队伍编号	MCB2302586
题号	A

基于 DeepLab v3+ 的坑洼道路图像语义分割

摘要

本文构建一个基于深度学习的语义分割模型 DeepLab v3+，融合空洞卷积、空间特征金字塔池化等模块，自行构建并标注数据集，采用数据增广方法扩充训练样本以驱动模型训练。

对于问题一，由于传统边缘特征提取算法与目标检测算法对于坑洼道路边缘提取任务均无法适用，故考虑采用**基于深度学习的语义分割方法**来对坑洼道路图像进行语义分割。由于题目仅公布测试数据，本文从公开数据集中提取 **400 张**坑洼道路图像，使用 **Lable me 工具**自行对数据集进行**像素级语义信息**标注。然后，使用**垂直翻转**、**水平翻转**、**高斯模糊**等数据增广手段丰富训练数据空间特征多样性，扩充训练样本总数达 1600 张。本文采用基于**编码器-解码器**结构的 DeepLab v3+ 模型对坑洼道路图像执行语义分割任务，使分割后图像与原图像保持尺寸一致。在编码器结构中，使用 **ResNet-101** 作为特征提取骨干网络用于提取坑洼特征。同时，该模型使用**空洞卷积**来扩大感受野范围，抑制语义信息损失，使用**空间特征金字塔池化**模块来提取坑洼道路图像中的**多尺度特征**，赋予模型更强的特征提取能力与泛化能力。在解码器结构中，模型采用**双线性插值法**对特征图进行上采样来扩大特征图尺寸，将输出图像恢复到与原始图像一致的尺寸。为了使 DeepLab v3+ 模型在坑洼道路数据集上更快更稳定地收敛，首先对模型进行**迁移学习预训练**，然后在坑洼道路数据集上进行微调。在训练过程中，选取**随机梯度下降**作为参数优化器，并对每轮次的学习率进行动态调整。本文选取 **PA**、**CPA**、**MIoU**、**FWIoU** 四个指标来评测模型的语义分割性能，选取 **DRN**、**MobileNet**、**ResNet-101** 来验证特征提取骨干网络的性能，选取 **U-Net**、**SegNet**、**DeepLab v3+** 来验证语义分割模型的性能。最后，DeepLab v3+ 语义分割模型在自主构建的测试数据集上的 AP、CAP、MIoU、FWIoU 分别达到 **0.9563**、**0.9353**、**0.8816**、**0.9177**。

对于问题二，为简化语义分割计算量，首先调用初赛的**基于伪样本生成增广与通道注意力增强的坑洼道路图像分类模型**对坑洼道路图像题目所给数据进行分类，从中分类出坑洼道路图像。然后使用问题一建立的坑 DeepLab v3+ 语义分割模型对筛选出的图像进行语义分割，输出**精确到像素点的语义信息图**。最后根据 RGB 图像通道矩阵中的非零数值统计坑洼像素点个数，从而计算坑洼面积占比。

关键字： **边缘检测　计算机视觉　图像分割**

目录

一、 问题重述

1.1 问题背景

通过数字图像分析技术辨识出存在坑洼的道路，不仅有助于保障交通安全，提高驾驶条件，还可以促进自动驾驶汽车智能化发展。除此之外，这一机器视觉任务对于地质勘探、航天科学和自然灾害等多个领域的研究和应用具有重要意义。

传统的分类算法通常难以识别出坑洼图像复杂多变的特征，因此在特征提取和分类任务上表现有限。近年来，深度学习技术在计算机视觉领域崭露头角，它具备卓越的特征提取和表示能力，可以自动捕捉图像的最关键信息，为解决坑洼道路检测问题提供了新途径。

在坑洼道路检测和识别任务中，研究者通常将道路图像分为两类：正常和坑洼。使用深度学习技术可以有效捕捉坑洼图像中轮廓、纹理和形态等重要特征，并将它们转化为更适合分类的表征，从而显著提升坑洼图像的识别性能。

1.2 问题描述

问题一: 为完成道路避障、道路修复等工作，需要更精准的识别坑洼的边缘、面积等特性。请建立模型提取道路中坑洼的边缘。

问题二：根据识别的坑洼边缘，估计坑洼的面积，并将附件 3：test_data.zip 中的图片计算每个图片中坑洼面积占整个图像面积的比例填写到 test_result2.csv 中，如果是正常道路占比填写 0，坑洼道路填写百分比整数部分。

二、 问题分析

对于问题一，需要提取图像中的坑洼边缘，传统的边缘检测算法如 Canny 算子、Sobel 算子等需要进行手工设计，且鲁棒性较差，难以准确地从整张图像中检测出坑洼边缘特征。因此，可以考虑使用抗干扰性强且特征提取能力强大的深度学习方法来检测坑洼边缘。然而单纯的目标检测方法只能在图像中检测到坑洼位置，而无法检测到具体的坑洼边缘，因此，应该对坑洼道路数据执行图像分割任务，从而精确地检测坑洼边缘。而基于深度学习的语义分割可以使用标注了语义信息的坑洼道路图像进行驱动训练，进行过训练的模型对图像中的坑洼边缘特征有较为精确的提取能力。

对于问题二，语义分割模型将会对输入图像中的每个像素点进行坑洼图案与正常背景进行二分类，每一类会标注不同的颜色。要计算整张图象中坑洼图案的面积占比，只需统计图像 RGB 通道中相应被标注为坑洼图案的像素个数，其与整张图像像素总数之比即为坑洼图案占比。

三、 模型假设

- 假设人工进行的像素级标注均为坑洼图像真实的语义情况。
- 本文已经固定模型参数随机种子, 对比实验时不存在模型初始化参数不一致的情况。

四、 符号说明

符号	符号解释
$\mathcal{D}_{train}$	训练数据集
$\mathcal{D}_{test}$	测试数据集
f_ϕ	语义分割网络参数
f_C	初赛问题中的分类模型
η	学习率
$\mathbf{x}$	图像
$\mathbf{x}_{seg}$	分割后图像
$S_{pothole}$	坑洼面积占比

五、 模型建立与求解

5.1 问题一：建立坑洼道路语义分割模型

在问题一中，为训练语义分割模型，本文收集公开数据并进行语义标注，构造 DeepLab v3+ 语义分割模型来对坑洼道路图像提取坑洼边缘特征。同时，为了验证本文构造模型的良好性能，进行了充分的对比实验。具体建模流程如图 1 所示。

