

|      |            |
|------|------------|
| 队伍编号 | MCB2301959 |
| 赛道   | A          |

## 基于 YOLOv8 的坑洼道路目标自动检测与特性提取

### 摘要

坑洼道路的高效检测可有效地推进无人驾驶、地质勘探、航天科学等领域的研究与应用。由于道路图像的环境背景复杂及多变性，传统算法难以提取有效特征，效果较差。因此，本文旨在建立**基于 YOLOv8 建立的图像目标检测与分割模型**，对坑道路坑洼的特性进行提取。

针对问题一，需要建立更精准的识别道路坑洼边缘特性的模型，用于完成道路避障、道路修复等工作。因此考虑对图像的坑洼进行**目标检测**，在此基础上对坑洼进行**图像分割**，从而可有效地提取坑洼的边缘。首先，结合初赛分析的相关结论，对道路坑洼的特征进行**机理分析**，包括**原图、热力图、RGB 直方图、灰度直方图的对比、边缘及轮廓检测、以及阈值分割**，有效地从机理层面分析其特性，为模型的建立提供机理层面的支撑。之后，对图像文件进行**预处理**，包括**人为再分类、对标记错误的图像修正，并剔除异常图像**，防止错误的标记影响模型效果；**文件夹分类**，方便后续模型数据的读取；**原数据集保留与拓充；测试集数据预读取**，防止破损文件影响预测过程。随后，建立**基于 YOLOv8s 的图像分割 (Segment) 及 YOLOv8n 的图像检测 (Detect)** 模型，对道路坑洼特性进行训练学习、验证、测试。在此之前，还需要对图像的坑洼处（边缘）进行标注，本文采用**ISAT-SAM 及 Labelimg** 分别对坑洼的区域及边缘进行标注，之后对标注文件进行**格式转换**，方便 YOLOv8 的读取，随后**划分训练集、测试集、验证集**，再对训练集采用**高斯滤波、双边滤波、旋转方法**对数据进行**初次增强**，最后配置训练所需文件，并开始训练模型。随着迭代的次数的增加，模型效果逐渐上升，且趋于稳定，待模型训练完成后，取迭代过程中**最优模型**对测试集数据的坑洼边缘进行提取，以“em5acvux.jpg”为例，提取效果见图 20。同时对模型效果进行详细分析，详见**模型的评估**，对于坑洼边缘的检测与分割，**精确率最高可分别达 94.43%、90.48%，mAP50 最高可分别达 0.49485、0.51992**。其余指标结果详见表 6，模型效果良好。

针对问题二，需要估算坑洼面积占比，因此考虑利用图像检测模型预测输出的坑洼边缘像素坐标，估算其像素面积，进而计算坑洼面积占比。同时，需要考虑到预测的坑洼的**多样复杂性**，可能存在**多个互不重叠的坑洼、边缘重叠的坑洼、坑洼标记重复**等情况，因此需要进行对应的处理，具体见**道路坑洼的面积占比估算**。此外，该方法需要考虑的情况较多，且估算存在一定误差。因此，提出一种**简便且准确**的方法，即利用**YOLOv8s-Segment** 模型提取的坑洼区域，进行**阈值分割、像素点二值化**，求目标像素点占整体像素点比值，即可较为准确地求得坑洼占比。本文采用坐标法进行估算，并将结果保存至“test\_result2.csv”中。

最后，本文对所建立的模型的优缺点进行了中肯的评价、提出了模型的改进措施以及对模型进行了一定推广。

关键词：计算机视觉；YOLOv8；图像检测；图像分割；可视化多维评估

# 目录

|                                 |    |
|---------------------------------|----|
| <b>一、 问题的提出</b> . . . . .       | 1  |
| 1.1 问题背景 . . . . .              | 1  |
| 1.2 问题要求 . . . . .              | 1  |
| <b>二、 问题的分析</b> . . . . .       | 1  |
| 2.1 初赛总结 . . . . .              | 1  |
| 2.2 问题的整体分析 . . . . .           | 2  |
| 2.3 各问逐一分析 . . . . .            | 3  |
| 2.4 行文思路 . . . . .              | 3  |
| <b>三、 模型的假设</b> . . . . .       | 3  |
| <b>四、 符号说明</b> . . . . .        | 4  |
| <b>五、 模型的建立与求解</b> . . . . .    | 4  |
| 5.1 初赛研究相关结论、坑洼特征分析 . . . . .   | 4  |
| 5.1.1 原图、热力图对比 . . . . .        | 5  |
| 5.1.2 RGB、灰度直方图 . . . . .       | 5  |
| 5.1.3 边缘、轮廓检测 . . . . .         | 6  |
| 5.1.4 阈值分割 . . . . .            | 7  |
| 5.2 图像文件预处理 . . . . .           | 7  |
| 5.2.1 人为再分类 . . . . .           | 8  |
| 5.2.2 文件夹分类 . . . . .           | 9  |
| 5.2.3 原数据集的保留与拓充 . . . . .      | 9  |
| 5.2.4 测试集数据预读取 . . . . .        | 9  |
| 5.3 YOLOv8 模型简述 . . . . .       | 9  |
| 5.4 坑洼特性提取模型的训练 . . . . .       | 11 |
| 5.4.1 图像坑洼标注 . . . . .          | 11 |
| 5.4.2 标注文件格式转换 . . . . .        | 11 |
| 5.4.3 划分训练集、测试集、验证集 . . . . .   | 12 |
| 5.4.4 训练集图像初次增强 . . . . .       | 12 |
| 5.4.5 训练文件配置 . . . . .          | 14 |
| 5.4.6 模型的训练 . . . . .           | 14 |
| 5.5 测试集图像坑洼特性预测 . . . . .       | 15 |
| 5.5.1 道路坑洼的边缘提取 . . . . .       | 15 |
| 5.5.2 道路坑洼的面积占比估算 . . . . .     | 16 |
| 5.6 模型的评估 . . . . .             | 19 |
| 5.6.1 模型评估指标 . . . . .          | 19 |
| 5.6.2 YOLOv8s-Segment . . . . . | 21 |
| 5.6.3 YOLOv8n-Detect . . . . .  | 23 |

|                   |    |
|-------------------|----|
| <b>六、模型的评价与推广</b> | 23 |
| 6.1 模型的评价         | 23 |
| 6.2 模型的推广         | 24 |
| <b>参考文献</b>       | 25 |
| <b>附录</b>         | 26 |

## 一、问题的提出

### 1.1 问题背景

坑洼道路的检测与识别工作是推动自动无人驾驶、地质勘探、航天科学及自然灾害等领域研究和应用的不可或缺的计算机视觉任务。然而传统的分类算法往往因坑洼图像的复杂及多变性，不能取得较好的效果。近年来，深度学习邻域技术的发展为坑洼道路的检测、为当前存在的亟待解决的问题提供了新的思想。

在实际工程应用中，非结构化的坑洼道路检测将遇到复杂多样的环境，例如，道路周围环境存在差异、光照条件变化等，这无疑增加了基于视觉检测道路的难度<sup>[5]</sup>。道路坑洼的检测是实现智能车辆导航的基础，只有快速准确地检测出道路的可行驶区域才能够真正实现自动驾驶。

深度学习拥有很强的特征提取与表示能力，可从图像中提取重要特征。于坑洼道路检测和识别而言，其可识别坑洼的轮廓、纹理、形态等特征，并将上述特征转换为更易分类的表现形式。在深度学习的基础上，迁移学习、知识蒸馏等技术可进一步提升分类性能以更加准确对新出现的道路图像自动识别。

### 1.2 问题要求

- 初赛：
  - **问题一：**基于图像文件，提取图像特征，以“正常”和“坑洼”为特征建立一个识别率高、速度快且分类准确的模型。
  - **问题二：**对问题一中所建立的模型进行训练，并对其进行多维度的评估分析。
  - **问题三：**利用训练模型识别测试集中的坑洼图像，并展示分析结果的合理性。
- 复赛：
  - **问题一：**建立可以更精准地识别坑洼边缘的模型，以满足道路避障、道路修复等工作的需求。
  - **问题二：**依据识别的坑洼边缘，对其面积进行估算，计算各坑洼占其图像面积的百分比。

## 二、问题的分析

### 2.1 初赛总结

针对**问题一**，需要提取给定图像的特征，建立一个用于识别其为正常或是坑洼道路的识别率高、速度快、分类准确的模型。首先，对图像文件**预处理**，包括**人为再分类**，对标记错误的进行**修正**，或**剔除异常图像**；再将两类图像置于各子文件中。其次，从**机理层**面对正常与坑洼道路进行比较分析，包括**原图**、**热力图**、**RGB 与灰度直方图的对比**、**边缘与轮廓检测**、**阈值分割**，从而有效地提取出**图像特征**。之后，利用 **opencv-python** 读取图像，并加入

**try-except** 异常处理语块，防止破损文件污染模型；此外，计算出正常与坑洼道路的个数比值为 **6.32**，即**数据集类别不平衡**，因此采用基于抽样的方法**平衡化**；之后，以 **8 : 2** 的比例划分数据集为**训练集与测试集**；考虑到现有的图像未能较全面地展现特征，因此**对训练集采用高斯、双边滤波，旋转处理进行图像增强**。随后，建立支持向量机、卷积神经网络的单一模型，但由于 CNN 中 softmax 函数的局限性，因而，**利用卷积神经网络对图像信息的特征进行提取**，依据上述分析，设定其输入层、卷积层、激活层、池化层、全连接层参数；在此基础上，建立**支持向量机分类模型**，对道路类别进行分类。

针对问题二，需要对问题一的模型进行训练，并进行多维度的评估分析，保证模型的准确性、快速性及普适性。因此，对问题一所建立的模型进行训练与测试，并从**模型耗时、准确率、分类报告、五折交叉验证，混淆矩阵、ROC/AUC 曲线**进行多维度评价。模型准确率在 **95 %** 的置信水平下达到 **(90 ± 8) %**；其精确率、召回率、F1 分数值在各维度下均于 **88 % 及以上**；ROC/AUC 曲线靠近图的左上角，曲线下面积达到 **0.90**。此外，还将其与单一模型进行多维度对比，**准确率最高可提升 3 %**，模型提升效果良好。

针对问题三，需要利用上述已建立的模型，对未知数据集图像进行分类，并保存结果。经过读取模型、读取数据、数据集预处理、预测得出**模型在 4942 张图像中，识别出坑洼道路为 3765 张，识别为正常道路为 1177 张**。

最后，对所建立的模型的优缺点进行了中肯的评价、提出了模型的改进措施以及对模型进行了一定推广。

## 2.2 问题的整体分析

该题是一个基于计算机视觉的坑洼道路的图像数据分析、检测类问题。

从**分析目的看**，本题需要分析道路坑洼图像并提取其边缘、面积等特性，建立一个识别率高、速度快、准确的普适性模型。因此本题需要完成以下两方面任务：**其一**，分析、研究道路坑洼的边缘特性，检测其边缘，为后续模型的建立提供支撑；同时，用以完成道路避障、道路修复等工作。**其二**，根据上述的提取的边缘特性，估算图像中坑洼的像素占比。

从**图像数据特征看**，本题的道路图像具有数量多，环境等情况复杂，以及坑洼形态多样等特征，极大地增加的模型学习的难度。因此优先考虑深度学习的方法，利用其强大的特征提取与表示能力，对道路图像进行特征提取。

从**模型的选择看**，本题图像数量多，特征复杂，且需要对道路中的坑洼边缘进行检测，同时估算其像素占比。因此，我们考虑图像增强技术，建立基于 Ultralytics 平台的 YOLO (You Only Look Once, YOLO) 算法，对道路的坑洼特性进行学习。

从**编程软件的选择看**，本题为图像大数据分析类，需要对大量的图片数据进行分析，并依据设问建立合适的模型，对坑洼道路的边缘进行提取，估算其像素占比，因此我们选择使用基于 Pycharm 内核的 Python Jupyter 对问题进行求解，其交互式的编程范式及轻量化，方便且高效。

## 2.3 各问逐一分析

- **问题一：**核心目的在于建立更精准的识别道路坑洼边缘特性的模型，用于完成道路避障、道路修复等工作。因此考虑对图像文件进行分割，有效地提取坑洼的边缘。首先，需要对图像文件进行预处理，包括人为再分类、文件夹分类，对标记错误的图像进行修正，并剔除异常的图像、原数据集保留与拓充、测试集数据预读取。之后从机理层面对道路的坑洼处进行分析，包括原图、热力图、RGB 直方图、灰度直方图的对比、边缘及轮廓检测、以及阈值分割，从而有效地从机理层面分析特性。在此基础上建立基于 YOLOv8s 的 segment 图像分割及 YOLOv8n 的 detect 图像检测模型，对道路的坑洼特性进行学习、训练。在此之前需要对图像进行标注，标注文件的格式转换，训练集、测试集、验证集三者的划分，对训练集进行数据增强，配置训练文件。待模型训练完成后，再读取迭代过程中最优模型对测试集数据的坑洼边缘进行提取。
- **问题二：**核心目的在于依据上述识别的坑洼边缘，估算坑洼面积占比。因此考虑结合上述建立的检测与分割模型，得到其边缘像素坐标，估算其像素面积，进而计算坑洼面积占比。这里需要考虑到预测的坑洼的多样性，可能存在多个坑洼、坑洼边缘重叠等情况，因此需要对上述情况进行对应的处理，得到其坑洼面积，从而求得坑洼面积占比。

## 2.4 行文思路

为了便于理清思路，绘制出本文解决问题的流程图，如图 1 所示。

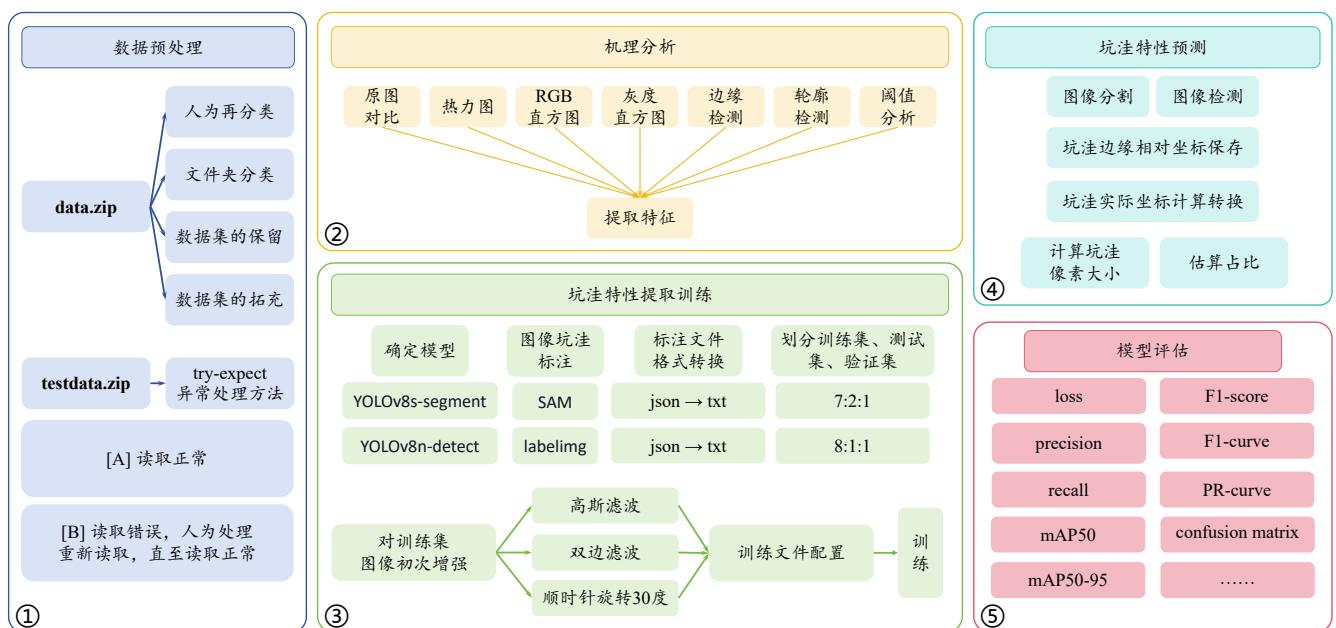


图 1 流程图

## 三、模型的假设

- **假设一：**所给数据集存在因某些原因而标注异常的图像数据的现象；
- **假设二：**所给数据集未能完整地表现出分类的特征。

## 四、符号说明

| 符号                 | 符号说明                    |
|--------------------|-------------------------|
| $W$                | 图像宽的像素值                 |
| $Y$                | 图像高的像素值                 |
| $W$                | 图像横轴像素点的像素值             |
| $Y$                | 图像纵轴像素点的像素值             |
| $I_f$              | 滤波后的图像像素值               |
| $I$                | 原始图像像素值                 |
| $G_s$              | 归一化权重                   |
| $\Omega$           | 滤波邻域窗口                  |
| $F_s$              | 空间域核函数                  |
| $F_f$              | 像素值域核函数                 |
| $(x_{i1}, y_{i1})$ | 同一张图中第 $i$ 个坑洼的左上角像素点坐标 |
| $(x_{i2}, y_{i2})$ | 同一张图中第 $i$ 个坑洼的右下角像素点坐标 |
| $X_i$              | 同一张图中第 $i$ 个坑洼的宽的像素值    |
| $Y_i$              | 同一张图中第 $i$ 个坑洼的高的像素值    |
| $S_{potholesi}$    | 同一张图中第 $i$ 个坑洼的像素面积     |
| $S$                | 某张图的像素面积                |
| $\eta$             | 坑洼面积占原图像面积比值百分数         |
| $L$                | 模型损失值                   |

