目录

1 实践目的

1. 理解和应用深度学习领域的目标检测算法。
2. 使用指定的数据集训练模型，并探索各种方法以提升目标检测的准确性。
3. 进行模型优化，以提高其性能表现。
4. 对训练完成的模型进行全面评估和比较分析。

2 课题分析

本课题为俯视角下的停车位检测，是一种目标检测任务，旨在识别和定位停车场中的每个停车位。以及判断是 Occupied 或 Vacant 这样的二分类任务。

2.1 目标检测介绍

目标检测（Object Detection）的任务是找出图像中所有感兴趣的目标（物体），确定它们的类别和位置，是计算机视觉领域的核心问题之一。由于各类物体有不同的外观、形状和姿态，加上成像时光照、遮挡等因素的干扰，目标检测一直是计算机视觉领域最具有挑战性的问题。

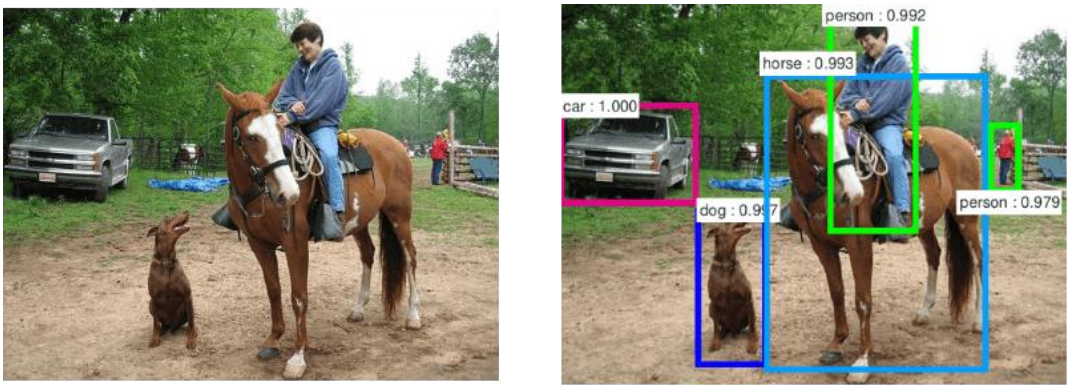


图 1 目标检测算法介绍

我在之前只是跑过一些目标检测的算法以及训练过 YOLO 系列的代码，并没有很深入了解目标检测算法的发展。借着这次的实践，阅读文献和查阅网上资料简单梳理了一下目标检测算法的分类。

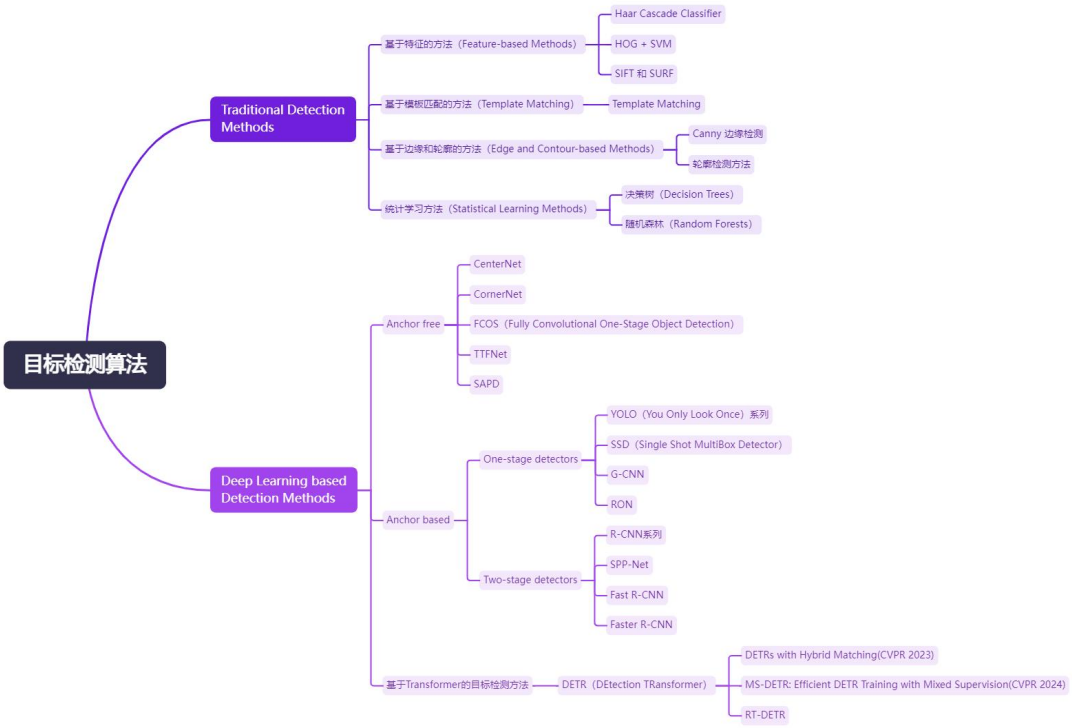


图 2 目标检测算法分类

目标检测算法可以根据其实现方式大致分为传统算法和深度学习算法两类。

2.1.1 传统目标检测算法：

传统目标检测算法主要依赖于手工设计的特征和特定的算法来识别目标。这些方法通常包括以下几种：

Haar 特征和级联分类器：使用 Haar 特征进行物体检测，经常与 AdaBoost 算法结合使用。

HOG（方向梯度直方图）特征和 SVM（支持向量机）：HOG 特征能够有效地描述物体的边缘和形状信息，结合 SVM 用于分类检测目标。

传统机器学习方法：如基于像素颜色、纹理等特征的分类器，如基于颜色直方图或纹理特征的分类器。

模板匹配：将预定义的模板与图像中的每个位置进行比较，以识别目标。

2.1.2 深度学习目标检测方法

深度学习目标检测方法在发展过程中大致可分为不同的子类别，包括

Anchor-free（无锚点）、Anchor-based（基于锚点）以及基于 Transformer 的算法。其中 Anchor-based 算法进一步分为一阶段和二阶段方法。

(1)Anchor-free（无锚点）方法：

这类方法试图摆脱传统的 Anchor 框架，不需要在输入图像中定义预定义的 Anchor 框来预测目标位置和边界框。代表性的算法包括 CornerNet 和 CenterNet。

CornerNet：使用目标的角点（corners）作为关键点来检测目标位置和边界框。

CenterNet：以目标中心点作为检测中心，同时预测目标的边界框大小和类别。

Anchor-based（基于锚点）方法：

这种方法依赖于预定义的 Anchor 框，通过网络预测这些 Anchor 框的偏移量和目标类别。主要分为一阶段和二阶段方法。

(2)One-stage detectors（一阶段方法）：

YOLO 系列：You Only Look Once，通过回归方式直接预测目标的位置和类别，速度较快，但对小目标和复杂场景的检测精度有所限制。

SSD：Single Shot Multibox Detector，通过在不同尺度上的特征图上预测目标的边界框和类别，具有较高的检测速度和准确率。

(3)Two-stage detectors（二阶段方法）：

RCNN 系列：包括 RCNN、Fast R-CNN、Faster R-CNN 等，这些方法首先生成候选区域，然后对这些区域进行分类和回归，具有更高的精确度但速度较慢。

Mask R-CNN：在 Faster R-CNN 的基础上扩展，同时预测目标的边界框和掩膜，用于实例分割任务。

(4)基于 Transformer 的目标检测方法：

最近，随着 Transformer 架构在自然语言处理领域的成功，它也被引入到视觉任务中，包括目标检测。

DETR：Detection Transformer，使用 Transformer 编码器-解码器结构来直接从输入图像中生成目标的位置和类别预测，摆脱了传统 Anchor 机制，有效地处理了对象之间的遮挡和尺度变化问题。

2.2 模型选择

本次项目选取的模式分别是一阶段方法的 YOLOv10 代表，二阶段方法的 Faster R-CNN 以及基于 Transformer 的 RT-DETR 模型。

YOLOv10 是清华大学研究人员在 UltralyticsPython 清华大学的研究人员在 YOLOv10 软件包的基础上，引入了一种新的实时目标检测方法，解决了 YOLO 以前版本在后处理和模型架构方面的不足。通过消除非最大抑制（NMS）和优化各种模型组件，YOLOv10 在显著降低计算开销的同时实现了最先进的性能。大量实验证明，YOLOv10 在多个模型尺度上实现了卓越的精度-延迟权衡。+

Faster R-CNN 是由 Ross B.Girshick 在 2016 年提出的，做为 two-stage 算法的经典之作，它在前两作（R-CNN 和 Fast-RCNN）的基础上，又有了很大改进，最终在 VOC2007 测试集测试 mAP 达到 73.2%，目标检测速度可达 5 帧/秒（但依然无法实现视频实时检测，所以后来有了 one-stage 著名的 YOLO 系列）。虽然提出的较早，但它至今仍是许多目标检测算法的基础，所以对其原理的学习和掌握有助于更好的了解后续模型和算法。

Real-Time Detection Transformer (RT-DETR) 由百度开发，是一种尖端的端到端物体检测器，可在保持高精度的同时提供实时性能。它基于 DETR（无 NMS 框架）的思想，同时引入了基于 conv 的骨干和高效混合编码器，以获得实时速度。RT-DETR 通过解耦尺度内交互和跨尺度融合，高效处理多尺度特征。该模型具有很强的适应性，支持使用不同的解码器层灵活调整推理速度，无需重新训练。RT-DETR 在 CUDA 等加速后端（TensorRT ）上表现出色，优于许多其他实时物体检测器。

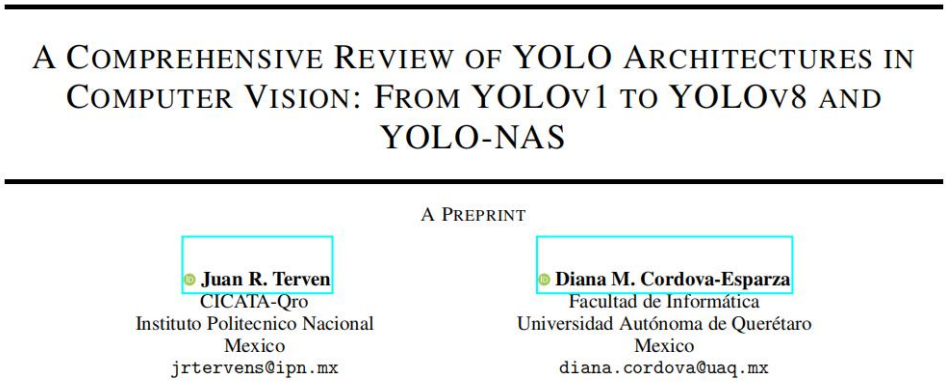
3 算法介绍

接下来的 3.1 部分我将根据我的理解总结 YOLO 系列，主要参考的论文是《A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS》。在 3.2 节，我将总结我所学到的 RCNN、FastRCNN 以及 FasterRCNN。最后 3.3 部分则是 RT-DETR 模型。

3.1 YOLO’s evolution

YOLO’s evolution

论文：https://arxiv.org/abs/2304.00501v6



A COMPREHENSIVE REVIEW OF YOLO ARCHITECTURES IN COMPUTER VISION: FROM YOLOV1 TO YOLOV8 AND YOLO-NAS

A PREPRINT

Juan R. Terven
CICATA-Qro
Instituto Politecnico Nacional
Mexico
jrtervens@ipn.mx

Diana M. Cordova-Esparza
Facultad de Informática
Universidad Autónoma de Querétaro
Mexico
diana.cordova@uaq.mx

图 3 YOLO’s evolution

摘要：YOLO 已经成为机器人、无人驾驶汽车和视频监控应用的核心实时物体检测系统。我们对 YOLO 的演变进行了全面的分析，研究了从最初的 YOLO 到 YOLOv8 每次迭代的创新和贡献。我们首先描述了标准指标和后处理；然后，我们讨论了每个模型的网络结构和训练技巧的主要变化。最后，我们总结了 YOLO 发展的基本经验，并提供了对其未来的看法，强调了提高实时物体检测系统的潜在研究方向。

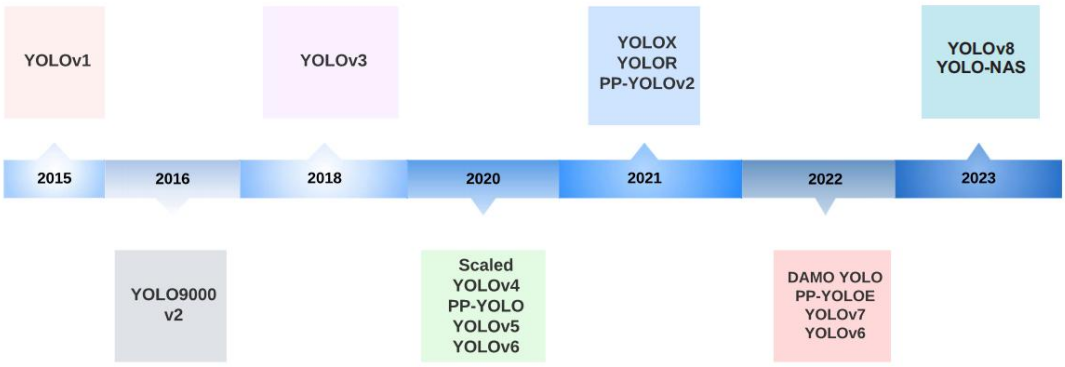


图 4 YOLO 版本的时间轴

YOLOv1

YOLO 将输入图像划每个边界框的预测由五个值组成：Pc、bx、by、bh、bw ，其中 Pc 是 bounding box 的置信度分数，反映了模型对 bbox 包含物体的置信度以及 bbox 的精确程度。bx 和 by 坐标是方框相对于网格单元的中心，bh 和 bw 是方框相对于整个图像的高度和宽度。YOLO 的输出是一个 S×S×(B×5+C)的张量，可以选择用非最大抑制（NMS） 来去除重复的检测结果。

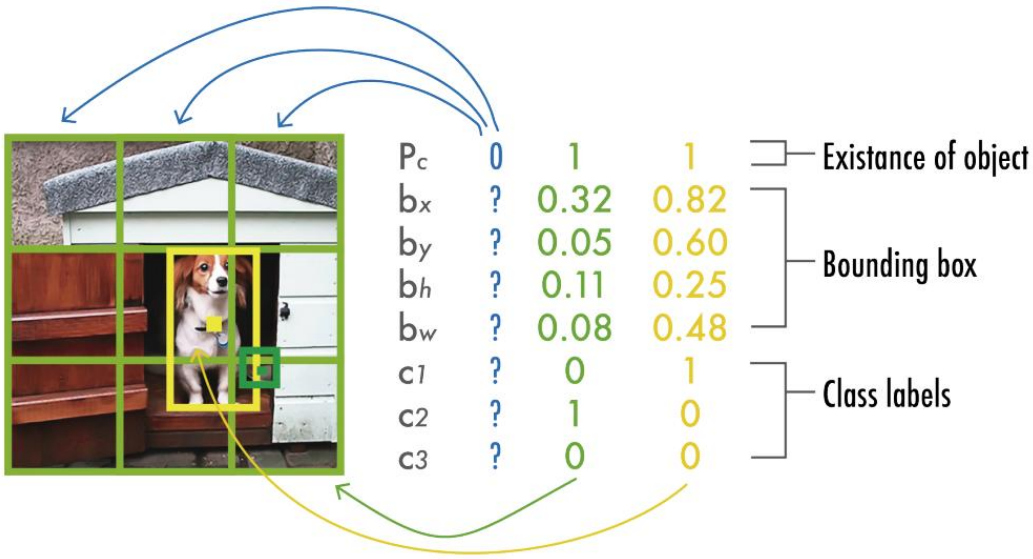


图 5yolo 预测结果

YOLOv1 架构包括 24 个卷积层，然后是两个全连接层，用于预测 bbox 坐标和概率。除了最后一个层使用线性激活函数外，所有层都使用了漏整流线性单元激活[40]。受 GoogLeNet[41]和 Network in Network[42] 的启发，YOLO 使用 1×1

卷积层来减少特征图的数量并保持相对较低的参数数量，作为激活层。

	Type	Filters	Size/Stride	Output
	Conv	64	$7 \times 7 / 2$	224×224
	Max Pool		$2 \times 2 / 2$	112×112
	Conv	192	$3 \times 3 / 1$	112×112
	Max Pool		$2 \times 2 / 2$	56×56
$1\times$	Conv	128	$1 \times 1 / 1$	56×56
	Conv	256	$3 \times 3 / 1$	56×56
	Conv	256	$1 \times 1 / 1$	56×56
	Conv	512	$3 \times 3 / 1$	56×56
	Max Pool		$2 \times 2 / 2$	28×28
$4\times$	Conv	256	$1 \times 1 / 1$	28×28
	Conv	512	$3 \times 3 / 1$	28×28
	Conv	512	$1 \times 1 / 1$	28×28
	Conv	1024	$3 \times 3 / 1$	28×28
	Max Pool		$2 \times 2 / 2$	14×14
$2\times$	Conv	512	$1 \times 1 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 2$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	FC		4096	4096
	Dropout 0.5			4096
	FC		$7 \times 7 \times 30$	$7 \times 7 \times 30$

