基于车载视角下多路标牌检测识别算法研究

严松舟 汪俊轩 吕法名

Contents

Abstract

本文围绕车载视角下的多路标牌检测问题，在对任务和数据的深度分析的基础上，提出采样策略和数据增强的方法应对其中的特点和挑战，并细致地试验了yolov8系列模型的微调训练以提高检测能力。我们也采用包括tensorrt加速在内的推理加速方法来追求更高的推理性能。最终在相应比赛中，我们取得了综合分数0.9065，排名第10的成绩。

1 引言

车载道路标牌识别技术（Vehicle mounted road sign recognition，VMRS）是指在车辆行驶过程中，通过车辆上的传感器、摄像头等设备，实时采集道路交通标志（Road sign）图像，并识别出标志中的文字、颜色、形状等信息，从而实现对道路标志的识别和理解，最终实现对交通状况的分析和监测，提高道路的安全性和效率。

车载道路标牌识别技术是一项重要的自动驾驶技术，对于自动驾驶汽车来说，实时识别道路标志是至关重要的。因为车辆需要根据道路标志的指示信息，遵守交通规则，避免行驶过程中发生交通事故。

车载道路标牌识别算法旨在使用车载摄像头捕捉的实时图像，识别并分类路标和交通标志，提供相关的信息给驾驶员或自动驾驶系统。

在本文的研究中，我们采用了独特的采样方式解决类别不平衡的问题，并且基于实际情况进行数据增强，本文采用yolov8模型[1]进行训练，并且使用tensorrt和推理预热的方法来提高模型的性能分。

2 数据集分析

2.1 目标性质

数据集中的路牌目标有以下重要性质：

- 高度一致：
  路牌通常是标准化的，同一类路牌在颜色、图案上相差很小。虽然老化、视角等因素依然会导致一些差别，但总体上对模型的泛化能力要求较弱。

- 小目标：
  样例集中可以看出，路牌的目标框通常是在具体的图案上，平均尺寸较小，且存在距离相对远的目标，这要求模型有相当的小目标检测能力。

- 丰富背景信息：
  车载视角+路牌的场景目标组合提供了丰富的先验信息。虽然目标框局限于图案，但一般地，目标的位置会与承载它的实体，如路牌、路标杆等高度相关，且在车载视角下有相对固定的位置区间，这些可能也是隐藏的特征。

2.2 模型选择

选择模型时，我们主要考虑以下因素：

- 推理性能：
  该任务有较高的实时性，并且竞赛中也确实有性能分，需要较高的推理性能，因此选择YOLO技术路线的模型。

- 小目标检测能力：
  模型需要具备相当的小目标学习和检测能力，在对比了yolov5和yolov8的相应指标后，我们选择使用yolov8系列模型。

[1] https://github.com/ultralytics/ultralytics

2.3 数据集分布

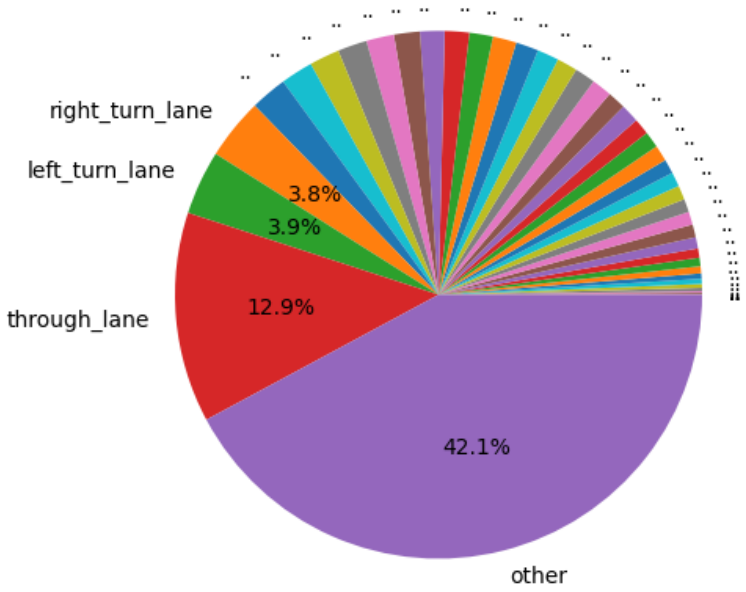

Figure 1: 数据集类别分布

比赛提供的数据集在分布上有两个显著特点：

- 类别分布不均衡：
  通过比赛中给出的数据集分布表可以看出，数据集分布极不均衡，虽然实际上的数据分布和网页上提供的有所出入，但总体规律是一致的，大致的数据分布如图1所示。
- 存在混杂类：数据集中存在一个庞大的other类，从样例集中可以看出，other应该包含了许多不同种类的路牌，这意味着这一类可能并不具备和其他类别同粒度级别的特征，将相对确定的类别与other区分开是一个挑战。