**注：**这里并未列出其余变量，这是由于它们在不同小节处有不同的含义，这些符号以及上述符号在使用时仍会在相应位置进行详细说明。

## 五、模型的建立与求解

### 5.1 初赛研究相关结论、坑洼特征分析

在初赛中，为了方便模型的分析、建立与求解，因此需要对图像数据进行机理分析，将标记为正常及坑洼的道路进行比较分析，并提取图像的特征。从以下几个方面进行分析：

- **原图、热力图对比：**分析坑洼与正常道路的表层区别。
- **RGB、灰度直方图：**分析坑洼与正常道路的图像信息分布。
- **边缘、轮廓检测：**分析坑洼与正常道路的边缘、轮廓特征。
- **阈值分割：**分析坑洼与正常道路的阈值分割特征。

但由于图片较多，故我们选择“normal133.jpg”及“potholes1.jpg”进行对比分析。以下是具体分析内容，该部分具体处理代码见附录-C.1，Comparative Analysis of Normal and Potholes [正常与坑洼道路的比较分析]。

### 5.1.1 原图、热力图对比

“normal133.jpg”及“potholes1.jpg”原图及热力图对比，如图2及图3所示<sup>1</sup>。

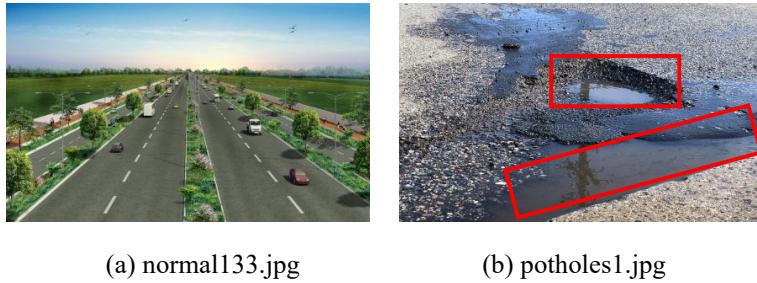


图2 原图对比

对比正常道路和坑洼道路的热力图，我们发现若整个道路的状况相同，没有坑洼，则那一片区域的热力值近似相同，热力图呈现的颜色相仿；若道路中的某一块状况与周围不同，如出现坑洼，则出现坑洼的区域热力值发生变化，使在热力图中呈现的颜色与周围存在差异。正常道路热力值基本一致，其热力图呈现色彩差别小，而坑洼道路中非坑洼处与坑洼处热力值相差较大，坑洼处与非坑洼处颜色相比存在较大差异。

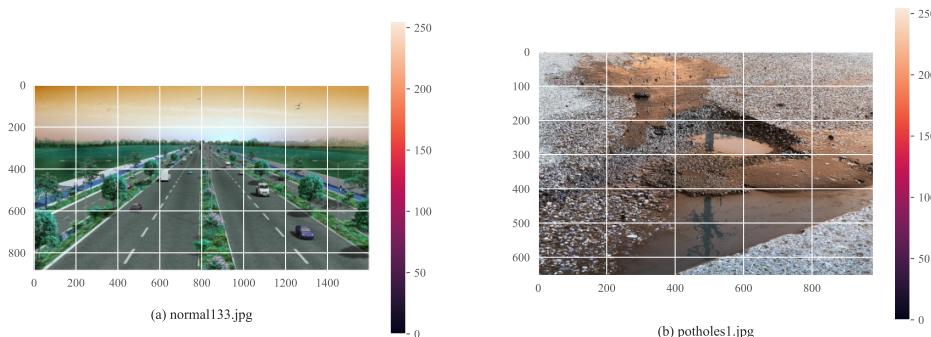


图3 热力图对比

### 5.1.2 RGB、灰度直方图

“normal133.jpg”及“potholes1.jpg”RGB及灰度直方图对比，如图4及图5所示。

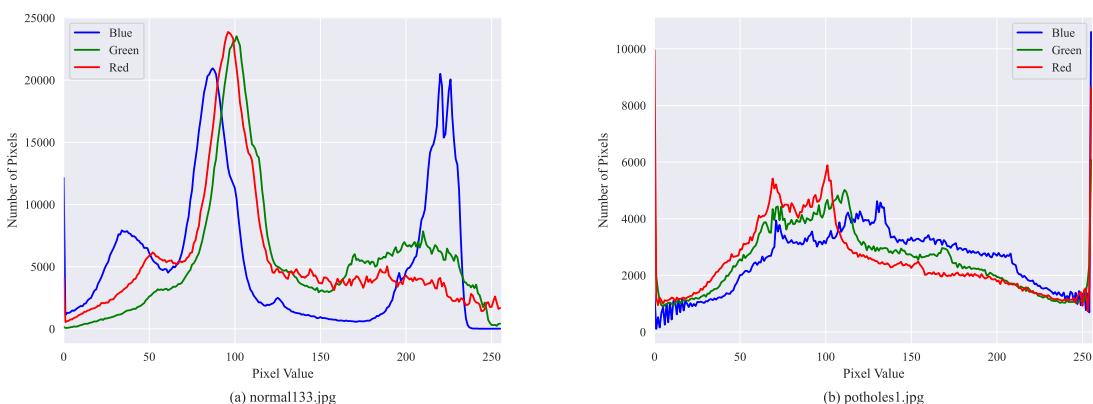


图4 RGB直方图对比

<sup>1</sup>本文所有图示、表格均已交叉引用，读者阅读PDF时可点击对应图表，进行跳转。

观察 RGB 直方图的对比，我们可以发现两者 RGB 三色整体变化具有一致性，但正常道路的 RGB 数值整体高于坑洼道路，且正常道路 RGB 峰值出现在像素值为 50~100 之间，而坑洼道路的 RGB 峰值出现在像素值为 0 与 250 处。

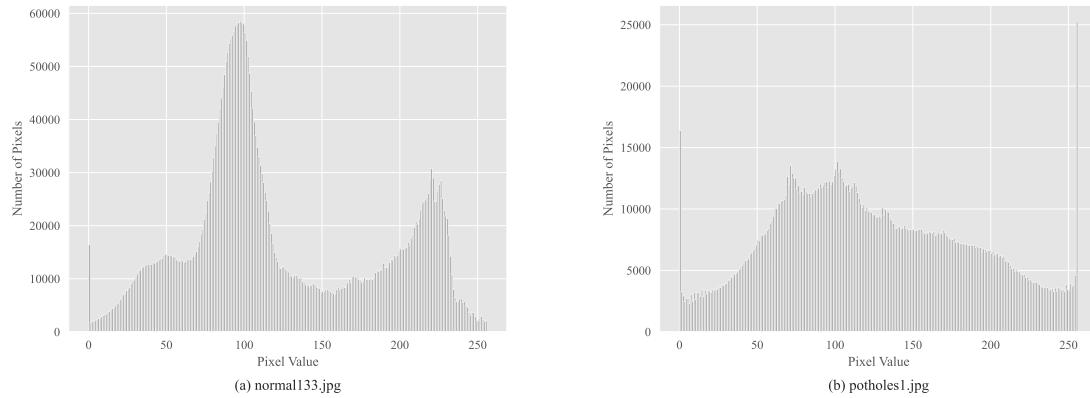


图 5 灰度直方图对比

观察灰度直方图的对比，我们可先发现灰度直方图中正常道路所呈现的峰值远高于坑洼道路，其中坑洼道路的峰值出现在图像两侧，而正常道路的峰值出现在图像的中心位置。此外正常道路数值波动明显大于坑洼道路。故我们认为灰度直方图所展现的数值大小、分布及波动情况可以较好体现道路的坑洼情况。

### 5.1.3 边缘、轮廓检测

“normal133.jpg” 及 “potholes1.jpg” 边缘、轮廓检测对比，如图 6 及图 7 所示。

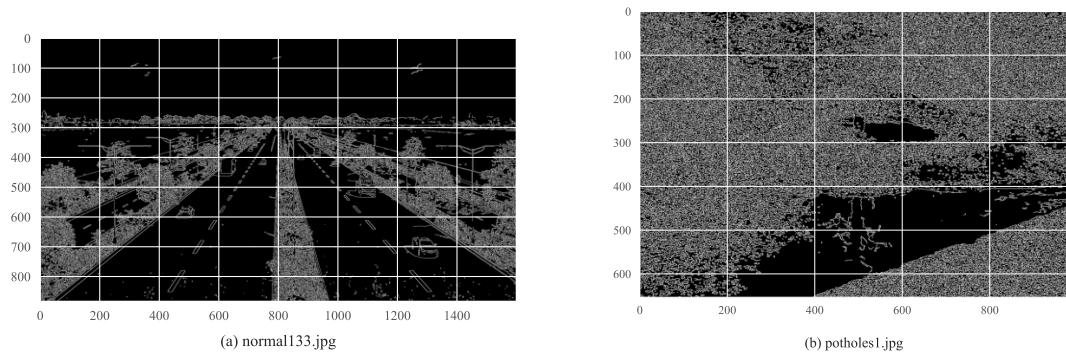


图 6 边缘检测

观察边缘检测的对比，我们可以发现边缘检测后正常道路色彩连续完整，而坑洼道路的坑洼部分在边缘检测后整体呈现深度黑色，与平整的砂石路面形成了强烈对比。故我们认为采用边缘检测可以较好判断道路状况。

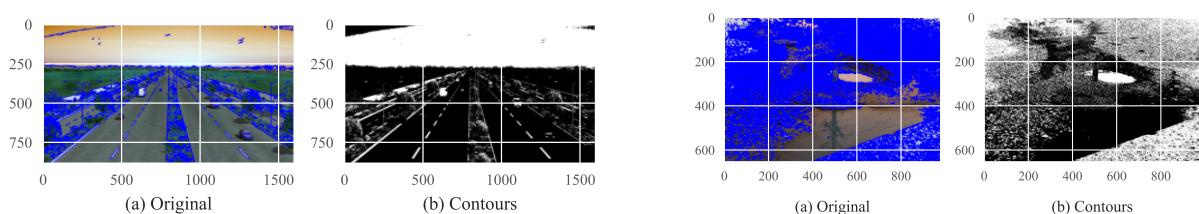


图 7 轮廓检测

对道路的图片进行轮廓检测后，我们可以发现检测后无坑洼道路的图片颜色相同，而对于坑洼道路，图片上有很明显的颜色差别。经过分析，我们认为不同的颜色代表了道路的平整与坑洼，进行轮廓检测后，可以通过道路颜色的异同判断道路是否坑洼。

#### 5.1.4 阈值分割

“normal133.jpg”及“potholes1.jpg”阈值分割对比，如图8所示。

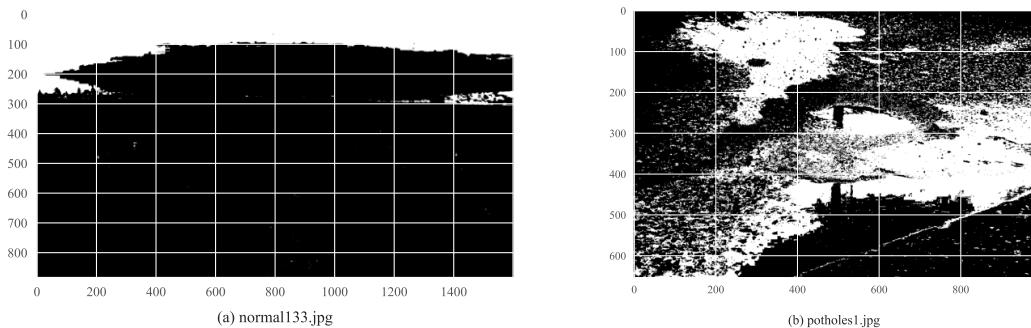


图 8 阈值分割

阈值分割将图片分为两类区域，对比正常道路与坑洼道路的阈值分割图，正常道路由于整个道路的状态相同，因此经阈值分割后，全部被划分为同一类，故颜色相同；而对于坑洼道路，道路中出现了与周围状态不同的区域，经过阈值分割后，呈现出不同的颜色，黑色为正常的道路，白色为道路中存在的坑洼，色彩对比强烈。

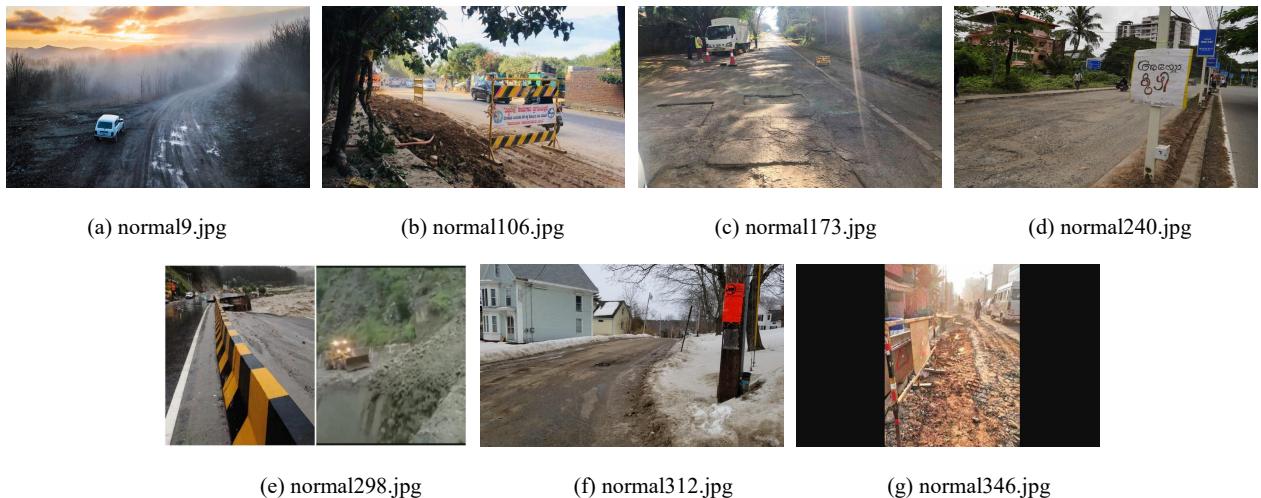


图 9 标注异常图片

#### 5.2 图像文件预处理

该题原数据集为“data.zip”，包含301张图片，均为“jpg”格式，且在文件名中进行了标注。其中，文件名中包含“normal”字符的表示正常道路，共266张，包含“potholes”字符的为坑洼道路，共35张。考虑到数据的标注严谨性，首先对其进行人为再分类，将标注异常的进行重新标注或剔除。此外，为方便后续模型的分析、建立与求解，我们将“normal”与“potholes”图像文件置于“DATA”文件下的“normal”及“potholes”文件夹中。

考虑到图像文件较少，用于训练的特征不足，因此对原数据集进行拓充，方法为从网络上获取坑洼道路的图像。

此外，还需要预先读取测试集图像，以防破损文件影响预测过程。

### 5.2.1 人为再分类

考虑到数据的标注严谨性，我们首先对其进行人为再分类，将标注异常的进行重新标注或剔除。对于所给数据集，我们筛选出了以下几张图片，其标注存在问题，需要进行重新标注或剔除，如图 9所示，图中所标注的为该图像的原始文件名。

对于上述“(e) normal298.jpg”文件，我们选择剔除，其余划分为“potholes”类。具体处理见表 1。表中“potholes”代表将该文件视为坑洼道路，而“删除”代表将该文件删除，不参与后续模型的分析、建立、训练等。

表 1 异常标注图片的处理

| 编号 | 原文件名          | 处理方式     |
|----|---------------|----------|
| a  | normal9.jpg   | potholes |
| b  | normal106.jpg | potholes |
| c  | normal173.jpg | potholes |
| d  | normal240.jpg | potholes |
| e  | normal298.jpg | 删除       |
| f  | normal312.jpg | potholes |
| g  | normal346.jpg | potholes |

