第 5 章基于目标检测的垃圾桶满溢状态识别

Tuesday 31st October, 2023

本章基于前面讲述的内容, 对于垃圾桶的满溢状态的识别检测进行了实验设计。试图创建一个可以布置在服务器和内存有限的设备上, 在视频或图像条件下给出预测结果的实验模型。于是选择运用 yolox-nano（不基于锚框）与 yolov4-tiny（基于锚框）两个轻量化模型，根据第四章 CSA 模块对于目标特征较好关注的特点，对两个模型进行一些优化改进。

Part I

判定标准

针对于垃圾是否满溢，制定了相关评判标准，并且决定是否提醒，做出流程如图所示：

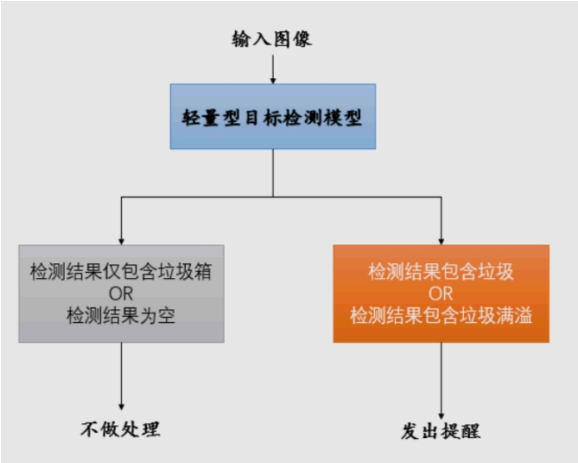

图 1: 判断流程图

Part II

实验环境与参数配置

与第四章实验环境一致, 实验训练 300 轮,batch size 设置为 128 张图片,yolox 的实验策略与原模型一致,yolov4-tiny 使用 adam(Adaptive Moment Estimation) 优化器，动量设置为 0.937，学习率下降方式为余弦退火法，学习率最小为初始学习率的 0.01.

Part III

评价指标

1 预测的准确性

1.1 IoU 指标

在第四章中介绍了精确率和召回率，精确率指的是所有正样本中被预测出来的比例，召回率指的是预测结果中正样本所占的比例。在计算中用到了

TP（True Positive，真正例，正样本被正确预测）；FP（False Positive，假正例，负样本被预测为正样本）；FN（False Negative，假负例，正样本被预测为负样本）。

精确率就是:Precision=$\frac{TP}{TP+FP}$=$\frac{TP}{\text{真实框数目}}$

召回率就是:Recall=$\frac{TP}{TP+FN}$=$\frac{TP}{\text{预测框数目}}$

在目标检测中，主要判别预测框与真实框的重叠部分。而判别真正例常用的指标为 IoU（Intersection of Union），IoU 实际上是测量真实值与预测值的相关度。假定 A 为预测框，B 为真实框，则 IoU 的公式可表示为$\frac{A\cap B}{A\cup B}$，即预测框预测正确的面积与两个锚框面积和之比。如图所示：

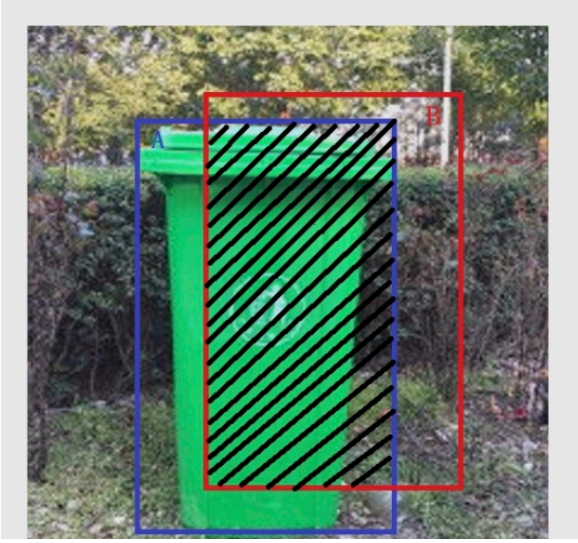

图 2: 预测框与真实框

图像中的阴影部分就是重合部分，与两个矩形框所形成面积之比就是 IoU。最大可以取 1，代表完全重合，最小可以取 0，代表没有重合部分。一般认为 IoU 大于 0.5 时预测结果较为准确。