2.4 采样方案

在解决类别分布不平衡问题时，采用关注稀疏类别的Focal Loss是一个办法，但考虑到比赛评分机制并不关注稀疏类别，且有些类别样例数实在太少，我们调整采样策略来缓解不平衡问题：

- 增加采样相对稀疏的类：
- 舍弃样例数极少的类：

调整后，数据集的类别分布如图2所示。

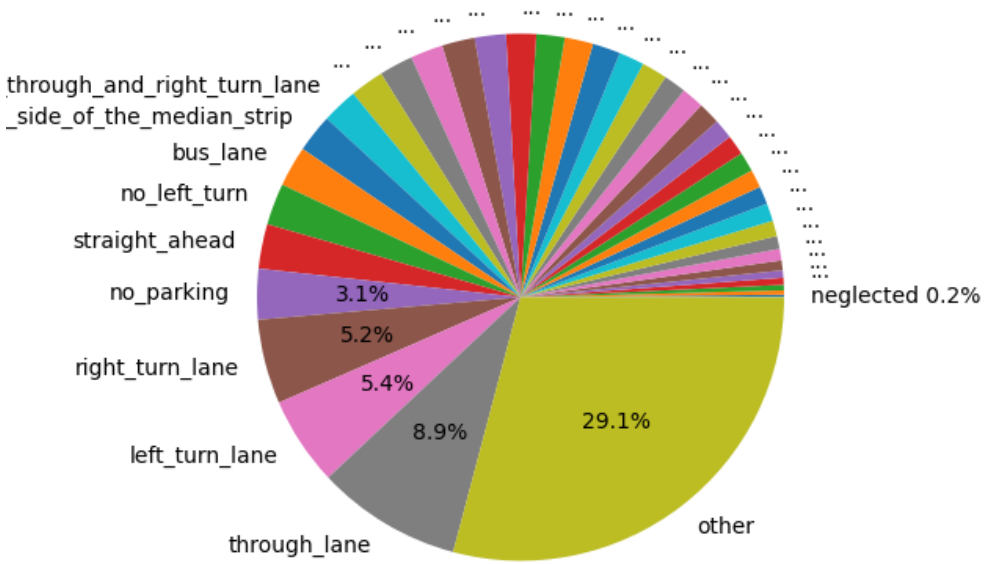

Figure 2: 调整后类别分布

3 数据增强

3.1 基本数据增强方法的选择

目标的性质对可用的数据增强方法作出了限制。例如，基于翻转、旋转的变换很可能破坏路牌的含义，甚至造成多类标牌的混淆。

此外，基于裁剪、拼贴的方法则可能破坏目标框周围的背景信息，损失路牌、路标杆等的特征。例如，我们在初步实验yolov8时，就测试了自带的mosaic数据增强，发现几乎总是产生负面影响。 因此，我们认为必须结合任务的现实特征，设计针对性的数据增强策略。

3.2 基于情景的数据增强

设计数据增强策略时，我们提出通过数据增强模拟现实可能出现的变化情况，从而提升模型泛化能力。基于这一思想，我们主要试验和应用以下两种来自OpenCV的数据增强[2]：

- 风格滤镜：
  我们基于现实中可能存在的昏暗场景、雨天、雾天等场景构造了相应的模拟滤镜，对图像进行相应变换。
  试验发现，昏暗滤镜对效果有一定提升，雨天滤镜影响不大，雾气滤镜则反而导致效果下降。

- 透视变换：
  我们认为观察路牌的透视关系，例如远近和方向，可能是同类路牌不同表现特征的最主要原因，尤其是对于稀疏类别，模型更需要视角不同的样例来学习真正的特征。
  我们通过分析目标框的方位和大小，决定采用的变换操作，主要包括放大、方向上的变换，并相应调整目标框。

示例图片和应用变换的效果见附录。

[2] `https://opencv.org/`

4 实验

4.1 实验环境

4.1.1 操作系统和软件环境

- 操作系统：Ubuntu 18.04
- CUDA 版本：11.1
- cuDNN 版本：8
- PyTorch 版本：1.10
- ATC Toolkit 版本：5.0.4
- JupyterLab 版本：3.3
- Visual Studio Code (VSCode) 版本：4.4.0
- PyTENSORRT 版本：8.2.3

4.1.2 硬件配置

- CPU：4核心
- 内存：32GB

4.2 训练

4.2.1 优化器选择

在本研究中，我们进行了一系列实验来比较Adam优化器和SGD优化器在训练深度神经网络模型时的表现。我们观察到了一个有趣的现象，即在训练的前几个epoch中，Adam优化器的性能明显优于SGD优化器，如表1所示。Adam在初始阶段能够更快地收敛，实现了更低的损失值。这可以归因于Adam优化器的自适应学习率和动量的特性，使得模型更快地找到了全局最优点。

