD 通过非最大抑制方法来消除大量重复的预测边界框。

(3) SSD 会导致预测目标的数目高于实际的目标数目。通常情况下，负样本的数量要比正样本的数量要多，这样导致了样本类别分布数量的不均衡。正如扩展阅读[5]所述："我们没有使用所有的负样本，而是对每个默认框使用其的置信度损失的最高值进行排序，然后取排在最前面的默认框。这样就使得负样本和正样本的比例最多是 3∶1。可以发现，这有助于提升模型优化速度和模型训练过程的稳定性。"

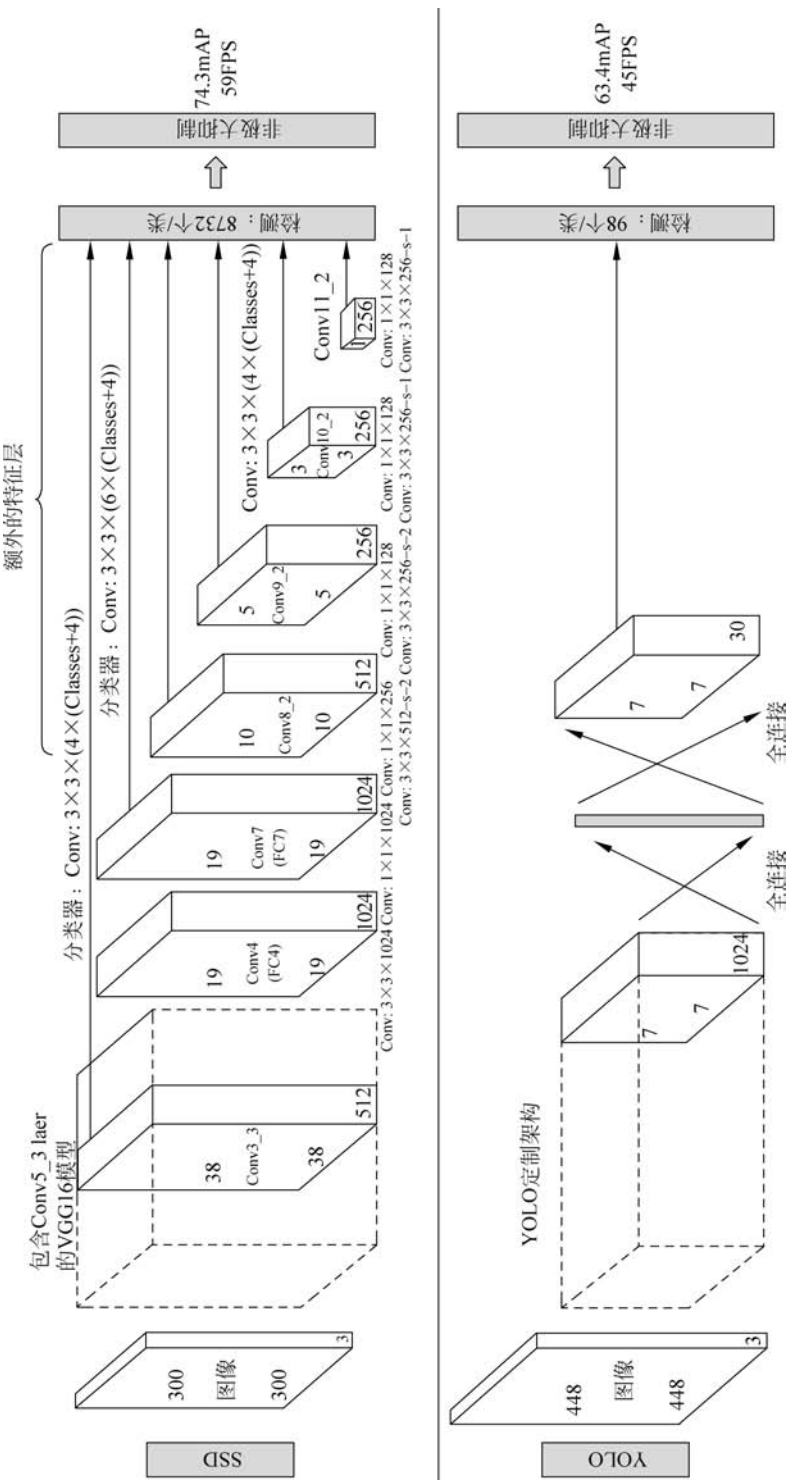


图 5-16　YOLO 和 SSD 的比较(图片来源：https://arxiv.org/pdf/1512.02325.pdf)

基于前述网络架构，总结出 SSD 具有如下几个特点。

(1) 对小目标的检测可能是 SSD 面临的一个挑战。为了解决这个问题，可以提高图像的分辨率。

(2) 模型的目标检测准确度与速度成反比；如果想提高检测速度，就需要增加边界框的数量。

(3) SSD 比 R-CNN 具有更高的分类误差，但它的定位误差比较小。

(4) SSD 可以很好地利用尺寸较小的卷积过滤器实现对目标类别的预测，并且可以使用多尺度特征图进行目标检测。这有助于提高模型的准确性。

SSD 的核心是对特征图使用小型卷积过滤器，由此实现对固定数目的默认边界框所属类别的分数预测和框偏移量的预测。

SSD 的准确度还有进一步提升的空间，它混淆了具有相似范畴的对象，而且 SSD 基于 VGG16 架构，需要消耗大量的模型训练时间。尽管如此，SSD 仍然是一个非常好的解决方案，可以很容易地用于端到端模型训练。它的检测速度非常快，可以实时地运行，而且比 Faster R-CNN 性能更好。