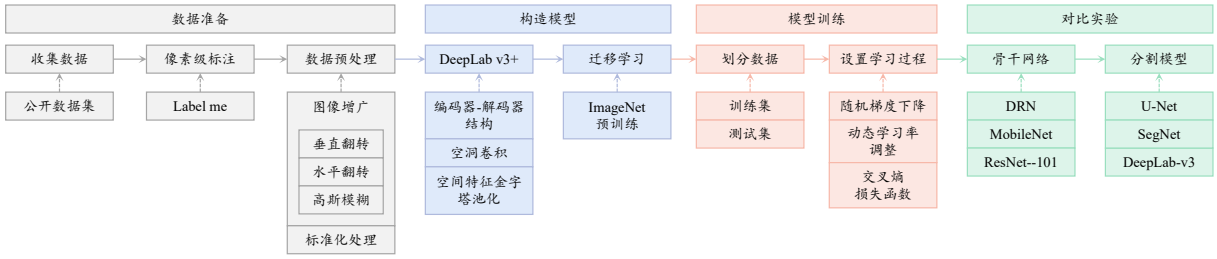

图 1　问题一建模流程

5.1.1 传统边缘检测算法与目标检测算法的缺陷

传统边缘检测算法 边缘检测是计算机视觉和图像处理中的基本任务之一，其目标是识别图像中物体的边界。边缘通常表示图像中灰度级别或颜色发生显著变化的地方，这些变化可能是物体之间的界限或图像中的重要特征。

题目要求准确地从图像中提取出坑洼边缘。然而，传统的基于模板的边缘检测算法（如：Sobel 算子、Canny 算子和 Laplacian 算子等）虽然能够从图像中提取出明显的边缘特征，但这些算子并不能精准地识别出那些是坑洼边缘，哪些是其他物体的边缘，只能笼统地提取出图像中的所有边缘特征，其检测效果如图 2 所示。

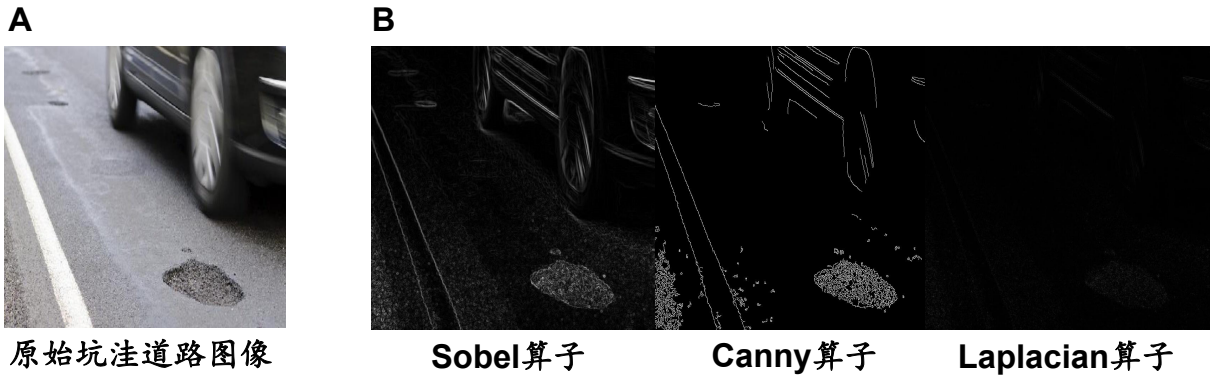

图 2 使用传统边缘检测算法进行坑洼道路图像边缘检测。(A) 原始坑洼道路图像。(B) 传统边缘检测算法提取出的图像边缘，从左到右的提取算法依次为 Canny 算子、Canny 算子、Laplacian 算子。

从图 2 中可以看出，Laplacian 算子提取的边缘非常模糊，Sobel 与 Canny 算子提取的边缘虽然更加清晰，但无法辨别哪些边缘属于坑洼，无法准确地刻画坑洼边缘特征。且当坑洼内部本身纹理特征较为复杂时，传统地边缘检测算法会将坑洼内部地纹理特征看作是边缘特征一同提取，无法做到仅提取边缘特征。

为进一步更加详实地检验传统边缘检测算法对坑洼道路图像中坑洼边缘的提取效果，本文从踢给测试数据中随机抽取 16 张图像，分别使用边缘检测的常用算子 Sobel 算子、Canny 算子、Laplacian 算子进行边缘检测，实验结果如图 3 所示。

从图 3 中可以看出，边缘检测算法只能从图像中提取不同物体间的边缘，无法精确地从众多复杂的边缘细节中提取出具体的坑洼边缘。故对于路况复杂、情况多种多样的坑洼道路图像来说，传统边缘检测算法无法准确地提取出坑洼边缘。

目标检测算法的缺陷 目标检测旨在识别图像中的特定物体并确定其位置，在坑洼道路图像中的具体应用效果如图 4 所示。此处使用开源的坑洼道路检测 YOLO 模型进行目标检测。