然而，随着训练的继续，我们注意到了一个有趣的趋势。随着epoch的增加，Adam优化器逐渐达到了一个性能瓶颈，模型的性能不再有显著的提升，如表1所示。相比之下，SGD优化器在后面的epoch中依然保持相对稳定的性能提升。这个现象与Adam优化器的自适应性有关，有时候可能会导致在全局最优点附近摆动，从而使模型的性能不再进一步提升。

最终，考虑到这一发现，我们决定选择SGD优化器作为最终的优化方法。尽管Adam在初始阶段表现出色，但SGD在长期训练中表现出更好的性能稳定性和模型收敛性，如表1所示。这个选择使得我们的模型在训练的后期能够更好地收敛到全局最优点，从而实现更好的性能。

我们的实验结果表明，优化器的选择在深度学习中具有重要意义，并且需要根据具体问题和数据来进行调整。这个发现为深度神经网络的训练提供了有益的指导，并有助于在实际应用中取得更好的性能。

Table 1: yolov8m训练过程中在验证集上的mAP50的变化

epoch	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
SGD	0.556	0.609	0.605	0.659	0.699	0.721	0.735	0.752	0.774
Adam	0.578	0.613	0.636	0.675	0.713	0.734	0.745	0.756	0.769

Table 2: yolov8m训练过程中在验证集上的mAP50的变化 续

epoch	10	11	12	13	14	15
SGD	0.788	0.799	0.818	0.825	0.826	0.833
Adam	0.770	0.778	0.785	0.793	0.804	0.810

4.2.2 模型性能分析

在本研究中，我们进行了一系列实验，使用Tesla T4GPU训练了Yolov8系列模型（包括yolov8n、yolov8s和yolov8m）。这些实验的目的是研究不同模型在参数规模和训练时间方面的性能差异，并以此为基础为我们的目标任务选择最合适的模型。

我们首先关注了模型的参数规模以及训练一个epoch所需的时间。下表3总结了我们的实验结果:

Table 3: 模型性能对比

模型	参数规模	训练一个小时的时间（小时）	训练时平均算力占用
yolov8n	225 layers, 3019623 parameters	12.5	48%
yolov8s	225 layers, 11153015 parameters	13	95%
yolov8m	295 layers, 25882375 parameters	25	98%

从表3可以看出，yolov8n相对较小的参数规模并没有在训练时间方面表现出显著的优势，尤其是与yolov8s相比。进一步的研究发现，yolov8n在训练过程中只能利用48%的显卡算力，而另外两个模型的利用率接近100%。这个发现揭示了为什么yolov8n的训练时间没有明显的竞争优势。因此，从训练时间成本的角度来看，我们决定弃用yolov8n，因为在相同的时间内，它无法达到yolov8s的性能水平。

对于yolov8s和yolov8m，它们各自具有不同的优势。从中期训练阶段的单位训练时间的mAP50提升值来看，这两个模型表现相似，并且明显高于yolov8n。然而，它们在其他方面有不同的特点。具体而言，yolov8s具有更快的推理速度，适用于强调实时性的应用场景。与此不同，yolov8m在模型性能的上限方面更有潜力，特别是在训练时间充足且可以容忍较慢推理速度的情况下。

4.2.3 训练数据及分析

将数据集按照6:1的比例划分为训练集和验证集，分别在yolov8n.pt，yolov8s.pt，yolov8m.pt三个预训练模型的基础上训练，由于比赛计算分数时，既考虑模型分，又考虑性能分，且不同的模型收敛速度不同，训练所需时间也不同，因此此处只展示40epoch来展示我们的训练过程。完整的训练数据可视化和训练中使用的超参数见附录。

优化器经过测试SGD效果优于ADAM，因此训练中优化器都是采用SGD，执行度损失、类别损失和变形损失以及mAP50的变化情况如下:

Table 4: Box Loss of Three Models

epoch/model	5	10	15	20	25	30	35	40
yolov8n	1.297	1.147	1.086	1.039	0.9995	0.9635	0.8805	0.8355
yolov8s	1.176	1.022	0.9615	0.9192	0.8763	0.842	0.7733	0.7228
yolov8m	1.101	0.9447	0.8698	0.796	0.7253	0.6706	0.6441	0.62

这个表格显示了三个不同的模型在40个epoch下的box loss的变化。box loss是一个衡量目标检测模型性能的指标，它反映了模型预测的边界框和真实的边界框之间的差异。box loss越小，说明模型越准确地定位了目标。

