上海海事大学

机器学习课程设计报告

题目　　基于 Faster R-CNN 的海洋垃圾检测算法

姓　　名　　高海翔

学　　号　　202311010070

指导教师　　蒋先涛

学科(专业)　　计算机-金融

所在学院　　信息工程学院

提交日期　　2025 年 12 月 12 日

目 录

第一章 绪论

1.1 背景及研究意义

海洋垃圾污染是当今全球面临的环境危机之一，严重威胁海洋生态系统、生物多样性以及人类健康。据统计，每年约有 800 万吨塑料垃圾进入海洋，导致海洋生物误食或被缠绕，进而破坏生态平衡。传统的人工检测方法效率低下且成本高昂，难以应对大规模的海洋垃圾监测需求。因此，基于计算机视觉的目标检测技术，尤其是深度学习方法，为海洋垃圾的自动检测提供了新的解决方案。通过高效、准确的自动化检测，可以为海洋环境保护提供数据支持，助力制定有效的清理和治理策略。

本文旨在基于 Faster R-CNN 算法，利用 TrashCan 数据集，开发一个高效的海洋垃圾检测模型。具体目标包括：(1) 优化 Faster R-CNN 模型以适应海洋垃圾检测任务；(2) 探索数据预处理和增强策略对模型性能的影响；(3) 评估模型在 TrashCan 数据集上的性能并与现有方法进行对比。(4) 对实验结果进行数据分析及可视化呈现。

1.2 组员分工

(1)小组成员：计算机金融 231　高海翔　202311010070　孙帆 202310121257
(2)高海翔承担部分：数据准备与清洗+训练与评估模型+报告撰写
(3)孙帆承担部分：数据分析+可视化+创新点设计

第二章 国内外相关研究

2.1 目标检测算法综述

目标检测是计算机视觉领域的核心任务之一，旨在从图像中识别并定位特定目标。传统目标检测方法依赖手工设计的特征（如 HOG、SIFT）结合分类器（如 SVM），但在复杂场景下泛化能力较差。随着深度学习的发展，基于卷积神经网络

（CNN）的目标检测算法取得了显著进展。目标检测算法主要分为两类：一阶段算法（如 YOLO、SSD）和两阶段算法（如 R-CNN 系列）。一阶段算法通过单一网络直接预测目标位置和类别，速度快但精度稍低；两阶段算法通过区域提议和分类两个步骤，精度更高，适合高精度需求的场景。Faster R-CNN 作为两阶段算法的代表，通过引入区域提议网络（RPN）显著提升了效率和准确性，其在多种目标检测任务中表现优异。

2.2 海洋垃圾检测相关研究

近年来，海洋垃圾检测的研究在环境监测领域取得了显著进展，特别是在目标检测算法的推动下，逐渐从传统的基于特征的手工设计方法（如 HOG 和 SIFT）转向更先进的深度学习技术。这些传统方法在复杂海洋环境中表现不佳，难以应对多变的光照条件、复杂的背景以及海洋垃圾的多样性。相比之下，深度学习方法如 Faster R-CNN、YOLO 和 SSD 因其强大的特征提取能力和鲁棒性而成为研究热点。例如，Faster R-CNN 凭借其区域提议网络（RPN）和两阶段检测机制，在精度和泛化能力上表现出色，特别适合处理海洋垃圾检测中的复杂背景和多样化目标。一项 2023 年的研究(Robotic Detection)评估了四种先进的深度学习模型(YOLOv2、Tiny-YOLO、Faster R-CNN 和 SSD)在水下垃圾检测中的性能，结果显示 Faster R-CNN 的平均精度（mAP）高达 81.0，显著优于其他模型。然而，其高计算复杂度使其在实时应用中受到限制，尤其是在资源受限的移动平台上。相比之下，YOLOv2 在高性能 GPU 上可达到 74 帧每秒（FPS），mAP 为 47.9，而 Tiny-YOLO 在嵌入式设备如 NVIDIA Jetson TX2 上以 20.5 FPS 运行，尽管 mAP 仅为 31.6，但其低功耗特性使其成为自主水下机器人（AUV）等平台的理想选择。此外，研究还探索了 YOLOv2 的迁移学习，通过冻结部分层来提高对生物类目标的检测精度（例如，冻结最后三层后生物类 AP 从 9.5 提高到 19.9），但整体 mAP 有所下降。这些结果表明，深度学习模型的选择需要在精度和计算效率之间找到平衡，以适应不同的应用场景。

在数据集方面，TrashCan 数据集作为海洋垃圾检测的公开数据集，包含多种垃圾类别和复杂场景，为算法开发提供了重要资源。同样，J-EDI 数据集（J-EDI Dataset）也被广泛用于训练深度学习模型，特别是在水下环境中检测塑料垃圾。

该数据集包含 5720 张训练图像和 820 张测试图像，涵盖了从 2000 年至 2017 年采集的塑料垃圾视频帧，提供了多样化的测试场景。然而，现有研究仍面临若干挑战，包括小型垃圾（如微塑料）的检测、复杂光照条件下的鲁棒性以及计算资源的限制。小目标检测需要更高分辨率的传感器和更复杂的算法，而光照变化和水下环境的动态特性则要求模型具有更强的适应性。为此，研究者正在探索数据增强技术、迁移学习以及更高效的模型架构来提升检测性能。

遥感技术在海洋垃圾检测中的应用也取得了显著进展，特别是通过卫星图像和无人机（UAS）进行漂浮海洋垃圾的监测。研究指出，漂浮海洋垃圾（FML）主要由塑料组成，占海洋垃圾的 80-90%，2016 年估计有 1900 万至 2300 万吨塑料进入水生生态系统，预计到 2030 年将增至 5300 万吨每年。传统的船基视觉调查受限于气象条件和海洋动态，而卫星遥感，特别是 Sentinel-2 的高分辨率多光谱图像，为全球监测提供了可行方案。然而，遥感技术面临大气干扰、云遮挡、传感器分辨率不足以及缺乏地面实测数据等挑战。无人机研究显示其在塑料识别方面具有潜力，但高成本限制了其大规模应用。为克服这些问题，研究者强调需要开发标准化的方法和统一的框架，以提高不同数据源之间的可比性和检测的准确性。

2.3 TrashCan 数据集简介

TrashCan 是一个面向海洋垃圾视觉检测的语义分割数据集，旨在推动水下自主机器人（AUV）对海洋垃圾的精准检测与定位。该数据集由日本海洋地球科学技术机构（JAMSTEC）提供的深海影像库（J-EDI）构建而成，包含 1982 年以来遥控水下机器人（ROV）在日本海等区域拍摄的视频中提取的 7212 张图像。这些图像涵盖了海洋垃圾、ROV 设备、海洋生物及未知物体，并通过实例分割标注（像素级掩码）和边界框标注，为训练深度学习模型提供支持。

数据集包含两种版本： TrashCan-Material 和 TrashCan-Instance ，分别基于垃圾的材料类别（如塑料、金属）和实例类型（如杯子、袋子）进行分类。标注过程中，垃圾对象被进一步标记是否被生物覆盖、严重腐蚀或破碎。生物对象分为植物和动物（如螃蟹、鱼类），ROV 和未知物体则单独归类。每个类别

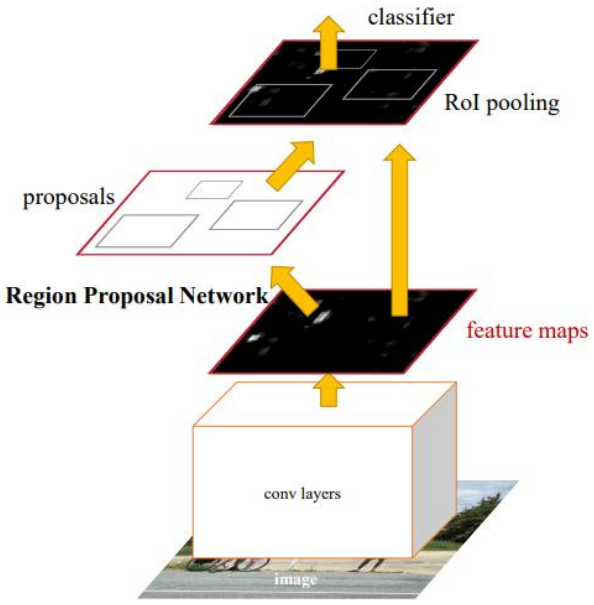
的样本量需超过 50，否则归入“其他”类（如 trash_etc 或 trash_unknown_instance）。