图 6YOLOv1 的架构

补充：mAP 和 NMS

AP (Average Precision)：平均精度，是一种用来衡量检测器性能的常见指标。在目标检测任务中，AP 是对每个类别的精确度（Precision）进行平均得到的。精确度衡量了检测到的物体中真正为该类别的比例。

mAP（mean average precision）是一个平均值，常用作目标检测中的检测精度指标 mAP 指标通过对于一个平均目标来检测任务中多个目标所对应不同 AP（average precision）值进行计算得到。目前各个经典算法都是使用 mAP 在

开源数据集上进行精度对比。在计算 mAP 之前，还需要使用到两个基础概念：准确率（Precision）和召回率（Recall）

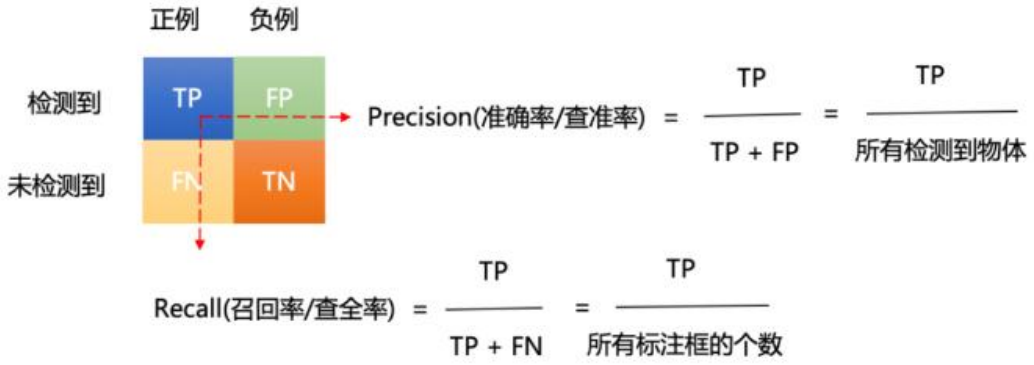


图 7 准确率、召回率

正例：正样本，即该位置存在对应类别的物体。

负例：负样本，即该位置不存在对应类别的物体。

TP（True Positives）：正样本预测为正样本的数量。

FP（False Positives）：负样本预测为正样本的数量。

FN（False Negative）：正样本预测为负样本的数量。

TN（True Negative）：负样本预测为负样本的数量。

IoU：物体检测的目的是通过预测边界框来准确定位图像中的物体。AP 指标包含了 "联合体上的交集"（IoU）措施，以评估预测的边界盒的质量。IoU 是预测界线盒和地面真实界线盒的交集面积与联合面积之比。它衡量的是地面实况和预测边界盒之间的重叠程度。

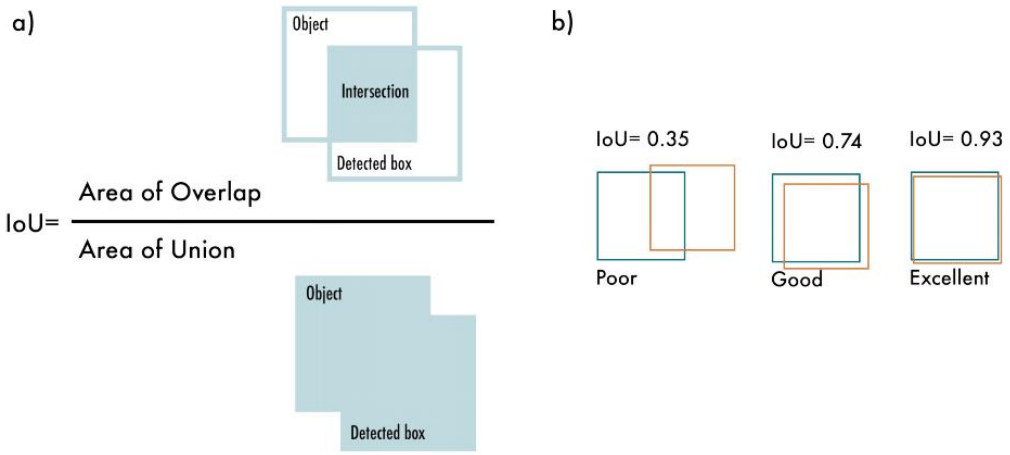


图 8 IOU 计算

mAP0.95：当 mAP 计算时，通常会考虑不同的 IoU（Intersection over Union）

阈值来衡量检测框的重叠程度。mAP0.95 指的是计算 mAP 时，使用 IoU 阈值为 0.95 来作为标准，计算所有类别的平均精度。

NMS（非极大值抑制）

非极大值抑制（NMS）是物体检测算法中使用的一种后处理技术，用于减少重叠边界盒的数量，提高整体检测质量。物体检测算法通常会在同一物体周围产生多个具有不同置信度分数的边界框。NMS 过滤掉多余的和不相关的边界盒，只保留最准确的边界盒。

图 9 NMS（非极大值抑制）

YOLOv2

论文中提到的改进如下：

批量归一化（Batch Normalization）：在所有卷积层上应用批量归一化，改善了模型的收敛性，并作为正则化器减少了过拟合的风险。

高分辨率分类器：YOLOv2 在 ImageNet 上以 224x224 的分辨率预训练，并进行了 10 次微调，以提高在 448x448 分辨率下的性能。

完全卷积架构：YOLOv2 采用完全卷积的结构，去除了密集层，这样可以接受任意尺寸的输入图像。

Anchor 预测边界框：引入 Anchor 机制预测边界框，每个网格单元预测多个 Anchor 的坐标和类别，代替了 YOLOv1 中的网格预测框架。

维度聚类：通过 k-means 聚类选择优化的 Anchor，帮助网络学习准确预测边界框，平衡了召回率和模型复杂性。

直接预测位置：YOLOv2 直接预测边界框相对于网格单元的位置坐标，每个边界框有五个参数来描述位置和置信度。

细粒度特征：YOLOv2 通过去除一个池化层，使得对 416×416 输入图像能够得到更细粒度的 13×13 特征图，有利于捕获更丰富的结构信息。

多尺度训练：由于不使用全连接层，YOLOv2 可以接受不同尺寸的输入，通过随机训练和变化的输入尺寸增强了模型对不同尺寸输入的鲁棒性。

YOLOv2 架构

	Type	Filters	Size/Stride	Output
	Conv	64	7 × 7 / 2	224 × 224
	Max Pool		2 × 2 / 2	112 × 112
	Conv	192	3 × 3 / 1	112 × 112
	Max Pool		2 × 2 / 2	56 × 56
1×	Conv	128	1 × 1 / 1	56 × 56
	Conv	256	3 × 3 / 1	56 × 56
	Conv	256	1 × 1 / 1	56 × 56
	Conv	512	3 × 3 / 1	56 × 56
	Max Pool		2 × 2 / 2	28 × 28
4×	Conv	256	1 × 1 / 1	28 × 28
	Conv	512	3 × 3 / 1	28 × 28
	Conv	512	1 × 1 / 1	28 × 28
	Conv	1024	3 × 3 / 1	28 × 28
	Max Pool		2 × 2 / 2	14 × 14
2×	Conv	512	1 × 1 / 1	14 × 14
	Conv	1024	3 × 3 / 1	14 × 14
	Conv	1024	3 × 3 / 1	14 × 14
	Conv	1024	3 × 3 / 2	7 × 7
	Conv	1024	3 × 3 / 1	7 × 7
	Conv	1024	3 × 3 / 1	7 × 7
	FC		4096	4096
	Dropout 0.5			4096
	FC		7 × 7 × 30	7 × 7 × 30

图 10 YOLOv2 架构

YOLOv2 使用的骨干架构被称为 Darknet-19，包含 19 个卷积层和 5 个 maxpooling 层。与 YOLOv1 的架构类似，它受到 Network in Network[42]的启发，在 3×3 之间使用 1×1 3×3 之间的卷积，以减少参数的数量。此外，如上所述，他们使用批量归一化来规范化并帮助收敛。

YOLOv3

YOLOv3 在边界框预测和分类方面进行了多项改进和优化，主要特点如下：

边界框预测:

YOLOv3 为每个 bounding box 预测四个坐标参数 tx,ty,tw,th，分别表示中心坐标和宽高。

每个 bounding box 还预测一个目标分数，这个分数表示该 bounding box 与 ground truth 的重合度。对于与 ground truth 重合度最高的 Anchor，分数为 1，对于其他 Anchor 为 0。这种设计与 Faster R-CNN 相比，YOLOv3 仅为每个 ground truth 分配一个 Anchor。

类别预测:

YOLOv3 采用二元交叉熵来训练独立的 logistic 分类器,而不是使用 softmax 进行分类。这种变化使得每个 bounding box 可以分配多个标签，适用于数据集中有标签重叠或者复杂标签情况下，例如一个物体同时属于人和男人两个类别。

新的骨干网络:

YOLOv3 使用一个更大的特征提取器，包含 53 个卷积层，并且引入了 Res 残余连接，以提升特征提取的效果和模型性能。

空间金字塔池（SPP）:

YOLOv3 引入了改进的 SPP 块，连接了多个不同大小的最大池化输出，每个池化核大小为 1×1、5×5、9×9、13×13，这样的设计增加了感受野，有助于模型在不同尺度上更好地捕获特征。

多尺度预测:

类似于特征金字塔网络，YOLOv3 在三个不同的尺度上进行预测，以适应不同大小物体的检测需求。

Bounding box 先验:

YOLOv3 通过 k-means 聚类确定了三个不同尺度的 Anchor 预设，与 YOLOv2 相比减少了 Anchor 的数量，但提升了效率和性能。

YOLOv3 架构

Num	Type	Filters	Size/Stride	Output
1	Conv/BN	32	3 × 3 / 1	416 × 416 × 32
2	Max Pool		2 × 2 / 2	208 × 208 × 32
3	Conv/BN	64	3 × 3 / 1	208 × 208 × 64
4	Max Pool		2 × 2 / 2	104 × 104 × 64
5	Conv/BN	128	3 × 3 / 1	104 × 104 × 128
6	Conv/BN	64	1 × 1 / 1	104 × 104 × 64
7	Conv/BN	128	3 × 3 / 1	104 × 104 × 128
8	Max Pool		2 × 2 / 2	52 × 52 × 128
9	Conv/BN	256	3 × 3 / 1	52 × 52 × 256
10	Conv/BN	128	1 × 1 / 1	52 × 52 × 128
11	Conv/BN	256	3 × 3 / 1	52 × 52 × 256
12	Max Pool		2 × 2 / 2	52 × 52 × 256
13	Conv/BN	512	3 × 3 / 1	26 × 26 × 512
14	Conv/BN	256	1 × 1 / 1	26 × 26 × 256
15	Conv/BN	512	3 × 3 / 1	26 × 26 × 512
16	Conv/BN	256	1 × 1 / 1	26 × 26 × 256
17	Conv/BN	512	3 × 3 / 1	26 × 26 × 512
18	Max Pool		2 × 2 / 2	13 × 13 × 512
19	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
20	Conv/BN	512	1 × 1 / 1	13 × 13 × 512
21	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
22	Conv/BN	512	1 × 1 / 1	13 × 13 × 512
23	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
24	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
25	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
26	Reorg layer 17			13 × 13 × 2048
27	Concat 25 and 26			13 × 13 × 3072
28	Conv/BN	1024	3 × 3 / 1	13 × 13 × 1024
29	Conv	125	1 × 1 / 1	13 × 13 × 125

图 11YOLOv3 架构

YOLOv3 中提出的架构主干被称为 Darknet-53。它用全连接层取代了所有的 max-pooling 层，并增加了 Res 残差连接。总的来说，它包含 53 个卷积层。下图显示了该架构的细节。

Layer	Filters	size	Repeat		Output size
Image					416 × 416
Conv	32	3 × 3/1	1		416 × 416
Conv	64	3 × 3/2	1		208 × 208
Conv	32	1 × 1/1	Conv		208 × 208
Conv	64	3 × 3/1	Conv	×1	208 × 208
Residual			Residual		208 × 208
Conv	128	3 × 3/2	1		104 × 104
Conv	64	1 × 1/1	Conv		104 × 104
Conv	128	3 × 3/1	Conv	×2	104 × 104
Residual			Residual		104 × 104
Conv	256	3 × 3/2	1		52 × 52
Conv	128	1 × 1/1	Conv		52 × 52
Conv	256	3 × 3/1	Conv	×8	52 × 52
Residual			Residual		52 × 52
Conv	512	3 × 3/2	1		26 × 26
Conv	256	1 × 1/1	Conv		26 × 26
Conv	512	3 × 3/1	Conv	×8	26 × 26
Residual			Residual		26 × 26
Conv	1024	3 × 3/2	1		13 × 13
Conv	512	1 × 1/1	Conv		13 × 13
Conv	1024	3 × 3/1	Conv	×4	13 × 13
Residual			Residual		13 × 13

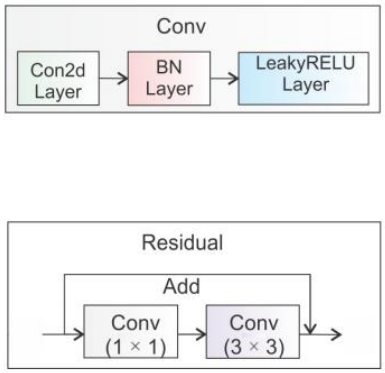


图 12 YOLOv3 模型

YOLOv3 多尺度预测

除了更大的结构，YOLOv3 的一个基本特征是多尺度预测，即在多个网格尺寸下的预测。这有助于获得更精细的方框，并大大改善了对小物体的预测，而这正是 YOLO 以前版本的主要弱点之一。

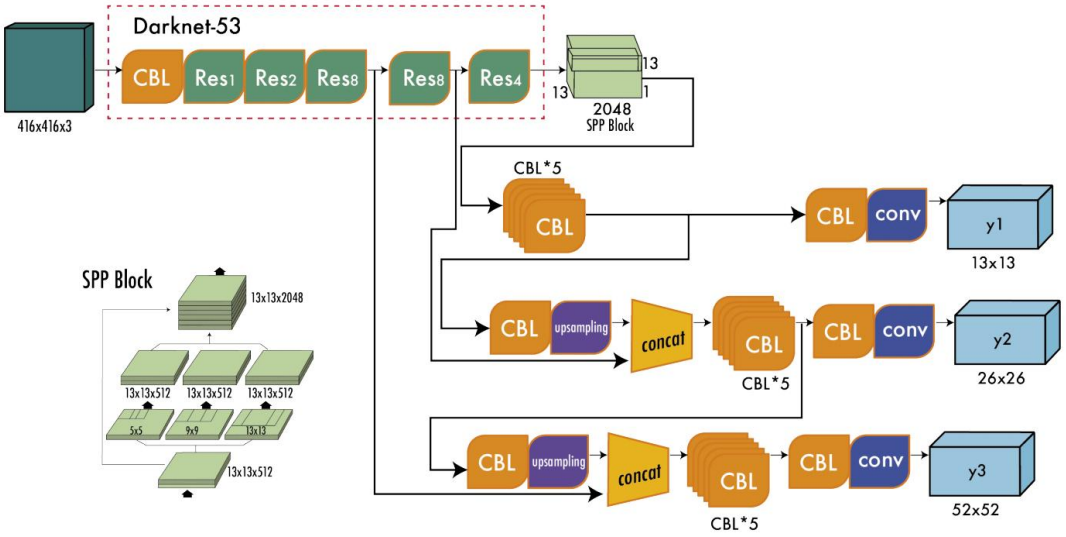


图 13 多尺度预测

Backbone，Neck 和 Head

这时，物体检测器的结构开始被描述为三个部分：Backbone，Neck 和 Head。

下图显示了一个高层次的 Backbone，Neck 和 Head 图。

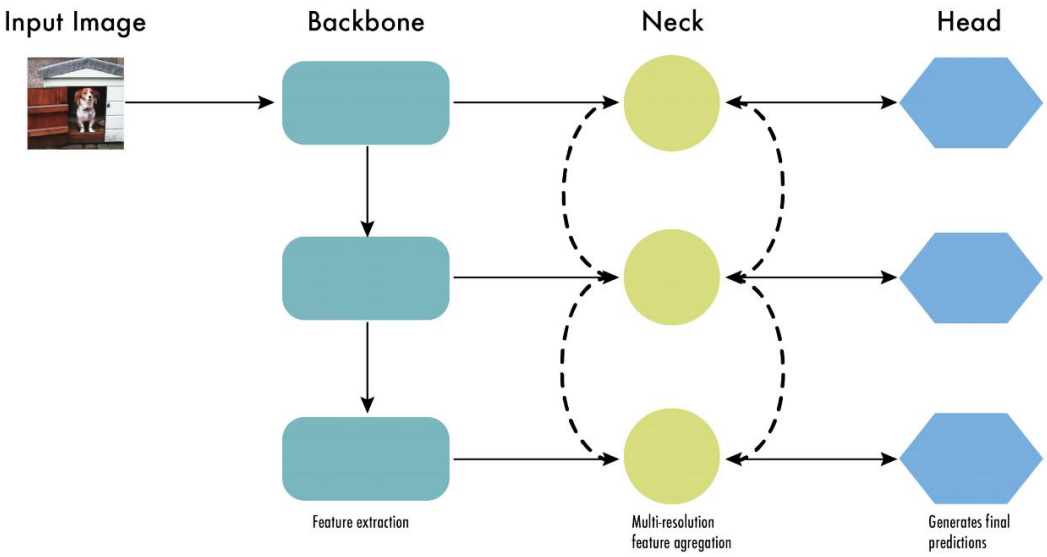


图 14 Backbone, Neck 和 Head

Backbone 负责从输入图像中提取有用的特征。它通常是一个卷积神经网络（CNN），在大规模的图像分类任务中训练，如 ImageNet。骨干网在不同的尺度上捕捉层次化的特征，在较早的层中提取低层次的特征（如边缘和纹理），在较深的层中提取高层次的特征（如物体部分和语义信息）。