从表格中可以看出，三个模型的box loss都随着epoch的增加而逐渐降低，说明模型都在不断地学习和优化。其中，yolov8m的box loss最低，达到了0.62，表明它是最优秀的模型。yolov8s的box loss次之，为0.7228，也表现出了较好的效果。yolov8n的box loss最高，为0.8355，相比于其他两个模型，还有一定的提升空间。

Table 5: Class Loss of Three Models

epoch/model	5	10	15	20	25	30	35	40
yolov8n	1.304	1.035	0.9308	0.858	0.8029	0.7519	0.647	0.5886
yolov8s	1.076	0.849	0.7565	0.702	0.6437	0.5966	0.5173	0.4581
yolov8m	0.9842	0.7531	0.6548	0.5636	0.4739	0.4051	0.37	0.3508

这个表格显示了三个不同的模型在40个epoch下的class loss的变化。class loss是一个衡量目标检测模型性能的指标，它反映了模型预测的类别和真实的类别之间的差异。class loss越小，说明模型越准确地识别了目标。

从表格中可以看出，三个模型的class loss都随着epoch的增加而逐渐降低，说明模型都在不断地学习和优化。其中，yolov8m的class loss最低，达到了0.3508，表明它是最优秀的模型。yolov8s的class loss次之，为0.4581，也表现出了较好的效果。yolov8n的class loss最高，为0.5886，相比于其他两个模型，还有一定的提升空间。

Table 6: Deformable Loss of Three Models

epoch/model	5	10	15	20	25	30	35	40
yolov8n	0.8873	0.8607	0.8532	0.8453	0.8406	0.8344	0.8287	0.8221
yolov8s	0.8687	0.8451	0.8366	0.8298	0.8253	0.8199	0.8155	0.8093
yolov8m	0.8579	0.8344	0.8255	0.8183	0.8121	0.8035	0.7978	0.795

这个表格显示了三个不同的模型在40个epoch下的deformable loss的变化。deformable loss是一个衡量目标检测模型性能的指标，它反映了模型预测的形变和真实的形变之间的差异。形变是指模型对输入图像的采样和池化的位置进行的偏移，以适应目标的几何变化。deformable loss越小，说明模型越灵活地适应了目标的形状。

从表格中可以看出，三个模型的deformable loss都随着epoch的增加而逐渐降低，说明模型都在不断地学习和优化。其中，yolov8m的deformable loss最低，达到了0.795，表明它是最优秀的模型。yolov8s的deformable loss次之，为0.8093，也表现出了较好的效果。yolov8n的deformable loss最高，为0.8221，相比于其他两个模型，还有一定的提升空间。

Table 7: mAP50 of Three Models

epoch/model	5	10	15	20	25	30	35	40
yolov8n	0.539	0.666	0.714	0.729	0.741	0.75	0.759	0.772
yolov8s	0.647	0.751	0.79	0.808	0.823	0.835	0.86	0.874
yolov8m	0.699	0.788	0.833	0.86	0.873	0.886	0.891	0.895

这个表格显示了三个不同的模型在40个epoch下的mAP50的变化。mAP50是一个衡量目标检测模型性能的指标，它反映了模型预测的边界框和真实的边界框之间的重叠程度和准确度。mAP50越高，说明模型越能够正确地识别和定位目标。

从表格中可以看出，三个模型的mAP50都随着epoch的增加而逐渐升高，说明模型都在不断地学习和优化。其中，yolov8m的mAP50最高，达到了0.895，表明它是最优秀的模型。yolov8s的mAP50次之，为0.874，也表现出了较好的效果。yolov8n的mAP50最低，为0.772，相比于其他两个模型，还有一定的提升空间。

综合上文中对于模型性能的分析，考虑到比赛是训练时间有限的情况，我们最终选择了yolov8s模型以获得更好的性能。

4.2.4 稀疏类别检测效果的提升

我们发现，采用基于数据增强的数据分布调整策略，确实能够显著提升稀疏类别检测的效果。相应训练结果如下（对照/应用后）：

Table 8: 稀疏类别检测效果

class	instances	precision	recall	mAP50
through & right turn	37/49	0.586/0.942	0.459/0.98	0.414/0.984
motor vehicle lane	23/116	0.322/0.928	0.478/0.931	0.471/0.968
stop and yield	64/98	0.674/0.967	0.875/0.98	0.879/0.987
through lane	1241/1106	0.926/0.937	0.938/0.941	0.942/0.953

可以看出，在样例较多的类别through lane提升不大的训练过程中，样例稀疏的类别上的表现出现非常显著的提升。需要说明的是，此处并非已经接近效果提升的边界，在训练后期through lane类别的mAP50可达到0.978，这更反映出调整分布的效果。事实上，在得到更多采样的35个类别中，19个呈现出效果的明显提升（mAP50变化值>0.1）。