使用目标检测算法虽然能够在图像中确定坑洼具体位置，但只能在原始图像的基础上标注出矩形的回归框，与实际的坑洼边缘存在较大误差。故目标检测算法虽然能在图

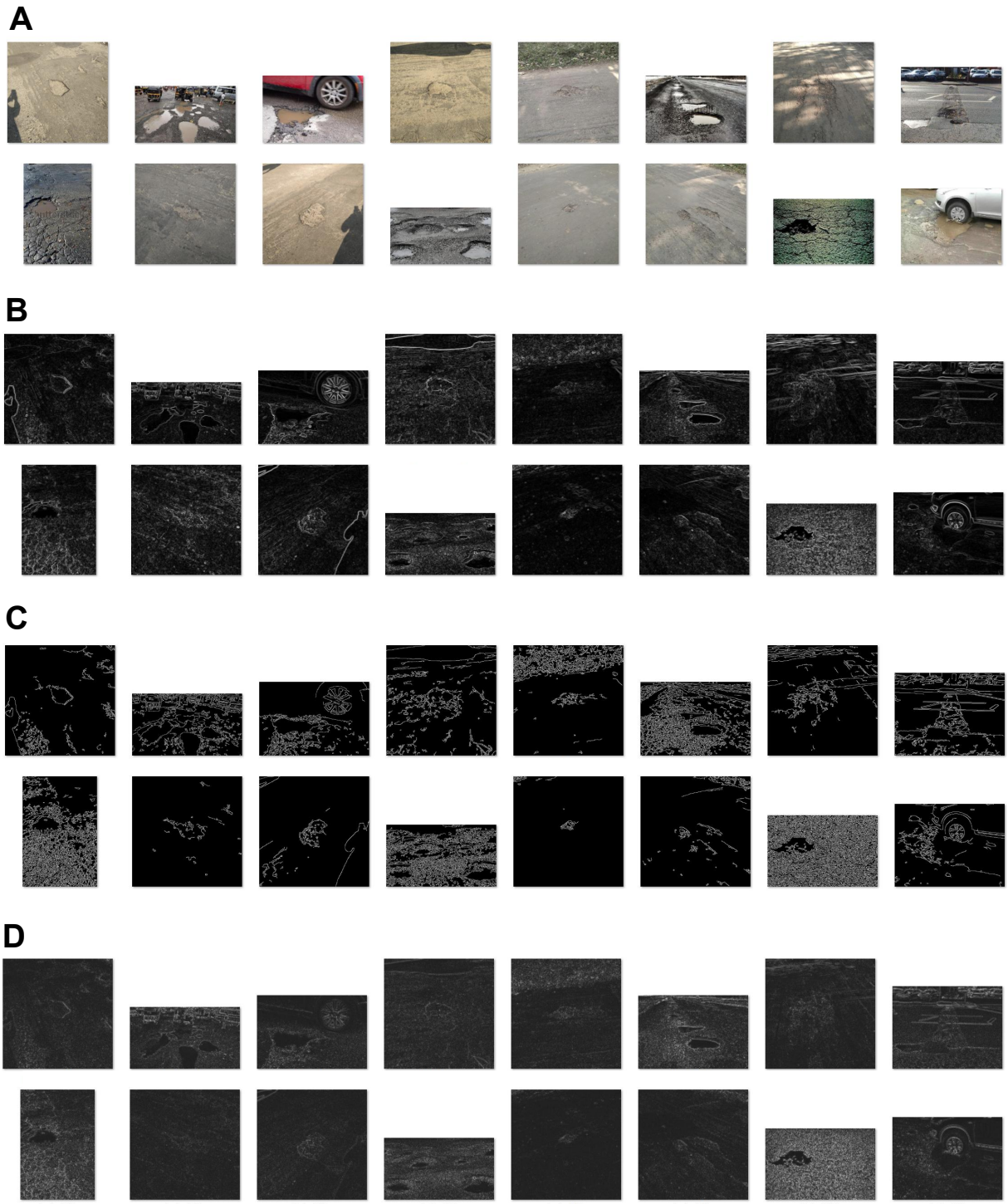

图 3　传统边缘检测算法实验效果。（A）原始坑洼图像。（B）Sobel 算子。（C）Canny 算子。（D）Laplacian 算子。

像中确定坑洼道路位置，大致估算坑洼面积，但面对非规则的复杂坑洼图像时无法准确提取坑洼边缘，只能使用矩形的回归框对坑洼位置进行拟合，计算不够精确。

并且坑洼道路图像场景复杂多样，有许特征空间较为复杂地难样本，如模糊、坑洼数量多、裂痕狭小等，为目标检测任务带来极大困扰。目标检测算法在难样本上的效果如图 5 所示。

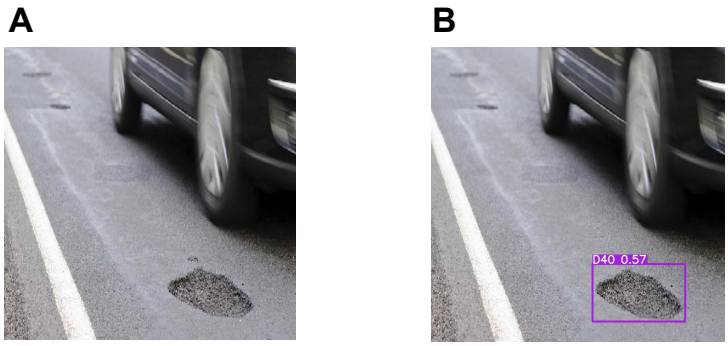

图 4　坑洼道路图像中的目标检测示意。（A）原始坑洼道路图像。（B）对图像进行目标检测，检测坑洼位置。

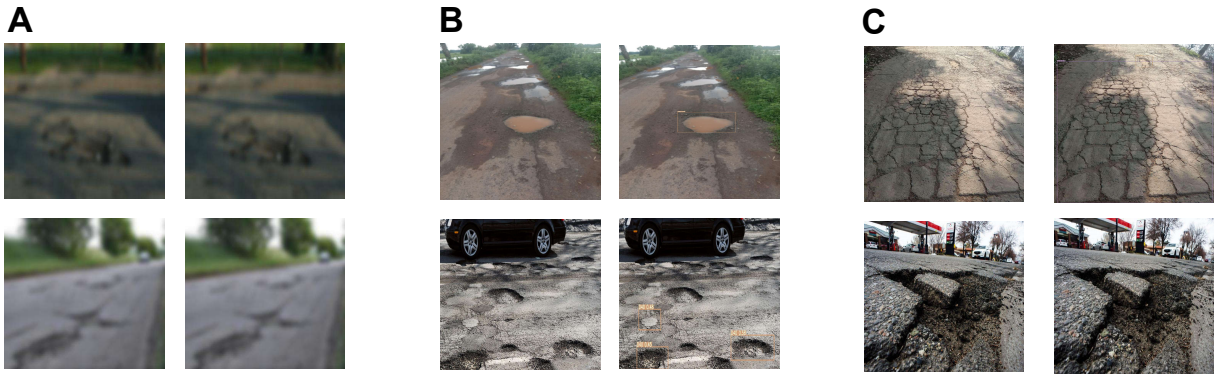

图 5　目标检测算法在坑洼道路图像难样本中的检测效果

可以明显看出，目标检测算法对模糊坑洼图像失去识别能力，对多坑图像仅能检测出图像中少量坑洼，无法识别出图像中所有坑洼，对于坑洼纹理特征较复杂的特征，输出整个回归框的结果误差太大，且对较细小的裂缝型坑洼识别能力有所下降。

5.1.2 坑洼道路图像语义分割

传统边缘检测算法能识别不规则物体边缘，但无法定位到图像中的坑洼，目标检测算法只能使用回归框标定的矩形区域，计算误差过大，无法精确识别纹理复杂、不规则的坑洼边缘。因此，为了准确的识别到路面坑洼区域并提取出坑洼边缘，可使用计算机视觉领域的语义分割算法对坑洼道路图像进行处理。

语义分割的目标是将图像中的每个像素分配到对应的语义类别，从而达到对图像进行像素级别的理解和分类。与目标检测任务不同，语义分割关注的是每个像素的语义，而不仅仅是物体的边界和位置，其基本工作流程框架如式 (1) 所示。

$$\mathbf{X}_{region} \longrightarrow M_s \Rightarrow \mathbf{X}_{mark} \tag{1}$$

其中 $\mathbf{X}_{region}$ 为语义分割模型的输入，表示原始图像；M_s 表示语义分割模型；$\mathbf{X}_{mark}$

为语义分割模型的输出，表示进行过像素级标记的图像。因此，与目标检测相比，对坑洼道路图像进行语义分割，不仅能识别出图像当中坑洼的位置，而且能更加精确的在图像中刻画坑洼的不规则边缘。