Neck 是连接 Backbone 和 Head 的一个中间部件。它聚集并细化骨干网提取的特征，通常侧重于加强不同尺度的空间和语义信息。颈部可能包括额外的卷积层、特征金字塔网络（FPN）[49]，或其他机制，以提高特征的代表性。

Head 是物体检测器的最后组成部分；它负责根据 Backbone 和 Neck 提供的特征进行预测。它通常由一个或多个特定任务的子网络组成，执行分类、定位，以及最近的实例分割和姿势估计。头部处理颈部提供的特征，为每个候选物体产生预测。最后，一个后处理步骤，如非极大值抑制（NMS），过滤掉重叠的预测，只保留置信度最高的检测。

在之后的 YOLO 模型中，我们将使用 Backbone，Neck 和 Head 来描述架构。

YOLOv4

YOLOv4 引入了多项关键变化和优化，以提升目标检测的性能和效率：

增强型架构:

采用跨阶段部分连接（CSPNet）改进的 Darknet-53 作为主骨干架构，结合 Mish 激活函数。

使用修改版的空间金字塔集合（SPP）和路径聚合网络（PANet），以及修改的空间注意模块（SAM）来增强特征提取和多尺度预测能力。

这些改进使得模型被称为 CSPDarknet53-PANet-SPP，旨在在减少计算量的同时保持高精度。

高级训练方法（bag-of-freebies）:

引入了马赛克增强，将四张图像合并为一张，有助于检测复杂背景中的物体。

实现了 DropBlock 作为替代 Dropout 的正则化方法，以及类标签平滑技术。

引入了 CIoU 损失和 Cross mini-batch normalization（CmBN），后者用于收集整个批次的统计数据，提升训练稳定性和效果。

自我对抗性训练（SAT）:

通过对输入图像进行对抗性攻击，使模型对于扰动更加稳健，这有助于在真实世界的复杂情况下保持检测准确性。

超参数优化:

使用遗传算法进行超参数优化，寻找最佳训练参数组合。

结合余弦退火调度器来调整学习率，以优化训练过程中的模型收敛速度和性能。

Yolov4 框架

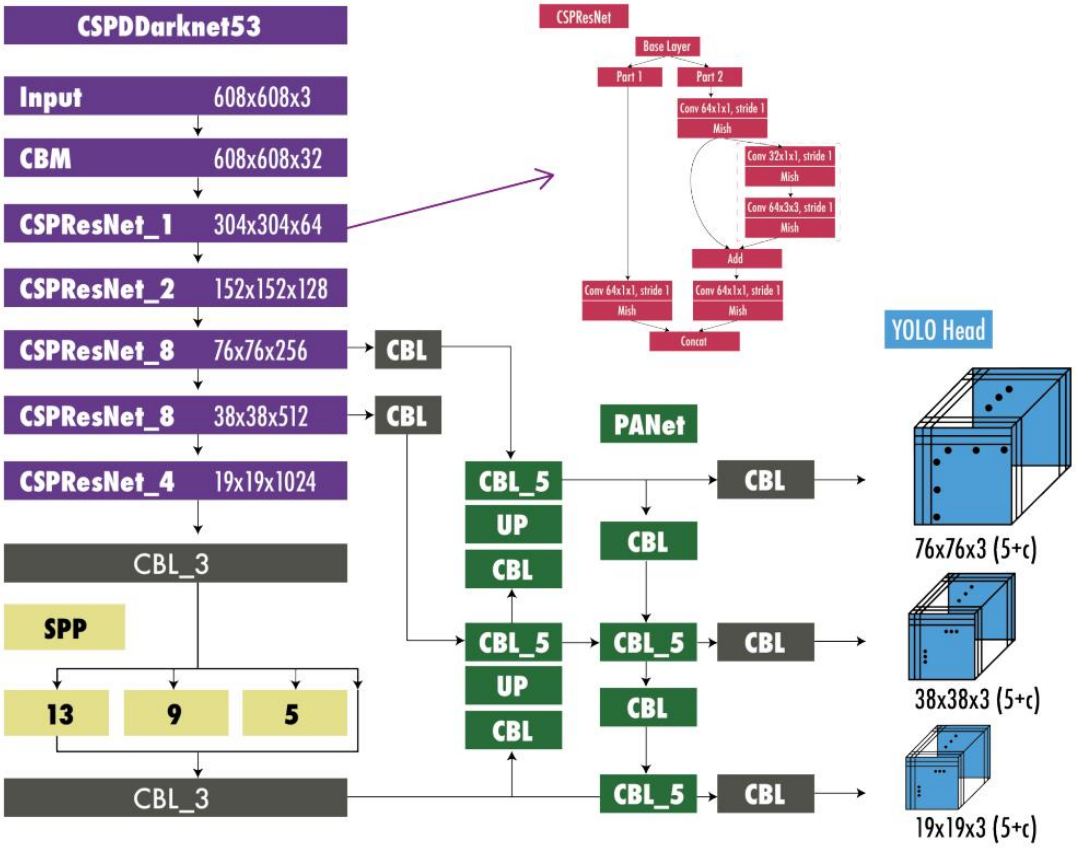


图 15 Yolov4 框架

YOLOv5

YOLOv5[72]是在 YOLOv4 之后几个月于 2020 年由 Glenn Jocher 发布。在写这篇文章时，还没有关于 YOLOv5 的科学论文，但从代码中，我们知道它使用了 YOLOv4 部分描述的许多改进，主要区别是它是用 Pytorch 而不是 Darknet 开发的。YOLOv5 是开源的，由 Ultralytics 积极维护，有 250 多个贡献者，并经常有新的改进。YOLOv5 很容易使用、培训和部署。Ultralytics 提供了一个 iOS 和 Android 的移动版本，以及许多用于标签、培训和部署的集成。

YOLOv5 提供了五个版本： YOLOv5n（ 纳米级）、YOLOv5s（ 小型）、YOLOv5m（ 中型）、YOLOv51（大型）和 YOLOv5x（特大型）。

YOLOX 相对于 YOLOv3 引入了几个关键变化和优化，以提升目标检测性能和效率：

无锚结构（Anchor-free）：

YOLOX 放弃了传统的锚点（Anchor）结构，转而采用无锚检测器的设计。这种结构简化了训练和解码过程，受到 CornerNet、CenterNet 和 FCOS 等先进无锚检测器的启发。相较于 YOLOv3，无锚结构使得平均精度（AP）提升了 0.9 个百分点。

多重正样本（Multi positives）：

为了解决无锚结构导致的正负样本不平衡问题，YOLOX 引入了中心采样方法，将中心 3×3 的区域作为正例区域。这种方法显著提高了平均精度（AP），增加了 2.1 个百分点。

解耦头（Decoupled head）：

YOLOX 将分类置信度和位置精度分离成两个独立的任务头部。这种解耦可以提升模型的训练和收敛速度，并且提高了平均精度（AP）1.1 个百分点。

高级标签分配（Advanced label assignment）：

受最佳传输（OT）问题的启发，YOLOX 提出了简化的标签分配算法 simOTA。这种方法处理多个对象的 bounding box 重叠时的标签分配模糊性，显著提高了平均精度（AP），增加了 2.3 个百分点。

强化增强（Stronger augmentation）：

YOLOX 引入了 MixUP 和 Mosaic 增强技术，这些技术在训练过程中有助于提升模型的泛化能力和鲁棒性。使用这些增强技术后，不再需要 ImageNet 预训练，平均精度（AP）提高了 2.4 个百分点。

YOLOv5 框架：

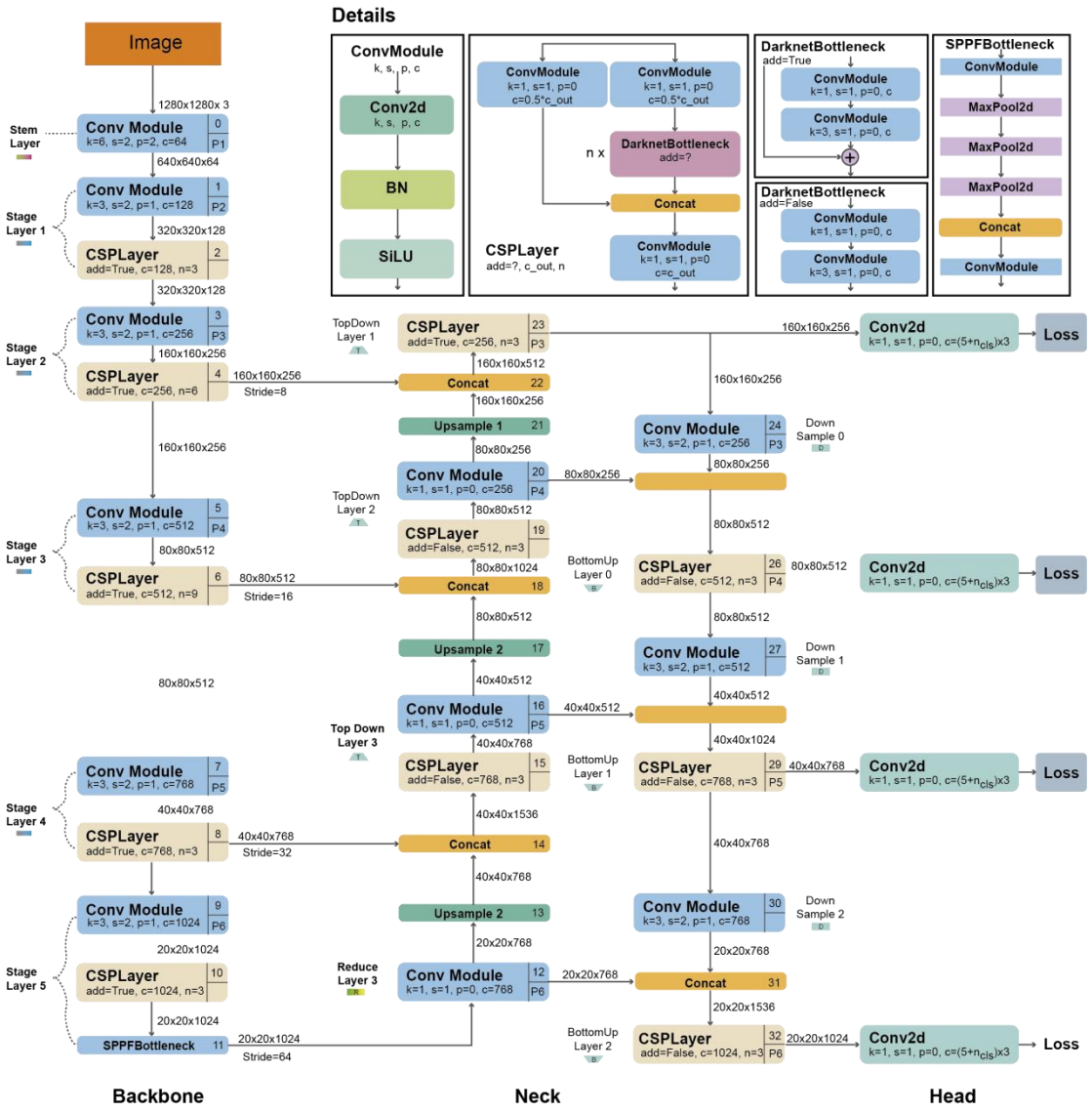


图 16 YOLOv5 框架

YOLOv6

YOLOv6 引入了多项创新，建立在无锚点检测器的基础上，以提升目标检测性能和效率：

EfficientRep 骨架：

YOLOv6 采用了一个新的骨干网络称为 EfficientRep，基于 RepVGG，这个骨

干网络具有更高的并行性和效率。对于大型模型，它使用了 PAN（Path Aggregation Network）来增强 RepBlocks 或 CSPStackRepBlocks。这些改进旨在提升模型的特征提取能力和计算效率。

高效的解耦头:

受到 YOLOX 的启发，YOLOv6 开发了一个高效的解耦头，将分类任务和回归任务分开处理。这种解耦能够更好地优化分类置信度和位置回归的准确性。

任务排列学习方法（TOOD）:

YOLOv6 使用了 TOOD 介绍的任务排列学习方法进行标签分配。这种方法能够更有效地处理多个对象的 bounding box 重叠问题，提高了标签分配的精度和一致性。

新的损失函数:

YOLOv6 采用了新的分类损失函数 VariFocal 损失和回归损失函数 SIoU/GIoU。这些损失函数设计旨在优化模型在目标检测任务中的性能，特别是在处理不均衡和复杂场景时表现更优。

自我蒸馏策略:

为了进一步提升模型的泛化能力和鲁棒性，YOLOv6 引入了自我蒸馏策略，用于分类和回归任务。这种策略通过利用模型自身的输出来调整训练过程，从而提升模型的性能和稳定性。

检测量化方案:

YOLOv6 框架：

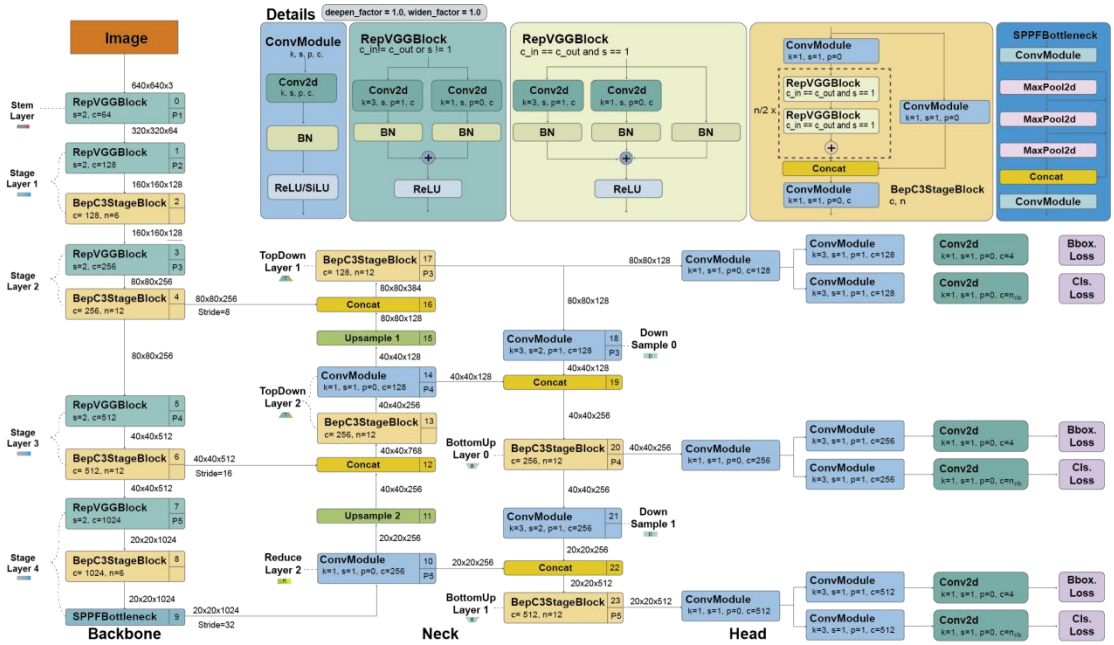


图 17 YOLOv6 框架

YOLOv7

YOLOv7 的架构变化和采用的技术总结如下：

扩展高效层聚合网络（E-ELAN）：

YOLOv7 引入了 E-ELAN，扩展自高效层聚合网络（ELAN），旨在通过控制梯度路径的长度来增强深度模型的学习效率和收敛性。E-ELAN 通过特征洗牌和合并不同组的特征，增强网络的学习能力，同时不破坏原始的梯度路径。