至此，已经介绍了用于实现目标检测的深度学习网络结构并讨论了其中的一些主要算法，下面将通过开发实际的 Python 代码来实现这个解决方案。在此之前，还需要了解迁移学习的概念，这是一个创新的解决方案，可以实现专家级别的最先进的算法。

5.5 迁移学习

迁移学习，顾名思义就是将作为学习成果的知识分享或传递给他人。在深度学习领域，研究人员和研究组织会创建新的神经网络架构。他们使用最先进的强大的多码处理器算力，在精心策划和选择的大型数据集上使用最先进的训练算法实现对模型的训练。

对我们来说，创造这样的智能就像是重新发明轮子。因此，使用迁移学习是在使用那些经过数百万数据点训练的网络模型。这就可以使用专业研究人员生成的智能，并且在实际的数据集上实现相同的功能。这个过程被称为迁移学习。

在迁移学习中，使用预先训练好的模型达到目的。预先训练好的模型拥有原始模型的最终权重。使用预训练模型的基本思想是，网络模型的初始层检测的目标的基本特征，随着网络层数的加深，基本特征开始逐渐组合成一些不同程度的形状信息。网络模型提取出来的基本特征可以用于任何类型的图像。所以，如果一个模型被训练用来区分手机，那么它就可以被训练用来区分汽车。

迁移学习的基本流程如图 5-17 所示，网络的第一层被冻结，用于提取边缘、线等低级特征。最后一层可以根据具体业务需要进行定制。

迁移学习比传统的机器学习更加快捷，需要的训练数据更少。第 8 章将讨论关于预训练模型的更多细节。迁移学习通过利用面向其他场合开发的解决方案解决现有的业务问题。本书的后续章节中使用迁移学习方法。

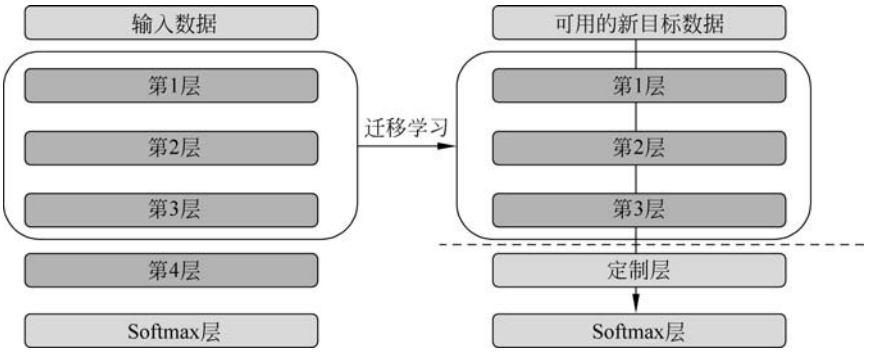


图 5-17　迁移学习使用预先训练好的网络

理论已经介绍完毕，现在开发目标检测解决方案。

5.6　实时的目标检测 Python 实现

我们将使用 YOLO 实现实时的目标检测。如果需要从网上下载网络模型的预训练的权重，可以从本章开头给出的链接中下载代码、权重、标签和预期输出。

（1）导入所有必要的库。

```
import cv2
from imutils.video import VideoStream
import os
import numpy as np
```

（2）从本地路径加载配置。需要加载权重、配置和标签，对一些关于检测的设置进行配置。

```
localPath_labels = "coco.names"
localPath_weights = "yolov3.weights"
localPath_config = "yolov3.cfg"
labels = open(localPath_labels).read().strip().split("\n")
scaling = 0.005
confidence_threshold = 0.5
nms_threshold = 0.005 # Non Maxima Supression Threshold Vlue
model = cv2.dnn.readNetFromDarknet(localPath_config, localPath_weights)
```

（3）开始处理视频。通过对目标模型访问实现对未连接的网络层的配置。

```
cap = VideoStream(src = 0).start()
layers_name = model.getLayerNames()
output_layer = [layers_name[i[0] - 1] for i in model.
getUnconnectedOutLayers()]
```