对坑洼道路进行语义分割的效果示意如图 6 所示。

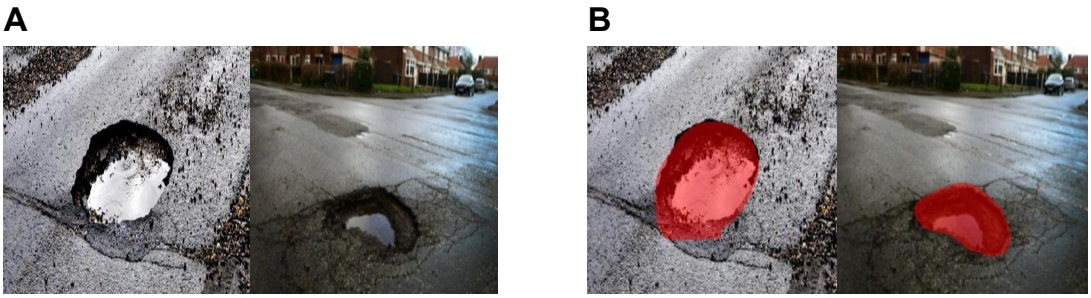

图 6　坑洼道路语义分割示意图。（A）原始坑洼道路图像。（B）对坑洼道路图像进行语义分割

通过观察图 6，可以明显看出语义分割技术能够精准的检测图像中不规则坑洼的形状边缘。与仅能输出矩形框的目标检测算法相比，语义分割在边缘检测地准确率上实现了显著的提升。

语义分割之所以能够从坑洼道路图像中准确地分割出完整的坑洼图案，得益于语义分割对图像执行像素级的分类，具体过程如算法 1所示。

Algorithm 1: 针对坑洼道路图像的语义分割

Data: 原始坑洼道路图像数据集 $\mathcal{D}(X)$

Result: 语义分割后的坑洼道路图像数据集 $\mathcal{D}'(X)$

// 从数据集选取训练样本;
for $i = 1$ **to** $N(\mathcal{D})$ **do**
    // $\mathbf{x}_i$ 代表 $\mathcal{D}(X)$ 中的第 i 张图像;
    for p_{xy} **to** $\mathbf{x}_i$ **do**
        对 p_{xy} 进行坑洼区域与正常区域的二分类;
        if $p_{xy}\in$ 坑洼区域 **then**
            对 p_{xy} 进行特殊颜色标记;
            p_{xy}^{mark} 替换 $\mathbf{x}_i$ 中的 p_{xy};
    $\mathbf{x}_i \Rightarrow \mathbf{x}_i^{mark}$;
    $\mathbf{x}_i^{mark} \longrightarrow \mathcal{D}'(X)$;
return 语义分割后的坑洼道路图像数据集 $\mathcal{D}'(X)$

语义分割中特征提取网络的主要结构 语义分割可看作是对图像中各像素点的分类任务，因此，语义分割模型也是建立在分类模型基础上的，主要利用卷积神经网络来提取特征并进行分类。语义分割任务中用于特征提取语义信息的主要结构为卷积层、池化层、激活函数，如图 7 所示。

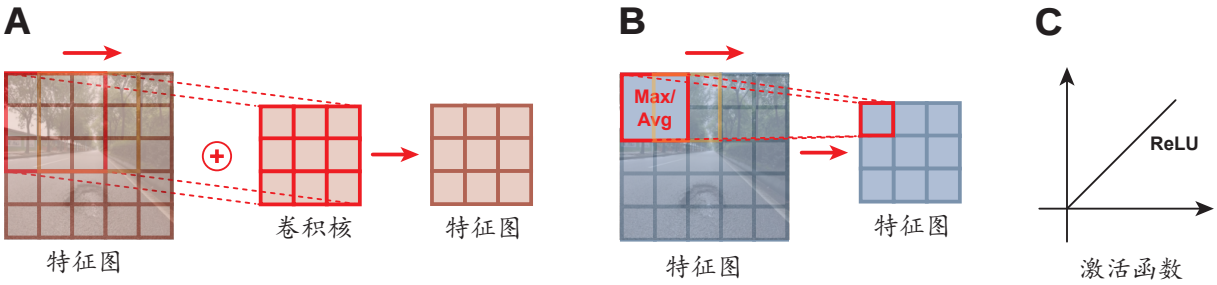

图 7 卷积神经网络工作示意图。（A）卷积层工作原理；（B）池化层工作原理；（C）激活函数工作原理。

- **卷积层：** 卷积神经网络的卷积层可以接收三维数据的输入，包括宽度、高度和 RGB 通道等信息，并且同样以的三维数据进行输出，有助于保留图像的形状特征和局部结构，能更好地提取图像的纹理特征。卷积层的工作原理如图 7A 所示，使用卷积操作来处理输入图像，如式 (2) 所示。其中 $\mathbf{W}$ 与 $\mathbf{X}$ 分别表示卷积核与完整输入图像，$*$ 表示卷积运算。

$$\mathbf{C} = \mathbf{W} * \mathbf{X} \tag{2}$$

具体过程如下，首先一个被称为卷积核的滤波器以一定的步长在输入图像上进行滑动，然后在每个像素点上，卷积核与输入图像的对应卷积核与输入图像的对应区域执行元素相乘并求和，产生一个单个的输出值。当卷积核滑动完整张输入图像后，得到的特征值依然是一个矩阵，这个输出矩阵便是卷积层提取出的特征图。具体的卷积操作如式 3所示。

$$\mathbf{C}_{ij} = \sum_{u=1}^{U}\sum_{v=1}^{V}\mathbf{W}_{uv}\mathbf{X}_{i-u+1,j-v+1} \tag{3}$$

其中图像 $\mathbf{X} \in \mathbb{R}^{M\times N}$，卷积核 $\mathbf{W} \in \mathbb{R}^{U\times V}$，$\mathbf{W}_{uv}$ 表示卷积核矩阵中的数值，$\mathbf{X}_{i-u+1,j-v+1}$ 表示与卷积核进行卷积运算的原图像素点，$\mathbf{C}_{ij}$ 表示经过卷积操作后得到的特征图中的像素点。

- **池化层：** 池化层的作用是从输入的特征图中进行特征选择，去除特征图中的冗余信息，简化与特征图参数有关运算，实现特征降维，防止出现过拟合现象。如图 7B 所示，对于一张图像 $\mathbf{X} \in \mathbb{R}^{M\times N}$，可以分成若干个大小相等的小块 $\mathbf{C}^i \in \mathbb{R}^{m\times n}, m < M, n < N$。然后在该区域中进行池化操作，选择出一个特征值来代表整个区域。对于图像 $\mathbf{X}$ 中的第 i 个小块区域 $\mathbf{C}^i_{m,n}$ 而言，常见的池化方法有最大池化与平均池