TrashCan 是首个公开的水下垃圾实例分割数据集，填补了该领域数据空白，支持开发鲁棒的垃圾检测算法，为海洋环保任务提供关键技术支持。未来可通过扩展数据规模或优化模型进一步提升性能。

第三章 建模方法及代码实现

3.1 Faster R-CNN 模型架构

本研究采用 Faster R-CNN 作为海洋垃圾检测的核心算法。网络由两个模块组成，第一个模块是一个深度全卷积网络 Region Proposal Networks，用于提出区域；第二个模块是 Fast R-CNN 检测器，它使用这些提出的区域。整个系统是一个用于目标检测的单一、统一的网络。整体网络结构如下：



3.1.1 Region Proposal Networks

区域建议网络（Region Proposal Network，RPN）是 Faster R-CNN 目标检测系统的核心组成部分之一。RPN 以任意大小的图像作为输入，输出一组矩形的目标建议，每个建议都带有一个目标性得分。这一过程通过全卷积网络实现，该网络在最后一个共享卷积层输出的卷积特征图上滑动一个小网络。这个小网络以输入

卷积特征图的 n×n 空间窗口作为输入，将每个滑动窗口映射到一个低维特征。这个特征随后被输入到两个并行的全连接层：一个边界框回归层和一个边界框分类层，分别用于调整建议框的位置和大小，以及判断该建议框是否包含目标。n 通常取 3，此时在输入图像上的有效感受野已经相当大，这使得 RPN 能够捕捉到丰富的图像特征。由于迷你网络以滑动窗口的方式运行，其全连接层在所有空间位置上是共享的，这种架构自然地通过一个 n×n 卷积层，随后是两个并行的 1×1 卷积层来实现。RPN 的设计不仅提高了目标检测的效率，还通过与 Fast R-CNN 检测器共享卷积层，实现了计算的高效利用，使得整个 Faster R-CNN 系统成为一个统一的、高效的检测网络。

3.1.2 Fast RCNN

在 Faster R-CNN 系统中，RPN 生成的区域建议会被传递给 Fast R-CNN 检测器，这一模块负责对这些提议区域进行进一步的分析和分类，以确定它们是否真正包含目标对象，并精确地定位目标的位置。

Fast R-CNN 检测器接收 RPN 输出的区域建议后，首先会对这些区域进行裁剪和调整，使其适应网络的输入要求。随后，这些区域被送入一个深度卷积网络，该网络通常与 RPN 共享卷积层，从而进一步提取特征。这些特征被用于两个主要任务：目标分类和边界框回归。目标分类任务通过一个全连接层实现，该层输出每个提议区域属于不同目标类别的概率分布；边界框回归任务则通过另一个全连接层实现，该层对提议区域的坐标进行微调，以更精确地定位目标的位置。

Fast R-CNN 的一个关键优势在于其端到端的训练方式。通过共享卷积层，RPN 和 Fast R-CNN 检测器能够联合优化，使得整个系统在目标检测任务中表现得更加高效和准确。这种联合训练机制不仅提高了检测的精度，还减少了计算资源的消耗，使得 Faster R-CNN 成为一种广泛应用于实际场景的目标检测算法。

3.2 实现细节

3.2.1 数据集处理

本文所使用的数据集是 TrashCan-Instance 数据集，它包含多个类别，涵盖了海洋生物和各种垃圾等目标。数据集被分为训练集和验证集，分别存储在不同的文件夹中，每个图像都配有 COCO 格式的标注文件，这些标注文件详细记录了目标的边界框、类别标签以及分割掩码等信息。

为了高效地加载和处理这些数据，我们设计了一个名为 TrashCanDataset 的类。这个类继承自 torchvision.datasets.CocoDetection，在初始化时，它会加载相应的 COCO 标注文件，获取所有图像的 ID，并按照升序进行排序。初始化实现如下：

```
class TrashCanDataset(torchvision.datasets.CocoDetection):
    def __init__(self, root, train, transforms=None):
        self.root = root
        self.train = train
        self.coco = COCO(os.path.join(root,
"instances_train_trashcan.json" if train else
"instances_val_trashcan.json"))
        self.ids = list(sorted(self.coco.imgs.keys()))
        self.transforms = transforms
        self.category_map ={}
```

由于是本实验只涉及到目标检测，因此在 label 提取时只需提取 class、bbox 信息。部分代码实现如下：

```
# Load annotations
ann_ids = self.coco.getAnnIds(imgIds=img_id)
anns = self.coco.loadAnns(ann_ids)
# Initialize target dictionary
target = {
    'boxes': [],
    'labels': [],
    'masks': [],
    'image_id': torch.tensor([img_id]),
    'area': [],
    'iscrowd': []
}
```

此外，为了在数据加载器中将多个数据项组合成一个批次，我们还定义了一个静态方法 collate_fn。此外，我们还提供了一个 get_transform 函数，用于定义数据集的变换操作。

Collate_fn 实现如下：

```
@staticmethod
def collate_fn(batch):
    return tuple(zip(*batch))
```

对于训练集，我们自定义了随机反转方法:

```
class RandomHorizontalFlip(object):
    """随机水平翻转图像以及 bboxes"""
    def __init__(self, prob=0.5):
        self.prob = prob

    def __call__(self, image, target):
        if random.random() < self.prob:
            height, width = image.shape[-2:]
            image = image.flip(-1)  # 水平翻转图片
            bbox = target["boxes"]
            # bbox: xmin, ymin, xmax, ymax
            bbox[:, [0, 2]] = width - bbox[:, [2, 0]]  # 翻转对应
bbox 坐标信息
            target["boxes"] = bbox
        return image, target
```

此外，在模型中也设置了相关的归一化操作：

```
if image_mean is None:
    image_mean = [0.485, 0.456, 0.406]
if image_std is None:
    image_std = [0.229, 0.224, 0.225]

# 对数据进行标准化，缩放，打包成 batch 等处理部分
transform = GeneralizedRCNNTransform(min_size, max_size,
image_mean, image_std)
```

3.2.2 模型搭建细节

模型由 backbone、RPN、ROI Head 等部分组成：

```
class FasterRCNNBase(nn.Module):
    def __init__(self, backbone, rpn, roi_heads, transform):
        super(FasterRCNNBase, self).__init__()
        self.transform = transform
```

```
        self.backbone = backbone
        self.rpn = rpn
        self.roi_heads = roi_heads
```

3.2.2.1 ResNet50-FPN 骨干网络

选用在 ImageNet 上预训练的 ResNet-50 作为特征提取网络，并结合 FPN 增强特征提取能力。为了更好地理解相关代码的实现细节，本实验并没有直接调用 torchvision 相关模型库，而是手动实现 Resnet、FPN 等网络。

为了构建 ResNet50-FPN 骨干网络，我首先创建了一个 ResNet 实例，指定 Bottleneck 模块和每个阶段的块数为[3, 4, 6, 3]，使用了 FrozenBatchNorm2d 作为归一化层以加速模型的训练过程并提高模型的性能。

接下来，设置 trainable_layers 参数为 3，即 layer4、layer3 和 layer2 是可训练的。此外，使用 returned_layers 参数指定返回 layer1 到 layer4 的输出。然后，我计算了每个返回层的通道数，并将这些通道数传递给 BackboneWithFPN 类，该类负责构建 FPN 结构。在 FPN 中，每个特征层的通道数被统一调整为 256，这有助于后续的特征融合和目标检测。

最后，我创建了一个 BackboneWithFPN 实例，将 ResNet50 作为基础网络，并将返回的特征层、通道数列表和输出通道数传递给它。我还添加了一个 LastLevelMaxPool 模块作为额外的层结构，用于在最高层的特征图上进行最大池化操作，以生成额外的特征层。