注意：建议研究模型的组件并打印，以便更好地理解它们。

图 5-18　目标的实时检测

（4）准备执行对目标的检测。这一步是整个解决方案的核心步骤。它检测目标和边界框，并将文本添加到边界框的顶部。从一个 while 循环开始，然后读取边界框。接下来是设置边界框的宽度和高度。然后通过循环方式遍历视频中的每一帧图像。如果获得的置信度高于之前设置的置信度阈值，就可以检测到目标。接下来，对检测到的目标进行标记，并在检测到的边界框上显示出相应的置信度分数。输出结果如图 5-18 所示，能够以 99.79％的准确率从视频流中实时检测到手机目标。

```
while True:
frame = cap.read()
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416),swapRB=True, crop=False)
model.setInput(blob)
nnoutputs = model.forward(output_layer)
confidence_scores = []
box_dimensions = []
class_ids = []
for output in nnoutputs:
for detection in output:
scores = detection[5:]
class_id = np.argmax(scores)
confidence = scores[class_id]
if confidence > 0.5 :
box = detection[0:4] * np.array([w, h, w, h])
(center_x, center_y, width, height) = box.
astype("int")
x = int(center_x - (width / 2))
y = int(center_y - (height / 2))
box_dimensions.append([x, y, int(width),
int(height)])
confidence_scores.append(float(confidence))
class_ids.append(class_id)
ind = cv2.dnn.NMSBoxes(box_dimensions, confidence_scores,
confidence_threshold, nms_threshold)
for i in ind:
i = i[0]
(x, y, w, h) = (box_dimensions[i][0], box_dimensions[i][1],box_dimensions[i][2], box_
dimensions[i][3])
```

```
cv2.rectangle(frame,(x, y), (x + w, y + h), (0, 255,255), 2)
label = "{}: {:.4f}".format(labels[class_ids[i]],
confidence_scores[i])
cv2.putText(frame, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,0,255), 2)
cv2.imshow("Yolo", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cv2.destroyAllWindows()
cap.stop()
```

这个解决方案可以识别真实世界中的目标，并且在目标周围创建一个边界框以及名称和置信度评分。可以将这个解决方案应用于多个场合，例如应用于自己定制的数据集，也可以用于检测图像和视频中的目标。

5.7　小结

目标检测是一个非常强大的解决方案，可以应用于很多领域和业务，几乎所有的行业都可以从目标检测技术中受益。目标检测可用于光学字符识别、自动驾驶、对象和人的跟踪、人群监视、安全机制等多个场合。这种计算机视觉技术正在改变实时处理系统能力的面貌。

本章讨论了关于目标检测的网络架构——R-CNN、Fast R-CNN、Faster R-CNN，YOLO 和 SSD。所有这些网络架构都基于深度学习的架构，在架构设计上都比较有创新性，其中有些网络架构的表现在某些方面优于其他的网络架构。一般来说，需要在处理速度和模型准确度之间进行取舍，所以必须根据业务问题仔细地选择适当的网络架构。

本章还讨论了迁移学习。迁移学习是一种崭新的解决方案，它使用已经使用数百万张图像样本训练过的预训练网络。迁移学习能够使研究人员利用强大的处理器产生的智能。它是一种工具，可以使每个人都能使用这些真正的深层网络，并根据需要实现对它们的定制。通过迁移学习方式使用经过预训练的 YOLO 模型，能够实现实时的目标检测。

目标检测可应用在很多实际的解决方案之中，但输入数据集将最终决定解决方案的准确度。因此，如果使用网络模型基于自定义的数据集，请在数据收集阶段进行一些重要工作。

习题

（1）解释锚盒和非极大性抑制的概念。

（2）边界框对目标检测的重要性有哪些？

（3）R-CNN、Fast R-CNN 和 Faster R-CNN 有什么不同，有什么改进？

（4）迁移学习如何改进神经网络解决方案？

（5）从 www. kaggle 下载 Open Images 2019 数据集。com/ c/openimages-2019 -

object-detection，并使用它创建一个基于 YOLO 的解决方案。

（6）从链接 https://public.roboflow.com/object-detection/chess-full 获取象棋数据集，并使用它根据本章的网络架构实现对棋子的定位。

（7）从链接 https://public.roboflow.com/object-detection/raccoon 获取浣熊数据集，并使用它创建一个目标检测解决方案。

（8）从链接 https://cocodataset.org/#home 获取 COCO 数据集，并使用这个数据集比较不同目标检测网络架构的性能。

（9）从链接 https://public.roboflow.com/object-detection/vehiclesopenimages 获取数据集 Vehicles-OpenImages 并创建一个目标检测解决方案。

拓展阅读

[1] Cong T，Fe Ng Y，Xing Y，et al. The Object Detection Based on Deep Learning[C]//4th International Conference on Information Science and Control Engineering（ICISCE）. IEEE Computer Society，2017.

[2] Howard A G，Zhu M，Chen B，MobileNets：Efficient Convolutional Neural Networks for Mobile Vision Applications[EB/OL]. https://arxiv.org/pdf/1704.04861v1。

[3] Sandler M，Howard A，Zhu M，et al. MobileNetV2：Inverted Residuals and Linear Bottlenecks[EB/OL]. https://arxiv.org/abs/1801.04381v4.

[4] Howard A，Sandler M，Chu G，et al. Searching for MobileNetV3[EB/OL]. https://arxiv.org/pdf/1905.02244v5.

[5] Redmon J，Divvala S，Girshick R，et al. You Only Look Once：Unified，Real-Time Object Detection[EB/OL]. https://arxiv.org/pdf/1506.02640v5.