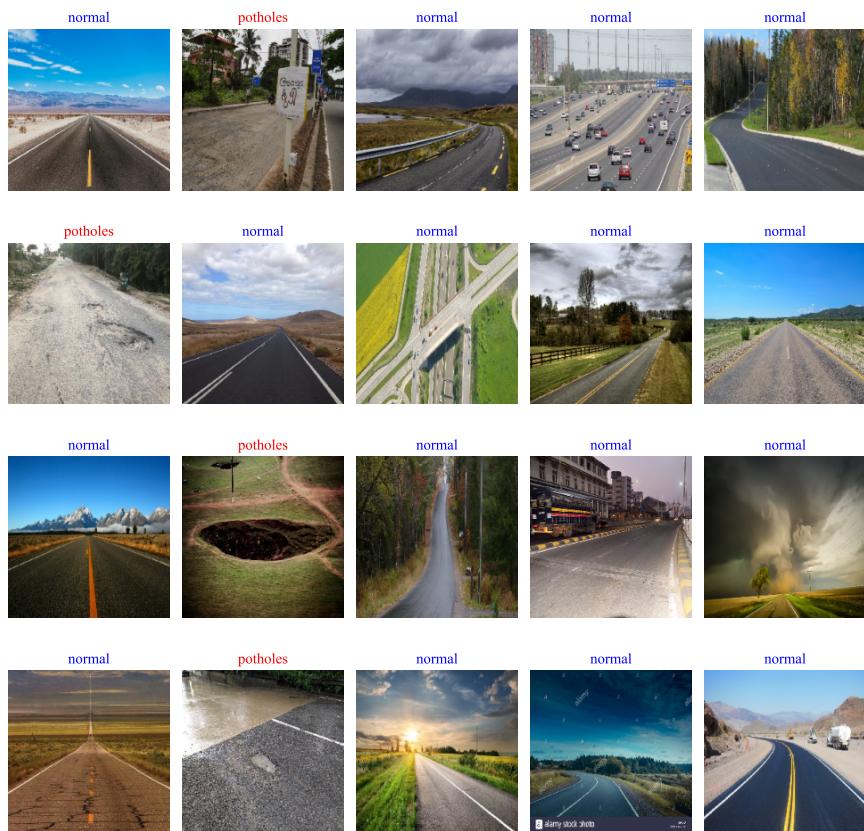


图 10 图像数据观测

### 5.2.2 文件夹分类

为了后续模型的分析、建立与求解,我们将“normal”与“potholes”图像文件置于“DATA”文件下的“normal”及“potholes”文件夹中,作为二分类的数据集。该处理我们使用 Python 的 os 及 shutil 库进行处理,首先指定根目录为“DATA”,再在其下创建“normal”与“potholes”文件夹,之后读取“DATA”文件夹下的所有文件,若文件名中含有“normal”字符,则将其放置于“normal”文件夹中,否则放置于“potholes”文件夹中。具体处理代码见附录-C.2, Data Preprocessing [数据预处理]。这里,我们可以大致观测数据集图像,如图 10 所示。

至此,我们已将所有数据进行了人为再分类,并将数据集分别放于“normal”及“potholes”文件夹中,方便后续的处理。

### 5.2.3 原数据集的保留与拓充

由于在复赛中,本题任务需要重点分析坑洼道路的特性,因此,这里舍弃“normal”类图像,保留“potholes”图像,共计 41 张。

考虑到保留的图像文件较少,学习特征不足,将会影响模型精度。同时,为了加强我们分析的可信度,我们从以下网站收集了坑洼图像,对“potholes”类图像数据进行拓充,来源见表 2 所示。<sup>2</sup>

表 2 拓充数据来源

| 数据源    | 网址  |
|--------|---|
| Kaggle | <a href="https://www.kaggle.com/">https://www.kaggle.com/</a>   |
| Github | <a href="https://github.com/">https://github.com/</a>           |
| 百度图片   | <a href="https://image.baidu.com/">https://image.baidu.com/</a> |

### 5.2.4 测试集数据预读取

此外,还需要预先读取测试集图像,以防破损文件影响预测过程。这里我们使用 try-except 异常处理方法,测试文件是否能够正常读取,方便后续进行预测。

经过测试,发现存在一个文件读取失败,为“guy7iodk.jpg”。因此,对于该张图片,我们对其进行人为处理,再重新读取。具体方法见测试集图像坑洼特性预测。

## 5.3 YOLOv8 模型简述

YOLOv8 是 Ultralytics 公司于 2023 年 1 月 10 日开源的 YOLOv5 的下一个重大更新版本,目前支持图像分类、物体检测、实例分割、目标追踪、姿态估计任务<sup>[1]</sup>。

YOLOv8 是一个 SOTA 模型,其建立在曾经的版本的基础上引入新的功能,并进行一定改进,从而进一步提升性能与灵活性。具体创新包括骨干网络 (Backbone Network)、Anchor-Free 检测头 (Anchor-Free Detection Head) 和损失函数 (Loss Function)<sup>[2]</sup>,模型结构如图 11 所示<sup>3</sup>。

<sup>2</sup>注:拓充的图为原数据集中尚未出现过的图像,即原数据集为新数据集的子集。

<sup>3</sup>该架构图来源于: [https://mmyolo.readthedocs.io/en/latest/recommended\\_topics/algorithm\\_descriptions/yolov8\\_description.html](https://mmyolo.readthedocs.io/en/latest/recommended_topics/algorithm_descriptions/yolov8_description.html), 原作者为 RangeKing@github, 其 Github 主页: <https://github.com/RangeKing>。

YOLOv8 主要可分为 Input 输入端、Backbone 骨干神经网络、Neck 混合特征网络层和 Head 预测层网络 4 个部分<sup>[3]</sup>。

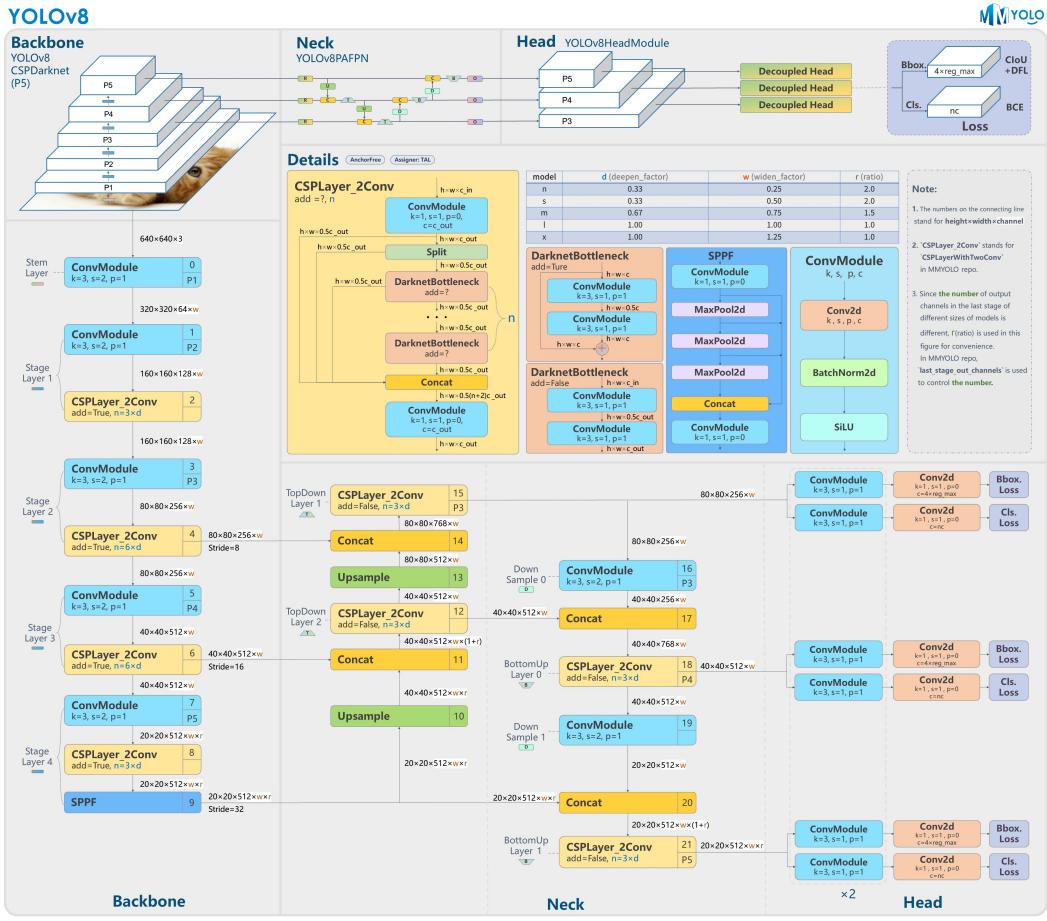


图 11 YOLOv8 模型架构

- Input:** 输入端方法包含的功能模块有：马赛克 (Mosaic) 数据增强、自适应锚框 (Anchor) 计算、自适应图片缩放和 Mixup 数据增强<sup>[4]</sup>。马赛克数据增强通过将 4 张图像进行随机的缩放、裁剪和打乱分布方式等操作来重新拼接图像，从而丰富检测的数据集。在每次训练数据之前，自适应锚框计算会选择最合适锚框尺寸，自动匹配最佳锚框，并将在目标检测中将两幅图像的像素值按图像透明度的通道信息进行线性融合。
- Backbone Network:** 骨干网络用来提取图像特征，包括注意力 (Focus) 机制模块、跨阶段局部网络 (Cross Stage Partial Network, CSP) 和空间金字塔池化 (Spatial Pyramid Pooling, SPP) 结构四个模块。先将图片进行切片，再将新得到的图片经过卷积操作，生成没有信息丢失的两倍下的采样特征图。
- Neck:** Neck 混合特征网络层由卷积层和 C2f 组成，通过路径聚合网络 (Path Aggregation Network, PAN) 和特征金字塔网络 (Feature Pyramid Network, FPN) 的结构对特征进行多尺度融合，将图像特征传递到预测层。
- Head:** Head 预测层采用解耦检测头 (Decoupled-Head)<sup>[6]</sup>，通过不同的分支进行运算，提升检测效果。

## 5.4 坑洼特性提取模型的训练

为对坑洼特性进行提取，我们需要建立基于 YOLOv8 的 segment 图像分割及 YOLOv8 的 detect 图像检测模型，对道路的坑洼特性进行学习、训练。在此之前需要对图像进行标注，标注文件的格式转换，训练集、测试集、验证集三者的划分，对训练集进行数据增强，配置训练文件。待模型训练完成后，再读取迭代过程中最优模型对测试集数据的坑洼边缘进行提取。

### 5.4.1 图像坑洼标注

首先，需要对图像中的坑洼位置进行标注，对于本文所建立的两个 YOLOv8 模型，使用不同的标注方法，标注过程示意图如图 12 所示。

- **YOLOv8s-Segment:** 基于 Segment Anything Model 的交互式半自动图像分割标注工具（Interactive Semi-automatic Annotation Tool for image segmentation with Segment Anything Model, ISAT-SAM）；
- **YOLOv8n-Detect:** Labelimg 标注。

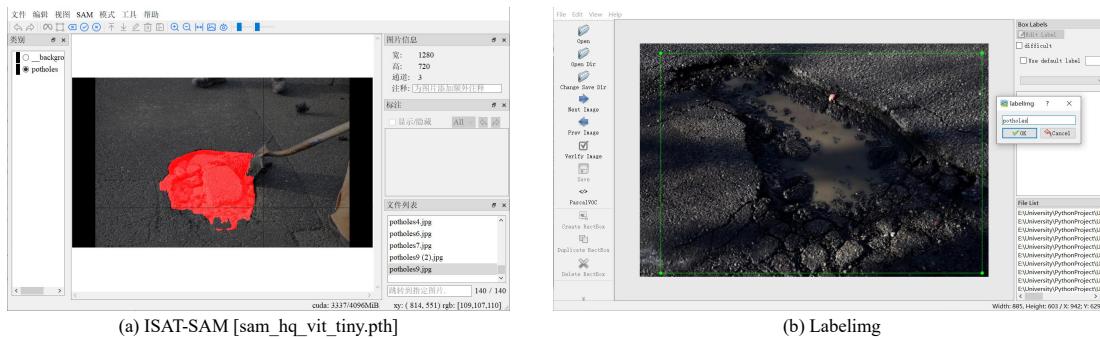


图 12 图像坑洼标注

两者比较如下：

- **ISAT-SAM:** ISAT-SAM 是一种基于 SAM 的交互式半自动图像分割标注工具，本文所使用的预训练模型为“sam\_hq\_vit\_tiny.pth”。该工具支持手动标注多边形、连点绘制、标注二次修改、重叠目标调整遮挡关系、标注结果预览、单独线程进行 Sam Encoder 计算，降低切换图片的卡顿感<sup>[7]</sup>。同时，其相比于其他人工标注方法，采用 SAM，大大提升了标注效率，且边缘处理更精准。标注输出格式为 JSON 文件。
- **Labelimg:** Labelimg 是一个可视化的图像标注工具。其是用 Python 编写的，并将 Qt 用于图形界面。批注以 PASCAL VOC 格式（ImageNet 使用的格式）另存为 XML 文件<sup>[8]</sup>。其仅可标注矩形，因此多用于图像目标的检测任务。

### 5.4.2 标注文件格式转换

由于上述两种标注方法生成的格式分别为 JSON 文件和 XML 文件，而 YOLOv8 模型所需的标注文件格式为 TXT 文件，因此需要对其进行格式转换。这里我们使用 Python 进行处理，具体处理代码见附录-C.4, `isat2txt [ista 转 txt]` 及附录-C.5 `xml2txt [xml 转 txt]`。

这里我们以“potholes22.jpg”为例，原图见图 13；两种标注方法转换前与转换后的对比如图 14 所示。



图 13 “potholes22.jpg” 原图



图 14 格式转换前后文件

#### 5.4.3 划分训练集、测试集、验证集

对于模型的学习，我们需要对数据集进行划分，划分为训练集、测试集以及验证集。

- **训练集：**用于模型的训练；
- **测试集：**用于对模型性能的评估分析，检验模型效果；
- **验证集：**用于在训练阶段验证模型训练效果，选择超参数。

对于本文所建立的两个 YOLOv8 模型，我们设置训练集、测试集、验证集的比例见表 3。

表 3 模型训练集、测试集、验证集的划分比例

| 模型名称            | 训练集 | 测试集 | 验证集 |
|-----------------|-----|-----|-----|
| YOLOv8n-Detect  | 8   | 1   | 1   |
| YOLOv8n-Segment | 7   | 2   | 1   |

具体实现代码见附录-C.6, Segment Spilt Data [图像分割数据集划分] 以及附录-C.7, Detect Spilt Data [图像检测数据集划分]。

#### 5.4.4 训练集图像初次增强

观察数据集，发现其特征多数存在相似性；同时，数据集样本较少。因此，为避免模型的过拟合与欠拟合，这里我们需要对训练集进行初步的数据增强<sup>4</sup>。从而增强模型对于未知数据集的泛化能力，提升模型的稳健性。

这里，我们选用线性滤波中的高斯滤波（Gauss Blur）、非线性滤波中的双边滤波（Bi-lateral Blur）、以及对图像进行顺时针旋转  $\frac{\pi}{6}$  处理。

- **高斯滤波**是一种线性平滑滤波。其基本原理是使用高斯核对图像进行卷积操作，进行加权平均的过程。每一个像素点的值，都由其邻域内的其他像素值和本身经过加权平均后得到。高斯核的标准差和大小决定了滤波器的效果，标准差越小，滤波器的效果越不明

<sup>4</sup>注意：这里数据增强是在划分训练集与测试集之后的，并且仅对训练集数据进行增强，并不对测试集进行划分。这是由于，若在划分训练集与测试集之前进行增强，则会造成数据泄露，影响模型的准确性，以及对未知数据的泛化能力。

显，但是不会导致图像的细节信息丢失。其权值随着距离中心像素点的距离增加而逐渐减小，从而保留了图像的边缘信息。使用公式如下：

$$G(X, Y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{X^2 + Y^2}{2\sigma^2}\right) \quad (1)$$

- **双边滤波**是一种基于高斯滤波的非线性滤波方法，目的是解决高斯滤波造成的边缘模糊。结合图像的空间邻近度和像素值相似度，同时考虑空域信息和灰度相似性，实现保边去噪。双边滤波器比高斯滤波多一个高斯核。它是基于像素颜色分布的高斯滤波函数，所以在边缘附近，当两个像素距离很近时，只有同时当颜色很接近时影响才会较大，反之，虽然距离很近，但颜色差距较大，那么平滑权重也会很小。

为实现双边滤波，我们首先定义滤波器的参数，即空间域核函数和像素值域核函数；然后计算出每个像素在空间域和像素值域上的权重；最后根据计算得到的权重，对每个像素的周围像素进行加权平均，以得到滤波后的像素值。其公式可表示为：

$$I_f(X, Y) = \frac{\sum_{(i,j) \in \Omega} I(X+i, Y+j) \cdot F_s(i, j) \cdot F_f[I(X, Y), I(X+i, Y+j)]}{G_s(X, Y)} \quad (2)$$

上式中各参量含义如下：

- $I_f(X, Y)$ : 滤波后的图像像素值；
- $I(X, Y)$ : 原始图像像素值；
- $G_s(X, Y)$ : 归一化权重；
- $\Omega$ : 滤波邻域窗口；
- $F_s(i, j)$ : 空间域核函数；
- $F_f[I(X, Y), I(X+i, Y+j)]$ : 像素值域核函数。

这里，我们随机选择一张图片，将其经过上述处理后的图像一一展示，如图 15 所示。经过上述处理后，在一定程度上可以增加训练集数据的多样性，从而模型可以学习到更多关于正常或坑洼道路的特征，因而有利于提升模型的泛化能力。