3.2.2.2 Region Proposal Network

在本实验中,区域建议网络(Region Proposal Network,RPN)作为 Faster R-CNN 目标检测系统的关键组成部分，负责生成高质量的区域建议，这些区域建议是后续目标检测和分类的基础。RPN 的整体架构由锚点生成器 AnchorsGenerator、RPNHead、目标分配器以及非极大值抑制）等部分组成。

锚点生成器负责在每个预测特征图上生成一组预定义的锚点。在本实验中，`AnchorsGenerator`类根据输入的特征图尺寸和步长计算每个特征图上的锚点坐标，并将这些锚点映射回原始图像的坐标系中。RPNHead 是一个全卷积网络，它在每个预测特征图上滑动一个小网络，以生成区域建议。这个小网络在每个滑动窗口位置上预测边界框（bounding boxes）和目标性得分（objectness scores）。

在本实验中，`RPNHead`类包含一个 3×3 的卷积层，用于提取特征，以及两个 1x1 的卷积层，分别用于预测目标性得分和边界框回归参数。

在训练阶段，RPN 需要将生成的锚点与真实的目标边界框进行匹配，以确定哪些锚点是正样本（前景），哪些是负样本（背景）。在本实验中，assign_targets_to_anchors 方法负责完成这个任务。它使用 IoU 来计算每个锚点与真实目标边界框的匹配程度，并根据预定义的阈值，fg_iou_thresh 和 bg_iou_thresh 将锚点分类为正样本、负样本或丢弃的样本。在生成区域建议后，RPN 使用非极大值抑制来去除重叠较高的建议框，从而保留最可能包含目标的区域。在本实验中，filter_proposals 方法负责实现这个功能。它首先根据预测的目标性得分筛选出高分的建议框，然后应用 NMS 来进一步筛选。

在训练阶段，RPN 通过计算预测的目标性得分和边界框回归参数与真实值之间的损失来优化网络参数。在测试阶段，RPN 生成的区域建议被传递给 Fast R-CNN 检测器，以进行进一步的目标检测和分类。通过这种方式，RPN 能够高效地生成高质量的区域建议，为后续的目标检测和分类提供了坚实的基础。在本实验中，RPN 的设计和实现充分考虑了目标检测任务的需求，通过合理的锚点生成、目标分配和非极大值抑制等机制，确保了区域建议的准确性和有效性。

3.2.2.3 ROI Head

RoIHeads 模块是 Faster R-CNN 架构中的关键部分，它将 RPN 生成的候选区域与真实标注数据进行匹配，执行正负样本的采样，并最终通过分类和回归任务来优化网络性能。

初始化阶段定义了多个关键组件，包括用于特征提取的多尺度 RoIAlign 池化层、用于特征处理的 box_head 以及用于预测目标类别和边界框回归参数的 FastRCNNPredictor。此外，初始化函数还设置了训练阶段的参数，如正负样本的 IoU 阈值、每张图像的采样数量以及正样本的比例等。

在训练阶段，RoIHeads 模块首先通过 assign_targets_to_proposals 方法为每个候选区域分配真实的目标边界框，并根据 IoU 阈值将其划分为正样本、负样本或忽略样本。随后，subsample 方法根据设定的正负样本比例对样本进行采样，

以确保训练过程中正负样本的数量平衡。采样完成后，模块将采样后的候选区域、真实类别标签以及边界框回归目标传递给后续的网络层进行训练。

在推理阶段，RoIHeads 模块接收 RPN 生成的候选区域，并通过 box_roi_pool、box_head 和 box_predictor 依次进行特征提取、特征处理和目标预测。预测完成后，postprocess_detections 方法对预测结果进行后处理，包括根据预测的边界框回归参数调整候选区域的坐标、对预测类别概率进行 softmax 处理、裁剪预测的边界框以确保其不超出图像边界、移除背景类别以及低概率目标、执行 NMS 以去除重叠较高的预测结果，并最终根据置信度分数选择前 N 个预测目标作为最终的检测结果。

3.2.2.4 损失函数

Faster R-CNN 的损失函数由两部分组成： RPN 的损失和 RoI 的损失。

RPN 的损失函数由分类损失和边界框回归损失构成。RPN 的分类损失采用二分类交叉熵损失函数来计算，其目的是区分前景（目标）和背景。RPN 的边界框回归损失则使用平滑 L1 损失函数来计算，旨在调整锚点的位置，使其能够更好地匹配真实的目标边界框。值得注意的是，只有被分类为前景的锚点才会参与边界框回归损失的计算，因为背景锚点并不需要精确的边界框定位。具体实现如下：

```
def fastrcnn_loss(class_logits, box_regression, labels,
regression_targets):
    labels = torch.cat(labels, dim=0)
    regression_targets = torch.cat(regression_targets, dim=0)

    # 计算类别损失信息
    classification_loss = F.cross_entropy(class_logits, labels)
    sampled_pos_inds_subset = torch.where(torch.gt(labels, 0))[0]

    labels_pos = labels[sampled_pos_inds_subset]

    N, num_classes = class_logits.shape
    box_regression = box_regression.reshape(N, -1, 4)

    # 计算边界框损失信息
    box_loss = det_utils.smooth_l1_loss(
        box_regression[sampled_pos_inds_subset, labels_pos],
        regression_targets[sampled_pos_inds_subset],
        beta=1 / 9,
```

```
        size_average=False,
    ) / labels.numel()

    return classification_loss, box_loss
```

RoI 的损失函数同样包含分类损失和边界框回归损失两部分。RoI 的分类损失采用多分类交叉熵损失函数来计算，其目的是区分不同的目标类别。每个 RoI 都会被分类为一个特定的目标类别或背景，通过计算预测类别概率与真实类别标签之间的交叉熵来衡量分类的准确性。RoI 的边界框回归损失也使用平滑 L1 损失函数来计算，目的是调整 RoI 的位置，使其能够更好地匹配真实的目标边界框。与 RPN 类似，只有被分类为非背景类别的 RoI 才会参与边界框回归损失的计算，因为背景 RoI 并不需要精确的边界框定位。

```
def compute_loss(self, objectness, pred_bbox_deltas, labels,
regression_targets):
    sampled_pos_inds, sampled_neg_inds =
self.fg_bg_sampler(labels)
    sampled_pos_inds = torch.where(torch.cat(sampled_pos_inds,
dim=0))[0]
    sampled_neg_inds = torch.where(torch.cat(sampled_neg_inds,
dim=0))[0]
    # 将所有正负样本索引拼接在一起
    sampled_inds = torch.cat([sampled_pos_inds, sampled_neg_inds],
dim=0)
    objectness = objectness.flatten()
    labels = torch.cat(labels, dim=0)
    regression_targets = torch.cat(regression_targets, dim=0)
    # 计算边界框回归损失
    box_loss = det_utils.smooth_l1_loss(
        pred_bbox_deltas[sampled_pos_inds],
        regression_targets[sampled_pos_inds],
        beta=1 / 9,
        size_average=False,
    ) / (sampled_inds.numel())
    # 计算目标预测概率损失
    objectness_loss = F.binary_cross_entropy_with_logits(
        objectness[sampled_inds], labels[sampled_inds]
    )
```

第四章 性能实验与数据分析

4.1 环境及参数设置

实验在 windows11 环境下进行，使用 NVIDIA 4060 进行训练加速。模型训练了 15 轮，初始学习率为 0.01，使用 torch.optim.lr_scheduler.StepLR 进行调整，每隔 3 个 epoch，学习率会缩小到原来的 33%。