4.3 推理

在本研究中，我们通过整合NVIDIA TensorRT加速库，成功地提升了YoloV8目标检测模型的推理速度[3]。TensorRT是一款专为深度学习推理任务设计的高度优化的库，它利用了多种技术来加速神经网络的推理过程。

TensorRT的核心加速原理包括网络优化、层融合、精度混合和并行计算。首先，TensorRT通过网络优化技术自动地优化网络结构，去除不必要的层和计算，从而减少了推理时的计算量。其次，TensorRT采用层融合技术，将多个卷积和激活函数合并为一个单一的层，减少了内存访问和计算的开销。此外，TensorRT还支持精度混合，允许在保持模型准确性的同时减少计算的数值精度，从而提高了推理速度。最后，TensorRT能够充分利用GPU的并行计算能力，进一步提高了推理效率。

然而，值得注意的是，在实验过程中，我们观察到了一个有趣的现象：单张图片的平均推理时间在测试过程中呈现出前面一部分显著大于后面一部分的趋势。为了解决这个问题，我们采取了一项额外的策略，即在实施推理之前先对模型进行1000张图片的预热推理，以达到稳定性。结果发现，这一策略显著提升了模型的性能。

此外，我们还观察到调整推理过程中使用的confidence threshold（conf）和intersection over union（iou）参数对model-score也有影响，但由于model-score的评分机制是黑盒，我们无法直接了解其具体评分机制。因此，我们不得不逐个尝试不同的超参数组合，以寻找最优配置，以实现更高的检测性能。

以下是我们的实验数据:

Table 9: 性能分（1s内推理的图片数量）

	无	tensorrt	tensorrt+预热推理
yolov8n	61.4046	102.1668	121.9999
yolov8s	43.5320	90.2180	102.4556
yolov8m	38.7026	58.7017	62.6147

Table 10: 平均单张图片耗时（ms）

	无	tensorrt	tensorrt+预热推理
yolov8n	16.2854	9.7879	8.1967
yolov8s	22.9716	11.0843	9.7603
yolov8m	25.8381	17.0352	15.9707

这些数据明确地表明了TensorRT的加速效果以及预热策略的有效性，为实时目标检测应用提供了强有力的支持。

[3] https://github.com/NVIDIA/TensorRT

5 分析与讨论

5.1 结果分析

从上述实验数据中可以看出，我们的模型都能够在一定程度上实现目标检测的任务。我们的模型在box loss, class loss, deformable loss和mAP50这四个指标上都有不同程度的改善，说明模型在定位、识别、适应和评价目标方面都有进步。其中，yolov8m和yolov8s表现相近，但是yolov8m的性能分低于其他模型，并且yolov8s在40个epoch后仍然没有达到收敛，继续训练下去分数更高，综合考虑，我们最终选择yolov8s。

我们的训练过程还有一些不足之处，需要进一步改进。首先，我们的模型的训练时间较长，可能需要更高效的优化算法或者更强大的计算资源。其次，我们的模型的泛化能力还有待验证，可能需要在更多的数据集上进行测试和调整。最后，我们的模型的可解释性还有待提高，可能需要更多的可视化工具或者更多的分析方法。

除此之外，相比于不使用任何加速技术，使用tensorrt和预热推理可以显著提高性能分，降低平均单张图片耗时，因此，tensorrt和预热推理可以有效地优化YOLOv8模型的推理效率。

不同大小的YOLOv8模型之间存在一个准确度和速度的权衡。一般来说，模型越大，准确度越高，但速度越慢。反之，模型越小，准确度越低，但速度越快。因此，在选择合适的YOLOv8模型时，需要根据不同的应用场景和性能需求，进行合理的折衷，在此处，综合考虑到性能分和模型分，我们最终采用yolov8s作为我们的目标检测模型。

5.2 讨论

5.2.1 YOLOv8s的优势

- YOLOv8s的模型大小和计算量相对YOLOv8n和YOLOv8m处于一个适中的程度，这意味着它兼顾了运算速度，也保证了较高的准确率，从而提高了模型的实用性和效率。
- YOLOv8s的模型经过了稀疏化、剪枝和finetune等优化步骤，这可能使得模型更加简洁和稳定，减少了冗余和过拟合的风险，从而提高了模型的泛化能力和鲁棒性。
- YOLOv8s的模型可能更适合于赛题的数据集和任务，赛题的数据具有小目标、高度一致和丰富背景信息的特点。

5.2.2 数据增强和采样策略的效果

数据显示，经过静态数据增强调整数据集分布后，模型在稀疏类别的检测能力上有了明显的提升，证实了我们的方法成功实现主要的目的。

另一方面，轻度数据增强对多数类别的检测能力提升相对有限，这可能是因为最终应用的风格滤镜相对单一和简单，无法覆盖复杂的现实情况。结合车载视角的性质，可以尝试添加水迹等更复杂的视觉效果来更真实地模拟现实情况。

最后，虽然样例集中的图片都相对清晰，但也有一张相对模糊的图片，如果采用动态模糊的数据增强方式，将对模型的泛化能力提出更高挑战，而效果很大程度上取决于完整训练和测试集中相应图片的比例，但也不失为值得一试的方法.