1.2 CoCo 评价指标

CoCo 评价指标中，每个真实标注框只算一次，会先给 IoU 设定一个值（通常大于 0.5）。

假设设定 IoU 值为 a, 此时
TP：IoU>a 的检测框个数
FP：IoU$\leq a$ 的检测框的个数
FN: 未检测到真实标注框的个数
例如：

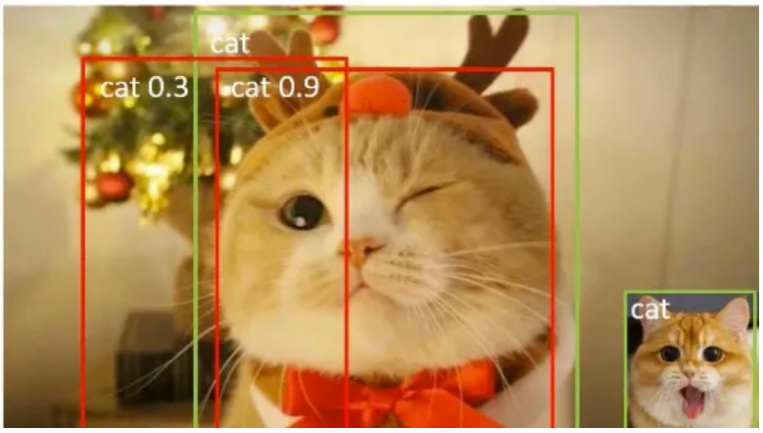

图 3: IoU 数目

根据检测框的大小，很容易得出较大绿色框的 IoU 值大于 0.5，为 FP；较小的 IoU 值小于 0.5，为 FP；还有一只猫没有被检测，为 FN。

根据不同的 Recall 值，与其对应的 Precision 值，可以画出对应图像，就是 P-R 曲线，AP 就是这条曲线下的面积。
而 map 就是各类别 AP 的平均值。即 AP 是衡量学出来的模型在每个类别上的好坏，map 是衡量学出来的模型在所有类别上的好坏。

在本文中，IoU 的值的设置从 0.5 开始，每隔 0.05 设定一个 IoU 阈值，最大阈值为 0.95，总共十个阈值。在每个阈值下计算类别的平均精确率，共得到 10 个 mAP（Mean Average Precision）值，计算这 10 个 mAP 的平均值即为 $\text{mAP}_{0.5:0.95}$。当目标检测任务含有多个类别目标时，先计算每一个类别的平均精确率，再进行平均。同时还采用阈值为 0.5 时的 $\text{mAP}_{0.5}$ 作为评价指标。
除了准确性评价指标之外，本实验还利用参数量来评估模型大小。

Part IV

实验结果

本章节利用 yolox-nano,yolov4-tiny 及改进后的模型进行实验对比。两组实验的训练和测试的输入均为 416×416。下表展示的是对 yolox-nano,yolov4-tiny 进行修改前后的实验结果。

模型	参数量(MB)	$mAP_{0.5:0.95}(\%)$	$mAP_{0.5}(\%)$
yolox-nano	0.8971	75.77	96.7
yolox-nano(+CSA)	0.8983(+0.0012)	76.4(+0.63)	97.0(+0.3)
yolov4-tiny	5.8787	68.12	96.71
yolov4-tiny（DP）	5.5409(-0.3378)	65.23(-2.89)	94.35(-2.36)
yolov4-tiny(DP+CSA)	5.5413(-0.3374)	65.42(-2.70)	94.67(-2.04)

图 4: 实验结果

根据图表可以看出，两个模型再加入 CSA 模块后，都有不同程度的提升,，说明 CSA 模块在一定程度上可以缓解轻量卷积带来的精度损失。

Part V

代码部分

评估部分主要分为三个代码块。

2 coco_classes

这一部分设置了一些参数，主要是对可能检测出的物体进行了罗列。

```
COCO_CLASSES = (
    "person",
    "bicycle",
    "car",
    "motorcycle",
    "airplane",
    "bus",
    "train",
    "truck",
    "boat",
    "traffic light",
    "fire hydrant",
    "stop sign",
    "parking meter",
    "bench",
    "bird",
    "cat",
    "dog",
    "horse",
    "sheep",
    "cow",
    "elephant",
    "bear",
    "zebra",
```