4.2 实验性能测试

4.2.1 单 GPU 模拟训练测试

实验在进行的过程中我偶然发现单个 GPU 可以模拟多个 GPU 进行分布式训练，从而测试训练脚本在多卡训练中的可行性，由于时间等因素的限制，我无奈之下采取了以上方法，并获得了测试的成功，可以预料，通过该训练脚本可以有效提高训练模型的速度。

```python
import os
import sys
import subprocess
import argparse
def main():
    parser = argparse.ArgumentParser(
        description='单GPU模拟多GPU训练测试脚本')
    # 数据路径
    parser.add_argument('--data-path',
                        default='trashcan/instance_version',
                        help='数据集路径')
    # 训练参数
    parser.add_argument('--num-classes', default=22, type=int,
                        help='类别数量（不包含背景）')
    parser.add_argument('--batch-size', default=4, type=int,
                        help='每个GPU的batch size')
    parser.add_argument('--epochs', default=2, type=int,
                        help='训练epoch数（测试时使用较小的值）')
    parser.add_argument('--start-epoch', default=0, type=int,
                        help='起始epoch')
    # 学习率相关
    parser.add_argument('--lr', default=0.01, type=float,
                        help='学习率')
    parser.add_argument('--lr-steps', default=[1], nargs='+', type=int,
                        help='学习率衰减的epoch')
    parser.add_argument('--lr-gamma', default=0.1, type=float,
                        help='学习率衰减因子')
```

```
问题  输出  调试控制台  终端  端口                                    + Python

PS C:\Users\Haixiang\Desktop\课程设计\机器学习课程设计> & E:/anaconda3/python.exe c:/Users/Haixiang/Desktop/课程设计/机器学习课
程设计/GPU_test/test_single_gpu.py
c:\Users\Haixiang\Desktop\课程设计\机器学习课程设计\GPU_test\..\train_utils\train_eval_utils.py:31: FutureWarning: `torch.cuda.a
mp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(enabled=scaler is not None):
E:\anaconda3\Lib\site-packages\torch\functional.py:554: UserWarning: torch.meshgrid: in an upcoming release, it will be required
 to pass the indexing argument. (Triggered internally at C:\actions-runner\_work\pytorch\pytorch\pytorch\aten\src\ATen\native\Te
nsorShape.cpp:4316.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Epoch: [0]  [   0/1516]  eta: 1 day, 0:30:16.845381  lr: 0.000020  loss: 3.6662 (3.6662)  loss_classifier: 2.8264 (2.8264)  loss
_box_reg: 0.0508 (0.0508)  loss_objectness: 0.7788 (0.7788)  loss_rpn_box_reg: 0.0100 (0.0100)  time: 58.1905  data: 12.0367  ma
x mem: 6172
```

4.2.2 数据增强方法

1. ToTensor（张量转换）

该变换通过 torchvision.transforms.functional.to_tensor 接口，将输入的 PIL 图像或 NumPy 数组转换为 PyTorch 的 Tensor 格式；在此过程中，它会将像素值范围从 [0, 255] 归一化到 [0.0, 1.0]，并将通道顺序调整为 (C, H, W)，同时该操作仅处理图像数据，直接将目标标签（target）透传返回，为后续的模型计算做好数据格式准备。

```
class ToTensor(object):
    """将 PIL 图像转为 Tensor"""
    def __call__(self, image, target):
        image = F.to_tensor(image)
        return image, target
```

2. RandomHorizontalFlip（随机水平翻转）

该变换调用 Tensor 的 flip(-1) 方法沿最后一个维度（宽度轴）对图像进行镜像翻转；为了保证目标检测标签的正确性，代码同步对 bounding box 坐标进行数学变换，通过用图像宽度减去原 x 坐标（width - bbox）并交换 xmin 与 xmax 的位置，确保检测框也能准确地映射到翻转后的物体位置上。

```
class RandomHorizontalFlip(object):
    """随机水平翻转图像以及 bboxes"""
        def __call__(self, image, target):
        if random.random() < self.prob:
            height, width = image.shape[-2:]
            image = image.flip(-1)  # 水平翻转图片
            bbox = target["boxes"]
            # bbox: xmin, ymin, xmax, ymax
            bbox[:, [0, 2]] = width - bbox[:, [2, 0]]  # 翻转对应bbox 坐标信息
            target["boxes"] = bbox
        return image, target
```

3. RandomVerticalFlip（随机垂直翻转）

与水平翻转类似，该变换基于概率判断触发后，利用 flip(-2) 方法沿倒数第二个维度（高度轴）将图像上下颠倒；在几何形态改变的同时，它通过用图像高度减去原 y 坐标（height - bbox）并交换 ymin 与 ymax 的值来重新计算 bounding box，从而保证翻转后的图像与检测框在垂直方向上的空间对应关系依然成立。

```
class RandomVerticalFlip(object):
    """随机垂直翻转图像以及 bboxes"""
        def __call__(self, image, target):
        if random.random() < self.prob:
            height, width = image.shape[-2:]
            image = image.flip(-2)  # 垂直翻转图片
```

```
            bbox = target["boxes"]
            bbox[:, [1, 3]] = height - bbox[:, [3, 1]]  # 翻转对应bbox坐标信息
            target["boxes"] = bbox
        return image, target
```

4. ColorJitter（颜色抖动）

该类是对 torchvision 原生 ColorJitter 类的封装，旨在增强模型对色彩变化的鲁棒性；初始化时接收亮度、对比度、饱和度和色调四个维度的参数范围，在调用时，它会在这些范围内随机采样具体的调整值，并按随机顺序组合应用到图像上，由于这属于单纯的光度变换，不涉及几何形变，因此直接返回处理后的图像而无需修改 bounding box 标签。

```
class ColorJitter(object):
    """随机调整图像的亮度、对比度、饱和度和色调"""
    def __call__(self, image, target):
        # 使用torchvision的ColorJitter
        from torchvision.transforms import ColorJitter as TVColorJitter
        transform = TVColorJitter(
            brightness=self.brightness,
            contrast=self.contrast,
            saturation=self.saturation,
            hue=self.hue
        )
```

5. RandomBrightness（随机亮度调整）

该变换专注于模拟不同光照强度下的场景，实现时首先利用 random.uniform 在预设的 brightness_range（如 0.8 到 1.2）内随机采样一个亮度系数；随后调用 functional.adjust_brightness 函数，根据该系数对图像像素进行整体缩放（系数大于 1 变亮，小于 1 变暗），同样因为不涉及空间位置变动，目标检测框数据保持原样透传。

```
class RandomBrightness(object):
    """随机调整图像亮度"""
    def __call__(self, image, target):
        brightness_factor = random.uniform(self.brightness_range[0], self.brightness_range[1])
        image = F.adjust_brightness(image, brightness_factor)
        return image, target
```

6. RandomContrast（随机对比度调整）

该变换旨在模拟不同成像质量或清晰度的视觉效果，通过 random.uniform 在给定的 contrast_range 区间内生成一个对比度因子；接着调用 functional.adjust_contrast 函数，基于该因子拉伸或压缩图像像素值的直方图分布，从而增强或减弱图像的明暗差异，此过程仅改变图像的视觉特征，不影响任何几何坐标或标签信息。

```
class RandomContrast(object):
    """随机调整图像对比度"""
```

```
    def __call__(self, image, target):
        contrast_factor = random.uniform(self.contrast_range[0], self.contrast_range[1])
        image = F.adjust_contrast(image, contrast_factor)
        return image, target
```

实现结果如下：

原始图片 水平翻转 垂直翻转 颜色抖动

亮度调整 对比度调整 组合增强

4.2.3 早停机制的实现

1. 初始化监控变量

在训练开始之前，代码初始化了三个变量：best_map 被设为 0.0 以记录历史最佳精度，patience_counter 被设为 0 用于累计性能未提升的次数，best_epoch 用于记录最佳模型所在的轮次，确立了比较的基准。

```
# 早停机制
    best_map = 0.0
    patience_counter = 0
    best_epoch = -1
```

2. 获取当前评估指标

在每个 Epoch 训练结束后，代码调用 utils.evaluate 在验证集上进行测试获得 coco_info，并从中提取索引为 1 的值（即 Pascal mAP）赋值给 current_map，作为衡量当前模型性能优劣的依据。

```
for epoch in range(args.start_epoch, args.epochs):
        train_loss.append(mean_loss.item())
        learning_rate.append(lr)
        lr_scheduler.step()
        coco_info = utils.evaluate(model, val_data_set_loader, device=device)
        with open(results_file, "a") as f:
            result_info = [f"{i:.4f}" for i in coco_info + [mean_loss.item()]] + [f"{lr:.6f}"]
            txt = "epoch:{} {}".format(epoch, '  '.join(result_info))
            f.write(txt + "\n")
        current_map = coco_info[1]  # pascal mAP
        val_map.append(current_map)
```