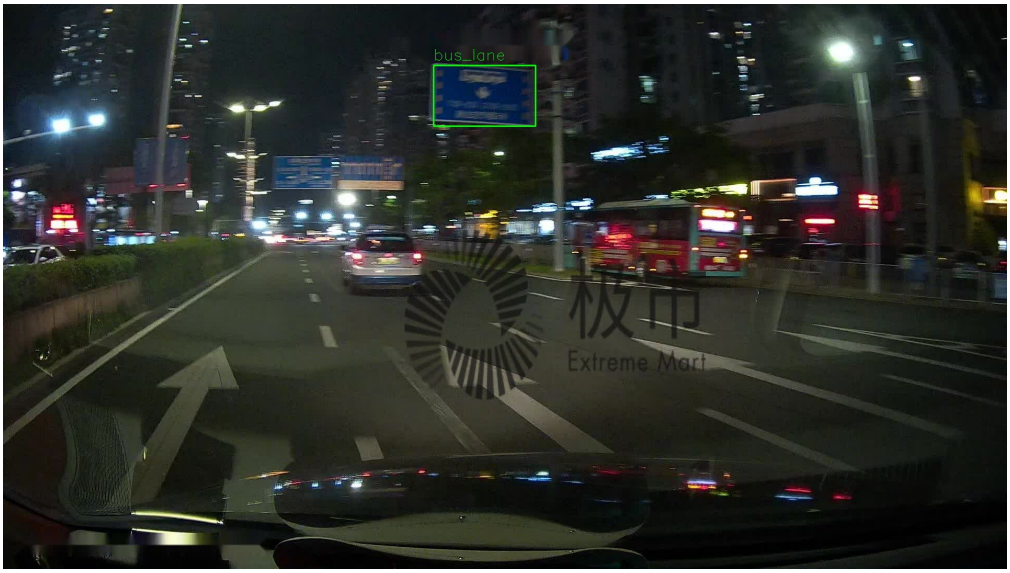

Figure 3: 模糊样例

5.2.3 遇到的问题与挑战

- 训练资源有限：
  平台训练资源有限，训练模型时往往需要一到两个小时的排队时间，导致试错成本过高，限制了我们的训练时间和迭代次数。
- 训练集数据无法直接接触：
  出于知识产权保护、隐私保护等原因，我们无法直接获得训练数据，无法对训练数据进行分析探索来调整我们的模型。
- 数据集不平衡：
  数据集存在严重的不平衡问题，个别类别在训练集和验证集上甚至都只有几个。

6 总结与不足

6.1 总结

在本次实验中，我们实现了YOLOv8模型的训练和部署。在数据预处理阶段，我们根据赛题所给数据具有分布不均衡以及存在混杂类的特点，使用增加采样相对稀疏的类和舍弃样例数极少的类的方法来缓解不平衡问题。又考虑的路牌的特点，避免了采取基于旋转、翻转的变换，除此之外，基于裁剪和拼贴的变换可能会破坏掉目标周围的背景信息，最终我们决定使用基于情景的数据增强，如风格滤镜和透视变换。

在模型训练阶段，我们比对了SGD和Adam两种优化器在实验中的表现，最终选择SGD优化器，在模型选择方面，我们详细比较了yolov8、yolov8m和yolov8s三个模型的在各项指标上的表现，最终选择yolov8s作为我们的目标检测模型。

为了提高我们模型的性能，我们使用了tensorrt和预热推理来加速模型的推理过程，最终我们实现了在10ms内完成一张图片的推理。

经过上述的实验，我们最终在比赛中取得了0.8961的模型分和102.4556的性能分。

6.2 不足与反思

我们的实验也依然存在很多的不足之处：

- 缺乏可解释性：
  由于比赛限制，我们无法给出模型的可视化结果，比如模型的特征图，激活图，热力图等，这些可以帮助我们更直观地看到模型是如何识别和定位路牌的，以及模型关注的区域和特征。
  除此之外，也没有给出模型的注意力机制的结果，比如模型的注意力权重，注意力分布，注意力矩阵