图 15 原图、高斯滤波、双边滤波、顺时针旋转 30 度

#### 5.4.5 训练文件配置

至此，我们还需要预训练模型，这里我们选用 YOLOv8s-seg 及 YOLO8n 分别作为图像分割及图像检测的基础模型，命名为 YOLOv8-Segment 及 YOLOav8-Detect。此外，我们还需要配置训练文件，主要配置文件见表 4。

表 4 模型配置文件

| 模型名称           | 配置文件                           |
|----------------|--------------------------------|
| YOLOv8-Segment | segment.yaml 与 yolov8-seg.yaml |
| YOLOv8-Detect  | detect.yaml                    |

#### 5.4.6 模型的训练

- **YOLOv8s-Segment**

在训练时，模型会生成数据集的相关特征，如图 16 所示。

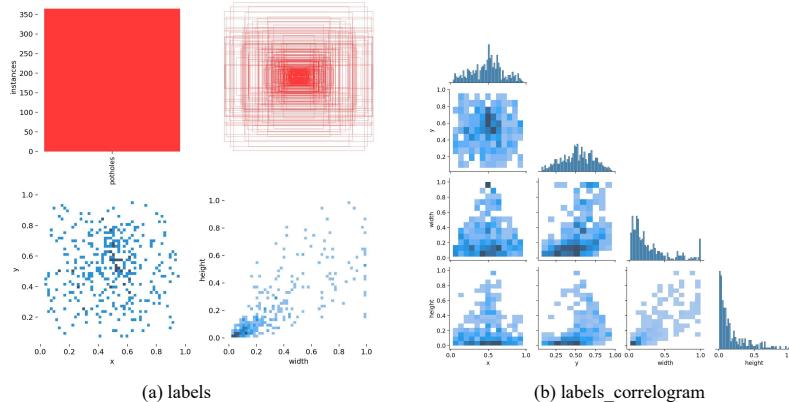


图 16 YOLOv8s-Segment 训练数据集特征

同时，YOLOv8 模型也会对训练集数据进行增强，有马赛克（Mosaic）数据增强、自适应锚框（Anchor）计算、自适应图片缩放和 Mixup 数据增强，如图 17 所示。

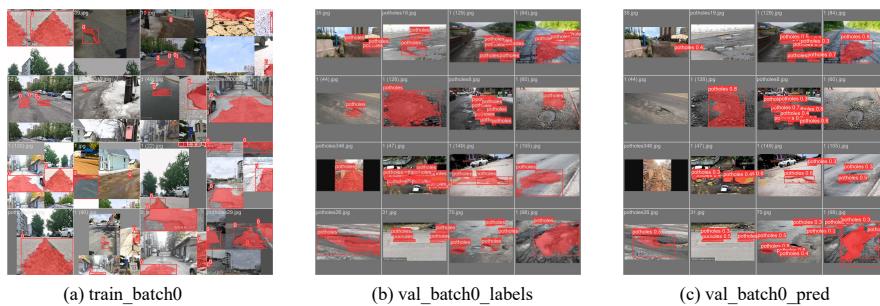


图 17 YOLOv8s-Segment 训练数据集增强

训练过程的每一次迭代的结果保存于“支撑材料/process/训练”文件夹中。训练结束后，会生成“best.pt”与“last.pt”模型文件，我们选择“best.pt”作为后续预测的图像分割模型。而对于模型的效果分析，详见[模型的评估](#)。

- YOLOv8n-Detect

对于该模型的训练与上述模型类似。在训练时，其也会生成数据集的相关特征，如图 18 所示。

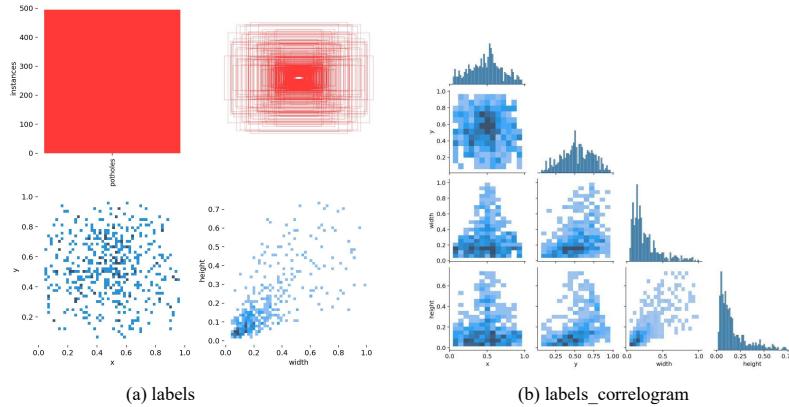


图 18 YOLOv8n-Detect 训练数据集特征

在训练过程中，模型也将对数据进行增强，部分效果如图 19 所示，由于篇幅原因，其余的详见附录。

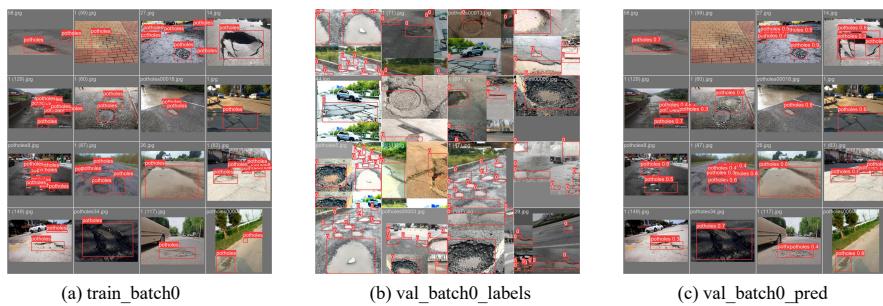


图 19 YOLOv8n-Detect 训练数据集增强

训练过程的每一次迭代的结果保存于“支撑材料/process/训练”文件夹中。训练结束后，会生成“best.pt”与“last.pt”模型文件，我们选择“best.pt”作为后续预测的图像检测模型。而对于模型的效果分析，详见[模型的评估](#)。

## 5.5 测试集图像坑洼特性预测

在对测试集图像坑洼特性预测前，首先要判断各图像文件是否能够读取，在[测试集数据预读取](#)中，经测试有一文件读取失败，读入的数组为空，将会导致后续批量预测的中断。这里我们考虑到，在系统中能够正常打开，而利用 opencv 或 pillow 库均打开失败。因此，我们考虑尝试重新另存为该张图片为 JPG 格式文件，再重新读取。经过测试，通过上述操作后，文件能够正常读取，且信息正确。至此 4942 张图片均能正常读取，方便后续批量、统一预测。

### 5.5.1 道路坑洼的边缘提取

利用上述训练好的模型，可以对未知数据集进行批量预测，提取其坑洼的边缘，现随机挑选一张图片，文件名为“em5acvux.jpg”，结果见图 20。可以发现，无论是图像分割还是图

像检测，其均能较好地提取出坑洼的边缘信息，但图像分割的效果更好，其能够更好地提取出坑洼的边缘，且能够将坑洼的形态进行较好的还原。

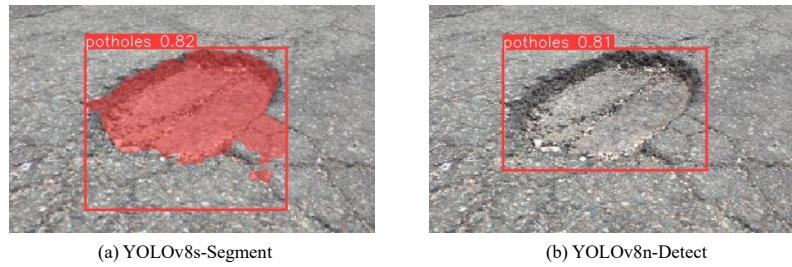


图 20 效果展示

对于上述起初读取失败的“guy7iodk.jpg”文件，这里我们也将结果展示，如图 21 所示。观察该图，我们可以发现，YOLOv8s-Segment 模型能够较好地提取出坑洼的边缘，但 YOLOv8n-Detect 模型提取的坑洼边缘信息不够完整，且存在一定的误差。

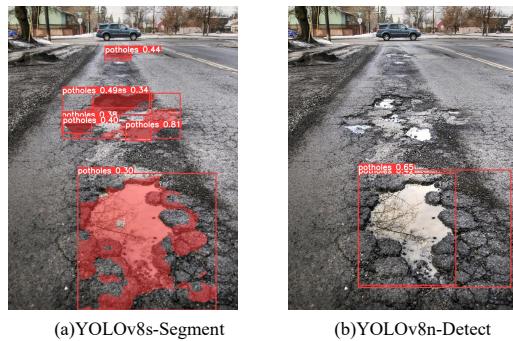


图 21 guy7iodk.jpg 结果展示

### 5.5.2 道路坑洼的面积占比估算

由于坑洼的形态多样，在对使用图像检测的矩形计算其面积占比时，需要考虑到以下五种基本情况，其有着不同的处理方法，以下进行具体叙述，记坑洼面积占原图像面积比值为<sup>5</sup> $\eta\%$ 。在估算面积占比之前，需要利用之前训练好的模型对未知数据集图像文件进行预测，得到坑洼的边框相对坐标信息。但由于在估算时，需要使用到绝对坐标信息，因此，我们需要对其进行转换。这里我们使用 Python 进行处理，具体处理代码见附录-C.10，Coordinate Transformation [坐标转换]。

- **图中无坑洼；** 在模型预测时，其支持输出预测的图像中坑洼的个数，利用 Python 的 pandas 库将其进行处理，即可得到各图预测的坑洼个数。当坑洼个数为 0 时，面积占比为 0。
- **图中仅有一处坑洼；** 当图中仅有一处坑洼时，可以直接计算该坑洼的面积，从而求得其占比。如图 22 所示。

<sup>5</sup>本文计算的  $\eta$  均向上取整。

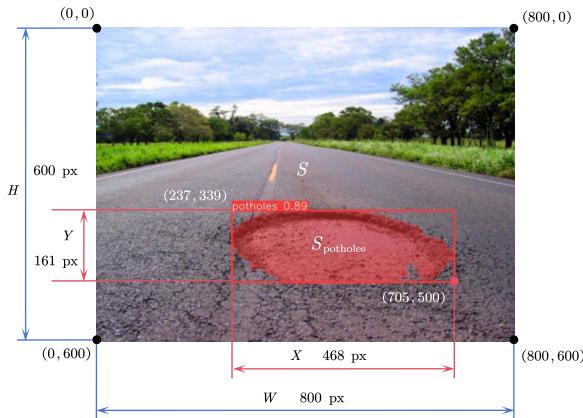


图 22 单一坑洼面积占比估算示意图

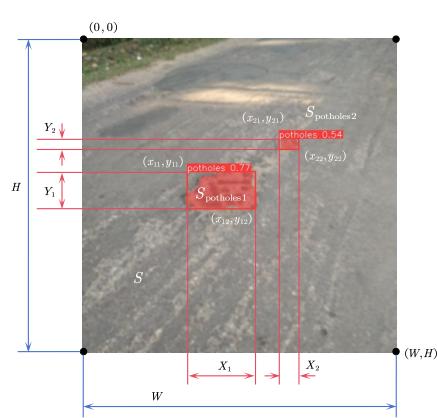


图 23 多个不重叠坑洼面积占比估算示意图

定义图像长、宽的像素值分别为  $W$ 、 $H$ 。则图像面积为:

$$S = W \cdot H \quad (3)$$

在图像坐标系中, 左上角为原点, 记坑洼边框的左上角坐标为  $(x_1, y_1)$ , 右下角坐标为  $(x_2, y_2)$ , 坑洼的长、宽的像素值分别为  $X$ ,  $Y$ , 则坑洼的面积为:

$$S_{\text{potholes}} = X \cdot Y = (x_2 - x_1) \cdot (y_2 - y_1) \quad (4)$$

因此坑洼面积占比为:

$$\eta = \frac{S_{\text{potholes}}}{S} \times 100 \quad (5)$$

例如图 22 的长、宽的像素值分别为  $W = 800$ ,  $H = 600$ , 坑洼的长、宽的像素值分别为  $X = 468$ ,  $Y = 161$ , 则坑洼面积占比为:

$$\eta = \frac{468 \times 161}{800 \times 600} \times 100 = 16 \quad (6)$$

- **图中有多个坑洼, 但两两互不重叠:** 该种情况, 如图 23 所示。其处理方法与图中仅有一处坑洼类似, 首先计算各坑洼的面积, 然后求和, 即可得到坑洼面积占比, 即:

$$\eta = \frac{\sum_{i=1}^n S_{\text{potholes},i}}{S} \times 100 = \frac{\sum_{i=1}^n [(x_{i2} - x_{i1}) \cdot (y_{i2} - y_{i1})]}{W \cdot H} \times 100 \quad (7)$$

- **图中有两坑洼为包含关系:** 当坑洼为包含关系时, 不可简单对其进行求和, 而应以最大的坑洼面积作为总的坑洼面积, 如图 24 即:

$$\eta = \frac{\max_{i=1}^n S_{\text{potholes},i}}{S} \times 100 = \frac{\max_{i=1}^n [(x_{i2} - x_{i1}) \cdot (y_{i2} - y_{i1})]}{W \cdot H} \times 100 \quad (8)$$

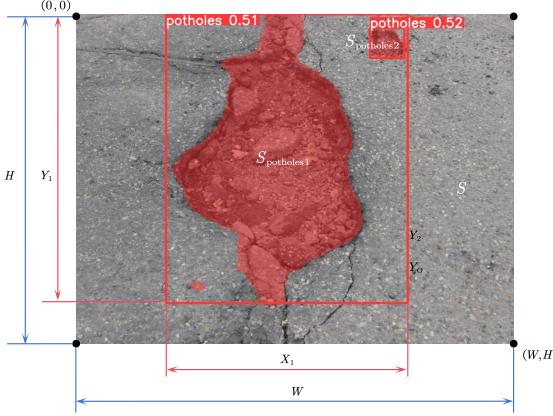


图 24 完全重叠坑洼面积占比估算示意图

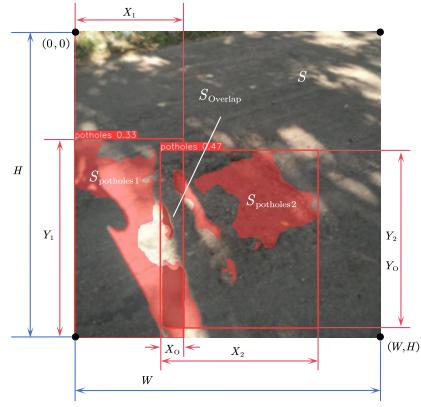


图 25 部分重叠坑洼面积占比估算示意图

- 图中有两坑洼部分重叠：对于该情况，如图 25 所示。基本方法是，先求得各坑洼的面积，然后求和，最后减去重叠部分的面积，即：

$$\eta = \frac{\sum_{i=1}^n S_{\text{potholes},i} - \sum S_{\text{overlap}}}{S} \times 100 = \frac{\sum_{i=1}^n [(x_{i2} - x_{i1}) \cdot (y_{i2} - y_{i1})] - \sum S_{\text{overlap}}}{W \cdot H} \times 100 \quad (9)$$

对于  $S_{\text{overlap}}$  的计算，需要分如图 26 所示的四种情况。

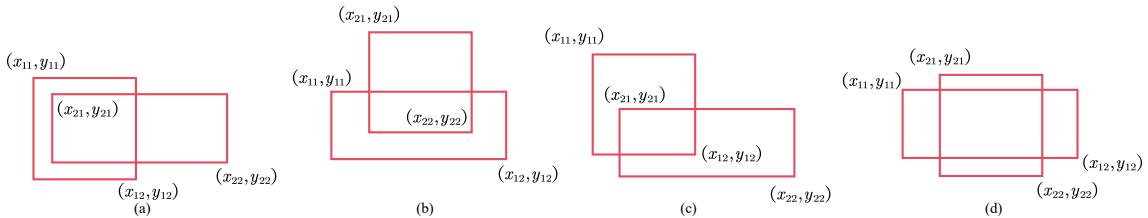


图 26 部分重叠四种情况

- a

$$S_{\text{overlap}} = (x_{12} - x_{21}) \cdot (y_{22} - y_{21}) \quad (10)$$

- b

$$S_{\text{overlap}} = (x_{22} - x_{21}) \cdot (y_{22} - y_{11}) \quad (11)$$

- c

$$S_{\text{overlap}} = (x_{12} - x_{21}) \cdot (y_{12} - y_{21}) \quad (12)$$

- d

$$S_{\text{overlap}} = (x_{22} - x_{21}) \cdot (y_{12} - y_{11}) \quad (13)$$

此外，由于坑洼图像的多样、复杂，部分图片可能存在上述基本情况的综合，因此，对于每一情况，均需要特定计算，再求和，从而求得其占比。

经过上述分析，整合如表 5 所示的数据集。表中仅罗列出了部分数据，且部分文件存在多行，这是由于其存在多个坑洼，因此，需要逐个文件进行处理。

现依据该表及上述分析对各图中坑洼占比进行计算，处理代码详见附录-C.11, **Proportional Calculation [面积占比估算]**。此外，该方法需要考虑的情况较多，且估算存在一定误差。因此，提出一种简便且准确的方法，即利用 YOLOv8s-Segment 模型提取的坑洼区域，进行阈值分割，像素点二值化，求目标像素点占整体像素点比值，即可较为准确地求得坑洼占比，具体见[模型的评价](#)。