基于串联的模型缩放：

YOLOv7 采用了一种基于串联的模型缩放策略，与传统的深度缩放不同，它保持了块的深度和宽度的比例，以保持模型的最佳结构。这种方法避免了因为输入通道和输出通道比例变化而导致的硬件资源使用量增加。

bag-of-freebies 中的技术应用：

重新参数化卷积（RepConvN）：受 RepConv 启发，YOLOv7 引入了重新参数化卷积，但移除了身份连接，以避免破坏残差和串联结构的优势。

辅助头和主导头的标签分配：YOLOv7 对辅助头进行粗略的标签分配，而对主导头进行更精细的标签分配，从而更有效地训练模型。

conv-bn-activation 中的批量归一化：这一技术将批量归一化的统计信息整合到卷积层的推理阶段，提高了模型的推断效率。

隐性知识：借鉴 YOLOR 中的隐性知识，帮助提升模型的性能和泛化能力。

指数移动平均线作为最终推断模型：采用指数移动平均线作为最终的推断模型，有助于提升模型的稳定性和准确性。

这些创新和技术应用使得 YOLOv7 在目标检测领域取得了显著的进展和优化。

YOLOv7 框架：

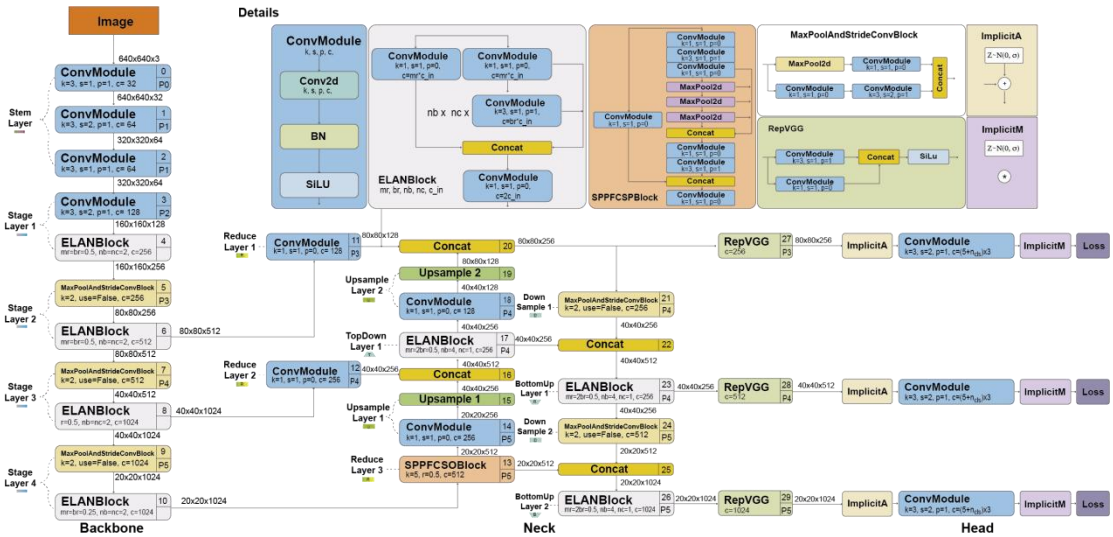


图 18 YOLOv7 框架

YOLOv8

YOLOv8 模型框架如下：

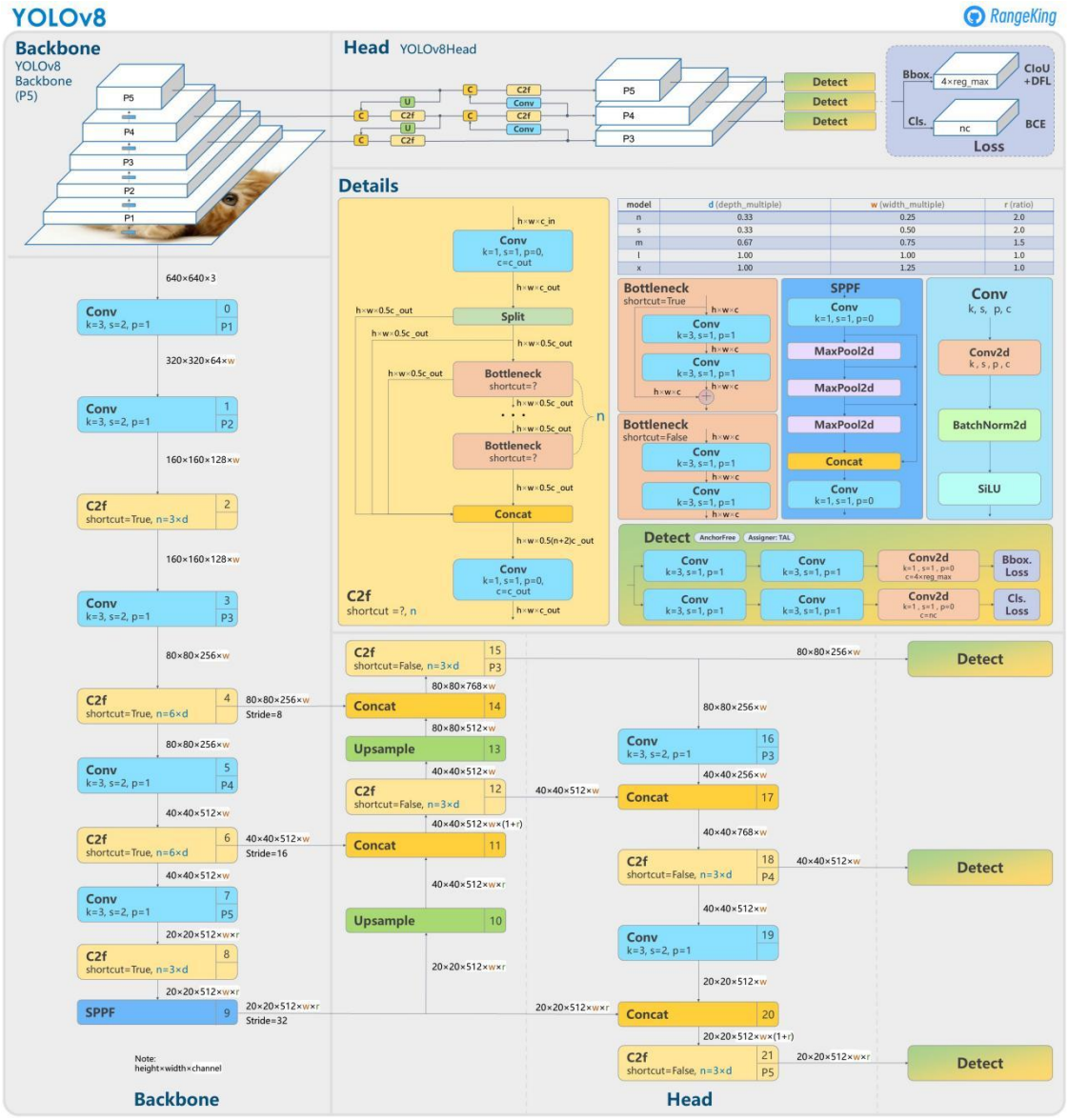


图 19 YOLOv8 模型框架

Backbone 和 Neck 的具体变化

a) 第一个卷积层的 kernel 从 6x6 变成了 3x3

b) 所有的 C3 模块换成 C2f，结构如下所示，可以发现多了更多的跳层连接和额外的 Split 操作

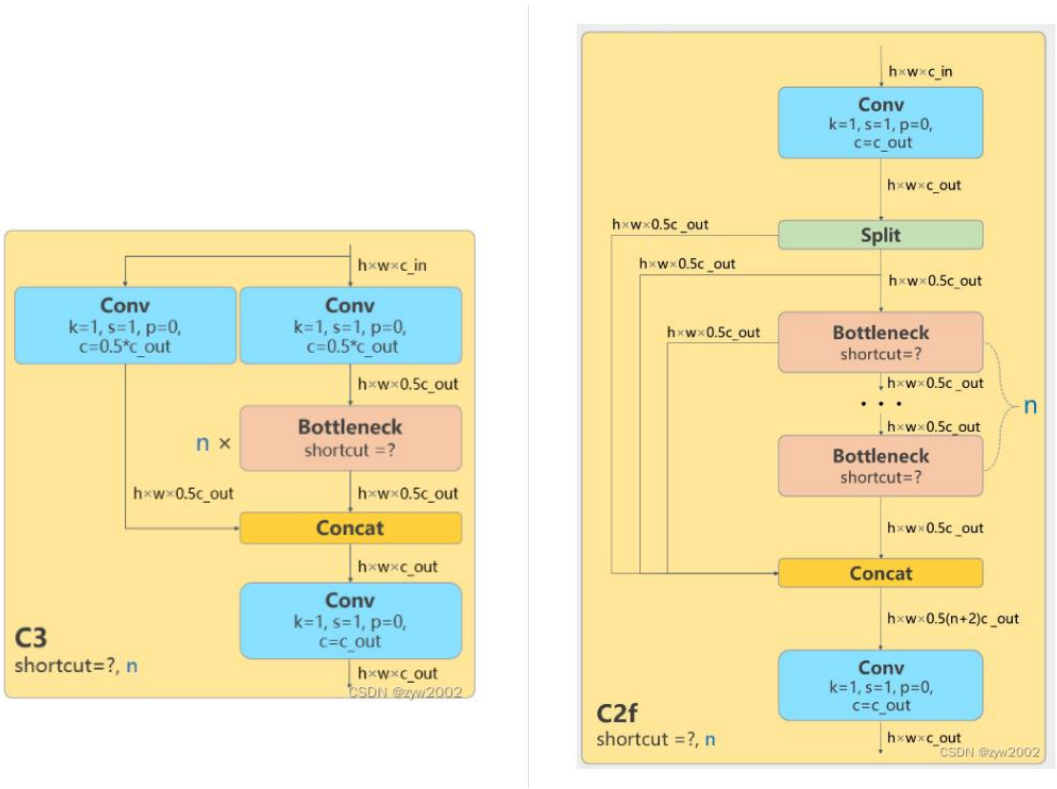


图 20 Backbone 和 Neck 的具体变化

c)去掉了 Neck 模块中的 2 个卷积连接层

d) Backbone 中 C2f 的 block 数从 3-6-9-3 改成了 3-6-6-3

e) 查看 N/S/M/L/X 等不同大小模型，可以发现 N/S 和 L/X 两组模型只是改了缩放系数，但是 S/M/L 等骨干网络的通道数设置不一样，没有遵循同一套缩放系数。如此设计的原因应该是同一套缩放系数下的通道设置不是最优设计，YOLOv7 网络设计时也没有遵循一套缩放系数作用于所有模型

Head 的具体变化

从原先的耦合头变成了解耦头，并且从 YOLOv5 的 Anchor-Based 变成了 Anchor-Free。

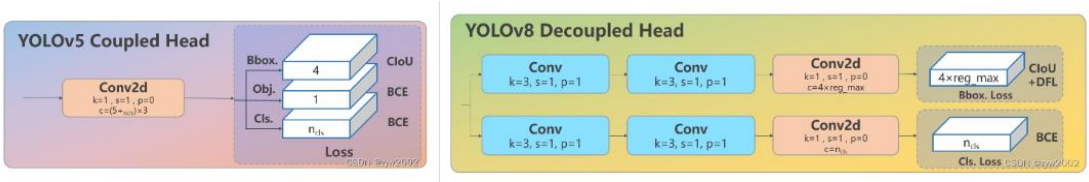


图 21 Head 的具体变化

从上图可以看出，不再有之前的 objectness 分支，只有解耦的分类和回归

分支，并且其回归分支使用了 Distribution Focal Loss 中提出的积分形式表示法。

模型推理过程

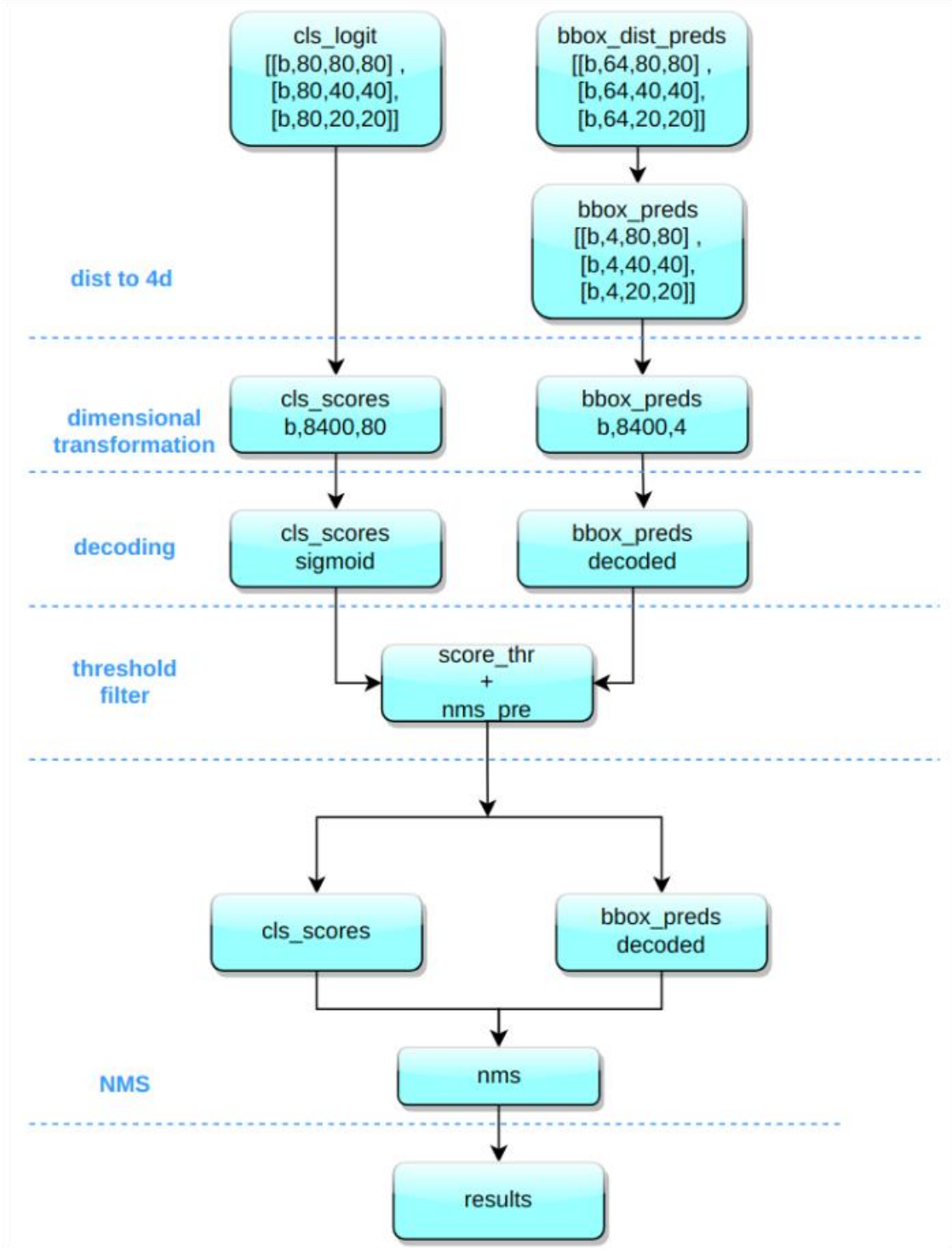


图 22 模型推理过程

YOLOV10

论文： https://arxiv.org/pdf/2405.14458

代码： GitHub - THU-MIG/yolov10: YOLOv10: Real-Time End-to-End Object Detection

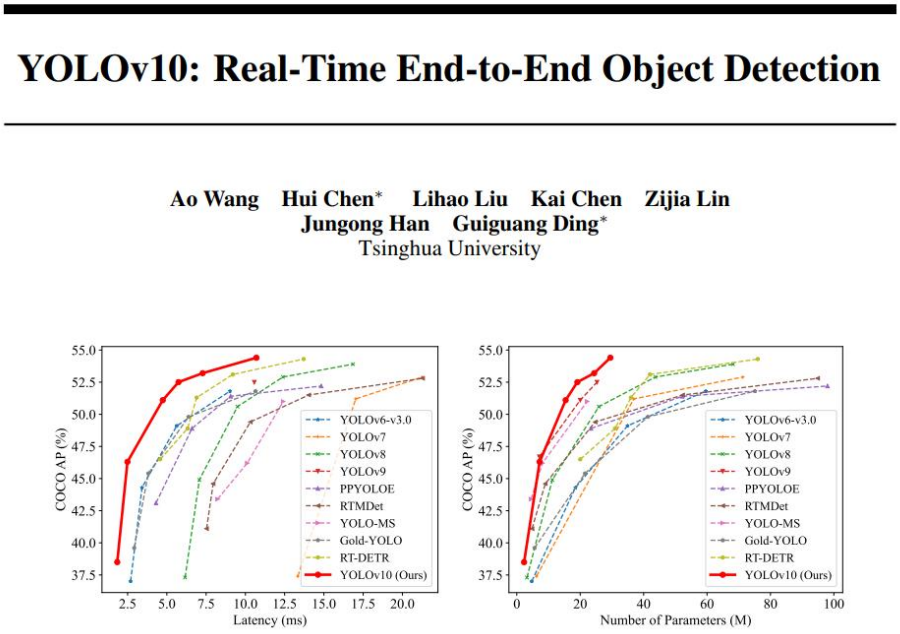


图 23 YOLOV10

读完摘要部分，简单概括就是用于后处理的非最大抑制（NMS）的依赖妨碍了 YOLOS 的端到端部署，并且影响了推理延迟。首先提出了用于 YOLOs 无 NMS 训练的持续双重分配。 模型的框架图如下。