等，这些可以帮助我们更清晰地看到模型是如何分配和调整注意力的，以及模型的注意力是否合理和有效。

- 缺乏对抗样本测试:
  对抗样本可以帮助我们更深入地看到模型的鲁棒性和稳定性，以及模型是否容易受到对抗攻击的影响。

- 缺乏超参数调优:
  超参数调优可以帮助我们找到最优的模型参数，以提高模型的性能和泛化能力。在这次实验中，我们没有进行充分的超参数优化，模型的性能挖掘不足。

- 比赛经验缺乏:
  比赛环境时间紧、训练资源有限且试错成本较高，我们缺乏参加相关比赛的经验，很多地方都没做到极致。

在经历了这次比赛之后，通过对我们对比赛的总结与反思，我们更加深刻的理解了课堂上学习到的知识，也积累了比赛、调优的经验，相信在下一次相关的比赛或工作中，我们能够做到更好。

A 附录

A.1 数据增强变换样例

注：为视觉效果，参数可能与实际训练中不同。

Figure 4: 风格滤镜

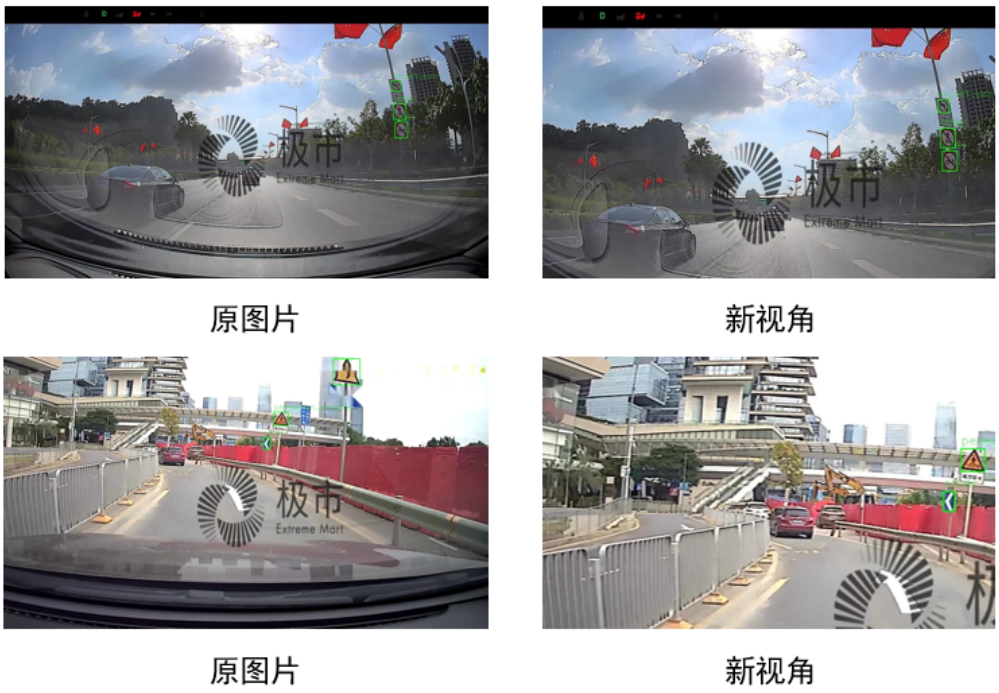

Figure 5: 透视变换

A.2 代码文件结构

注：此处代码是本地开发的原型，与工作台中实际运行的版本可能存在细微差异。

- **run.sh**–训练脚本，有关实际训练的步骤请参考该脚本中的调用。
- **data.yaml**–按yolov8要求格式的配置文件，指定类别和必要文件位置。
- **split_train_val.py**–数据集划分脚本，将数据集划分为训练集和验证集。
- **voc_label.py**–数据集处理脚本，将数据集标签转换为yolov8要求的格式、应用静态数据增强。
- **aug.py**–实现数据增强的函数原型。
- **ji.py**–按极市平台要求编写的测试脚本，实现推理过程中的初始化和单张图片的处理。

A.3 超参数

参数名	取值	说明
lr_0	0.01	初始学习率
cos_{lr}	True	余弦学习率衰减
$optimizer$	SGD	优化器
$momentum$	0.937	SGD动量
$epoch$	40	（每次）训练轮数
$resume$	False	断点续训
$batchsize$	16	批次大小
d_b	-0.5	昏暗滤镜亮度
d_c	0.8	昏暗滤镜对比度参数
p_{aug}	0.1	非稀疏类别增强概率
$conf$	0.5	推理置信度阈值
iou	0.7	推理交并比阈值
$imagesize$	640	推理图像尺寸