图 5: coco_classes

3 coco

```python
def __init__(
    self,
    data_dir=None,
    json_file="instances_train2017.json",
    name="train2017",
    img_size=(416, 416),
    preproc=None,
    cache=False,
):
    """
    COCO dataset initialization. Annotation data are read into memory
    Args:
        data_dir (str): dataset root directory
        json_file (str): COCO json file name
        name (str): COCO data name (e.g. 'train2017' or 'val2017')
        img_size (int): target image size after pre-processing
        preproc: data augmentation strategy
    """
    super().__init__(img_size)
    if data_dir is None:
        data_dir = os.path.join(get_yolox_datadir(), "COCO")
    self.data_dir = data_dir
    self.json_file = json_file

    self.coco = COCO(os.path.join(self.data_dir, "annotations", self
    remove_useless_info(self.coco)
    self.ids = self.coco.getImgIds()
```

图 6: 初始化

这一函数主要进行初始化，用于获取筛选类别的的 ID，图片的 ID。

```python
def load_anno_from_ids(self, id_):
    im_ann = self.coco.loadImgs(id_)[0]
    width = im_ann["width"]
    height = im_ann["height"]
    anno_ids = self.coco.getAnnIds(imgIds=[int(id_)], iscrowd=False)
    annotations = self.coco.loadAnns(anno_ids)
    objs = []
    for obj in annotations:
        x1 = np.max((0, obj["bbox"][0]))
        y1 = np.max((0, obj["bbox"][1]))
        x2 = np.min((width, x1 + np.max((0, obj["bbox"][2]))))
        y2 = np.min((height, y1 + np.max((0, obj["bbox"][3]))))
        if obj["area"] > 0 and x2 >= x1 and y2 >= y1:
            obj["clean_bbox"] = [x1, y1, x2, y2]
            objs.append(obj)

    num_objs = len(objs)

    res = np.zeros((num_objs, 5))

    for ix, obj in enumerate(objs):
        cls = self.class_ids.index(obj["category_id"])
        res[ix, 0:4] = obj["clean_bbox"]
        res[ix, 4] = cls

    r = min(self.img_size[0] / height, self.img_size[1] / width)
```

图 7: 标注 id

这一函数主要是用于获取满足过滤条件的标注的 ID。

```python
def _cache_images(self):
    logger.warning(
        "\n********************************************************************************\n"
        "You are using cached images in RAM to accelerate training.\n"
        "This requires large system RAM.\n"
        "Make sure you have 200G+ RAM and 136G available disk space for training COCO.\n"
        "********************************************************************************\n"
    )
    max_h = self.img_size[0]
    max_w = self.img_size[1]
    cache_file = os.path.join(self.data_dir, f"img_resized_cache_{self.name}.array")
    if not os.path.exists(cache_file):
        logger.info(
            "Caching images for the first time. This might take about 20 minutes for COCO"
        )
        self.imgs = np.memmap(
            cache_file,
            shape=(len(self.ids), max_h, max_w, 3),
            dtype=np.uint8,
            mode="w+",
        )
        from tqdm import tqdm
        from multiprocessing.pool import ThreadPool

        NUM_THREADs = min(8, os.cpu_count())
        loaded_images = ThreadPool(NUM_THREADs).imap(
            lambda x: self.load_resized_img(x),
```

图 8: 图片的保存与加载

```python
def load_resized_img(self, index):
    img = self.load_image(index)
    r = min(self.img_size[0] / img.shape[0], self.img_size[1] / img.shape[1])
    resized_img = cv2.resize(
        img,
        (int(img.shape[1] * r), int(img.shape[0] * r)),
        interpolation=cv2.INTER_LINEAR,
    ).astype(np.uint8)
    return resized_img
```