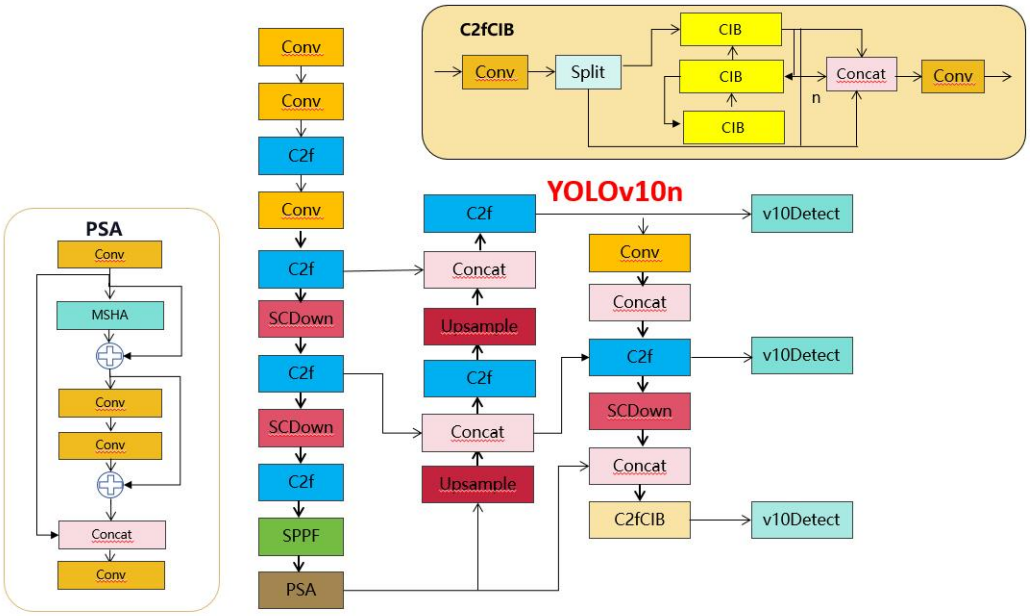


图 24 模型的框架图

论文中主要提出了三个创新的结构分别是 C2fUIB、PSA、SCDown，下面分别来介绍一下这三个主要的创新模块 。

C2fCIB 介绍

C2fCIB 只是用 CIB 结构替换了 YOLOv8 中 C2f 的 Bottleneck 结构。

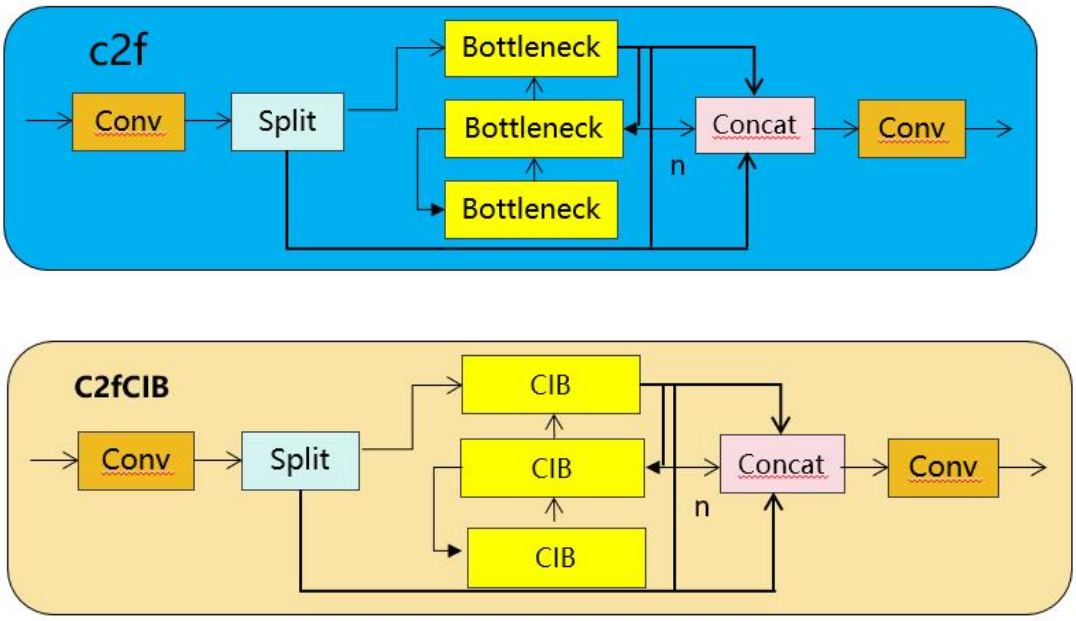


图 25 C2fCIB 介绍

PSA 介绍

具体来说，我们在 1×1 卷积后将特征均匀地分为两部分。只将一部分输入到由多头自注意力模块（MHSA）和前馈网络（FFN）组成的 NPSA 块中。然后，两部分通过 1×1 卷积连接并融合。此外，遵循将查询和键的维度分配为值的一半，并用 BatchNorm 替换 LayerNorm 以实现快速推理。

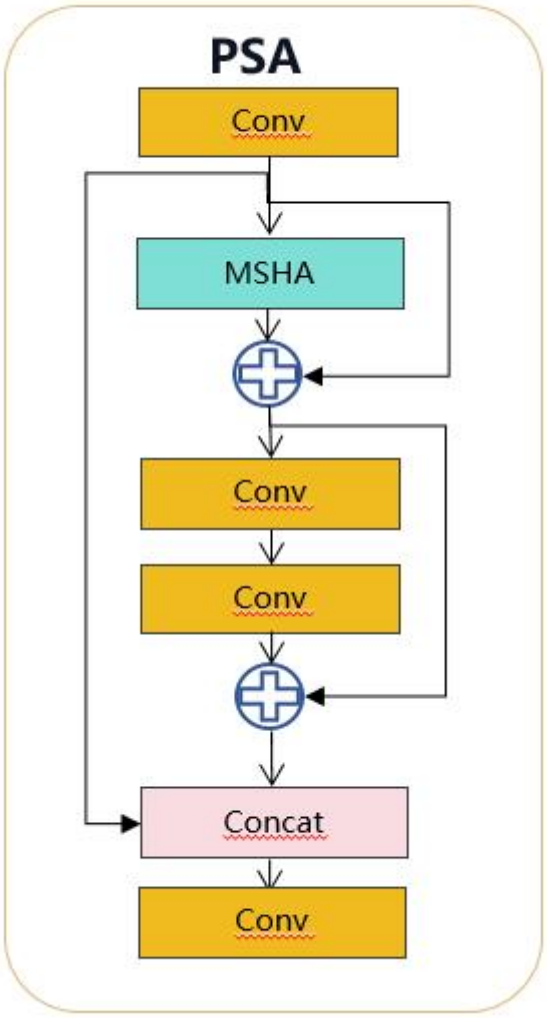


图 26 PSA 介绍

SCDown 介绍

OLOs 通常利用常规的 3×3 标准卷积，步长为 2，同时实现空间下采样（从 H×W 到 H/2×W/2）和通道变换（从 C 到 2C）。这引入了不可忽视的计算成本 O(9HWC^2)和参数数量 O(18C^2)。相反，我们提议将空间缩减和通道增加操作解耦，以实现更高效的下采样。具体来说，我们首先利用点对点卷积来调整通道维

度，然后利用深度可分离卷积进行空间下采样。这将计算成本降低到 O(2HWC^2 + 9HWC)，并将参数数量减少到 O(2C^2 + 18C)。同时，它最大限度地保留了下采样过程中的信息，从而在减少延迟的同时保持了有竞争力的性能。

参考 https://cloud.tencent.com/developer/article/2426044

简介如下：

时间	版本	主要内容	作者
2015	v1	Joseph Redmon & Ali Farhadi	Joseph Redmon & Ali Farhadi
2016	v2	加入 batch normalization, anchor boxes 和 dimension clusters.	Joseph Redmon & Ali Farhadi
2018	v3	更高效的骨干网络、引入 spatial pyramid 和 PAN	Joseph Redmon & Ali Farhadi
2020	v4	Mosaic 数据增强、new loss、SPP 模块、CSP 模块、Mish 激活函数、CmBN 归一化	Alexey Bochkovskiy
2020	v5	灵活的配置参数、超参优化策略、focus 结构	XXXX
2021	x	Decoupled Head、anchor free	旷视

2022	v6	引入 RepConv，量化相关内容，Loss	美团
2022	v7	Efficient Layer Aggregation Network (ELAN) 模块，MP 模块	Alexey Bochkovskiy & 台湾中央研究院
XXXX	v8	C2f 模块	Ultralytics
2024	v9	加入 Programmable Gradient Information (PGI) and Generalized Efficient Layer Aggregation Network (GELAN)	XXXX
2024	v10	Dual label assignments、整体高效的网络设计、空间-通道解耦下采样	清华

表 1 YOLO 发展总结

3.2 RCNN->FastRCNN->FasterRCNN

RCNN

R-CNN 系列论文（R-CNN,fast-RCNN,faster-RCNN）是使用深度学习进行物体检测的鼻祖论文，其中 fast-RCNN 以及 faster-RCNN 都是沿袭 R-CNN 的思路。

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

图 27 RCNN

我之前是有 CNN 的基础的。读完 RCNN 的论文，可以总结 RCN。用 CNN 提取出 Region Proposals 中的 featues，然后进行 SVM 分类与 bbox 的回归。

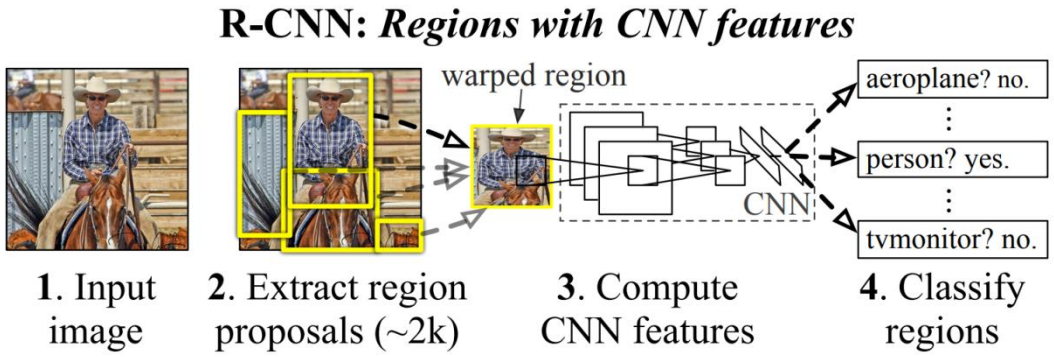


图 28R-CNN 目标检测算法流程

如上图所示，R-CNN 目标检测算法流程主要分四个步骤：

1.采用 SS(Selective Search)方法，使一张图片生成 1000~2000 个候选区域(region proposals)。

2.对每一个候选区域，都使用深度神经网络（AlexNet）提取特征，得到 1*4096 的特征向量。

3.将每一个特征向量送入每一类的 SVM 分类器，判断是否属于该类。

4.对已分类的推荐框进行线性回归，对这些框进行精细地调整，得到更加准确的边界框坐标。

区域推荐：

指的是为了目标检测而提出的候选区域，使得后续的目标检测算法可以专注于这些可能包含目标的区域，而不必检查整张图像。作者提到了几种流行的区域推荐方法，如物体性（objectness）、选择性搜索（selective search）、类别无关物体推荐（category-independent object proposals）等，以及一些特定应用场景下的方法。

特征提取：

描述了使用深度学习中的卷积神经网络（CNN）来提取区域推荐中每个候选区域的特征向量。文中使用了基于 Caffe 实现的 AlexNet，通过多层卷积和全连接层，将每个候选区域的图像转换为一个 4096 维的特征向量。这些特征向量可以被后续的目标检测算法使用来识别图像中的物体。

图像预处理：

为了将不同尺寸和比例的候选区域输入到 CNN 中，文中描述了简单的图像转换方法，即在候选区域周围加上 16 像素的填充，并进行各向异性缩放，以适应 CNN 的固定输入大小（277x277 像素）。这种预处理方法可以提高检测性能（mAP 提高了 3 到 5 个百分点）。

FastRCNN

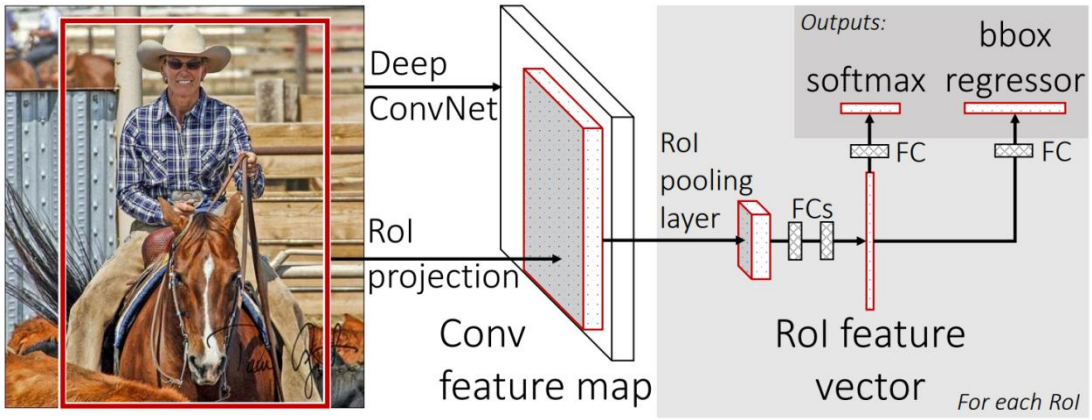


图 29Fast R-CNN 算法的流程

如上图所示，Fast R-CNN 算法的流程主要分为下面三个步骤：

1.依然先使用SS(Selective Search)方法，使一张图片生成1000~2000个候选区域。
2.将图像输入到一个CNN（VGG-16）得到相应的特征图，然后将已经生成的候选框投影到特征图上获得相应的特征矩阵。
3.将每个特征矩阵通过ROI Pooling层缩放到7*7大小，然后将特征图展平，在通过一系列全连接层得到预测的类别信息和目标边界框信息。

Fast R-CNN在改进和优化R-CNN的过程中引入了几个关键的创新点：

ROI Pooling层：

Fast R-CNN 提出了ROI (Region of Interest) Pooling层，用于解决R-CNN中特征重复提取的问题。在R-CNN中，每个候选区域都需要单独送入CNN提取特征，这导致了冗余的计算。相比之下，Fast R-CNN通过在整个图像上进行一次CNN的前向传播，得到一个特征图（feature map），然后将每个候选区域映射到这个特征图上，并利用ROI Pooling层将每个候选区域池化为固定大小的特征向量（通常是7x7大小）。这样一来，每个候选区域只需要进行一次特征提取，大大提高了计算效率。

整合多个模型：

在R-CNN中，需要分别训练用于提取候选区域特征的CNN模型、用于分类的SVM模型，以及用于回归边界框的回归器。这使得训练过程复杂且耗时。Fast R-CNN将这三个模块整合到一个统一的网络结构中，即用于特征提取、分类和回归的部分都包含在一个网络中。这种整合简化了训练流程，提高了训练的速度和效率。

Faster RCNN

2015年，由任少卿、何凯明、Ross Girshick、孙剑组成的微软研究团队，提出了Region Proposal Networks取代了原来的SS算法，几乎不消耗计算时间，