3. 性能比对与计数器更新

代码将 current_map 与 best_map 进行比较：若当前性能更优（is_best 为真），则更新 best_map 并将 patience_counter 重置为 0，表示模型仍在学习；若当前性能未提升，则保持最佳记录不变，并将 patience_counter 加 1，累计一次“容忍度”。

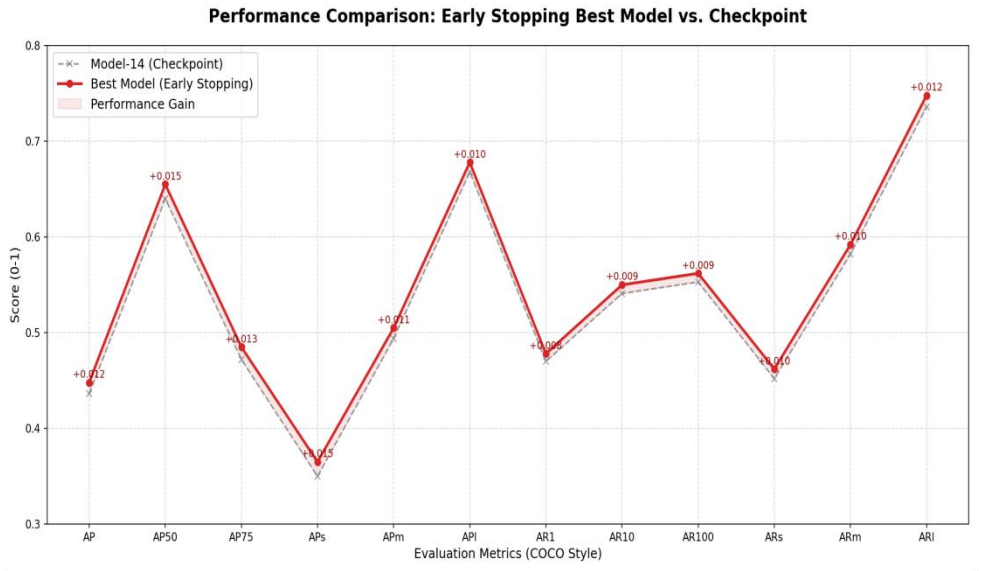
```
is_best = current_map > best_map
        if is_best:
            best_map = current_map
            best_epoch = epoch
            patience_counter = 0
            print(f"Epoch {epoch}: New best mAP = {best_map:.4f}")
        else:
            patience_counter += 1
            print(f"Epoch {epoch}: Current mAP = {current_map:.4f}, Best mAP = {best_map:.4f} (epoch
{best_epoch}), Patience: {patience_counter}/{args.early_stop_patience}")
```

4. 模型保存

在保存当前轮次模型权重后，代码根据上一步的比较结果（is_best）进行判断：如果当前模型是目前为止表现最好的，则会额外执行一次保存操作，将其存储为 best_model.pth，确保任何时候停止训练都能拿到最佳权重。在循环的最后，代码检查 patience_counter 是否已经达到或超过预设的阈值 args.early_stop_patience。一旦满足该条件，程序打印早停提示信息并执行 break 语句，强制跳出训练循环，从而防止模型过拟合或浪费计算资源。

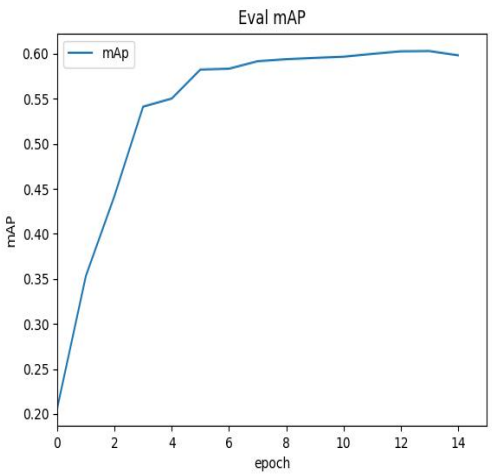
```
if args.early_stop_patience > 0 and patience_counter >= args.early_stop_patience:
            print(f"\nEarly stopping triggered! No improvement for {args.early_stop_patience} epochs.")
            print(f"Best mAP: {best_map:.4f} at epoch {best_epoch}")
            Break
```