图 9: 分辨率修改

这一部分是把获取的图片修改分辨率后保存在磁盘上并且可以进行加载，可以加快读图时间，提高效率。

4 AP

```python
def per_class_AP_table(coco_eval, class_names=COCO_CLASSES, headers=["class", "AP"], colums=6):
    per_class_AP = {}
    precisions = coco_eval.eval["precision"]
    # dimension of precisions: [TxRxKxAxM]
    # precision has dims (iou, recall, cls, area range, max dets)
    assert len(class_names) == precisions.shape[2]

    for idx, name in enumerate(class_names):
        # area range index 0: all area ranges
        # max dets index -1: typically 100 per image
        precision = precisions[:, :, idx, 0, -1]
        precision = precision[precision > -1]
        ap = np.mean(precision) if precision.size else float("nan")
        per_class_AP[name] = float(ap * 100)

    num_cols = min(colums, len(per_class_AP) * len(headers))
    result_pair = [x for pair in per_class_AP.items() for x in pair]
    row_pair = itertools.zip_longest(*[result_pair[i::num_cols] for i in range(num_cols)])
    table_headers = headers * (num_cols // len(headers))
    table = tabulate(
        row_pair, tablefmt="pipe", floatfmt=".3f", headers=table_headers, numalign="left",
    )
    return table
```

图 10: AP 部分

对于输入的不同类别的进行比对计算精确率

```python
def evaluate(
    self,
    model,
    distributed=False,
    half=False,
    trt_file=None,
    decoder=None,
    test_size=None,
):
    """
    COCO average precision (AP) Evaluation. Iterate inference on the test dataset
    and the results are evaluated by COCO API.

    NOTE: This function will change training mode to False, please save states if neede

    Args:
        model : model to evaluate.

    Returns:
        ap50_95 (float) : COCO AP of IoU=50:95
        ap50 (float) : COCO AP of IoU=50
        summary (sr): summary info of evaluation.
    """
    # TODO half to amp_test
    tensor_type = torch.cuda.HalfTensor if half else torch.cuda.FloatTensor
    model = model.eval()
```

图 11: CoCo 评估函数

返回不同 IoU 值下的 AP 值，并求出加和.

```python
def convert_to_coco_format(self, outputs, info_imgs, ids):
    data_list = []
    for (output, img_h, img_w, img_id) in zip(
        outputs, info_imgs[0], info_imgs[1], ids
    ):
        if output is None:
            continue
        output = output.cpu()

        bboxes = output[:, 0:4]

        # preprocessing: resize
        scale = min(
            self.img_size[0] / float(img_h), self.img_size[1] / float(img_w)
        )
        bboxes /= scale
        bboxes = xyxy2xywh(bboxes)

        cls = output[:, 6]
        scores = output[:, 4] * output[:, 5]
        for ind in range(bboxes.shape[0]):
            label = self.dataloader.dataset.class_ids[int(cls[ind])]
            pred_data = {
                "image_id": int(img_id),
                "category_id": label,
                "bbox": bboxes[ind].numpy().tolist(),
                "score": scores[ind].numpy().item(),
                "segmentation": [],
            }  # COCO json format
```

图 12: 格式转换

这一个函数是将所获得的图片 ID，类别，位置信息等转换为 CoCo 数据集中的所需要的类型，并且保存为 json 文件储存。

```python
def evaluate_prediction(self, data_dict, statistics):
    if not is_main_process():
        return 0, 0, None

    logger.info("Evaluate in main process...")

    annType = ["segm", "bbox", "keypoints"]

    inference_time = statistics[0].item()
    nms_time = statistics[1].item()
    n_samples = statistics[2].item()

    a_infer_time = 1000 * inference_time / (n_samples * self.dataloader.batch_size)
    a_nms_time = 1000 * nms_time / (n_samples * self.dataloader.batch_size)

    time_info = ", ".join(
        [
            "Average {} time: {:.2f} ms".format(k, v)
            for k, v in zip(
                ["forward", "NMS", "inference"],
                [a_infer_time, a_nms_time, (a_infer_time + a_nms_time)],
            )
        ]
    )

    info = time_info + "\n"
```

图 13: 预测

这部分是对预测后的结果与真实值比对后，得到的每个类别的 AP 值与 AR 值并且进行相关信息的输出。