化。最大池化如式 (4) 所示，即选取区域内的最大值来代表整个区域。

$$\mathbf{Y}^i_{m,n} = \max_{j \in \mathbf{C}^i_{m,n}} x_j \tag{4}$$

平均池化如式 (5) 所示，即计算 $\mathbf{C}^i_{m,n}$ 内的平均特征值来代表整个区域。

$$\mathbf{Y}^i_{m,n} = \frac{1}{m \times n} \sum_{j \in \mathbf{C}^i_{m,n}} x_j \tag{5}$$

- **激活函数：**激活函数是卷积神经网络中的非线性变换，能让网络拥有捕捉更复杂特征的能力。激活函数同时能够通过抑制一部分神经元的活性来缓解网络训练中出现的梯度消失问题，也能学习到更加稳定的特征，提高网络的泛化能力。图 7C 所示的是卷积神经网络中最常用的激活函数 ReLU，其表达式如式 (6)所示。

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{6}$$

  相比其他复杂的激活函数，ReLU 函数只需要进行简单的比较与赋值操作，计算上更加高效，且具有良好的稀疏性质。

语义分割网络的特性　普通分类任务是基于图像全局信息的处理，只需最终预测一个全局概率，因此特征图维度降低，但是语义信息更高级。而语义分割需要的是精确到像素点级的处理，过小的特征图会损失很多信息。两种任务要求的特征向量处理如图 8 所示。

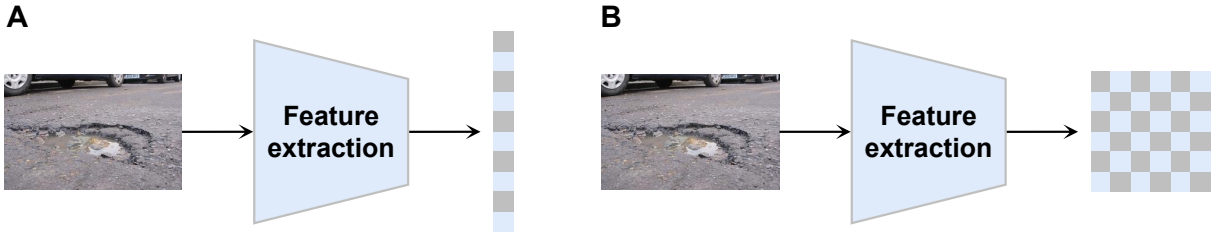

图 8　分类任务与语义分割任务要求的特征向量形式不同。（A）分类任务需要输出维度小语义信息高维的特征向量。（B）语义分割任务需要精准到像素级的语义信息，特征向量尺寸尽可能大。

如果在语义分割任务中对图像进行与分类任务相似的降维处理，将导致严重的语义信息损失，大大降低图像中语义分割的精度。但是如果一直让网络总的特征图保持原始大小，将会产生非常大的计算量，无法训练深层次的网络。而降维后的特征图虽然损失了语义信息，但同时也降低了计算量，使得深度网络的训练更加可行。因此，对于语义分割任务来说，平衡这两点尤为关键。

用于处理语义分割的全卷积神经网络（Fully Convolution Network, FCN）[3] 将图像降维成 32×32 的特征图，虽然使特征图含有丰富的语义但是空间信息损失严重导致分

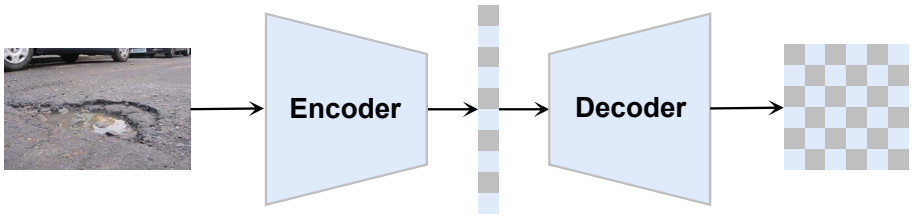

图 9　语义分割中的编码器-解码器结构

割不准确。语义分割的另一种解决方案是采用编码器-解码器（Encoder-Decoder）结构，如图 9 所示。

其中编码器是下采样模块，负责从原始图像中提取高维语义信息，输出一个形状尺寸较小的特征向量。解码器复杂将较小的特征向量恢复成原始图像大小，避免特征降维所引起的语义信息损失。

本文使用基于编码器-解码器结构的 DeepLab v3+ 模型对坑洼道路图像进行语义分割。

5.1.3 获取训练数据

获取原始坑洼道路图像　题目要求对 4942 张坑洼图像进行边缘检测，然而对于语义分割任务而言，要训练图像分割模型，需要对应原始图像的语义标记图像，如图 10 所示。

图 10　坑洼道路语义分割中的原始图像与语义标签

而题中仅给出 4942 张测试数据，且缺少语义标签，为使模型训练结果公平公正，体现模型在测试数据上的泛化能力，本文将提给 4942 张图像全部作为测试数据 $\mathcal{D}_{test}$，完全不接触训练过程。本文从来自阿里云天池实验室的坑洼道路公开数据集“Pothole Detection 坑洼检测”中选取样本作为训练集 $\mathcal{D}_{train}$。

阿里云天池实验室“Pothole Detection 坑洼检测”是用于坑洼道路目标检测任务的公开数据集，共包含 665 张坑洼道路图像，并且每张图像带有 PASCAL VOC 格式的边

界框注释标签。然而对于语义分割任务而言，需要带有像素级语义信息的标签才能驱动模型训练，故本文从公开数据集中选取 400 张坑洼道路图像，并自行进行数据标注。选取的数据集中部分图像如图 11 所示。

图 11　坑洼道路公开数据集中的部分图像示例

标注数据　为了对原始坑洼道路图像构造语义信息标签，本文使用 Labelme 工具对选取的 400 张坑洼道路图像进行语义信息标注。Labelme 是开源的图像标注工具，常用做计算机视觉领域的目标检测、语义分割等任务的图像标注。使用 Labelme 为坑洼道路图像进行的语义标注如图 12 所示。