使得生成推荐区域过程变得非常高效，解决了 Fast R-CNN 的瓶颈问题。

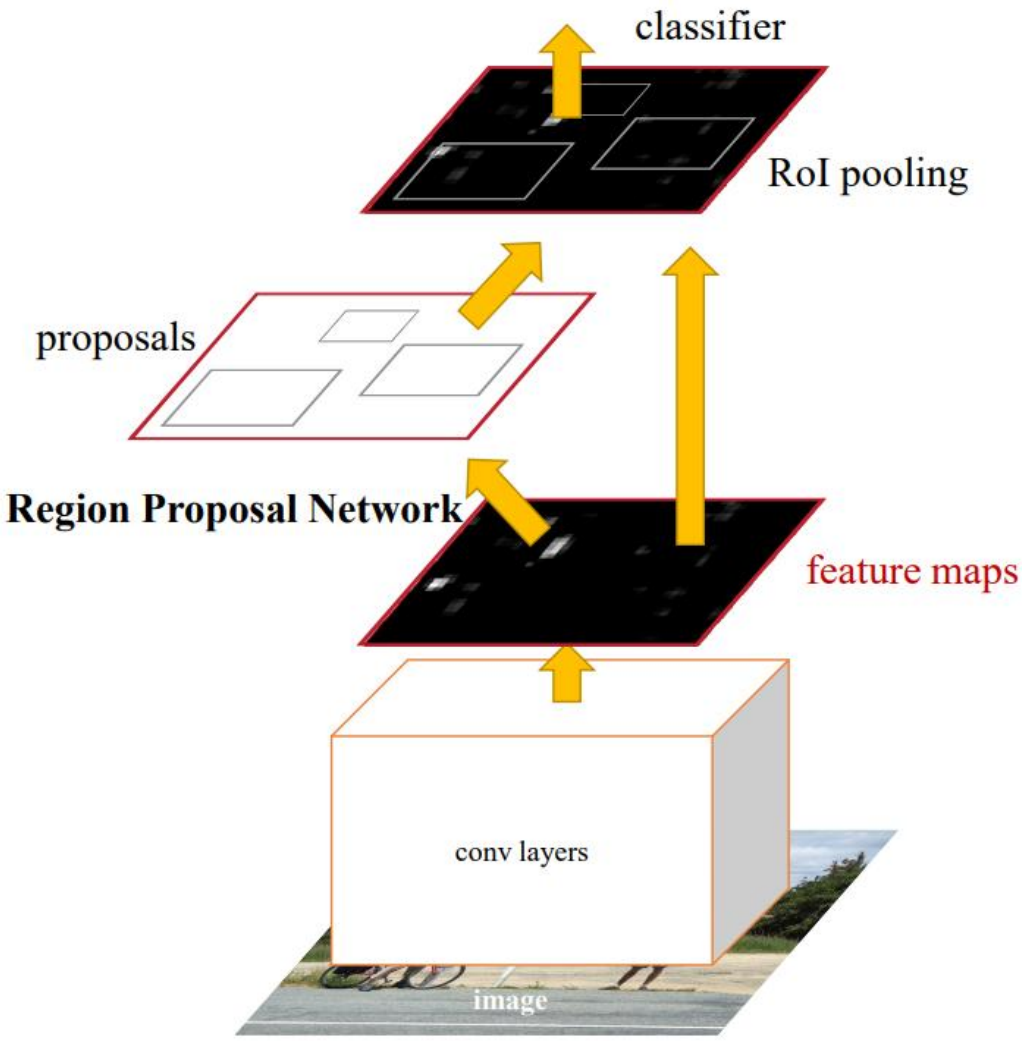


图 30 Faster R-CNN 算法流程

如上图所示，Faster R-CNN 算法流程主要有以下 4 个步骤：

1.Conv layers。首先将图像输入到 CNN（VGG-16）提取图像特征，得到的 feature maps 将被共享用于后面的 RPN 和 ROI Pooling。

2.Region Proposal Networks。RPN 用于生成推荐区域。该网络通过 softmax 判断 anchors 属于 positive 还是 negative，再利用边界框回归修正 anchors 获得精确的推荐框 proposals。

3.ROI Pooling。该层以 feature maps 和 proposals 同时作为输入，综合这些信息后提取 proposal feature maps，送入后续全连接层判定目标类别。

4.Classifer。将 proposal feature maps 输入全连接层与预测 proposals

的类别；同时再次进行边界框回归，获得检测框最终的精确位置。

Region Proposal Networks：

区域推荐网络（RPN）接收（任意大小的）图像作为输入，并输出一组矩形的目标推荐框，每个目标推荐框都有一个客观评分。我们使用全卷积网络对该过程进行建模，我们将在本节中对其进行描述。因为我们的最终目标是与 Fast R-CNN 目标检测网络共享计算，所以我们假设两个网络共享一组共同的卷积层。在我们的实验中，我们研究了具有 5 个可共享卷积层的 Zeiler 和 Fergus 模型（ZF）和具有 13 个可共享卷积层的 Simonyan 和 Zisserman 模型（VGG-16）。

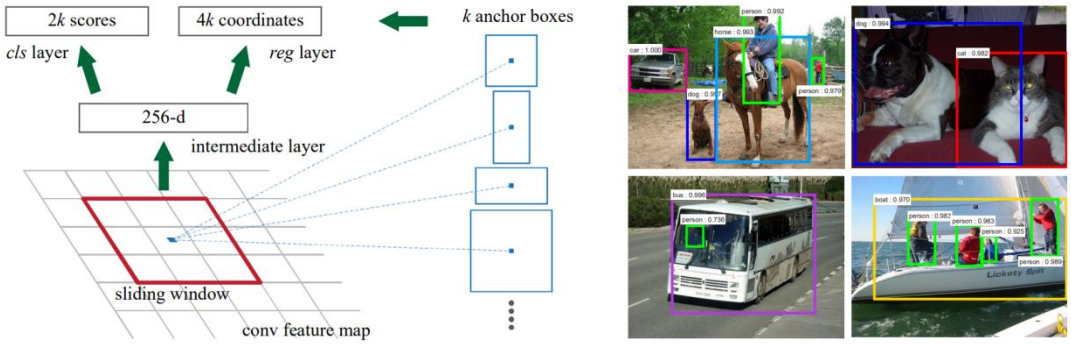


图 31 推荐区域

为了生成推荐区域，我们在最后共享的卷积层输出的卷积特征图上滑动一个小型网络。这个小网络将输入卷积特征图的 n×n 空间窗口作为输入。每个滑动窗口都映射到一个较低维的特征（ZF 为 256-d，VGG 为 512-d，后面是 ReLU）。此功能被馈入两个同级的全连接层——边界框回归层（reg）和框分类层（cls）。在本文中，我们使用 n = 3，注意输入图像上的有效接收场很大（ZF 和 VGG 分别为 171 和 228 像素）。在图 3 的单个位置（左）显示了此微型网络。请注意，由于微型网络以滑动窗口的方式运行，因此完全连接的层将在所有空间位置上共享。自然地，该体系结构由 n×n 卷积层和两个同级 1×1 卷积层（分别用于 reg 和 cls）实现。

3.3 RT-DETR

DETR 发展历程

模型名称	发布时间	主要原理	引入的新内容	backbone
DETR	2020	DETR使用卷积backbone和一个编码器-解码器Transformer结构，可以端到端地进行目标检测(此时还达不到实时检测)	首次将Transformer用于目标检测领域	ResNet-50/ ResNet-101
DAB-DETR	2022.1	DAB-DETR在DETR的基础上引入了动态锚框作为查询。它直接使用盒子坐标作为在transformer解码器中的查询，并且这些查询会在每层中动态更新。	引入了动态锚框作为查询	ResNet50-DC5
DN-DETR	2022.3	DN-DETR通过引入查询去噪来加速DETR的训练。它通过额外地将带有噪声的真实边界框输入到transformer解码器中，并训练模型来重建原始框，从而有效减少二分图匹配的难度，加速收敛。	进入了查询去噪机制	ResNet
DINO	2021.4	DINO是在DN-DETR和DAB-DETR的基础上进一步改进的模型，它结合了去噪和动态锚框的思想。	引入了去噪和动态锚框	
H-DETR	2022.7	H-DETR提出了一种基于混合匹配方案的方法，它结合了原始的一对一匹配分支和一个辅助的一对多匹配分支。在推理阶段，仅使用原始的一对一匹配分支，保持DETR的端到端特性和相同的推理效率。	提出了一种基于混合匹配方案的方法	
Group-DETR	2022.7	Group-DETR引入了组别一对多分配的概念，以支持快速的DETR训练。它在训练过程中进行了一些简单的修改，如采用组别分配策略来匹配多个正样本对象查询。	引入了组别一对多分配的概念	
Lite-DETR	2023.3	Lite-DETR旨在提供一个高效版本的DETR模型。它专门针对效率设计，通过模型架构和计算需求的优化实现这一目标。	轻量化	
RT-DETR	2023.11	RT-DETR由百度开发，是一种先进的端到端对象检测器，提供实时性能的同时保持高准确率。它利用视觉变换器ViT高效处理多尺度特征，通过解耦内尺度交互和跨尺度融合	具有内尺度特征交互（AIFI）和跨尺度特征融合模块（CCFM）。该模型还包括一个带有辅助预测头的变换器解码器	HGNet/ResNet

图 32 DETR 发展历程

RT-DETR 网络结构

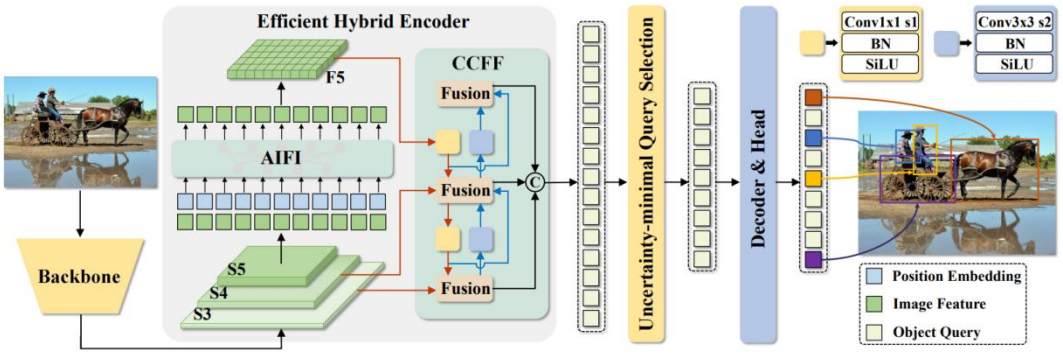


图 33 RT-DETR 网络结构

从结构上来看，RT-DETR 可以分为三大块：主干网络、颈部网络以及解码器。我们分别来说一下这三大块。

主干网络：

选择的网络架构： RT-DETR 可以采用多种 CNN 主干网络，如 ResNet 系列、百度自家研发的 HGNet，以及最近流行的 ViT（Vision Transformer）系列，例如 SwinTransformer 及其变体。这些网络用于从输入图像中提取特征，以供后续的目标检测任务使用。

特征提取的考量： 虽然 ViT 系列在最初表现出色，但后来的研究表明，训练技巧对 ViT 的性能至关重要。因此，选择 CNN 架构作为主干网络可以更好地支持实时性能要求，尤其是对硬件资源有限的场景，因为 CNN 在推理速度和实用性上有优势。

输出尺度： 主干网络从图像中提取三个尺度的特征输出，其输出步长分别为 8、16 和 32 像素。这些尺度对应着不同粒度的特征表示，有助于在不同尺度下捕捉目标的信息。

颈部网络：

在 RT-DETR 中，颈部网络是负责处理主干网络输出特征的一层 Transformer Encoder 层，称为 AIFI（Attention-based Intra-scale Feature Interaction）模块。

AIFI 模块概述：

AIFI 模块实际上是一个标准的 Transformer Encoder 层，包含多头自注意力（MSAH）和 Feed Forward Network（FFN），用于处理主干网络输出的特征。这些特征首先被拉平成向量形式，然后通过 AIFI 模块进行处理，处理过程如图 7 所示。处理后的特征再调整回二维形式，以便进行后续的跨尺度特征融合。

解码器：

检测头设计:

RT-DETR 的解码器采用了基于 cross attention 的 Transformer 解码器，并结合了多层感知器（MLP）作为检测头。这种设计借鉴了 DINO 的解码器结构，使用线性复杂度的 deformable attention，这有助于在保持推理速度的同时提高模型的检测准确性。

标签匹配:

RT-DETR 在标签分配过程中采用了经典的匈牙利匹配方法，这是 DETR 系列常用的方法。相比较于传统的匈牙利匹配，RT-DETR 参考了 DINO 的实现，在考虑到一些噪声样本的匹配时，进行了一定的优化，以提升训练效率和收敛速度。

损失函数:

在损失函数的设计上，RT-DETR 使用了 GIoU 损失和 L1 损失作为回归损失，这与传统的目标检测方法类似，旨在优化检测框的位置预测。与传统的类别损失不同的是，RT-DETR 引入了 "IoU 软标签" 的概念。

"IoU 软标签" 将预测框与真实框之间的 IoU 作为类别预测的标签，而不是传统的离散的 0 或 1。这种方法的优势在于可以更好地对齐类别和位置的预测，避免了类别预测过早学习的问题，特别是在定位不够准确时。

为了适应这种连续值标签，RT-DETR 使用了可变焦点损失（Variable Focal Loss, VFL），而不是传统的 focal loss，这一损失函数的应用也在 YOLOv6 中有所体现。

4 实现细节

4.1 数据预处理

首先查看所给数据集，发现共 150 张图片标注图像，class=2。其中 Occupied2443 个，Vacant318 个。这个数据分布存在明显的类别不平衡问题。当一个类别的样本量远远超过另一个类别时，模型可能会偏向于预测样本数量更多的类别。在这种情况下，模型可能会更频繁地预测 Occupied 类别，而对 Vacant 类别的预测效果不佳。

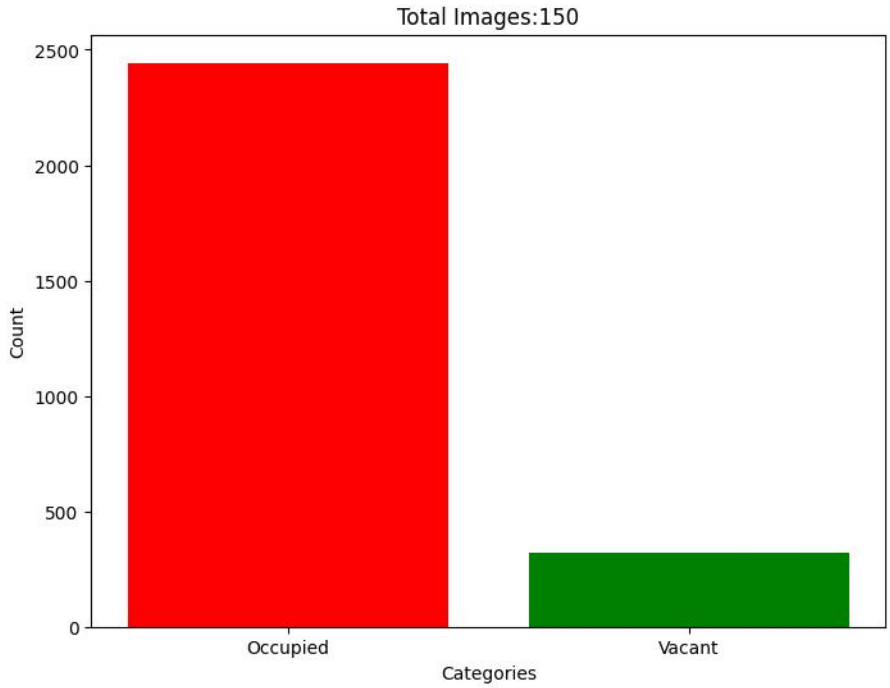


图 34 数据集标签情况

扩容数据集 1：PKLot

来源：PKLot Computer Vision Project

数据集 1 共计 12415 张图像，对应标签为 space-empty、space-occupied。

图 35PKLot 数据集展示

扩容数据集 2：CNRPark+EXT

来源：a preliminary dataset composed by 12,000 images collected in different days of July 2015 from 2 cameras.