Table 11: 超参数表

A.4 训练过程数据可视化

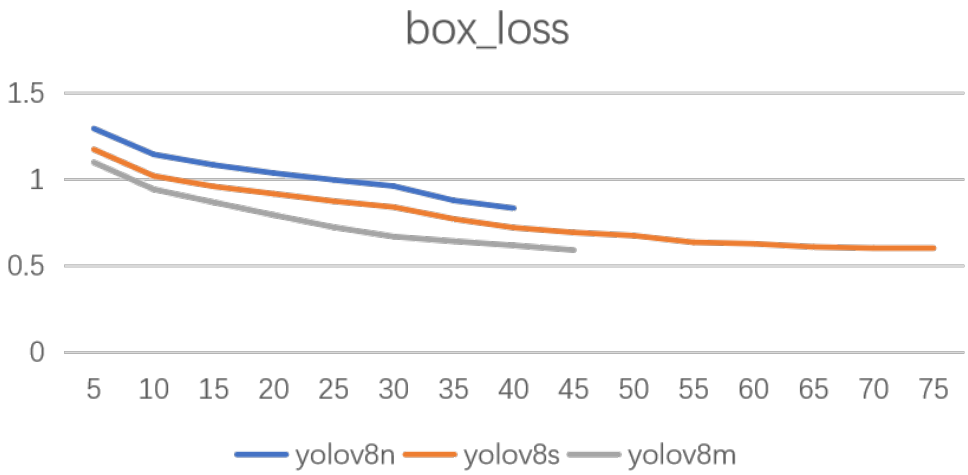

Figure 6: box loss

Figure 7: class loss

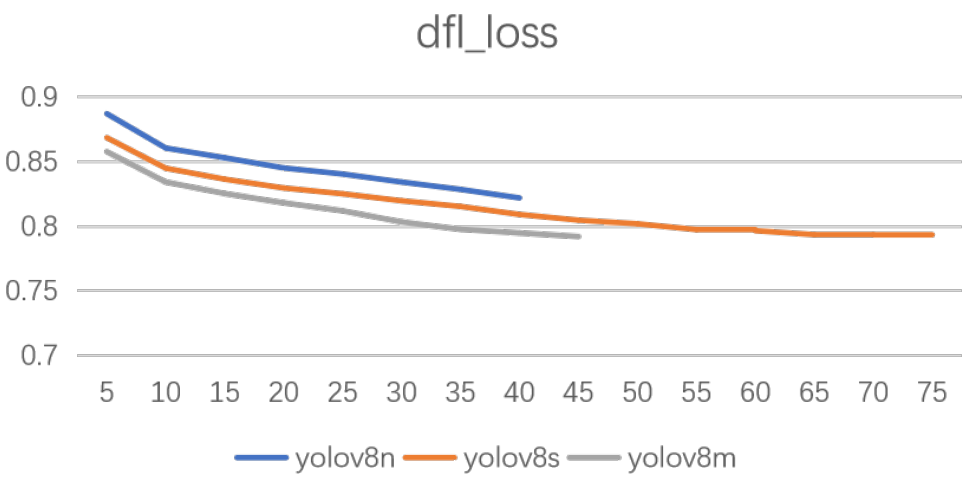

Figure 8: deformable loss

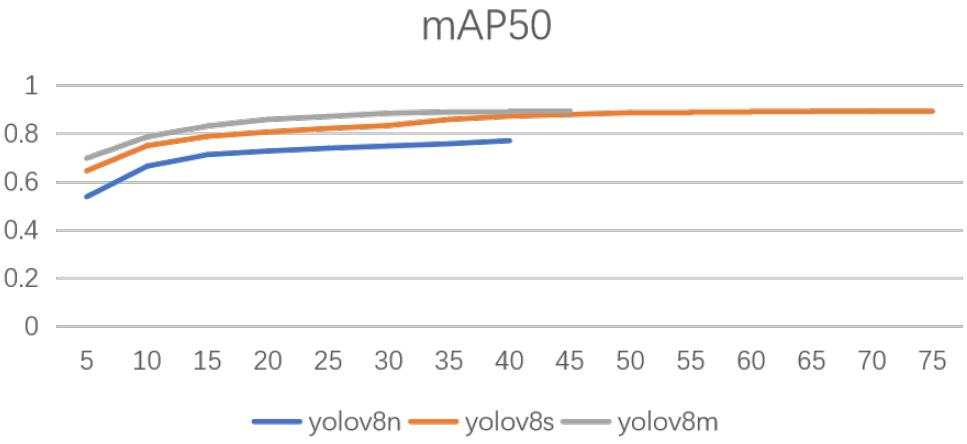

Figure 9: mAP50 loss

A.5 比赛结果

10	怪异奇男子	0.8961	102.4556	0.9065	2023-12-04 11:20:12

Figure 10: 比赛结果