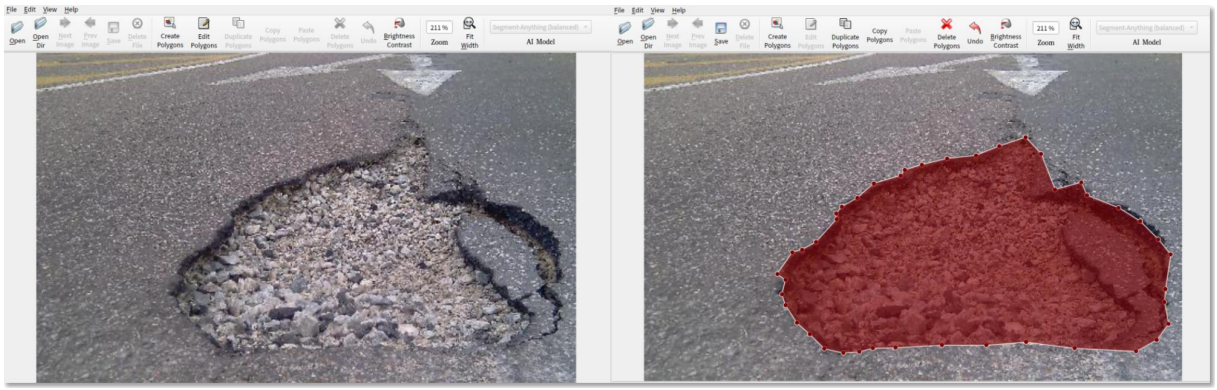

图 12　Labelme 语义标注示意

5.1.4 数据预处理

在坑洼道路图像语义分割任务中，适当地对坑洼道路图像数据进行预处理操作有助于提高模型的性能和鲁棒性，增强语义分割模型对复杂路面场景和变化的适应能力。在本文中，由于自行标注的数据集样本总数较少，故使用水平翻转、垂直翻转、高斯模糊等数据增广手段来扩充训练样本数量，并对所有训练样本进行标准化处理以提高提高模型稳定性。

数据增广　本文使用水平翻转、垂直翻转、高斯模糊三种数据增广方法来扩充原有数据样本，原有数据样本总数为 400 张，每种数据增广方法在所有原始样本进行操作处理，故每种增广手段扩充 400 张图像，增广后数据总数为 1600 张图像。三种数据增广方法效果如图 13 所示。

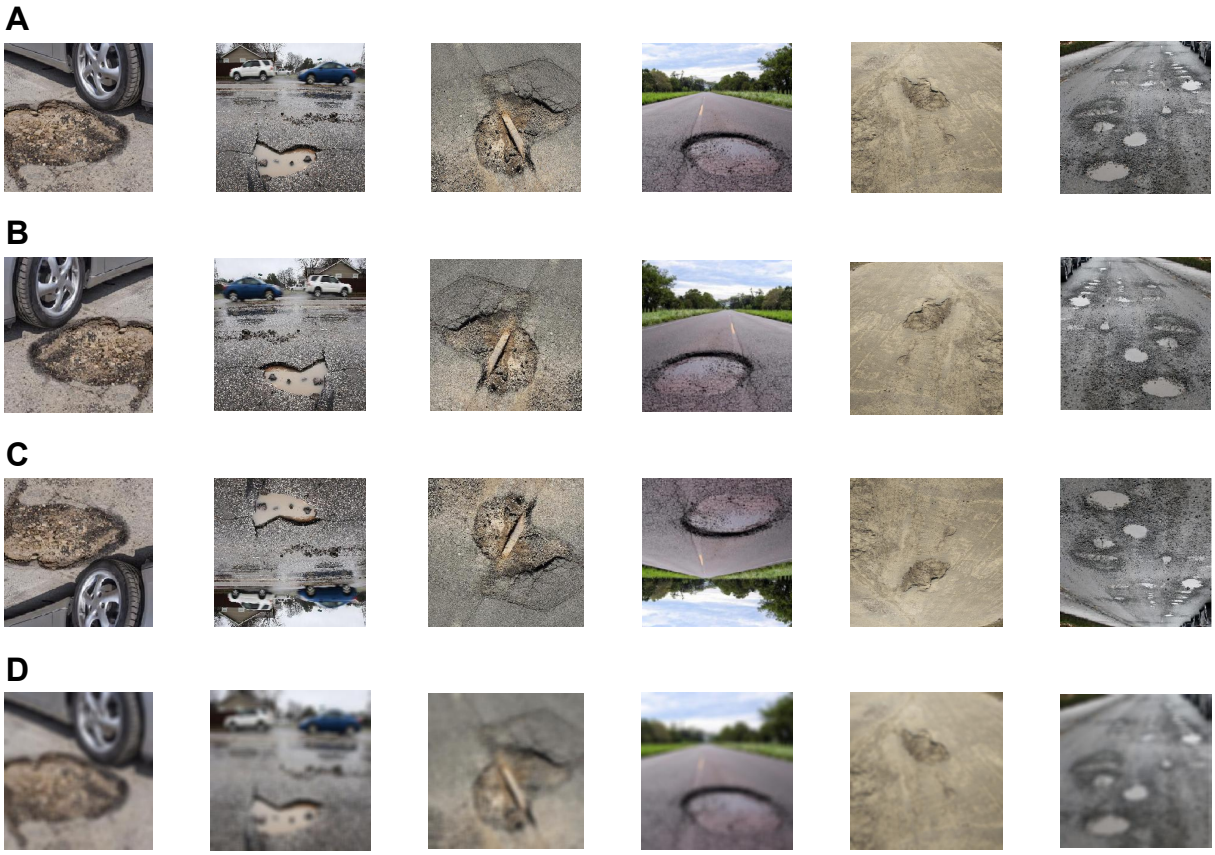

图 13　坑洼道路图像数据增广。（A）原始图像。（B）水平翻转。（C）垂直翻转。（D）高斯模糊。

对图像进行翻转可以增加数据集中的空间特征复杂性，模型在学习过程中将会见到水平和垂直方向上的不同特征，使模型对目标物体在图像中的位置变化不敏感，让模型更能适应实际场景中物体的不同姿态和方向变化，进一步帮助模型在不同的空间位置中准确地分割出坑洼图案。

考虑到坑洼道路图像获取时的成像条件不同，模型需要面临不同清晰度的坑洼道路图像，因此，本文使用高斯模糊来处理图像，增广清晰度较低的图像，以强化模型对清晰度不理想坑洼道路的分割能力。

高斯模糊通过应用高斯函数对图像进行卷积，使图像中像素值的梯度减小来使图像变模糊。其基本原理是在图像的每个像素上应用一个高斯核，将该像素及其周围的的像素按照高斯分布的权重进行加权平均。高斯核的尺寸和标准差决定了模糊的程度，尺寸越大、标准差越小，模糊效果越显著。在离散形式下，高斯模糊的具体数学表达如式 (7) 所示。

$$G(i,j) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right) \tag{7}$$

设 K 是一个尺寸为 $(2k+1) \times (2k+1)$ 的离散高斯核，k 是一个非负整数，那么 i 和 j 分别表示 K 的行和列索引，σ 是高斯分布的标准差。那么对原始图像 $\mathbf{x}$ 的高斯模糊卷积操作如式 (8) 所示。

$$\mathbf{x}_{\text{blurred}}(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} \mathbf{x}(x+i, y+j) \cdot G(i,j) \tag{8}$$