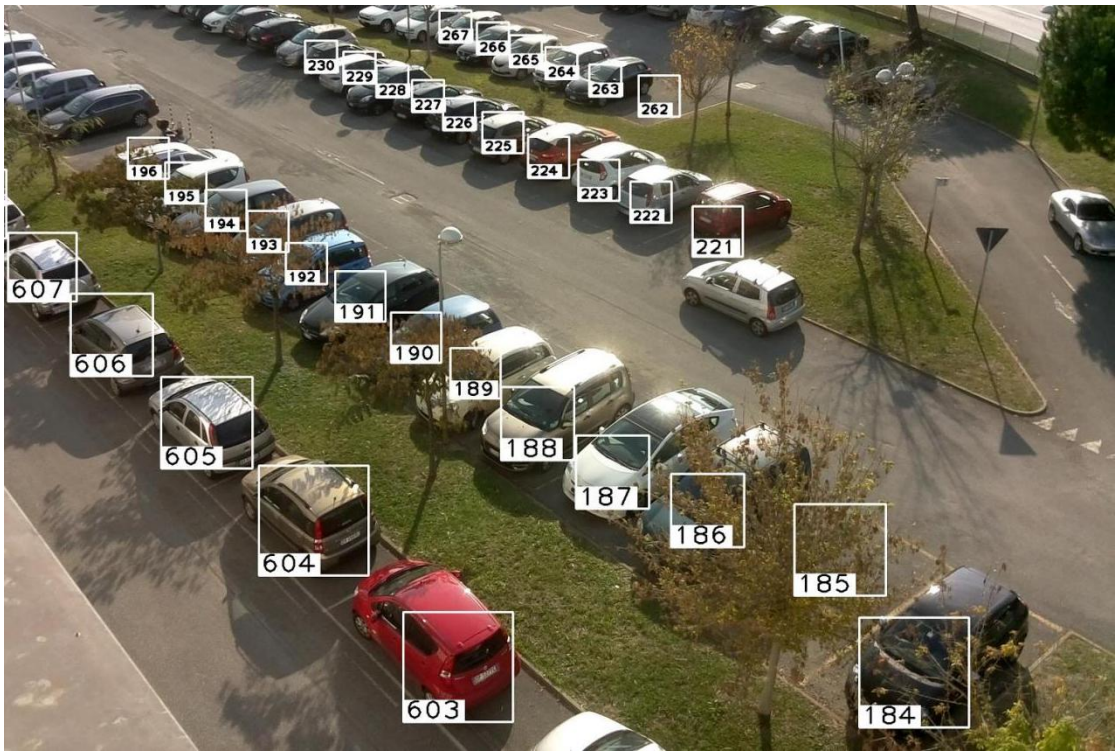


图 36 CNRPark+EXT 数据集展示

在扩充过数据集后，我们按照 7:2:1 来划分训练集、验证集、测试集。

4.2 数据集标注转换

对于原始数据集给的 XML 格式，解析如下：

labels: 这是一个包含多个标注对象的列表，每个对象代表一个检测到的目标。每个标注对象包括以下字段：

name: 物体的类别名称。在这里，所有标注都属于"Occupied"类别，表示被占用的物体或区域。

x1, y1, x2, y2: 这四个值描述了物体的边界框坐标信息。

(x1, y1) 是边界框左上角的坐标。

(x2, y2) 是边界框右下角的坐标。

考虑到后面训练的时候不同模型支持不同类型的数据集，这里在对数据集扩容后将数据集分别转换为 COCO 标注格式、VOC txt 标注格式、以及 YOLO 标注格式。这里不做详细说明。

4.3 实验平台

Ubuntu 20.04.6 LTS

GPU:TU106 [GeForce RTX 2060 Rev. A]

炼起来很慢，而且 bc 开不大。所以后面均采用迁移学习来训练。

4.2 YOLOV10

4.2.1 代码运行环境

```
[conda]
conda create -n yolov10 python=3.9
conda activate yolov10
pip install -r requirements.txt
```

pip install -e .

requirements.txt

torch==2.0.1

torchvision==0.15.2

onnx==1.14.0

onnxruntime==1.15.1

pycocotools==2.0.7

PyYAML==6.0.1

scipy==1.13.0

onnxsim==0.4.36

onnxruntime-gpu==1.18.0

gradio==4.31.5

opencv-python==4.9.0.80

psutil==5.9.8

py-cpuinfo==9.0.0

huggingface-hub==0.23.2

safetensors==0.4.3

4.2.2 参数设置

（1）创建 yaml 参数文件，加载数据集

```
# dataset root dir

path: /home/hb/1_machine_vision/Parking_detection/data/yolo_dataset



train: train/images   # train images (relative to 'path') 128 images

val: val/images   # val images (relative to 'path') 128 images

test: test/images # test images (optional)
```

```
# Classes

names:
```

```
0: Vacant
1: Occupied
```

（2）下载预训练权重

下载 YOLOv10-X 预训练权重

（3）修改参数

YOLOv10 关于模型的各种参数都在 ultralytics/cfg/default.yaml 中（其实是与 YOLOv8 一致的）。训练 500 轮。

- lr0: 0.01
- 初始学习率。这个参数决定了训练开始时模型权重调整的速度。在训练的早期阶段，较高的初始学习率可以加快收敛速度，但过高可能导致训练不稳定。
- lrf: 0.01
- 最终学习率。通过初始学习率乘以最终学习率因子，决定训练过程中学习率的衰减。较低的最终学习率因子可以使模型在接近收敛时更加稳定。
- momentum: 0.937
- SGD 动量或者 Adam 优化器的 beta1 参数。这个参数决定了优化器在更新过程中保留之前更新的方向的比例，有助于加速收敛和减少震荡。
- weight_decay: 0.0005
- 优化器的权重衰减参数。它通过减少模型的复杂度来防止过拟合，通常设置为一个较小的正则化项。
- warmup_epochs: 3.0
- 学习率预热的时期。这个参数表示在正式训练开始之前，用较低的学习率进行一些训练步骤，帮助模型更快地达到稳定状态。
- box: 7.5
- 目标定位损失的权重。这个参数决定了目标检测中位置精度的重要性，较高的权重可以加强模型对物体位置的准确预测。
- cls: 0.5
- 目标分类损失的权重。在多类别检测任务中，这个参数调整了分类损失对模型训练的影响力，确保模型能够准确地分类目标。
- mosaic: 1.0

- 图像马赛克的概率。马赛克增强技术将多个图像片段混合在一起作为输入图像，可以提升模型对复杂场景的泛化能力。

4.3 Faster R-CNN

4.3.1 代码运行环境

[conda]
torch
torchvision
tensorboard
scipy==1.2.1
numpy==1.17.0
matplotlib==3.1.2
opencv_python==4.1.2.30
tqdm==4.60.0
Pillow==8.2.0
h5py==2.10.0

4.3.2 参数设置

- Cuda: True
- 是否使用 Cuda。如果没有 GPU 可用，将其设置为 False。
- seed: 11
- 用于固定随机种子，确保每次训练结果的一致性。
- train_gpu: [0]
- 训练使用的 GPU 编号列表。在多 GPU 环境下，每张卡上的 batch 大小为总 batch 大小除以卡的数量。
- fp16: False
- 是否使用混合精度训练。可以减少显存占用，要求 PyTorch 版本在 1.7.1 以上。

- classes_path: 'model_data/voc_classes.txt'
- 指向包含类别信息的 txt 文件路径，用于训练时加载数据集的类别信息。
- model_path: 'model_data/voc_weights_resnet.pth'
- 预训练模型的权重文件路径。如果设置为空字符串''，则不加载整个模型的权重。
- input_shape: [640, 640]
- 输入图像的大小。
- backbone: "resnet50"
- 使用的主干网络模型。可以是'vgg'或'resnet50'等。
- pretrained: False
- 是否使用主干网络的预训练权重。如果设置了 model_path，则此参数失效。
- anchors_size: [8, 16, 32]
- 先验框的大小列表，用于生成检测框。
- Init_Epoch: 0
- 冻结阶段开始的训练轮数，用于断点续训时设置。
- Freeze_Epoch: 50
- 冻结阶段的总训练轮数，此阶段模型主干被冻结。
- Freeze_batch_size: 4
- 冻结阶段的训练批次大小。
- UnFreeze_Epoch: 500
- 解冻阶段的总训练轮数，解冻主干网络进行完整训练。
- Unfreeze_batch_size: 4
- 解冻阶段的训练批次大小。
- Freeze_Train: True
- 是否进行冻结阶段的训练。建议设置为 True，然后根据情况调整。
- Init_lr: 0.0001
- 初始学习率。
- Min_lr: 1e-06
- 学习率的最小值，默认为初始学习率的 0.01 倍。

- optimizer_type: "adam"
- 使用的优化器类型，可以是'adam'或'sgd'。
- momentum: 0.9
- 优化器的动量参数。
- weight_decay: 0
- 权重衰减，用于防止过拟合。
- lr_decay_type: 'cos'
- 学习率衰减类型，可以是'step'或'cos'。
- save_period: 5
- 多少个 epoch 保存一次模型权重。
- save_dir: 'logs'
- 保存模型权重和日志文件的目录。
- eval_flag: True
- 是否在训练时进行验证集评估。
- eval_period: 5
- 多少个 epoch 评估一次验证集。
- num_workers: 6
- 用于数据加载的线程数，设置为 1 表示不使用多线程。
- train_annotation_path: '2007_train.txt'
- 训练集图片路径和标签文件路径。
- val_annotation_path: '2007_val.txt'
- 验证集图片路径和标签文件路径。

4.4 RT-DETR

4.4.1 代码运行环境

[conda]

torch==2.0.1

torchvision==0.15.2

onnx==1.14.0

onnxruntime==1.15.1

pycocotools

PyYAML

scipy

4.4.2 参数设置

- sync_bn: True
- 是否使用同步的 Batch Normalization。在多 GPU 训练时，如果设置为 True，会保证不同 GPU 上的 Batch Normalization 层的统计量保持一致。
- find_unused_parameters: False
- 是否查找未使用的参数。在使用多 GPU 训练时，设置为 True 可以提高效率，但在某些情况下可能会导致性能下降。
- use_amp: False
- 是否使用自动混合精度（Automatic Mixed Precision，AMP）。当设置为 True 时，模型训练过程中会使用混合精度来减少显存使用。
- scaler:
  - type: GradScaler
  - enabled: True
  - 梯度缩放器的配置，用于混合精度训练时梯度的缩放管理。
- use_ema: False
- 是否使用指数移动平均（Exponential Moving Average，EMA）。当设置为 True 时，模型的权重会通过 EMA 进行平滑处理，用于模型集成和稳定训练过程。
- ema:
  - type: ModelEMA
  - decay: 0.9999
  - warmups: 2000
  - EMA 的配置，包括衰减率（decay）和热身步数（warmups）。
- epoches: 200

- 总的训练轮数（epochs），指定模型训练的总迭代次数。
- clip_max_norm: 0.1
- 梯度裁剪的阈值，用于控制梯度的最大范数，防止梯度爆炸。
- optimizer:
  - type: AdamW
  - lr: 0.0001
  - betas: [0.9, 0.999]
  - weight_decay: 0.0001
  - 优化器的配置，包括类型（AdamW）、学习率（lr）、动量参数（betas）和权重衰减（weight_decay）。
- lr_scheduler:
  - type: MultiStepLR
  - milestones: [1000]
  - gamma: 0.1
  - 学习率调度器的配置，多步学习率调度器，在指定的里程碑（milestones）处降低学习率，降低率由 gamma 指定。

5 实验结果与分析

5.1 YOLOV10

(1)结果展示：

图 37 真实标签，预测结果

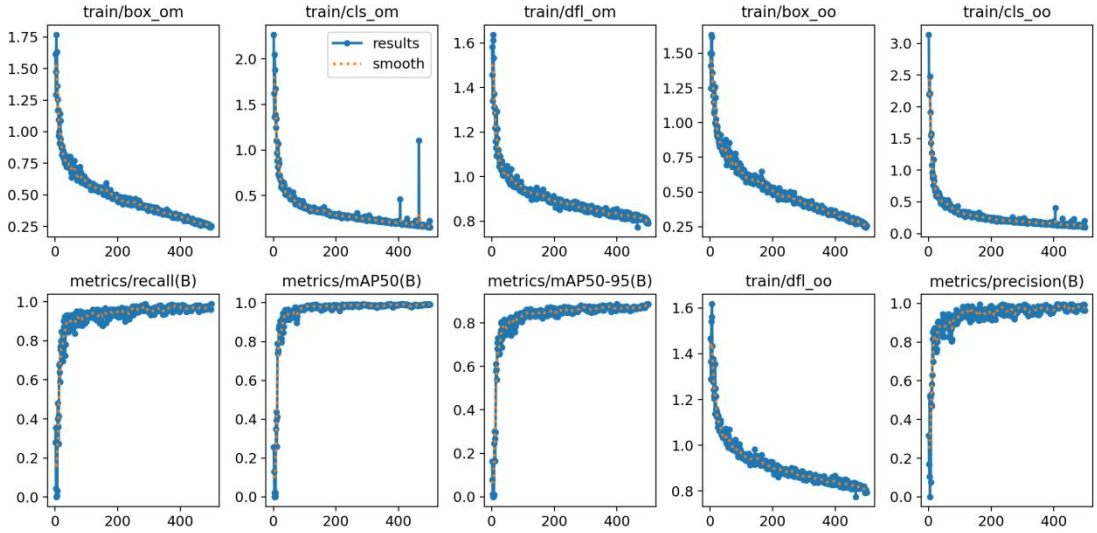


图 38 训练过程

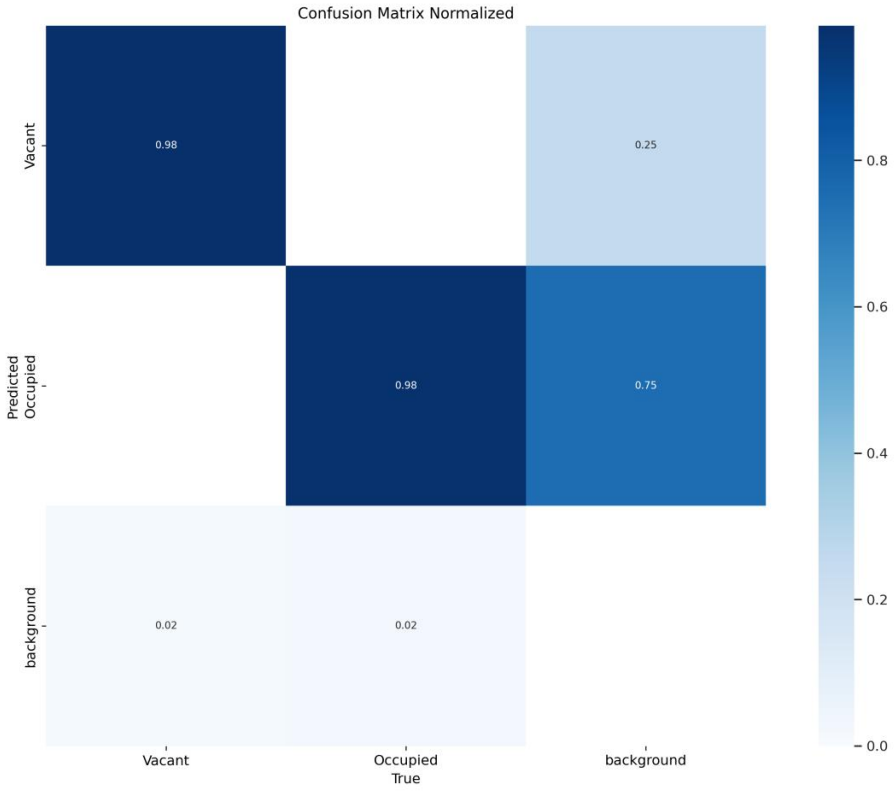


图 39 混淆矩阵

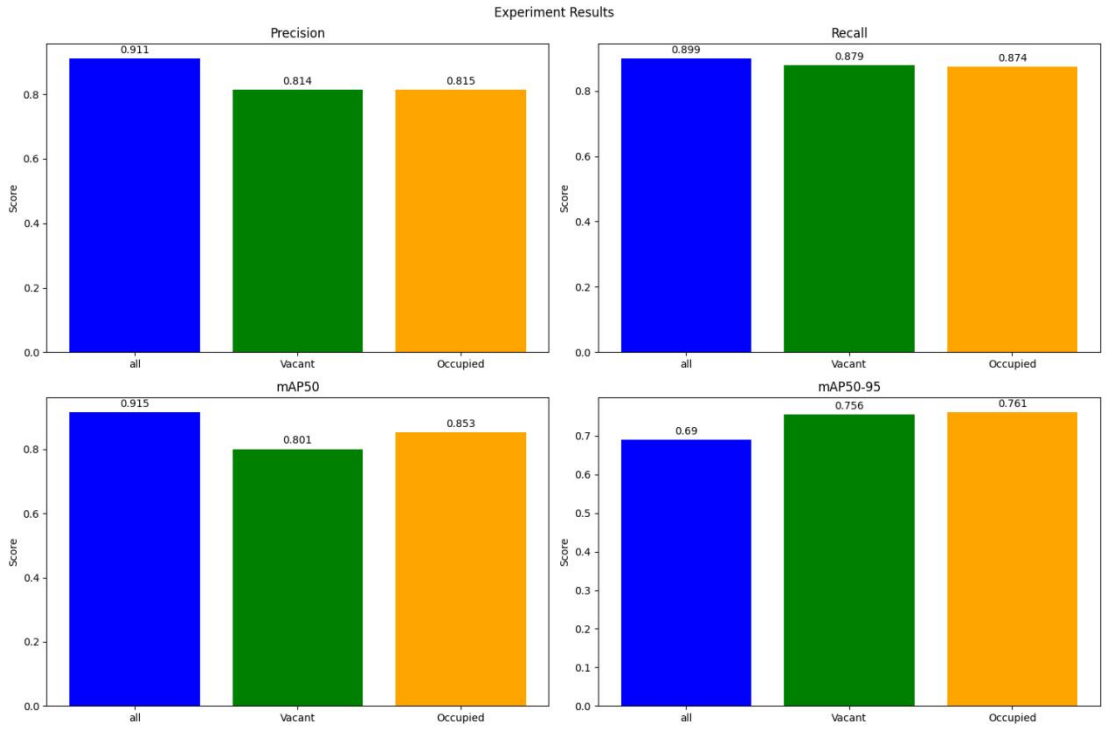


图 40 测试集测试结果

性能评价指标介绍

Precision（精确率）：

精确率是指在所有预测为正类的样本中，实际为正类的比例。计算公式为：

$$Precision= TP/TP+FP$$

其中，TP 表示真正例（True Positive），FP 表示假正例（False Positive）。

Recall（召回率）：

召回率是指在所有实际为正类的样本中，被正确预测为正类的比例。计算公式为：

$$Recall= TP/TP+FN$$

其中，FN 表示假负例（False Negative）。

mAP50（平均精度，IoU=0.50）：

mAP（Mean Average Precision）是用于衡量目标检测模型性能的指标。mAP50 表示在 IoU 阈值为 0.50 时的平均精度。

mAP50-95：

mAP50-95 表示在 IoU 阈值从 0.50 到 0.95（步长为 0.05）之间的平均精度。这是一个更严格和全面的评估指标。

(2)分析：

精确率（Precision）：模型在识别正类样本时非常准确，尤其是在处理所有数据时精确率更高。

召回率（Recall）：模型在识别真实正类样本方面具有较高的能力，且在不同类别之间差异不大。

mAP50：虽然在所有类别上模型的平均精度较高，但在空闲类别上的表现略逊色，这可能需要进一步优化。

mAP50-95：在更严格的 IoU 阈值范围内（0.50 到 0.95），模型的平均精度有所下降，但仍然保持在较高水平，特别是在空闲和占用类别上的表现相对平衡。