表 5 道路坑洼面积占比估算计算数据集

| 索引   | fnames       | 置信度      | $x_1$      | $y_1$      | $x_2$       | $y_2$       | Npotholes | height | width |
|------|--------------|----------|------------|------------|-------------|-------------|-----------|--------|-------|
| 0    | 00c9sdfq.jpg | 0.783255 | 54.144627  | 119.418549 | 127.293488  | 149.223495  | 2         | 259    | 194   |
| 1    | 00c9sdfq.jpg | 0.727551 | 86.166367  | 92.886162  | 143.883499  | 117.423920  | 2         | 259    | 194   |
| 2    | 00yf8rxj.jpg | 0.685144 | 0.000000   | 268.575745 | 714.198425  | 439.384949  | 1         | 450    | 800   |
| 3    | 01lx7dej.jpg | 0.345122 | 723.983704 | 628.620300 | 1671.653564 | 1621.744507 | 1         | 3264   | 2448  |
| 4    | 01paoyy0.jpg | 0.447922 | 62.188206  | 54.767998  | 338.213837  | 239.532608  | 1         | 300    | 447   |
| ...  | ...          | ...      | ...        | ...        | ...         | ...         | ...       | ...    | ...   |
| 6506 | zynixpux.jpg | 0.266729 | 55.626934  | 116.560684 | 126.989532  | 136.572723  | 3         | 300    | 481   |
| 6507 | zyylqfs0.jpg | 0.829763 | 645.354004 | 481.915741 | 1075.006836 | 717.232178  | 1         | 816    | 1200  |
| 6508 | zzhe6767.jpg | NaN      | NaN        | NaN        | NaN         | NaN         | 0         | 555    | 986   |
| 6509 | zzipq9zn.jpg | NaN      | NaN        | NaN        | NaN         | NaN         | 0         | 300    | 463   |
| 6510 | zzvomOax.jpg | 0.711966 | 0.000822   | 369.854126 | 1447.335327 | 984.154907  | 1         | 1080   | 1623  |

## 5.6 模型的评估

### 5.6.1 模型评估指标

为更好地评估模型，我们针对上述模型进行多维度评估，这里我们用到的有：

- **损失值 (Loss):** 损失函数由预测的中心坐标损失、预测边界框的宽高损失、预测的类别损失、预测的置信度损失四个部分组成[\[9\]](#)。

– 中心坐标损失：

$$L_1 = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \ell_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \quad (14)$$

该式计算了相对于预测的边界框位置  $(x, y)$  的损失值。定义当  $\ell_{ij}^{\text{obj}}$  的值为 1 时，如果网格单元  $i$  中存在目标，则第  $j$  个边界框预测值对该预测有效。

– 预测边界框的宽高损失：

$$L_2 = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \ell_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad (15)$$

– 预测类别损失：

$$L_3 = \sum_{i=0}^{S^2} \ell_i^{\text{obj}} \sum_{c \in \text{classes}} [p_i(c) - \hat{p}_i(c)]^2 \quad (16)$$

– 预测置信度损失：

$$L_4 = \sum_{i=0}^{S^2} \sum_{j=0}^B \ell_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \ell_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (17)$$

其中  $C$  为置信度得分,  $\hat{C}$  是预测边界框与基本事实的交叉部分。当一个单元格有对象时,  $\ell_i^{\text{obj}}$  为 1, 否则为 0。

因此损失值为:

$$L = L_1 + L_2 + L_3 + L_4 \quad (18)$$

- 精确率 (Precision)、召回率 (Recall), F1 分数值 (F1-Score):

– 精确率

$$\text{Precision} = \frac{TP}{TP + FP} \quad (19)$$

– 召回率

$$\text{Recall} = \frac{TP}{TP + FN} \quad (20)$$

– F1 分数值

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (21)$$

对于  $TP$ 、 $FN$ 、 $FP$ 、 $TN$ , 其中 T 为 True, F 为 False, 这两个字母表示预测值与实际值是否相同; P 为 Positive, N 为 Negative, 这两个字母表示预测出的是属于正类还是负类。因此  $TP$  为预测为正类且实际为正类的样本数,  $FN$  为预测为负类且实际为正类的样本数,  $FP$  为预测为正类且实际为负类的样本数,  $TN$  为预测为负类且实际为负类的样本数。

此外, 对于模型的精确率、召回率, 我们可以根据定义可以发现若这两项值较大, 则模型效果较好。同时根据定义, 我们可以发现模型的精确率、召回率在理想情况下是相差较小的, 我们可以根据图示结果验证, 符合预期效果。对于模型的 F1 分数值, 其为精确率与召回率的调和平均数<sup>[10]</sup>, 因此当精确率与召回率均有较好表现时, F1 分数值会有较优秀表现。我们也可对(21) 式进行一定变换, 可以得到:

$$F1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \quad (22)$$

根据该式, 我们可以得出上述结论。

- 混淆矩阵 (Confusion Matrix): 矩阵每一行表示样本标签的实际类别, 在本题中表示道路类型: 为正常还是坑洼的实际标签; 每一列表示样本标签的预测类别, 在本题中表示道路类型: 为正常还是坑洼的预测标签。因此该图示的主对角线数据之和即为模型预测准确的样本数。这里此外还需要引入四项值, 而混淆矩阵可以直观地观察到预测准确与错误的情况, 以及模型对于每一类别的区分程度。
- 精确率-召回率曲线 (Precision-Recall Curve, PR-Curve): 该图像可表现出分类的预测精度与召回率之间的关系<sup>[11]</sup>。图像的填充区域越大, 分类效果越优。

- **mAP:** 其中 AP 是某一类别的精确率-召回率曲线 (PR-Curve) 下的面积，而 mAP 则是计算所有类别的 PR 曲线下面积的平均值，即：

$$AP = \sum_{i=1}^n (R_i - R_{i-1}) P_i \quad (23)$$

$$mAP = \frac{1}{Q} \sum_{q=1}^Q AP(q) \quad (24)$$

在本题中，由于我们只研究道路坑洼，因此 mAP 与 AP 是相等的，即(24) 式中  $Q = 1$ 。

- **mAP50:** 50 表示目标的交集与并集之比 IoU (Intersection over Union) 阈值为 50%，即当预测框与真实框的 IoU 大于 50% 时，该预测框为正确预测框。若记真实区域为  $A$ ，预测区域为  $B$ ，则 IoU 为：

$$IoU = \frac{A \cap B}{A \cup B} \quad (25)$$

- **mAP50-95:** 50 – 95 表示 mAP 阈值为 50 到 mAP 阈值为 95，间隔 5，取得 10 个 mAP 值，再对这十个值取平均<sup>[12]</sup>。
- **epoch 过程中各项指标变化图:** 该图可直观地观察到模型在训练过程中各项指标的变化情况，从而判断模型是否收敛，以及模型的收敛速度。

### 5.6.2 YOLOv8s-Segment

对于图像分割模型，绘制其 Box 及 Mask，即该模型会对坑洼标注矩形框的同时也会标注其边缘信息，如图 27 及图 28 所示。

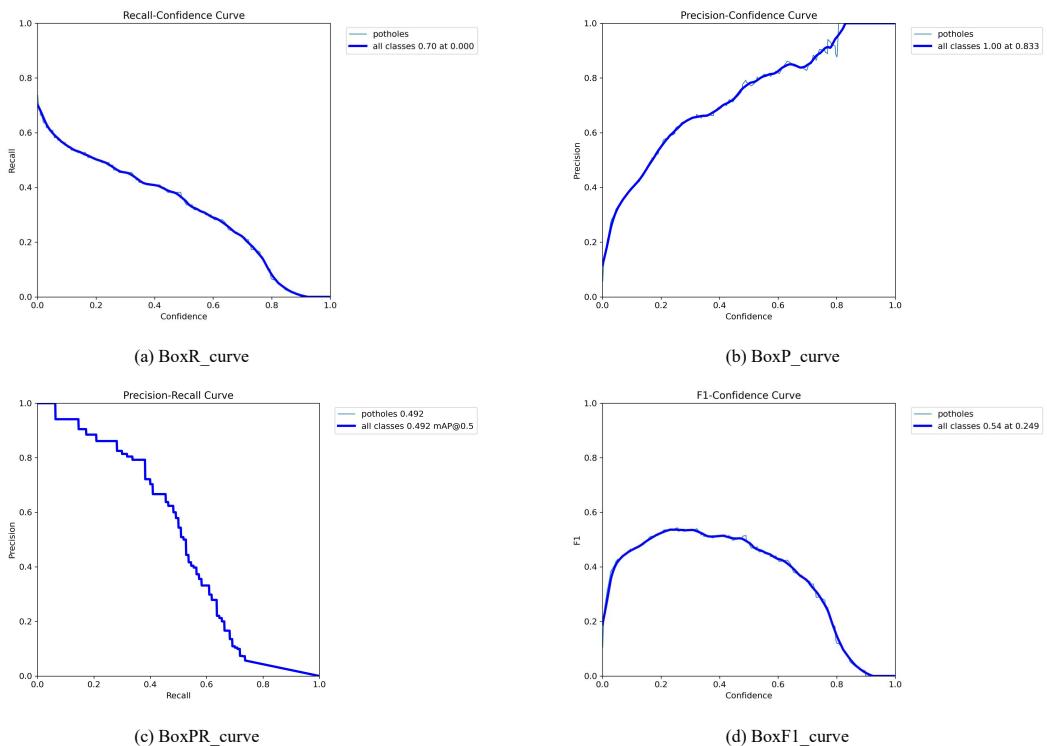


图 27 YOLOv8s-Segment Box

观察上述图示，可以发现各项指标在 Box 中较为优秀，而在 Mask 中相对较低，这是由于在标注矩形框时，仅需要坑洼的最边界信息，而无需具体到其轮廓、形态等特征。而边缘分割时，需要获取其边缘信息，需要具体到其轮廓、形态等特征，因此该任务相对于 Box 标注更为困难，因此其指标相对较低。

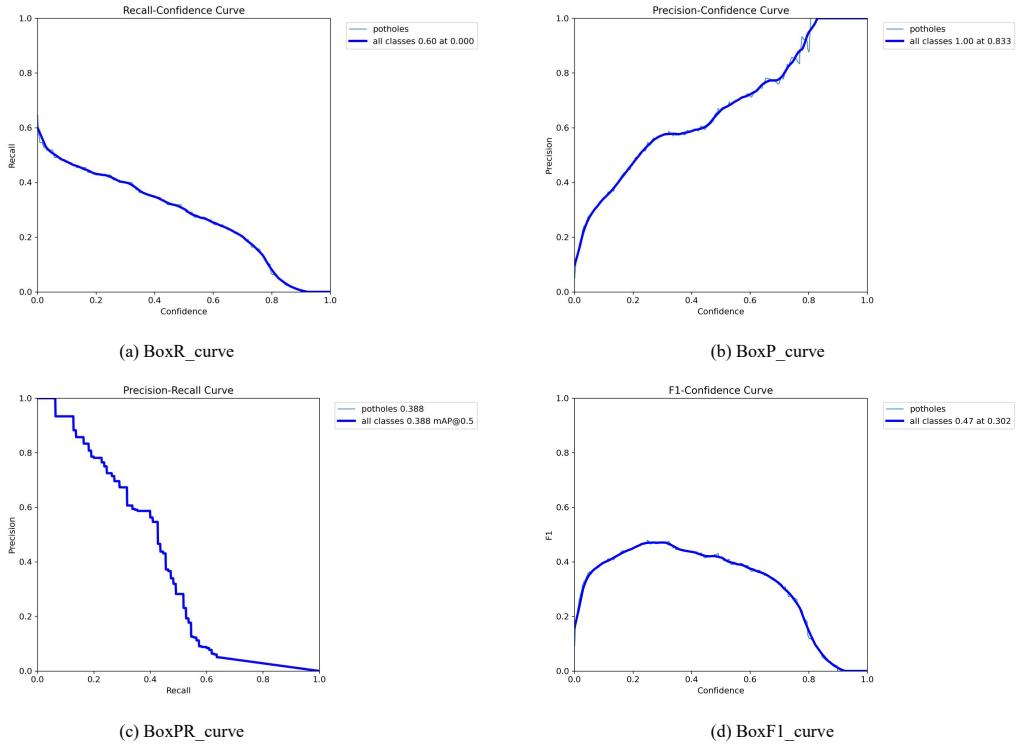


图 28 YOLOv8s-Segment Mask

同时，绘制模型的混淆矩阵热力图，如图 29 所示，采用两种计算，其一为数量统计，其二为数据标准化统计。可以发现对于坑洼的识别，其坑洼预测正确率为 0.51%，对于 background 并未进行训练，因此表现出的效果较差。考虑到各坑洼的轮廓、形态、所在区域的多样及复杂，其预测效果仍可以接受。但出于未来模型的改进，我们仍需要对其进行改进，以提高其预测效果，在模型的评价中，我们也将进行一定叙述。

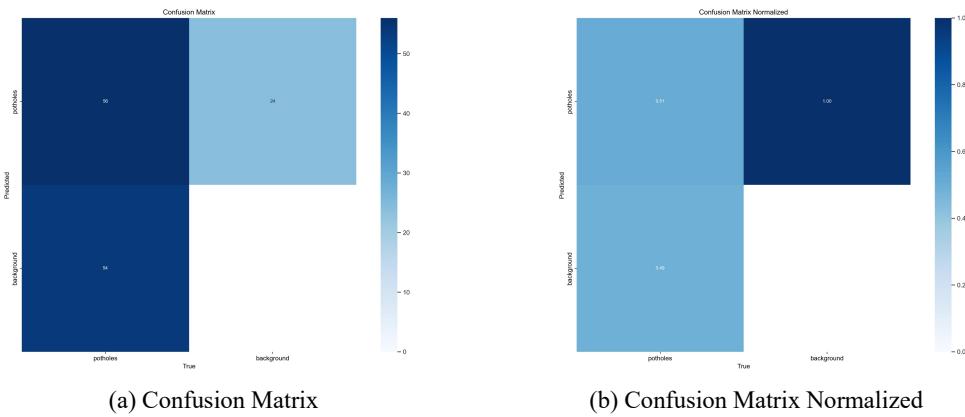


图 29 YOLOv8s-Segment Confusion Matrix

此外，绘制训练过程中每一次 Epoch 的各项指标，如图 30 所示。观察该图，我们可以发现无论是训练集还是验证集的损失值随着迭代的次数的增加，其在整体上有下降趋势。但在

前 20 轮迭代过程中，损失值大幅度上升，在此之后，平稳下降，说明模型效果在逐步上升。同时，可以发现，模型的无论是 Box 还是 Mask 的精确率、召回率、mAP50、mAP50-95 在整体上均随着迭代次数的增加而上升，且在此之后仍有上升趋势。

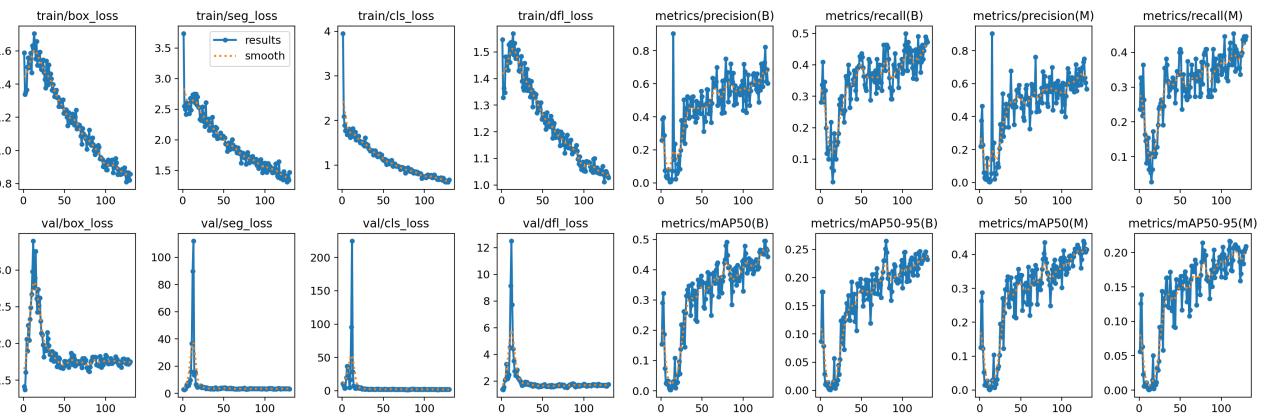


图 30 YOLOv8s-Segment Results

### 5.6.3 YOLOv8n-Detect

对于图像检测模型，其仅有 Box 衡量参数方法，而无 Mask 方法，因此仅绘制其 Box 体系下的各项指标。其余分析与 YOLOv8s-Segment 类似，此处不再赘述，相关图示可在附录中翻阅，如图 33 至图 35 所示。