其中 $\mathbf{x}_{\text{blurred}}$ 为高斯模糊后的图像。

标准化处理 标准化是深度学习中常用的预处理技术，将输入数据归一到一个较小的范围，可以减少梯度消失或梯度爆炸等问题，有助于模型更好地收敛，更容易找到全局最优解。

要对坑洼道路数据集进行标准化处理，首先计算数据集整体的均值 $mean$ 和方差 std。RGB 图像中的每个通道 c，$mean$ 的计算如式 (9) 所示。其中 H 和 W 分别是图像的宽和高，$\mathbf{x}_c(i,j)$ 是图像在位置 (i,j) 处通道 c 的像素值。

$$mean_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{x}_c(i,j) \tag{9}$$

对每个通道 c，标准差 std 的计算如式 (10) 所示。

$$std_c = \sqrt{\frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(\mathbf{x}_c(i,j) - mean_c\right)^2} \tag{10}$$

上述计算在整个数据集中的图像进行，对数据集中所有图像的所有通道进行累加求均值和标准差，求得数据集中的均值和标准差具体数值后，对数据集中所有数据进行如式 (11) 所示的标准化处理。

$$\mathbf{x}_{\text{Normalized}}(i,j) = \frac{\mathbf{x}(i,j) - mean}{std} \tag{11}$$

其中 $\mathbf{x}_{\text{Normalized}}$ 为标准化处理后的图像，$\mathbf{x}_{\text{Normalized}}(i,j)$ 为 $\mathbf{x}_{\text{Normalized}}$ 中位于 (i,j) 位置的像素值。

5.1.5 DeepLab v3+

建模动机 大多数用于语义分割的深度神经网络面临着两大问题。一是物体多尺度问题，语义分割任务中数据图像内的物体通常具有不同的尺度，而模型需要能够有效地识

别和分割这些不同尺度地物体。其次，深度神经网络的多次下采样会造成特征图分辨率变小，语义信息损失，从而导致预测精度降低，边界信息丢失。

DeepLab v3+ 通过引入空洞卷积、ASPP 模块等解决了上述问题，同时进一步优化了大尺度图像的计算量，使得语义分割任务速率高且分割精准。

网路结构 为了解决网络计算耗时过长、语义信息不够充分等问题，DeepLab v3+ 引入编码器-解码器结构。解码器进行特征提取，编码器通过上采样操作将特征图还原为低层次特征的分辨率大小。其具体网络结构如图 14 所示。

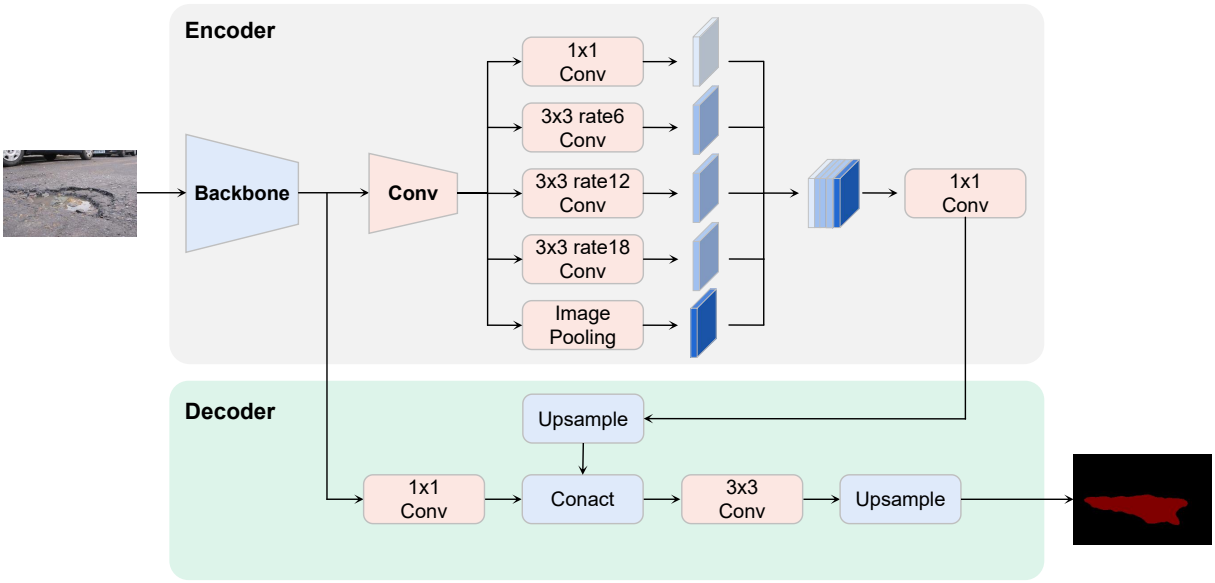

图 14 对坑洼道路执行语义分割任务的 DeepLab v3+ 总体结构。

在图 14 中，Encoder 表示提取图像特征的编码器结构，Decoder 表示还原特征图大小的解码器结构，Backbone 表示 DeepLab v3+ 中的特征提取骨干网络。

首先, 将图像输入到带有空洞卷积和深度可分离卷积的骨干网络模型中提取低层次特征；然后，将图像的低层次特征输入到空间金字塔池化模块（Atrous Spatial Pyramid Pooling, ASPP）中进一步提取更深层次的多尺度特征，并使用尺寸为 1×1 的卷积层将多尺度特征融合成高层次特征；最后，对高层次特征进行双线性插值的上采样，并将其与低层次特征进行融合，通过卷积层与上采样操作输出最后的语义分割图像。

空洞卷积 空洞卷积是 DeepLab v3+ 网络中的重要组成部分。我们在进行语义分割时需要在扩大感受野的同时尽可能减少语义信息的丢失，使用空洞卷积能够在扩大卷积层中感受野的同时保持分辨率维持稳定。