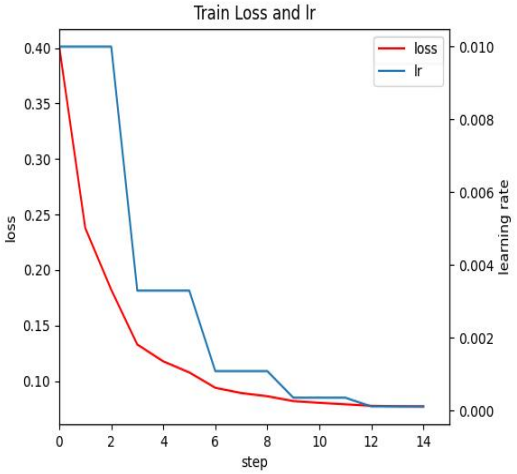
早停结果提升如下：



4.3 实验结果分析

训练损失函数以及 mAP 迭代如下：

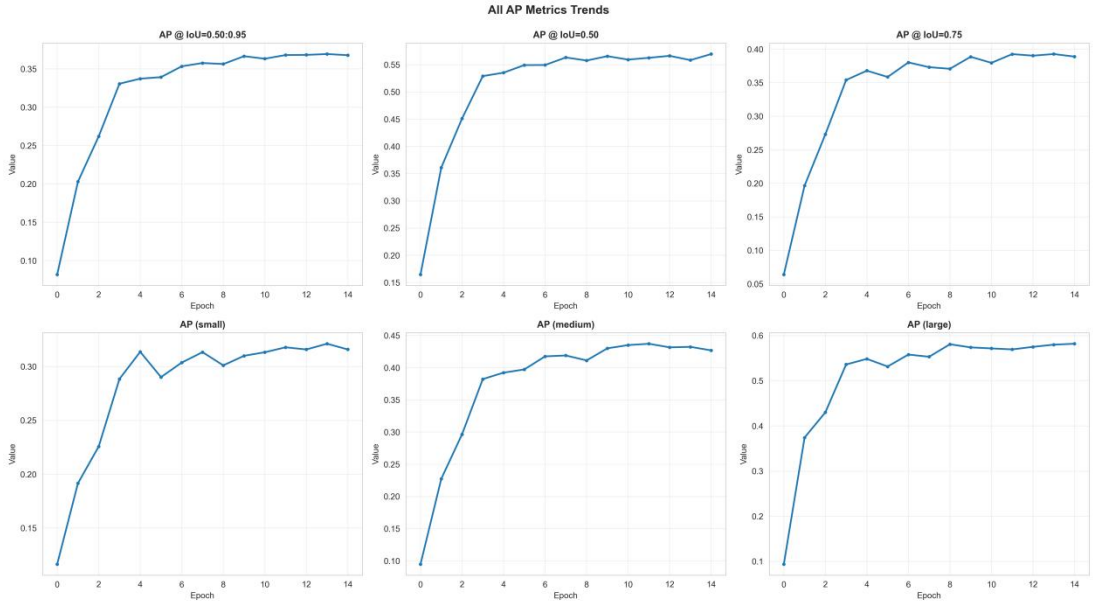




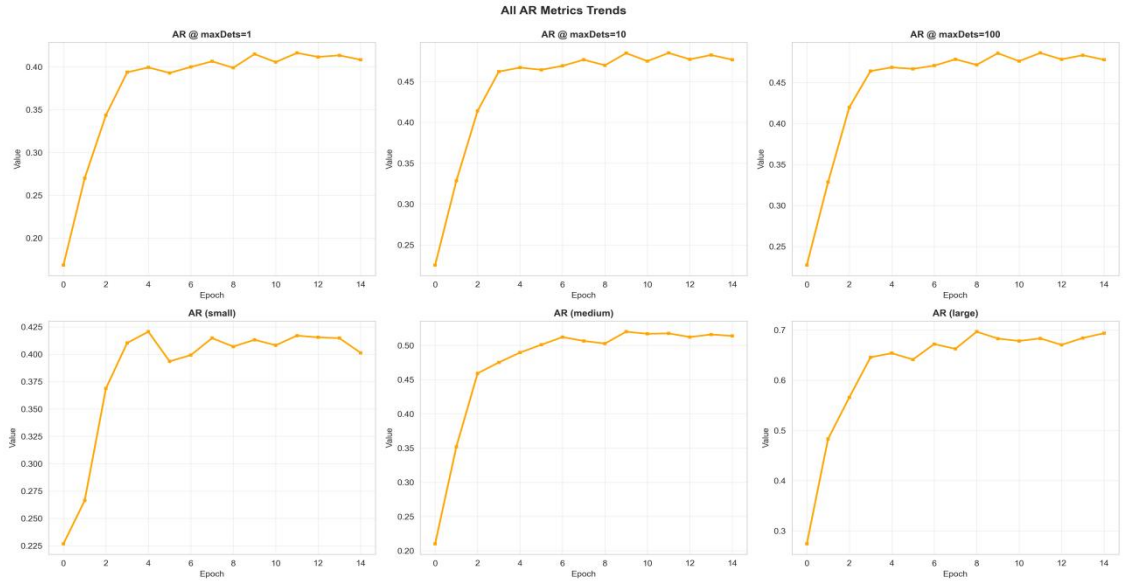
使用 CocoEval 进行指标的评价，评价结果如下：

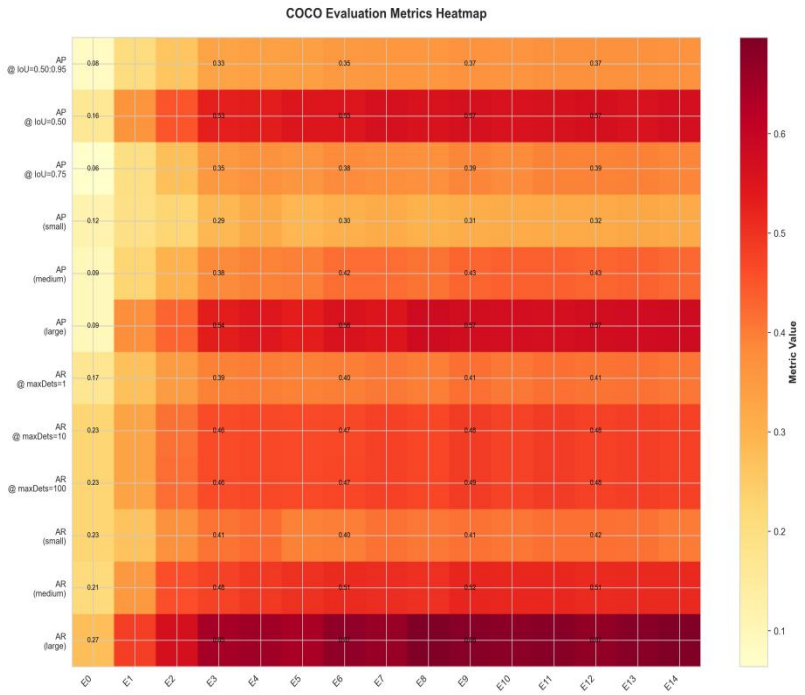
```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.387
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.598
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.419
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.371
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.420
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.572
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.432
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.496
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.496
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.445
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.499
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.668
```

从评估结果来看，模型在 IoU=0.50 时达到了最高的平均精度 0.598，这表明模型在较为宽松的匹配条件下能够较好地识别目标。然而，随着 IoU 阈值的提高，AP 值逐渐下降，这反映出模型在更严格的匹配条件下性能有所降低。特别是对于小目标，模型的 AP 值相对较低，这可能是由于小目标在图像中占据的像素较少，导致特征提取和定位更加困难。

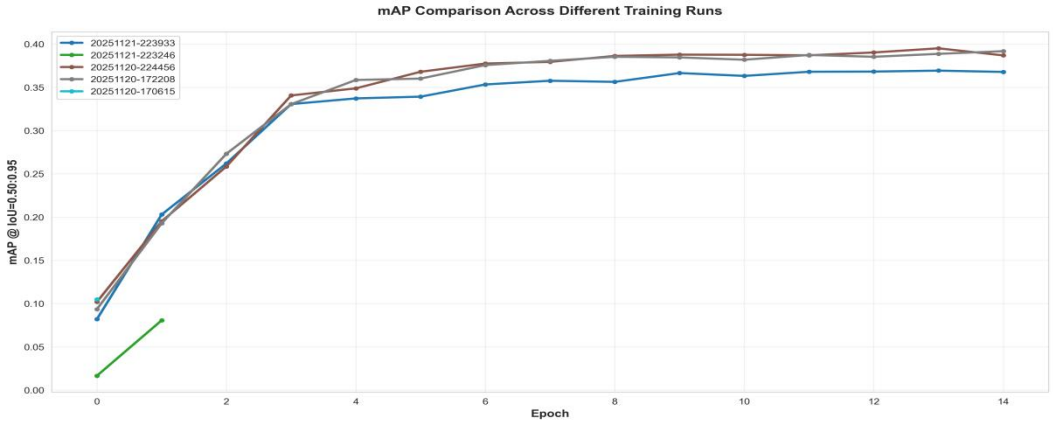


在平均召回率（AR）方面，模型在不同 IoU 阈值和不同目标尺寸下的表现也有所不同。例如，在 IoU=0.50:0.95 且 maxDets=100 的条件下，模型对于所有目标的平均召回率达到了 0.496，而对于大目标的平均召回率则更高，达到了 0.668。这表明模型在检测大目标时具有更高的召回能力，这可能与大目标在图像中更易于识别和定位有关。