同时，我们可以得到上述两种模型的最终效果指标，见表 6。

表 6 YOLOv8s-Segment 及 YOLOv8n-Detect 模型指标

| 衡量指标      | YOLOv8s-Segment |         | YOLOv8n-Detect |
|-----------|-----------------|---------|----------------|
|           | Box             | Mask    | Box            |
| precision | 0.90475         | 0.90475 | 0.94427        |
| recall    | 0.50000         | 0.45455 | 0.84615        |
| mAP50     | 0.49485         | 0.43971 | 0.51992        |
| mAP50-95  | 0.26490         | 0.21676 | 0.23120        |

## 六、模型的评价与推广

### 6.1 模型的评价

- 模型的优点

1. 在建立 YOLOv8 模型之前，首先对图像的特征进行了详细分析，为后续建模提供了有效的理论支持；
2. 在对图像坑洼边缘标记时，采用 ISAT-SAM 方法，交互式半自动标注，大大提升了标注效率与准确度，为后续模型的训练提供了有效的数据支持；
3. 对于坑洼边缘的检测与分割，我们选用 YOLOv8 模型，其速度快，准确度高，效果优秀；

4. 建立分割与检测模型，适用于不同场景的需求；
5. 在图像文件预处理时，对数据进行了人为再分类，使数据更准确有效；
6. 在图像数据预处理时，对训练集进行高斯、双边滤波与旋转增强，提升模型的泛化能力；
7. 在 YOLOv8 训练时，采用 Mosaic 数据增强，增加了数据的多样性，丰富了图片的背景，同时也增加了目标个数。
8. 在对图像坑洼面积占比估算时，采用简单的矩形估算，使得计算较为简便，空间复杂度较低。

- **模型的缺点**

1. 在对训练集进行数据增强时，方法可能并不是最佳的，因此在对特征进行学习时可能会有所遗漏；
2. 在计算坑洼面积占比时采用矩形方框估算，准确性相对较低，可能不适用于精度较高的场景。
3. 由于计算机设备受限，并未选用能力更强的 YOLOv8 的 m、l、x 模型。

- **模型的改进**

1. 在进行预测时，可以结合优化算法及修改 YOLO 架构，以提高预测精度。如分布偏移卷积（DSconv）<sup>[13]</sup>、Gather-and-Distribute（GD）机制<sup>[14]</sup>等方法，提高多尺度特征融合能力，并在模型尺度上实现延迟和精度之间的理想平衡。
2. 在计算坑洼面积时可用多边形估算替代矩形估算，使准确性提高，但计算复杂度也会相应提高。此外还可通过标记的图像进行阈值分割，将红色部分（标记为坑洼的）记为 1，否则记为 0，从而计算 0 与 1 的像素点个数，再通过像素点个数计算坑洼面积占比<sup>[15]</sup>。具体解决方案见附录-C.13, Proportional Calculation [面积占比估算（像素点法）]。

## 6.2 模型的推广

此模型具备分析及处理有明显特征的复杂图像的能力，因此除了估算坑洼占比，为道路避障和道路修复等工作提供便利外，我们也可以将其应用于人工智能驾驶的智能避障层面，以提供给乘客更加舒适的乘坐体验，同时也减少了坑洼可能对车辆造成的损坏，降低企业的车辆维护成本。以此类推，除了人工智能驾驶层面，此模型还可应用于环境保护，通过对坑洼面积占比的估算，帮助相关专家及工作人员了解地形地貌对于生态环境的影响，为环境保护工作提供一定的数据支持。

## 参考文献

- [1] 知乎. 用 YOLOv8 一站式解决图像分类、检测、分割 [EB/OL].<https://zhuanlan.zhihu.com/p/655889189>.
- [2] MMYOLO.Algorithm principles and implementation with YOLOv8 —MMYOLO 0.6.0 documentation[EB/OL].[https://mmyolo.readthedocs.io/en/latest/recommended\\_topics/algorithm\\_descriptions/yolov8\\_description.html](https://mmyolo.readthedocs.io/en/latest/recommended_topics/algorithm_descriptions/yolov8_description.html).
- [3] 刘瑞锦, 何章鸣. 基于 YOLOv8 的卫星遥感图像快速目标检测方法 [J]. 空间控制技术与应用,2023,49(05):89-97.
- [4] Zhang H, Cisse M, Dauphin Y N, et al. mixup: Beyond empirical risk minimization[J]. arXiv preprint arXiv:1710.09412, 2017.
- [5] 曹江华. 复杂背景下非结构化道路可行驶区域检测研究 [D]. 浙江科技学院,2021.
- [6] Song G, Liu Y, Wang X. Revisiting the sibling head in object detector[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020: 11563-11572.
- [7] Github.yatengLG/ISAT\_with\_segment\_anything: Labeling tool by SAM(segment anything model),supports SAM, sam-hq, MobileSAM etc. 交互式半自动图像标注工具 [EB/OL].[https://github.com/yatengLG/ISAT\\_with\\_segment\\_anything](https://github.com/yatengLG/ISAT_with_segment_anything).
- [8] 知乎.【教程】标注工具 Labelimg 的安装与使用 [EB/OL].<https://zhuanlan.zhihu.com/p/550021453>.
- [9] CSDN.YOLO 目标检测中损失函数 loss 的理解及部分代码实现 [EB/OL].<https://blog.csdn.net/shengyan5515/article/details/84036734>.
- [10] 知乎. 模型评测:PRECISION、RECALL、F1-score[EB/OL].<https://zhuanlan.zhihu.com/p/519982682>.
- [11] Yellowbrick.Precision-Recall Curves - Yellowbrick v1.5 documentation[EB/OL].<https://www.scikit-yb.org/en/latest/api/classifier/prcurve.html>.
- [12] CSDN. 目标检测评估指标 mAP: 从 Precision,Recall, 到 AP50-95 【未完待续】 [EB/OL].<https://blog.csdn.net/Albert233333/article/details/132752216>.
- [13] Nascimento M G, Fawcett R, Prisacariu V A. Dsconv: Efficient convolution operator[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 5148-5157.
- [14] Wang C, He W, Nie Y, et al. Gold-YOLO: Efficient Object Detector via Gather-and-Distribute Mechanism[J]. arXiv preprint arXiv:2309.11331, 2023.
- [15] CSDN. 利用目标检测分割图进行目标占比计算 [EB/OL].<https://blog.csdn.net/zhn3648/article/details/127458929>.

## 附录

### [A] 支撑文件列表

支撑文件列表如下（列表中不包含原始数据集及测试数据集）：

| 文件（夹）名           | 描述                       |
|------------------|--------------------------|
| html             | 包括所有解决问题的源程序运行结果         |
| ipynb            | 包括所有解决问题的源程序源代码          |
| models           | 已训练好的模型文件                |
| process          | 包括所有模型训练、预测的过程结果         |
| py               | 包括所有解决问题的源程序输出 python 文件 |
| yaml             | 包括所有模型训练的配置文件            |
| test_result2.csv | 坑洼面积占比估算结果               |

## [B] 使用的软件、环境

### B.1 使用的软件及版本

- TeX Live 2022
- Visual Studio Code 1.85.0
- WPS Office 2023 冬季更新 (15990)
- Python 3.10.4 [MSC v.1929 64 bit (AMD64)] on win32
- Pycharm 2023.3 (Professional Edition)

### B.2 模型训练所用计算机配置

- Intel(R) Core(TM) i5-10200H CPU @ 2.40GHz
- NVIDIA GeForce GTX 1650 Ti
- NVIDIA CUDA 11.7.102 driver
- 16.0 GB RAM
- Windows 10 家庭中文版 22H2

### B.3 Python 环境下所用使用到的库及其版本

| 库                        | 版本     | 库             | 版本         |
|--------------------------|--------|---------------|------------|
| jupyter                  | 1.0.0  | jupyter-lsp   | 2.2.0      |
| jupyter_client           | 8.5.0  | matplotlib    | 3.8.0      |
| jupyter_core             | 5.4.0  | numpy         | 1.22.4+mkl |
| jupyter_server           | 2.9.1  | opencv-python | 4.8.1.78   |
| jupyter_server_terminals | 0.4.4  | os            | 内置库        |
| jupyter-console          | 6.6.3  | pandas        | 2.0.3      |
| jupyter-events           | 0.8.0  | scikit-learn  | 1.3.0      |
| jupyterlab               | 4.0.7  | shutil        | 内置库        |
| jupyterlab_server        | 2.25.0 | torch         | 1.11.0     |
| jupyterlab-pygments      | 0.2.2  | tqdm          | 4.64.0     |
| jupyterlab-widgets       | 3.0.9  | ultralytics   | 8.0.206    |

## [C] 问题解决源程序

### C.1 Comparative Analysis of Normal and Potholes [正常与坑洼道路比较分析]

---

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import cv2
8 import numpy as np
9 import matplotlib.pyplot as plt
10
11 # In[2]:
12
13
14 normalImg = cv2.imread('DATA\\normal\\normal133.jpg')
15 potholesImg = cv2.imread('DATA\\potholes\\potholes1.jpg')
16
17 # In[3]:
18
19
20 plt.rcParams['font.sans-serif'] = ['Times New Roman']
21 plt.rcParams['axes.unicode_minus'] = False
22
23
24 # In[4]:
25
26
27 def cv_show(img):
28     b, g, r = cv2.split(img)
29     img = cv2.merge([r, g, b])
30     plt.imshow(img)
31
32
33 # In[5]:
34
35
36 cv_show(normalImg)
37
38 # In[6]:
39
40
41 cv_show(potholesImg)
42
```

```

43 # In[7]:
44
45
46 color = ('b', 'g', 'r')
47
48 for i, col in enumerate(color):
49     histr = cv2.calcHist([normalImg], [i], None, [256], [0, 256])
50     plt.plot(histr, color=col)
51
52 plt.legend(['Blue', 'Green', 'Red'])
53 plt.xlim([0, 256])
54 plt.xticks(fontsize=10)
55 plt.yticks(fontsize=10)
56 plt.title('a) normal133.jpg', y=-0.2, fontsize=12)
57 plt.xlabel('Pixel Value', fontsize=11)
58 plt.ylabel('Number of Pixels', fontsize=11)
59 plt.savefig('Figures\\normal133RGB直方图.pdf', bbox_inches='tight')
60
61 # In[8]:
62
63
64 color = ('b', 'g', 'r')
65
66 for i, col in enumerate(color):
67     histr = cv2.calcHist([potholesImg], [i], None, [256], [0, 256])
68     plt.plot(histr, color=col)
69
70 plt.legend(['Blue', 'Green', 'Red'])
71 plt.xlim([0, 256])
72 plt.xticks(fontsize=10)
73 plt.yticks(fontsize=10)
74 plt.title('b) potholes1.jpg', y=-0.2, fontsize=12)
75 plt.xlabel('Pixel Value', fontsize=11)
76 plt.ylabel('Number of Pixels', fontsize=11)
77 plt.savefig('Figures\\potholes1RGB直方图.pdf', bbox_inches='tight')
78
79 # In[9]:
80
81
82 plt.style.use('ggplot')
83 plt.hist(normalImg.ravel(), 256, [0, 256], color='grey')
84 plt.title('a) normal133.jpg', y=-0.2, fontsize=12)
85 plt.xlabel('Pixel Value', fontsize=11)
86 plt.ylabel('Number of Pixels', fontsize=11)

```

```

87 plt.savefig('Figures\\normal133灰度直方图.pdf', bbox_inches='tight')
88
89 # In[10]:
90
91
92 plt.style.use('ggplot')
93 plt.hist(pothelesImg.ravel(), 256, [0, 256], color='grey')
94 plt.title('(b) potheles1.jpg', y=-0.2, fontsize=12)
95 plt.xlabel('Pixel Value', fontsize=11)
96 plt.ylabel('Number of Pixels', fontsize=11)
97 plt.savefig('Figures\\potheles1灰度直方图.pdf', bbox_inches='tight')
98
99 # In[11]:
100
101
102 # 边缘检测
103 gray = cv2.cvtColor(normalImg, cv2.COLOR_BGR2GRAY)
104 edges = cv2.Canny(gray, 100, 200)
105 plt.imshow(edges, cmap='gray')
106 plt.title('(a) normal133.jpg', y=-0.2, fontsize=12)
107 plt.savefig('Figures\\normal133边缘检测.pdf', bbox_inches='tight')
108
109 # In[12]:
110
111
112 # 边缘检测
113 gray = cv2.cvtColor(pothelesImg, cv2.COLOR_BGR2GRAY)
114 edges = cv2.Canny(gray, 100, 200)
115 plt.imshow(edges, cmap='gray')
116 plt.title('(b) potheles1.jpg', y=-0.2, fontsize=12)
117 plt.savefig('Figures\\potheles1边缘检测.pdf', bbox_inches='tight')
118
119 # In[13]:
120
121
122 plt.imshow(normalImg)
123 plt.colorbar()
124 plt.title('(a) normal133.jpg', y=-0.3, fontsize=12)
125 plt.savefig('Figures\\normal133热力图.pdf')
126
127 # In[14]:
128
129
130 plt.imshow(pothelesImg)

```

```

131 plt.colorbar()
132 plt.title('`potholes1.jpg`', y=-0.3, fontsize=12)
133 plt.savefig('Figures\\potholes1热力图.pdf')
134
135 # In[15]:
136
137
138 # 阈值分割
139 hsv = cv2.cvtColor(normalImg, cv2.COLOR_BGR2HSV)
140 lower_blue = np.array([90, 50, 50])
141 upper_blue = np.array([130, 255, 255])
142 mask = cv2.inRange(hsv, lower_blue, upper_blue)
143 plt.imshow(mask, cmap='gray')
144 plt.title('`normal133.jpg`', y=-0.2, fontsize=12)
145 plt.savefig('Figures\\normal133阈值分割.pdf', bbox_inches='tight')
146
147 # In[16]:
148
149
150 # 阈值分割
151 hsv = cv2.cvtColor(potholesImg, cv2.COLOR_BGR2HSV)
152 lower_blue = np.array([90, 50, 50])
153 upper_blue = np.array([130, 255, 255])
154 mask = cv2.inRange(hsv, lower_blue, upper_blue)
155 plt.imshow(mask, cmap='gray')
156 plt.title('`potholes1.jpg`', y=-0.2, fontsize=12)
157 plt.savefig('Figures\\potholes1阈值分割.pdf', bbox_inches='tight')
158
159 # In[17]:
160
161
162 # 转换为灰度图像
163 gray = cv2.cvtColor(normalImg, cv2.COLOR_BGR2GRAY)
164 # 二值化
165 ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
166 # 轮廓检测
167 contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
168 # 绘制轮廓
169 cv2.drawContours(normalImg, contours, -1, (0, 0, 255), 3)
170
171 plt.subplot(1, 2, 1)
172 plt.title('`Original`', y=-0.4, fontsize=12)
173 plt.imshow(normalImg)
174 plt.subplot(1, 2, 2)

```

```

175 plt.imshow(binary, cmap='gray')
176 plt.title('（b） Contours', y=-0.4, fontsize=12)
177 plt.savefig('Figures\\normal133轮廓检测.pdf', bbox_inches='tight')
178
179 # In[18]:
180
181
182 # 转换为灰度图像
183 gray = cv2.cvtColor(pothelesImg, cv2.COLOR_BGR2GRAY)
184 # 二值化
185 ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
186 # 轮廓检测
187 contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
188 # 绘制轮廓
189 cv2.drawContours(pothelesImg, contours, -1, (0, 0, 255), 3)
190
191 plt.subplot(1, 2, 1)
192 plt.title('（a） Original', y=-0.4, fontsize=12)
193 plt.imshow(pothelesImg)
194 plt.subplot(1, 2, 2)
195 plt.imshow(binary, cmap='gray')
196 plt.title('（b） Contours', y=-0.4, fontsize=12)
197 plt.savefig('Figures\\potheles1轮廓检测.pdf', bbox_inches='tight')

```

---

## C.2 Data Preprocessing [数据预处理]

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import os
8 import shutil
9
10 # 指定目录"DATA"
11 path = "DATA"
12
13 # 在"DATA"文件夹中创建"normal"和"potheles"文件夹
14 os.mkdir(os.path.join(path, "normal"))
15 os.mkdir(os.path.join(path, "potheles"))
16
17 # 读取"DATA"文件夹, 若文件名中含有"normal", 则将其放置于"normal"文件夹中, 否则放置于"potheles"文件夹中
18 files = os.listdir(path)

```

```
19 for file in files:
20     if "normal" in file:
21         shutil.move(os.path.join(path, file), os.path.join(path, "normal"))
22     else:
23         shutil.move(os.path.join(path, file), os.path.join(path, "potholes"))
```

---

### C.3 Images Display [图像展示]

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import cv2
8
9 img = cv2.imread("DATA\\potholes\\potholes1.jpg", cv2.IMREAD_COLOR)
10 img = cv2.resize(img, [256, 256])
11
12 Gaussian = cv2.GaussianBlur(img, (3, 3), 1)
13 Bilateral = cv2.bilateralFilter(img, 9, 75, 75)
14 Rotate = cv2.warpAffine(img, cv2.getRotationMatrix2D((img.shape[1] / 2, img.shape[0] /
15             2), 30, 1),
16                         (img.shape[1], img.shape[0]))
17
18 cv2.imwrite("potholes1_Gaussian.jpg", Gaussian)
19 cv2.imwrite("potholes1_Bilateral.jpg", Bilateral)
20 cv2.imwrite("potholes1_Rotate.jpg", Rotate)
```

---

### C.4 isat2txt [isat 转 txt]