模型在整体上表现优异，特别是在精确率和召回率方面，表现出色。尽管在更严格的 mAP50-95 指标下，总体平均精度有所下降，但依然具有较强的检测能

力。在不同类别之间，模型对空闲类别的精度略低于占用类别，可能需要进一步调整和优化以提升空闲类别的检测性能。

5.2 Faster R-CNN

(1)结果展示

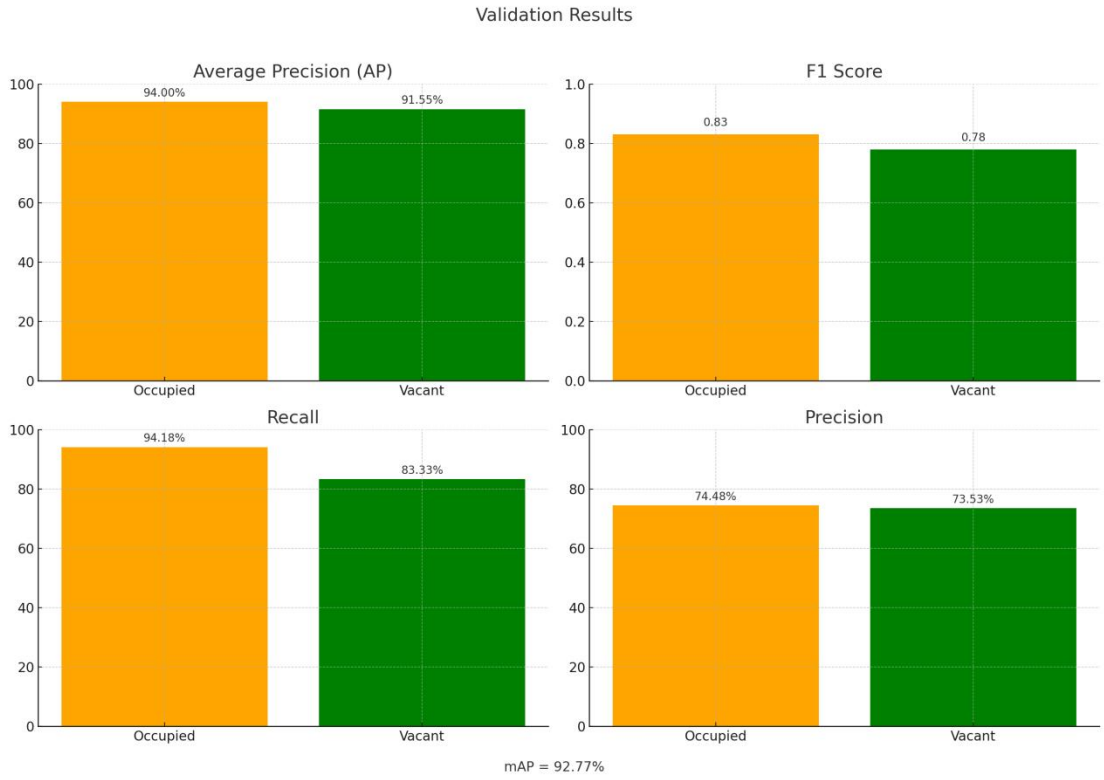


图 41 测试集测试结果

(2)分析：

1. 平均精度 (Average Precision, AP)

Occupied: 94.00%

Vacant: 91.55%

模型在检测“Occupied”类别时的平均精度较高，达到 94.00%。对于“Vacant”类别，平均精度略低，但也有 91.55%。总体来说，模型在这两个类别上的检测精度都很高。

2. F1 分数 (F1 Score)

Occupied: 0.83

Vacant: 0.78

F1 分数综合了精确率和召回率的性能，对于“Occupied”类别为 0.83，对于“Vacant”类别为 0.78。两者都显示出较强的综合性能。

3. 召回率 (Recall)

Occupied: 94.18%

Vacant: 83.33%

召回率方面，模型在“Occupied”类别上表现更为突出，达到 94.18%。而在“Vacant”类别上的召回率为 83.33%。高召回率表明模型在识别实际存在的目标时效果较好。

4. 精确率 (Precision)

Occupied: 74.48%

Vacant: 73.53%

精确率显示模型在预测为正类中的准确性。“Occupied”类别的精确率为 74.48%，略高于“Vacant”类别的 73.53%。

总体平均精度 (mAP)

mAP: 92.77%

总体平均精度（mAP）为 92.77%，这表明模型在处理所有类别时整体表现出色。

总结分析

模型在检测“Occupied”类别上表现尤为突出，不仅在平均精度和召回率上占据优势，同时 F1 分数也显示出较好的综合性能。

“Vacant”类别的检测性能也相当不错，尽管在平均精度和召回率上略逊于“Occupied”类别，但总体表现依然很强。

这两个类别的精确率接近，表明模型在预测正类样本时具有相似的准确性。

5.3 RT-DETR

(1)结果展示

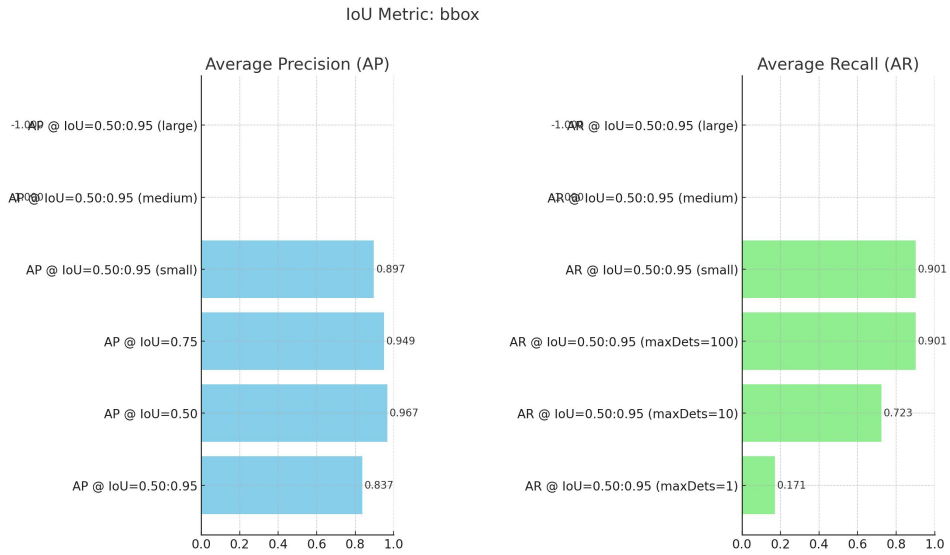


图 42 测试集测试结果

(2)分析：

性能评价指标可视化及分析

平均精度 (Average Precision, AP)

AP @ IoU=0.50:0.95: 0.837

AP @ IoU=0.50: 0.967

AP @ IoU=0.75: 0.949

AP @ IoU=0.50:0.95 (small): 0.897

AP @ IoU=0.50:0.95 (medium): -1.000

AP @ IoU=0.50:0.95 (large): -1.000

平均召回率 (Average Recall, AR)

AR @ IoU=0.50:0.95 (maxDets=1): 0.171

AR @ IoU=0.50:0.95 (maxDets=10): 0.723

AR @ IoU=0.50:0.95 (maxDets=100): 0.901

AR @ IoU=0.50:0.95 (small): 0.901

AR @ IoU=0.50:0.95 (medium): -1.000

AR @ IoU=0.50:0.95 (large): -1.000

分析

1. 平均精度 (AP)

总体表现:

模型在 IoU=0.50 的条件下表现最佳，平均精度达到 0.967，这表明在低重叠阈值条件下，模型的检测精度非常高。

在 IoU=0.75 条件下，平均精度略低于 IoU=0.50，但仍然达到 0.949，显示出在较高重叠阈值下模型仍然保持良好性能。

综合各个 IoU 阈值的平均精度 (0.50:0.95) 为 0.837，这表明模型在不同重叠阈值下的综合表现较为平衡。

小目标检测:

模型在检测小目标时表现较好，平均精度为 0.897。

中等和大目标检测:

模型在中等和大目标检测上的平均精度为-1.000，表明在这两个尺寸范围内没有足够的数据或检测结果。

2. 平均召回率 (AR)

总体表现:

在允许最多 1 个检测框的情况下，平均召回率为 0.171，这表明在限制条件下，模型的召回率较低。

当允许最多 10 个检测框时，平均召回率显著提升至 0.723。

在允许最多 100 个检测框时，平均召回率达到最高的 0.901，表明模型在允许更多检测框的情况下能够更好地识别目标。

小目标检测:

模型在检测小目标时的平均召回率为 0.901，与总体最高召回率一致，显示出在小目标检测方面的强大能力。

中等和大目标检测:

模型在中等和大目标检测上的平均召回率为-1.000，表明在这两个尺寸范围内没有足够的数据或检测结果。

结论

模型在 IoU=0.50 和 0.75 条件下的平均精度较高，综合不同 IoU 阈值的平均精度也表现良好。

小目标检测的表现尤为突出，但在中等和大目标检测上缺乏数据或检测结果。
平均召回率随着允许检测框数量的增加而显著提升，在最多 100 个检测框的情况下达到最高。

6 结论

6.1 结论

模型的总体性能：

在不同类别和不同 IoU 阈值条件下，模型都表现出了较高的平均精度和召回率，特别是在低 IoU 阈值和小目标检测中表现尤为突出。
总体平均精度（mAP）和 F1 分数显示模型在各类检测任务中的综合性能较强。

类别差异：

“Occupied”类别的检测性能优于“Vacant”类别，特别是在召回率和 F1 分数方面，这表明模型在检测这些类别时的精度和召回率存在一定的差异。
目标尺寸影响：

模型在小目标检测中表现良好，但在中等和大目标检测上缺乏有效数据或检测结果，这可能是由于训练数据不均衡或模型在处理较大目标时存在问题。

6.2 改进策略

数据扩展：增加中等和大目标的训练数据，以提高模型在这些尺寸目标上的检测性能。
模型优化：进一步优化模型结构和训练参数，特别是针对“Vacant”类别的检测精度和召回率进行改进。
评估指标：在实际应用中，综合考虑不同 IoU 阈值和检测框数量对模型性能的影响，以确保模型在多种条件下都能保持良好性能。

7 感想

完成这一系列项目后，我对目标检测领域有了更全面和深入的理解。从最初的传统目标检测算法到如今的深度学习方法，每一步的探索都让我感受到了技术发展的飞速和创新的无限可能。

项目初期

在项目开始时，我花了大量时间研究目标检测的基础知识，了解了传统方法和现代深度学习方法的演变。通过对比传统的 Haar 特征与 Adaboost、HOG 特征与 SVM 等方法，我认识到这些方法在处理复杂场景时的局限性。这也激发了我对深度学习方法的兴趣。

模型选择与算法分析

在深入学习了 YOLO、Faster R-CNN、RT-DETR 等主流目标检测模型后，我选择了这三个模型进行实验对比。通过研究 YOLO 的发展历程和 RCNN 系列模型的演变，我掌握了这些算法的基本原理和改进点。特别是在了解 RT-DETR 时，我看到了 Transformer 在目标检测中的新应用，感受到了这项技术的前沿性。

实现细节与实验过程

实际动手实现这些模型的过程充满了挑战。为了确保实验的可重复性和结果的可靠性，我花了大量时间在数据预处理、数据集划分和参数调优上。每一次实验的成功都让我感到兴奋，每一次失败也让我学到了宝贵的经验。

实验结果与分析

通过对不同模型的实验结果进行详细分析，我不仅掌握了如何评估模型性能，还学会了如何根据具体需求选择合适的模型。YOLO 的快速检测速度、Faster R-CNN 的高精度、RT-DETR 的创新性都给我留下了深刻的印象。在实际应用中，我了解到没有一种模型是完美的，选择适合的模型需要权衡速度、精度和资源等多方面因素。

综合体会

整个项目让我深刻体会到了理论与实践结合的重要性。书本上的知识只有在实际应用中才能真正发挥作用。而通过不断实验、调试和分析，我不仅提高了技术能力，还培养了严谨的科研态度和解决问题的能力。

展望未来

通过这次项目，我对目标检测领域有了更深的热爱和更强的信心。我相信，随着技术的不断发展，目标检测将在更多领域发挥重要作用。我也期待在未来的学习和工作中，能够应用这些知识和技能，为科技进步贡献自己的力量。

总的来说，这是一段充实而有意义的学习经历，让我对未来在数据科学和计算机视觉领域的深入探索充满期待。

8 项目代码介绍

Parking-Slot-Detection：

https://github.com/CHB-learner/Parking-Slot-Detection

```
├── conda 环境.txt
├── data
|       ├── archive_coco
|       ├── archive_download
|       ├── archive_voc
|       ├── archive_YOLO
|       ├── yolo_dataset
|       └── YOLO_split.py
├── faster-rcnn-pytorch-master
|       ├── 2007_train.txt
|       ├── 2007_val.txt
```

```
|       ├─── frcnn.py
|       ├─── get_map.py
|       ├─── img
|       ├─── LICENSE
|       ├─── logs
|       ├─── model_data
|       ├─── nets
|       ├─── predict.py
|       ├─── README.md
|       ├─── requirements.txt
|       ├─── summary.py
|       ├─── train.py
|       ├─── utils
|       ├─── voc_annotation.py
|       ├─── VOCdevkit
|       └─── 常见问题汇总.md
├─── RT-DETR
|       ├─── configs
|       ├─── README.md
|       ├─── requirements.txt
|       ├─── src
|       └─── tools
├─── YOLO10
|       ├─── app.py
|       ├─── CONTRIBUTING.md
|       ├─── docker
|       ├─── docs
|       ├─── examples
|       ├─── figures
```

```
|       ├──── flops.py
|       ├──── gradio_cached_examples
|       ├──── LICENSE
|       ├──── logs
|       ├──── main.py
|       ├──── mkdocs.yml
|       ├──── pyproject.toml
|       ├──── README.md
|       ├──── requirements.txt
|       ├──── runs
|       ├──── tests
|       ├──── ultralytics
|       ├──── ultralytics.egg-info
|       └──── wget-log
```

9 参考文献

[1]Zou, Zhengxia, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. "Object detection in 20 years: A survey." Proceedings of the IEEE 111, no. 3 (2023): 257-276.

[2]Terven, Juan, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas." Machine Learning and Knowledge Extraction 5, no. 4 (2023): 1680-1716.

[3]Reis, Dillon, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. "Real-time flying object detection with YOLOv8." arXiv preprint arXiv:2305.09972 (2023).

[4]Wang, Ao, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. "Yolov10: Real-time end-to-end object detection." arXiv preprint arXiv:2405.14458 (2024).

[5]Girshick, Ross, 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, pp. 580-587. 2014.

[6]Girshick, Ross. "Fast r-cnn." In Proceedings of the IEEE international conference on computer vision, pp. 1440-1448. 2015.

[7]Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).

[8]Carion, Nicolas, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. "End-to-end object detection

with transformers." In European conference on computer vision, pp. 213-229. Cham: Springer International Publishing, 2020.
[9]Zhao, Yian, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. "Detrs beat yolos on real-time object detection." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16965-16974. 2024.
[10]He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition." IEEE transactions on pattern analysis and machine intelligence 37, no. 9 (2015): 1904-1916.
[11]parkingSpace. "parking space finder Dataset." Roboflow Universe, Roboflow, Jun 2024. Available at: https://universe.roboflow.com/parkingspace/parking-space-finder-wjxkw. Accessed on: 2024-07-07.
[12]Amato, Giuseppe, et al. "Deep learning for decentralized parking lot occupancy detection." *Expert Systems with Applications*, vol. 72, 2017, pp. 327-334. Pergamon.
[13]Amato, Giuseppe, et al. "Car parking occupancy detection using smart camera networks and deep learning." In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, IEEE, 2016, pp. 1212-1217.
[14]Amato, Giuseppe, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, Carlo Meghini, and Claudio Vairo. "Deep learning for decentralized parking lot occupancy detection." Expert Systems with Applications 72 (2017): 327-334.
[15]Amato, Giuseppe, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, and Claudio Vairo. "Car parking occupancy detection using smart camera networks and deep learning." In Computers and Communication (ISCC), 2016 IEEE Symposium on, pp. 1212-1217. IEEE, 2016.