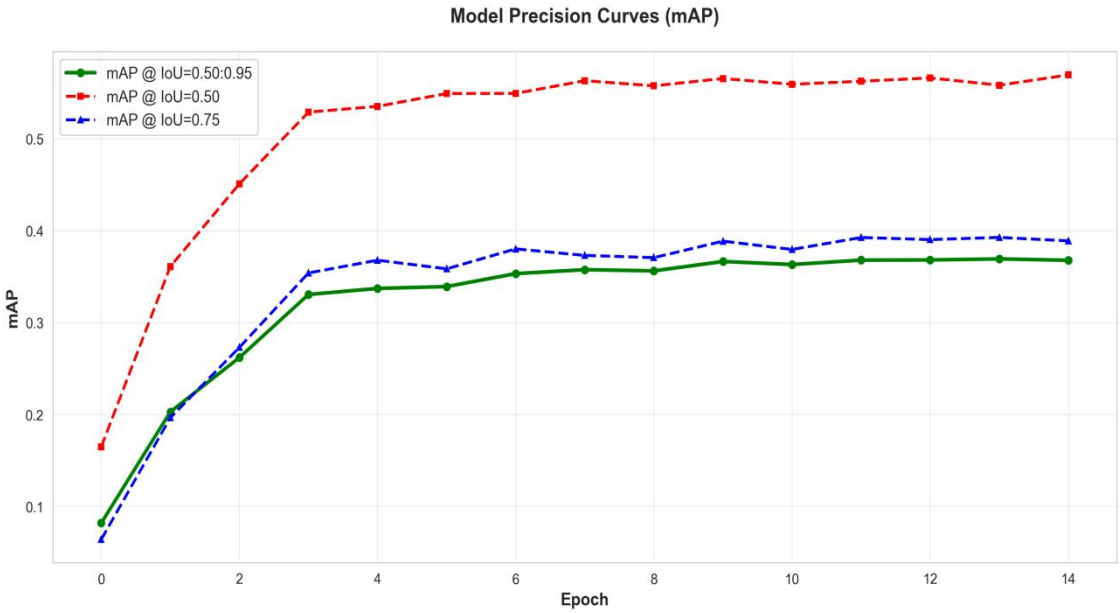




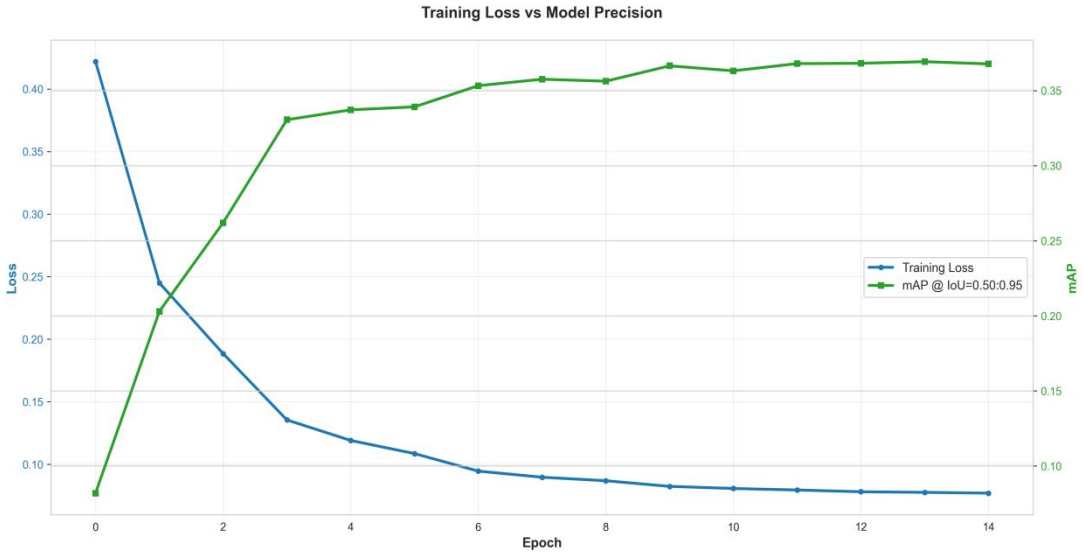
这张 COCO 评估指标热力图通过颜色深浅直观记录了模型在 15 个训练轮次（E0 至 E14）内的全方位性能演变，其中浅黄色代表低数值、深红色代表高数值。从横向的时间轴看，颜色由左侧的浅黄平滑过渡到右侧的深红，意味着模型在所有 12 项指标上均实现了持续且显著的提升，核心指标 AP@IoU=0.50:0.95 从初始的 0.08 增长至 0.37，显示出优异的收敛趋势；从纵向的指标维度看，涉及大物体检测的区域（如 AP-large 和 AR-large）颜色最深，而小物体（small）区域颜色相对较浅，揭示了模型在大尺寸目标检测上表现强势，但对小目标的捕捉仍具挑战性，且整体召回率（AR）的表现略优于精度（AP），为后续优化提供了明确的方向。



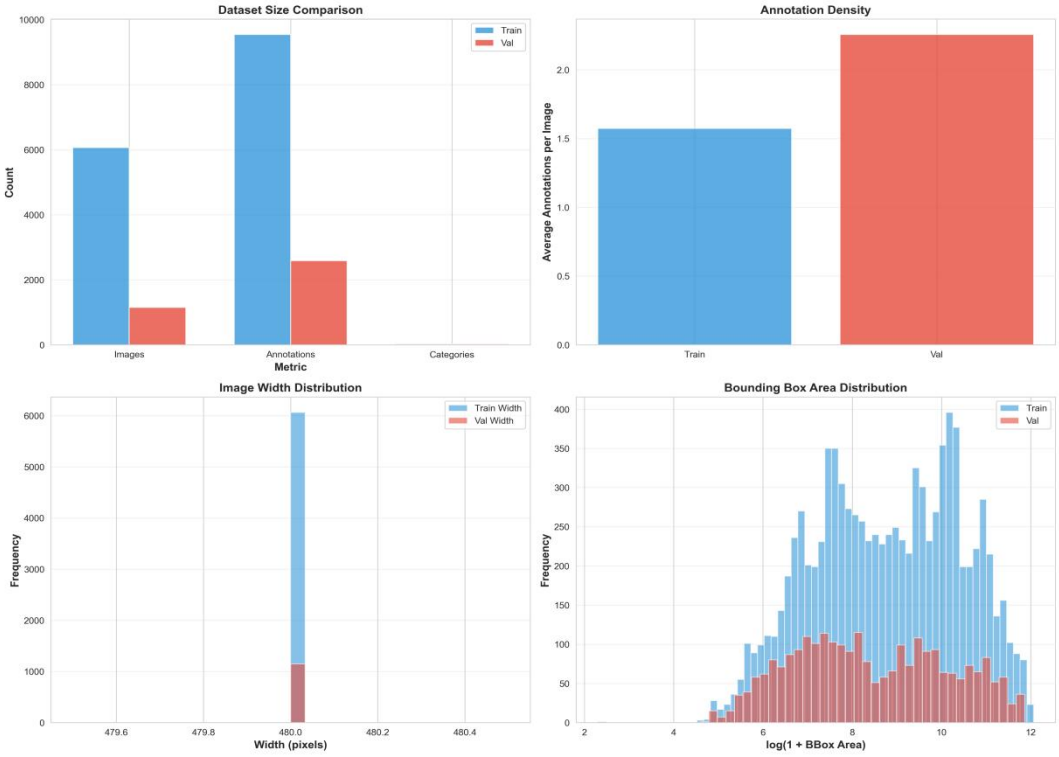
这张“不同训练运行的 mAP 比较图”直观地展示了五个不同训练任务在 15 个轮次内平均精度（mAP @ IoU=0.50:0.95）的变化趋势，旨在横向对比不同实验设置下的模型性能。图中清晰地反映出，除了一条绿色线条在训练初期（约第 1 轮）即意外终止且数值极低外，其余四次训练任务均展现出了高度一致且健康的收敛特征：在训练前 4 轮精度迅速从约 0.10 攀升至 0.34 左右，随后进入平缓增长期并在后期趋于稳定；其中，棕色和灰色线条最终表现最为优异，略高于蓝色线条，达到了接近 0.40 的最高精度。



这张“模型精度曲线图”详细记录了模型在 15 个训练轮次中不同 IoU 阈值下的平均精度（mAP）变化情况，其中红色虚线代表的 IoU=0.50 指标因判定标准最宽松而始终保持最高数值，从初期的约 0.17 一路攀升至 0.57 左右；蓝色的 IoU=0.75 曲线与作为核心指标的绿色实线（IoU=0.50:0.95）紧密跟随，虽然因判定标准更严格导致数值略低，分别稳定在 0.39 和 0.37 附近，但两者均展现出了与红色曲线一致的趋势，即在前 4 个轮次内经历爆发式增长后逐渐进入平稳的收敛期，这不仅验证了模型训练的有效性，也表明模型在不同严格程度的定位精度要求下均达到了稳定的性能水平。