```
1 import json
2 import os
3 import shutil
4
5 category_mapping = {"potholes": 0}
6 # ISAT格式的实例分割标注文件
7 ISAT_FOLDER = "./annotations"
8 # YOLO格式的实例分割标注文件
9 YOLO_FOLDER = "./labels"
10
11 # 创建YoloV8标注的文件夹
12 if not os.path.exists(YOLO_FOLDER):
13     os.makedirs(YOLO_FOLDER)
14
15
```

```

16 # 载入所有的ISAT的JSON文件
17 for filename in os.listdir(ISAT_FOLDER):
18     if not filename.endswith(".json"):
19         # 不是json格式，跳过
20         continue
21     # 载入ISAT的JSON文件
22     with open(os.path.join(ISAT_FOLDER, filename), "r") as f:
23         isat = json.load(f)
24     # 提取文件名(不带文件后缀)
25     image_name = filename.split(".") [0]
26     # Yolo格式的标注文件名，后缀是txt
27     yolo_filename = f"{image_name}.txt"
28     # 写入信息
29     with open(os.path.join(YOLO_FOLDER, yolo_filename), "w") as f:
30         # 获取图像信息
31         # - 图像宽度
32
33         image_width = isat["info"]["width"]
34
35         # try:
36         #     image_width = isat["info"]["width"]
37         # except KeyError:
38         #     break
39         # - 图像高度
40         image_height = isat["info"]["height"]
41     # 获取实例标注数据
42     for annotation in isat["objects"]:
43         # 获取类别名称
44         category_name = annotation["category"]
45         # 如果不在类别名称字典里面，跳过
46         if category_name not in category_mapping:
47             continue
48         # 从字典里面查询类别ID
49         category_id = category_mapping[category_name]
50         # 提取分割信息
51         segmentation = annotation["segmentation"]
52         segmentation_yolo = []
53         # 遍历所有的轮廓点
54         for segment in segmentation:
55             # 提取轮廓点的像素坐标 x, y
56             x, y = segment
57             # 归一化处理
58             x_center = x / image_width
59             y_center = y / image_height

```

```

60     # 添加到segmentation_yolo里面
61     segmentation_yolo.append(f"{{x_center:.4f} {y_center:.4f}}")
62     segmentation_yolo_str = " ".join(segmentation_yolo)
63     # 添加一行Yolo格式的实例分割数据
64     # 格式如下: class_id x1 y1 x2 y2 ... xn yn\n
65     f.write(f"{category_id} {segmentation_yolo_str}\n")

```

---

### C.5 xml2txt [xml 转 txt]

---

```

1 import xml.etree.ElementTree as ET
2 import os, cv2
3 import numpy as np
4 from os import listdir
5 from os.path import join
6
7 #将给定的XML格式的标注文件转换为YOLOv8所需的TXT格式，并记录数据集中的类别信息
8 classes = []
9
10 def convert(size, box):
11     dw = 1. / (size[0])
12     dh = 1. / (size[1])
13     x = (box[0] + box[1]) / 2.0 - 1
14     y = (box[2] + box[3]) / 2.0 - 1
15     w = box[1] - box[0]
16     h = box[3] - box[2]
17     x = x * dw
18     w = w * dw
19     y = y * dh
20     h = h * dh
21     return (x, y, w, h)
22
23
24 def convert_annotation(xmlpath, xmlname):
25     with open(xmlpath, "r", encoding='utf-8') as in_file:
26         txtname = xmlname[:-4] + '.txt'
27         txtfile = os.path.join(txtpath, txtname)
28         tree = ET.parse(in_file)
29         root = tree.getroot()
30         filename = root.find('filename')
31         img = cv2.imdecode(np.fromfile('{}/{}.{}'.format(imgpath, xmlname[:-4], postfix),
32                                         np.uint8), cv2.IMREAD_COLOR)
33         h, w = img.shape[:2]
34         res = []
35         for obj in root.iter('object'):
            cls = obj.find('name').text

```

```

36         if cls not in classes:
37             classes.append(cls)
38             cls_id = classes.index(cls)
39             xmlbox = obj.find('bndbox')
40             b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),
41                   float(xmlbox.find('ymax').text))
42             bb = convert((w, h), b)
43             res.append(str(cls_id) + " " + " ".join([str(a) for a in bb]))
44         if len(res) != 0:
45             with open(txtfile, 'w+') as f:
46                 f.write('\n'.join(res))
47
48
49 if __name__ == "__main__":
50     postfix = 'jpg'
51     imgpath = 'VOCdevkit/JPEGImages'
52     xmlpath = 'VOCdevkit/Annotations'
53     txtpath = 'VOCdevkit/txt'
54
55     if not os.path.exists(txtpath):
56         os.makedirs(txtpath, exist_ok=True)
57
58     list = os.listdir(xmlpath)
59     error_file_list = []
60     for i in range(0, len(list)):
61         try:
62             path = os.path.join(xmlpath, list[i])
63             if ('.xml' in path) or ('.XML' in path):
64                 convert_annotation(path, list[i])
65                 print(f'file {list[i]} convert success.')
66             else:
67                 print(f'file {list[i]} is not xml format.')
68         except Exception as e:
69             print(f'file {list[i]} convert error.')
70             print(f'error message:\n{e}')
71             error_file_list.append(list[i])
72     print(f'this file convert failure\n{error_file_list}')
73     print(f'Dataset Classes:{classes}')

```

---

## C.6 Segment Spilt Data [图像分割数据集划分]

---

```

1 import os
2 import random
3 from tqdm import tqdm

```

```

4
5 # 指定 images 文件夹路径
6 image_dir = "./images"
7 # 指定 labels 文件夹路径
8 label_dir = "./labels"
9 # 创建一个空列表来存储有效图片的路径
10 valid_images = []
11 # 创建一个空列表来存储有效 label 的路径
12 valid_labels = []
13
14 # 遍历 images 文件夹下的所有图片
15 for image_name in os.listdir(image_dir):
16     # 获取图片的完整路径
17     image_path = os.path.join(image_dir, image_name)
18     # 获取图片文件的扩展名
19     ext = os.path.splitext(image_name)[-1]
20     # 根据扩展名替换成对应的 label 文件名
21     label_name = image_name.replace(ext, ".txt")
22     # 获取对应 label 的完整路径
23     label_path = os.path.join(label_dir, label_name)
24     # 判断 label 是否存在
25     if not os.path.exists(label_path):
26         # 删除图片
27         os.remove(image_path)
28         print("deleted:", image_path)
29     else:
30         # 将图片路径添加到列表中
31         valid_images.append(image_path)
32         # 将label路径添加到列表中
33         valid_labels.append(label_path)
34         # print("valid:", image_path, label_path)
35 # 遍历每个有效图片路径
36 for i in tqdm(range(len(valid_images))):
37     image_path = valid_images[i]
38     label_path = valid_labels[i]
39     # 随机生成一个概率
40     r = random.random()
41     # 判断图片应该移动到哪个文件夹
42     # train: valid: test = 7:2:1
43     if r < 0.1:
44         # 移动到 test 文件夹
45         destination = "./datasets/test"
46     elif r < 0.3:
47         # 移动到 valid 文件夹

```

```

48     destination = "./datasets/valid"
49 else:
50     # 移动到 train 文件夹
51     destination = "./datasets/train"
52 # 生成目标文件夹中图片的新路径
53     image_destination_path = os.path.join(destination, "images", os.path.basename(
54         image_path))
55     # 移动图片到目标文件夹
56     os.rename(image_path, image_destination_path)
57 # 生成目标文件夹中 label 的新路径
58     label_destination_path = os.path.join(destination, "labels", os.path.basename(
59         label_path))
60     # 移动 label 到目标文件夹
61     os.rename(label_path, label_destination_path)
62 print("valid images:", valid_images)
63 # 输出有效label路径列表
64 print("valid labels:", valid_labels)

```

---

## C.7 Detect Spilt Data [图像检测数据集划分]

---

```

1 import os, shutil
2 from sklearn.model_selection import train_test_split
3
4 # 用于将图像和标注文件按照一定比例划分为训练集、验证集和测试集，并将它们分别复制到相应的文
5    件夹中
6 val_size = 0.1
7 test_size = 0.1
8 postfix = 'jpg'
9 imgpath = 'VOCdevkit/JPEGImages'
10 txtpath = 'VOCdevkit/txt'
11 os.makedirs('images/train', exist_ok=True)
12 os.makedirs('images/val', exist_ok=True)
13 os.makedirs('images/test', exist_ok=True)
14 os.makedirs('labels/train', exist_ok=True)
15 os.makedirs('labels/val', exist_ok=True)
16 os.makedirs('labels/test', exist_ok=True)
17
18 listdir = [i for i in os.listdir(txtpath) if 'txt' in i]
19 train, test = train_test_split(listdir, test_size=test_size, shuffle=True, random_state
=0)
20 train, val = train_test_split(train, test_size=val_size, shuffle=True, random_state=0)
21 print(f'train set size:{len(train)} val set size:{len(val)} test set size:{len(test)}')
22
23 for i in train:

```

```

24     shutil.copy('{}.{}.{}'.format(imgpath, i[:-4], postfix), 'images/train/{}.{}'.format(i[:-4], postfix))
25     shutil.copy('{}.{}'.format(txtpath, i), 'labels/train/{}'.format(i))
26
27 for i in val:
28     shutil.copy('{}.{}.{}'.format(imgpath, i[:-4], postfix), 'images/val/{}.{}'.format(i[:-4], postfix))
29     shutil.copy('{}.{}'.format(txtpath, i), 'labels/val/{}'.format(i))
30
31 for i in test:
32     shutil.copy('{}.{}.{}'.format(imgpath, i[:-4], postfix), 'images/test/{}.{}'.format(i[:-4], postfix))
33     shutil.copy('{}.{}'.format(txtpath, i), 'labels/test/{}'.format(i))

```

---

## C.8 YOLOv8s Segment [YOLOv8s 图像分割]

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 from ultralytics import YOLO
8
9 # In[3]:
10
11
12 model = YOLO('yolov8s-seg.pt')
13 results = model.train(data="dataset/VOCdevkit/datasets/potholes.yaml", epochs=300,
14                         device='cpu')
15 # In[2]:
16
17
18 model = YOLO('potholesSegment.pt')
19
20 # In[3]:
21
22
23 resultOne = model('potholes\\part1', save=True)
24
25 # In[4]:
26
27
28 resultTwo = model('potholes\\part2', save=True)

```

```
29
30 # In[5]:
31
32
33 resultThree = model('potholes\\part3', save=True)
34
35 # In[3]:
36
37
38 resultFour = model('potholes\\part4', save=True)
39
40 # In[4]:
41
42
43 resultFive = model('potholes\\part5', save=True)
44
45 # In[3]:
46
47
48 resultSix = model('potholes\\part6', save=True)
49
50 # In[4]:
51
52
53 resultSeven = model('potholes\\part7', save=True)
54
55 # In[3]:
56
57
58 resultEight = model('potholes\\part8', save=True)
```

---

## C.9 YOLOv8n Detect [YOLOv8n 图像检测]

---

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 from ultralytics import YOLO
8
9
10 # In[47]:
```

```

13 model = YOLO("potholesDetect.pt")
14 resultsOne = model.predict(source="potholes\\partOne", save=True, save_conf=True,
15     save_txt=True, imgsz=416)
16
17 # In[48]:
18
19
20 model = YOLO("potholesDetect.pt")
21 resultsTwo = model.predict(source="potholes\\partTwo", save=True, save_conf=True,
22     save_txt=True, imgsz=416)

```

---

## C.10 Coordinate Transformation [坐标转换]

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import os
8 import cv2
9 import torch
10 import numpy as np
11
12 # In[2]:
13
14
15 label_path = 'potholes\labels'
16 image_path = 'potholes\images'
17
18
19 def xywhn2xyxy(x, w=416, h=416, padw=0, padh=0):
20     y = x.clone() if isinstance(x, torch.Tensor) else np.copy(x)
21     y[:, 0] = w * (x[:, 0] - x[:, 2] / 2) + padw # top left x
22     y[:, 1] = h * (x[:, 1] - x[:, 3] / 2) + padh # top left y
23     y[:, 2] = w * (x[:, 0] + x[:, 2] / 2) + padw # bottom right x
24     y[:, 3] = h * (x[:, 1] + x[:, 3] / 2) + padh # bottom right y
25     return y
26
27
28 folder = os.path.exists('potholes\\labelsNew')
29 if not folder:
30     os.makedirs('potholes\\labelsNew')
31

```

```

32 folderlist = os.listdir(label_path)
33 for i in folderlist:
34     label_path_new = os.path.join(label_path, i)
35     with open(label_path_new, 'r') as f:
36         lb = np.array([x.split() for x in f.read().strip().splitlines()], dtype=np.
37             float32) # labels
38         print(lb)
39         read_label = label_path_new.replace(".txt", ".jpg")
40         read_label_path = read_label.replace(label_path, image_path)
41         print(read_label_path, "ddd")
42         img = cv2.imread(str(read_label_path))
43         h, w = img.shape[:2]
44         lb[:, 1:] = xywhn2xyxy(lb[:, 1:], w, h, 0, 0)
45
46         for _, x in enumerate(lb):
47             class_label = int(x[0])
48
49             with open('potholes\\labelsNew\\' + i, 'a') as fw:
50                 fw.write(str(int(x[0])) + ' ' + str(x[5]) + ' ' + str(x[1]) + ' ' +
51                         str(x[2]) + ' ' + str(x[3]) + ' ' + str(
52                             x[4]) + '\n')

```

---

## C.11 Proportional Calculation [面积占比估算（坐标法）]

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import os
8 import cv2
9 import numpy as np
10 import pandas as pd
11
12 # In[2]:
13
14
15 labelsList = os.listdir('potholes\\labelsNew')
16 imagesList = os.listdir('potholes\\images')
17
18 # In[3]:
19
20
21 labelsDf = pd.DataFrame(labelsList, columns=['fnames'])

```

```

22 labelsDf
23
24 # In[4]:
25
26
27 Npotholes = pd.read_csv('potholes\\detectResults.txt', sep=' ', header=None)
28 Npotholes = Npotholes.drop([0, 1, 3, 5, 6], axis=1)
29 Npotholes.columns = ['fnames', 'Npotholes']
30 Npotholes['fnames'] = Npotholes['fnames'].str.split('\\').str[-1]
31 Npotholes['Npotholes'] = Npotholes['Npotholes'].str.replace('(', '')
32 Npotholes['fnames'] = Npotholes['fnames'].str.replace(':', '')
33 Npotholes
34
35 # In[5]:
36
37
38 Npotholes['Npotholes'] = Npotholes['Npotholes'].str.replace('no', '0')
39 Npotholes['Npotholes'] = Npotholes['Npotholes'].astype('int')
40 Npotholes
41
42 # In[6]:
43
44
45 data = pd.DataFrame(columns=['Type', '置信度', 'x1', 'y1', 'x2', 'y2', 'fnames'])
46 data
47
48 # In[7]:
49
50
51 for i in labelsList:
52     with open('potholes\\labelsNew\\' + i, 'r') as f:
53         content = f.readlines()
54         lb = np.array([x.strip().split() for x in content], dtype=np.float32) # labels
55         lb = pd.DataFrame(lb, columns=['Type', '置信度', 'x1', 'y1', 'x2', 'y2'])
56         lb['fnames'] = i
57
58     data = data.append(lb, ignore_index=True)
59
60 data
61
62 # In[8]:
63
64
65 data = data.drop('Type', axis=1)

```

```

66 data = data[['fnames', '置信度', 'x1', 'y1', 'x2', 'y2']]
67 data['fnames'] = data['fnames'].str.replace('.txt', '.jpg')
68 data
69
70 # In[9]:
71
72
73 data = pd.merge(data, Npotholes, on='fnames', how='right')
74 data
75
76 # In[10]:
77
78
79 potholesImages = pd.DataFrame(imagesList, columns=['fnames'])
80 potholesImages
81
82 # In[11]:
83
84
85 for i in imagesList:
86     img = cv2.imread('potholes\\images\\' + i)
87     height, width, channels = img.shape
88     potholesImages.loc[potholesImages['fnames'] == i, 'height'] = height
89     potholesImages.loc[potholesImages['fnames'] == i, 'width'] = width
90
91 potholesImages
92
93 # In[12]:
94
95
96 data = pd.merge(data, potholesImages, on='fnames', how='left')
97 data
98
99 # In[13]:
100
101
102 data['S'] = data['height'] * data['width']
103 data['X'] = data['x2'] - data['x1']
104 data['Y'] = data['y2'] - data['y1']
105 data['Spotholes'] = data['X'] * data['Y']
106 data['Spotholes/S'] = data['Spotholes'] / data['S'] * 100
107 data
108
109 # In[14]:

```