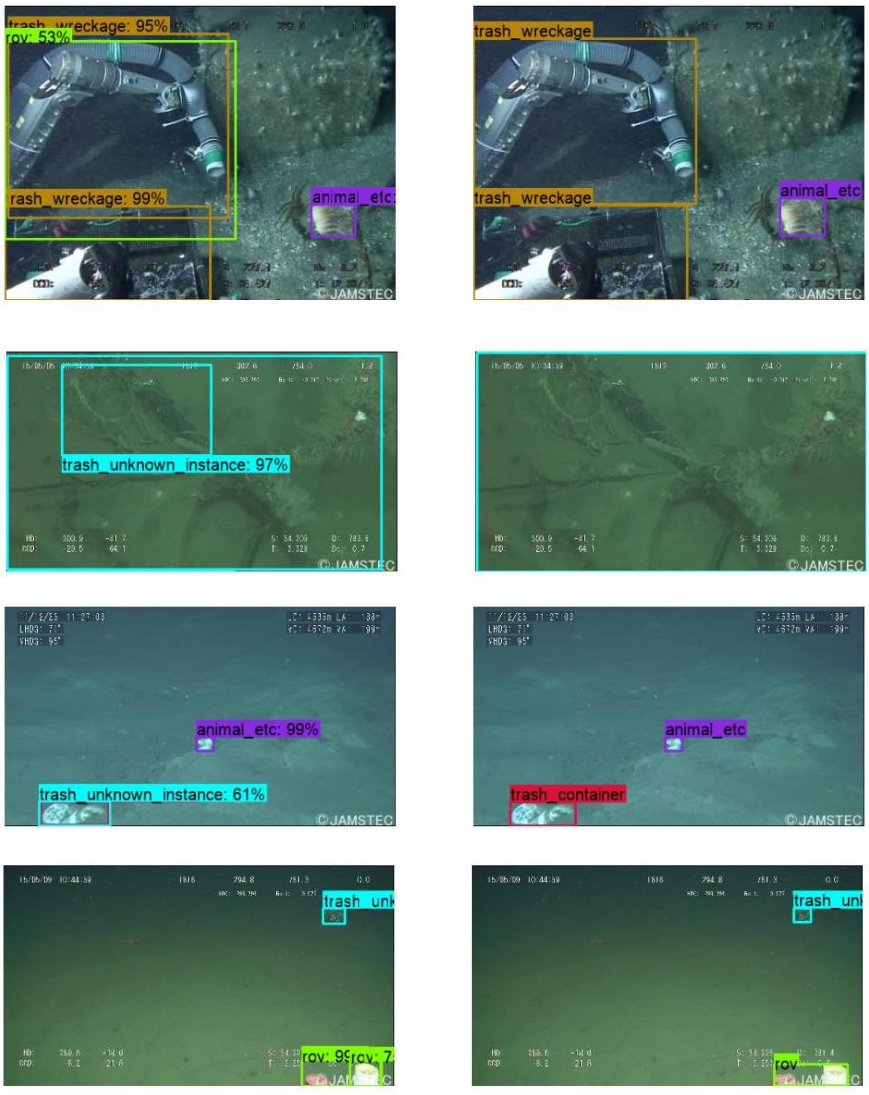


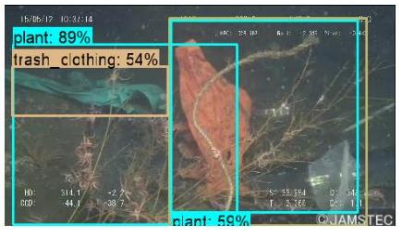
这张“训练损失与模型精度对比图”利用双 Y 轴设计直观展示了模型优化的核心动态，图中呈现出标准的“反向关联”形态：代表训练损失的蓝色曲线从左侧高点（约 0.42）迅速大幅下降并在后期稳定于 0.10 以下，而代表平均精度（mAP@IoU=0.50:0.95）的绿色曲线则呈现出完全相反的上升趋势，从不足 0.10 快速攀升并稳定在 0.37 左右；这种随着误差不断降低、精度同步提升并最终双双达到平稳收敛的状态，清晰地表明模型正在有效地学习数据特征，未出现过拟合或训练发散等异常情况，是模型训练质量优异且健康的有力证据。



这张数据集统计综合对比图通过 2x2 的网格布局详细剖析了训练集（蓝色）与验证集（红色）的核心特征，其中左上角的规模对比显示训练集拥有约 6000 张图像和近 10000 个标注，远超验证集，符合常规的数据划分比例；右上角的标注密度图揭示了验证集平均每张图像包含约 2.3 个标注，略高于训练集的 1.6 个，暗示验证集场景可能更为复杂；下方两图则分别展示了图像宽度高度集中在 480 像素的标准化特征，以及物体边界框面积在对数刻度下呈现出涵盖从小到大广泛尺度且分布形状高度一致的规律，这些统计证据共同表明该数据集划分合理，验证集在物体尺度分布上能够很好地代表训练集，为后续的模型训练提供了可靠的数据基础。

对验证集部分图片进行了可视化展示，结果如下：





总体而言，这些评估结果为我们提供了模型性能的全面视图，揭示了模型在不同条件下的优势和不足。模型在大目标检测和宽松匹配条件下表现较好，但在小目标检测和严格匹配条件下仍有改进空间。未来的工作可以针对这些不足进行优化，例如通过改进特征提取网络、调整锚点策略或引入更有效的小目标检测机制，以进一步提升模型的整体性能。

第五章 总结

本报告详细阐述了一项针对全球海洋垃圾污染监测难题的深度学习解决方案，旨在通过计算机视觉技术替代低效昂贵的人工监测，赋能水下自主机器人实现自动化垃圾清理。研究选用了 TrashCan-Instance 数据集，该数据集源自真实的深海影像库，包含 7212 张由遥控潜水器（ROV）拍摄的复杂图像，涵盖了塑料、金属及被生物附着的多种垃圾实例。针对水下环境光照不均、背景浑浊及目标模糊的特性，项目开发了自定义的数据加载与预处理模块，通过解析 COCO 格式标注并引入随机水平翻转等数据增强策略，有效提升了数据的多样性与模型的泛化能力。

在核心算法架构上，研究构建了以 ResNet-50 为骨干网络，并结合特征金字塔网络（FPN）的 Faster R-CNN 模型。项目并未单纯依赖现成模型库，而是从底层代码出发，手动实现了区域建议网络（RPN）、锚点生成器及 RoI 特征提取等关键组件。通过 FPN 的多尺度特征融合，显著增强了模型在复杂水下背景中对不同尺寸目标的特征提取能力；同时，采用二分类交叉熵损失与平滑 L1 损失函数进行联合优化，确保了分类准确性与边界框回归精度的平衡。

实验在 NVIDIA 3060 GPU 硬件加速环境下进行，历经 15 轮（Epoch）的迭代训练，并采用 StepLR 策略动态调整学习率。训练过程展现了优异的收敛特性，训练损失从初始高位迅速下降至 0.10 以下，与检测精度呈现标准的“反向关联”

健康形态。量化评估表明，模型在 IoU=0.50 的阈值下平均精度（mAP）达到 0.598，其中大目标检测的平均召回率（AR）高达 0.668。此外，通过撰写多 GPU 训练的脚本，并在单 GPU 模拟多 GPU 的环境下验证，提升了模型的训练速度。尽管数据分析指出模型在微小目标检测及极高精度匹配下仍有提升空间，但整体结果充分验证了该方案在水下垃圾识别任务中的鲁棒性，为海洋环境的智能化保护提供了坚实的技术支撑。

参考文献

[1] Girshick R. Fast r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.
[2] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 39(6): 1137-1149.
[3] Hong J, Fulton M, Sattar J. Trashcan: A semantically-segmented dataset towards visual detection of marine debris[J]. arXiv preprint arXiv:2007.08097, 2020.
[4] Mahsereci M, Balles L, Lassner C, et al. Early stopping without a validation set[J]. arXiv preprint arXiv:1703.09580, 2017.
[5] Aggarwal L P. Data augmentation in dermatology image recognition using machine learning[J]. Skin Research and Technology, 2019, 25(6): 815-820.
[6] Shorten C, Khoshgoftaar T M. A survey on image data augmentation for deep learning[J]. Journal of big data, 2019, 6(1): 1-48.
[7] Davey S J, Rutten M G, Cheung B. A comparison of detection performance for several track-before-detect algorithms[J]. EURASIP Journal on Advances in Signal Processing, 2007, 2008(1): 428036.
[8] 杨锁荣, 杨洪朝, 申富饶, 等. 面向深度学习的图像数据增强综述[J]. 软件学报, 2025, 36(03): 78-79.
[9] 董子源, 韩卫光. 基于卷积神经网络的垃圾图像分类算法[J]. 计算机系统应用, 2020, 29(8): 199-204.
[10] 马志远, 余粟. 基于 Faster-RCNN 网络的表格检测算法研究[J]. 智能计算机与应用, 2020, 10(12): 24-27.
[11] 张阳婷, 黄德启, 王东伟, 等. 基于深度学习的目标检测算法研究与应用综述[J]. Journal of Computer Engineering & Applications, 2023, 59(18).
[12] 强伟, 贺昱曜, 郭玉锦, 等. 基于改进 SSD 的水下目标检测算法研究[J]. 西北工业大学学报, 2020, 38(4): 747-754.