```

110
111
112 data['Spotholes/S'] = data['Spotholes/S'].fillna(data['Spotholes/S'].mean())
113 data['Spotholes/S'] = data['Spotholes/S'].apply(np.ceil)
114 data['Spotholes/S'] = data['Spotholes/S'].astype('int')
115 data
116
117 # In[15]:
118
119
120 dataNew = data.groupby('fnames').apply(lambda x: x.loc[x['Spotholes/S'].idxmax()])
121 dataNew
122
123 # In[16]:
124
125
126 dataNew = dataNew.reset_index(drop=True)
127 dataNew
128
129 # In[17]:
130
131
132 dataR = pd.read_csv('未知数据集预测效果.csv')
133 dataR = dataR.drop(['imgType', 'label', 'isTrue'], axis=1)
134 dataR
135
136 # In[18]:
137
138
139 Results = pd.merge(dataR, dataNew, on='fnames', how='left')
140 Results
141
142 # In[19]:
143
144
145 # 修改guy7iodk.jpg行, height=1587, width=1200, S=1587*1200=1904400, Spotholes=954792,
     Spotholes/S=50
146 Results.loc[Results['fnames'] == 'guy7iodk.jpg', 'height'] = 1587
147 Results.loc[Results['fnames'] == 'guy7iodk.jpg', 'width'] = 1200
148 Results.loc[Results['fnames'] == 'guy7iodk.jpg', 'S'] = 1904400
149 Results.loc[Results['fnames'] == 'guy7iodk.jpg', 'Spotholes'] = 954792
150 Results.loc[Results['fnames'] == 'guy7iodk.jpg', 'Spotholes/S'] = 50
151 Results
152

```

```

153 # In[20]:
154
155
156 Results.loc[Results['imgClass'] == 'normal', 'Spotholes/S'] = 0
157 Results
158
159 # In[21]:
160
161
162 Results['Spotholes/S'] = Results['Spotholes/S'].astype('int')
163 Results
164
165 # In[22]:
166
167
168 Results.to_csv('SpotholesS.csv', index=False)

```

---

## C.12 Detect Segment Analyze [图像检测与图像分割效果分析]

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import pandas as pd
8
9 dataDetect = pd.read_csv('detectResults.txt', sep=' ', header=None)
10 dataSegment = pd.read_csv('segmentResults.txt', sep=' ', header=None)
11
12 # In[2]:
13
14
15 dataDetect = dataDetect.drop([0, 1, 3, 5, 6], axis=1)
16 dataDetect.columns = ['fnames', 'Detect-Npotholes']
17 dataDetect['fnames'] = dataDetect['fnames'].str.split('\\').str[-1]
18 dataDetect['Detect-Npotholes'] = dataDetect['Detect-Npotholes'].str.replace('(', '')
19 dataDetect['fnames'] = dataDetect['fnames'].str.replace(':', '')
20 dataDetect['Detect-Npotholes'] = dataDetect['Detect-Npotholes'].str.replace('no', '0')
21 dataDetect['Detect-Npotholes'] = dataDetect['Detect-Npotholes'].astype('int')
22 dataDetect
23
24 # In[3]:
25
26

```

```

27 dataSegment = dataSegment.drop([0, 1, 3, 5, 6], axis=1)
28 dataSegment.columns = ['fnames', 'Segment-Npotholes']
29 dataSegment['fnames'] = dataSegment['fnames'].str.split('\\').str[-1]
30 dataSegment['Segment-Npotholes'] = dataSegment['Segment-Npotholes'].str.replace('(', '')
31 dataSegment['fnames'] = dataSegment['fnames'].str.replace(':', '')
32 dataSegment['Segment-Npotholes'] = dataSegment['Segment-Npotholes'].str.replace('no', '0')
33 dataSegment['Segment-Npotholes'] = dataSegment['Segment-Npotholes'].astype('int')
34 dataSegment
35
36 # In[4]:
37
38
39 data = pd.merge(dataDetect, dataSegment, on='fnames', how='left')
40 data
41
42 # In[5]:
43
44
45 # DetectFalse为Detect-Npotholes为0的个数
46 # DetectTrue为Detect-Npotholes不为0的个数
47 # SegmentFalse为Segment-Npotholes为0的个数
48 # SegmentTrue为Segment-Npotholes不为0的个数
49 DetectFalse = 0
50 DetectTrue = 0
51 SegmentFalse = 0
52 SegmentTrue = 0
53 for i in range(len(data)):
54     if data['Detect-Npotholes'][i] == 0:
55         DetectFalse += 1
56     else:
57         DetectTrue += 1
58     if data['Segment-Npotholes'][i] == 0:
59         SegmentFalse += 1
60     else:
61         SegmentTrue += 1
62
63 # In[6]:
64
65
66 DetectTrue / (DetectFalse + DetectTrue)
67
68 # In[7]:

```

```
69  
70  
71 SegmentTrue / (SegmentFalse + SegmentTrue)
```

---

### C.13 Proportional Calculation [面积占比估算（像素点法）]

---

```
1 import os  
2 from PIL import Image  
3 import numpy as np  
4  
5 path = './image/' # 图片路径  
6 dataname = 'data.txt' # 存放计算结果  
7 red_data = 'red.txt' # 存放红色像素数量  
8 black_data = 'black.txt' # 存放黑色像素数量  
9  
10 files = os.listdir(path) # 读取图片文件夹  
11 for file in files:  
12     img = Image.open(path+file) # 读取图片  
13     img_array = np.array(img)      # 将图片转换成二进制数组  
14     shape = img_array.shape      # 得到数组尺寸大小  
15  
16  
17     black = 0  
18     red = 0  
19     for i in range(0,shape[0]):    # 遍历图片像素  
20         for j in range(0,shape[1]):  
21             value = img_array[ i , j ] # 得到像素每一点的值  
22             if value == 0:           # 图片为二值化图片，黑色为0，红色为1  
23                 black += 1 # 黑色像素数量  
24             if value == 1:  
25                 red += 1 # 红色像素数量  
26  
27     per_image = red/(black+red) # 红色像素占整张图像素比例  
28     print(per_image)  
29  
30     with open (dataname,'a') as f: # 将结果写入txt文档  
31         f.write(str(per_image)+"\n")  
32     with open (red_data , 'a') as f:  
33         f.write(str(red)+"\n")  
34     with open (black_data , 'a') as f:  
35         f.write(str(black)+"\n")
```

---

## [D] 模型训练配置文件

### D.1 segment.yaml

---

```
1 # Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [
2   path/to/imgs1, path/to/imgs2, ...]
3 # path: ./datasets # dataset root dir
4 train: E:/University/PythonProject/UltralyticsPotholes/dataset/VOCdevkit/datasets/train
5   /images # train images (relative to 'path') 128 images
6 val: E:/University/PythonProject/UltralyticsPotholes/dataset/VOCdevkit/datasets/valid/
7   images # val images (relative to 'path') 128 images
8 test: E:/University/PythonProject/UltralyticsPotholes/dataset/VOCdevkit/datasets/test/
9   images # test images (optional)
10
11 # Classes
12 names:
13   0: potholes
```

---

### D.2 detect.yaml

---

```
1 # dataset path
2 train: E:/University/PythonProject/UltralyticsMaster/dataset/images/train
3 val: E:/University/PythonProject/UltralyticsMaster/dataset/images/val
4 test: E:/University/PythonProject/UltralyticsMaster/dataset/images/test
5
6 # number of classes
7 nc: 1
8
9 # class names
10 names: ['potholes']
```

---

### D.3 YOLOv8-seg.yaml

---

```
1 # Parameters
2 nc: 1 # number of classes
3 scales: # model compound scaling constants, i.e. 'model=yolov8n-seg.yaml' will call
4   yolov8-seg.yaml with scale 'n'
5   # [depth, width, max_channels]
6   n: [0.33, 0.25, 1024]
7   s: [0.33, 0.50, 1024]
8   m: [0.67, 0.75, 768]
9   l: [1.00, 1.00, 512]
10  x: [1.00, 1.25, 512]
11
12 # YOLOv8.0n backbone
13 backbone:
14   # [from, repeats, module, args]
15   - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
```

```

15   - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
16   - [-1, 3, C2f, [128, True]]
17   - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
18   - [-1, 6, C2f, [256, True]]
19   - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
20   - [-1, 6, C2f, [512, True]]
21   - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
22   - [-1, 3, C2f, [1024, True]]
23   - [-1, 1, SPPF, [1024, 5]] # 9
24
25 # YOLOv8.On head
26 head:
27   - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
28   - [[-1, 6], 1, Concat, [1]] # cat backbone P4
29   - [-1, 3, C2f, [512]] # 12
30
31   - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
32   - [[-1, 4], 1, Concat, [1]] # cat backbone P3
33   - [-1, 3, C2f, [256]] # 15 (P3/8-small)
34
35   - [-1, 1, Conv, [256, 3, 2]]
36   - [[-1, 12], 1, Concat, [1]] # cat head P4
37   - [-1, 3, C2f, [512]] # 18 (P4/16-medium)
38
39   - [-1, 1, Conv, [512, 3, 2]]
40   - [[-1, 9], 1, Concat, [1]] # cat head P5
41   - [-1, 3, C2f, [1024]] # 21 (P5/32-large)
42
43   - [[15, 18, 21], 1, Segment, [nc, 32, 256]] # Segment(P3, P4, P5)

```

---

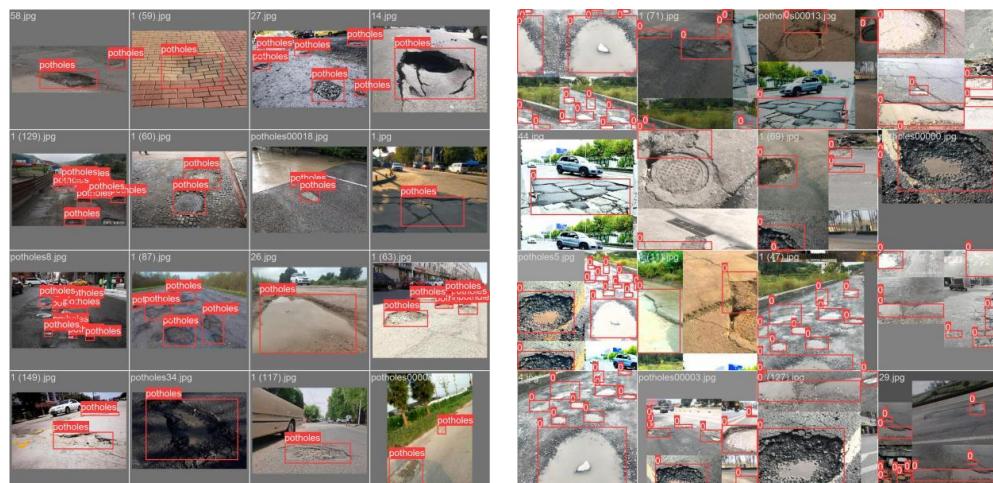
## [D] 图示



(a) train\_batch1

(b) train\_batch2

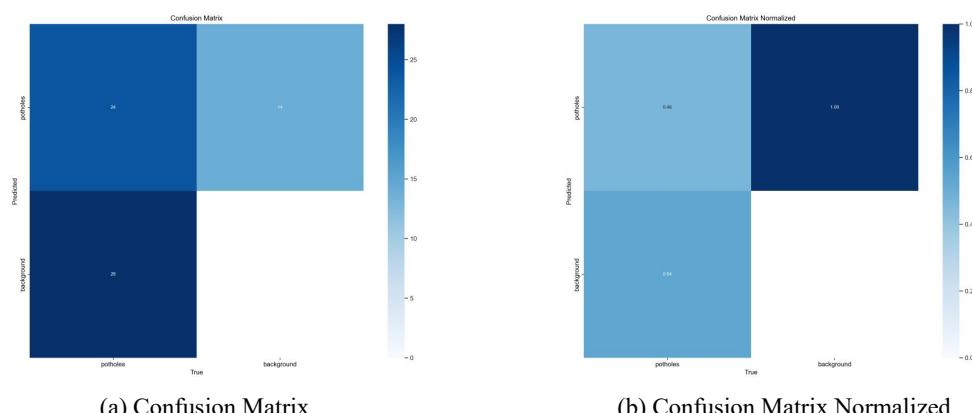
图 31 附 Segment 训练



(a) train\_batch1

(b) train\_batch2

图 32 附 Detect 训练



(a) Confusion Matrix

(b) Confusion Matrix Normalized

图 33 YOLOv8s-Detect Confusion Matrix

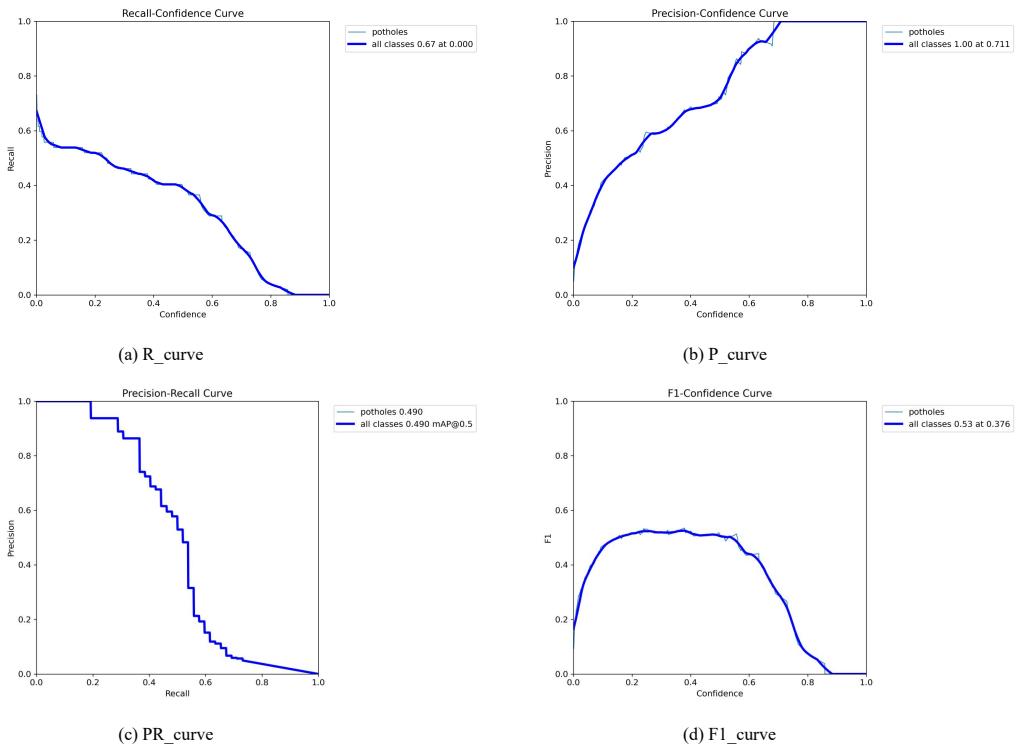


图 34 YOLOv8n-Detect Box

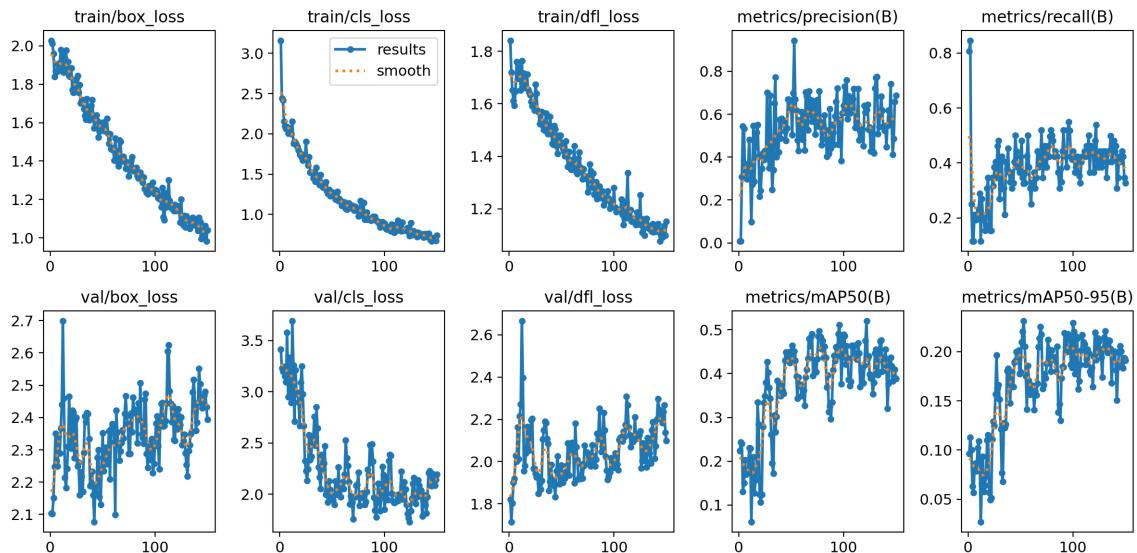


图 35 YOLOv8s-Detect Results