在空洞卷积中，r 控制着感受野的大小，在卷积核尺寸不变的情况下，r 越大则感受野越大，具体控制效果如图 15 所示。

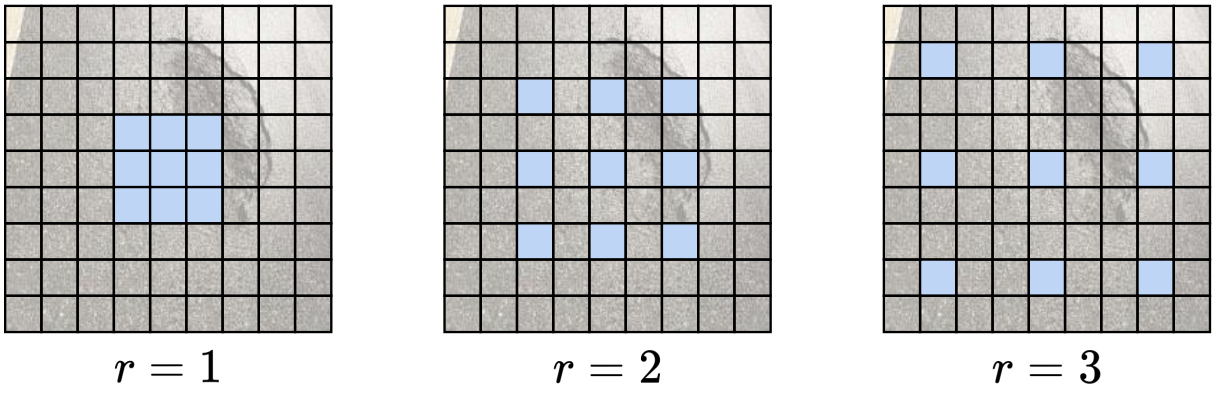

图 15　空洞卷积示意图。从左到右为 r 值逐渐增大的感受野区域。

空洞卷积对特征图位置 i 的感受野运算如式 (12)所示。

$$\mathbf{Y}[i] = \sum_{k} \mathbf{x}[i + r * k] \times \mathbf{W}[k] \tag{12}$$

其中 **X** 为输入的原始图像，**Y** 为空洞卷积层卷积运算后得到的特征图，i 表示特征图上的像素位置，**W** 表示卷积核，r 为膨胀卷积中的膨胀系数。空洞卷积操作相当于将输入 X 在每个空间维度上，与两个连续的卷积核之间插入 $r-1$ 个零值而产生的上采样卷积核进行卷积。

空间金字塔池化　当骨干模型网络提取低层次特征之后，DeepLab v3+ 使用空洞卷积构成的空间金字塔池化模块进一步提取多尺度特征。DeepLab v3+ 中的空间金字塔详细结构如图 16 所示。

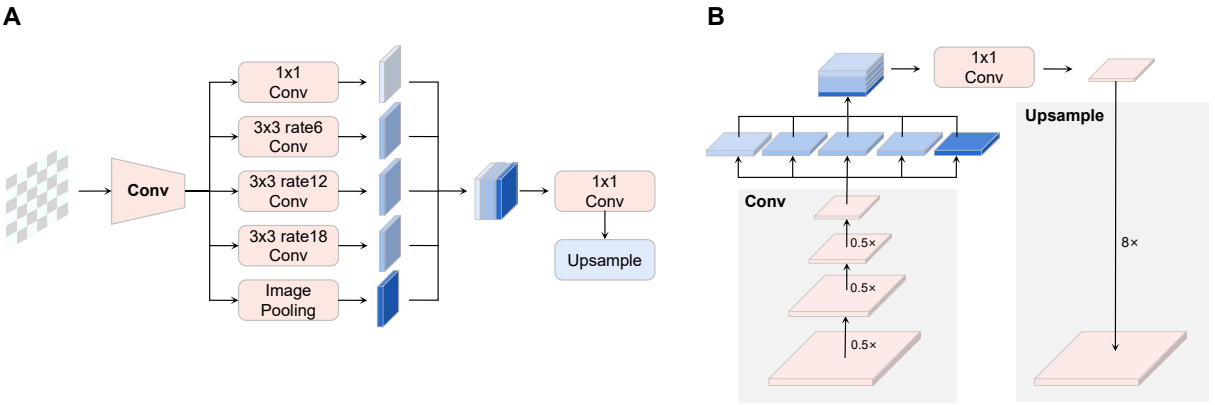

图 16　DeepLab v3+ 中的空间金字塔池化。(A) 空间金字塔网络结构。(B) 特征图在空间金字塔中的变化流程与详细网络模块。

特征图输入到空间金字塔中，首先经过三个尺寸为 3×3 且 r 不一样的空洞卷积层核一个尺寸为 1×1 的卷积层，提取出特征图中的不同尺度特征，再使用一个全局平均池化层对原特征图进行处理，将处理后的特征图与之前的四张不同尺度特征图堆叠在一

起形成多尺度特征，最后将多尺度特征输入到 1×1 中的卷积层得到新的融合特征，最后对融合特征进行八倍的上采样，得到与原始特征图大小相同的特征图。

在坑洼道路的边缘检测任务中，受到拍摄角度、图像尺寸大小等因素的影响，图像中的坑洼图案也呈现出不一样的大小和尺寸，因此，在语义分割网络中加入空间金字塔模块有助于挖掘不同图像当中具有不同尺度特征的坑洼边缘特征。

解码器结构 解码器结构如图 14 下方所示。首先，上采样层对编码器输出的多尺度融合特征进行四倍的双线性插值上采样。双线性插值是一种用于在二维空间中找到非整数坐标上的像素插值方法，通常用于图像放大，以及改善图像质量使图像在变换过程中变得更加平滑。

双线性插值方法具体过程如下：考虑二维空间上的四个相邻点 Q_{11}，Q_{12}，Q_{21}，Q_{22}，和目标点 (x, y)，首先计算在 x 轴方向上的两个插值，分别位于 Q_{11} 和 Q_{21} 之间以及 Q_{12} 和 Q_{22} 之间，插值可通过如式 (13) 所示的线性插公式得到。

$$\begin{cases} R_1 &= Q_{11} + (x_2 - x) \cdot \frac{Q_{21} - Q_{11}}{x_2 - x_1} \\ R_2 &= Q_{12} + (x_2 - x) \cdot \frac{Q_{22} - Q_{12}}{x_2 - x_1} \end{cases} \tag{13}$$

再使用水平插值得到的 R_1 和 R_2 进行垂直插值，如式 (14) 所示。

$$P = R_1 + (y_2 - y) \cdot \frac{R_2 - R_1}{y_2 - y_1} \tag{14}$$

P 即为目标点 (x, y) 处的插值结果。

多尺度融合特征经过插值放大后，与来自骨干特征提取网络的低层次特征进行连接，低层次特征进行融合之前还会经过一个卷积核尺寸为 1×1 的卷积层以减少通道数量。因为低层次特征包含大量的特征通道，容易使训练变得非常困难。特征融合后，使用一个 3×3 的卷积来对融合后的特征进一步细化，并再次进行四倍的双线性插值上采样，使语义分割后的图像大小与原始输入图像大小一致。

特征提取骨干网络 本文选取特征提取任务中的经典网络残差神经网络 ResNet 作为 DeepLab v3+ 中的特征提取骨干网络模型。同时，为了挖掘图像中坑洼部分的深层特征，使用 ResNet 中层数较深的 ResNet-101 对坑洼图像执行特征提取任务。

残差神经网络（ResNet） [2] 创造性地引入了残差模块与跳跃连接，极大程度上改善了深层网络的梯度消失与梯度爆炸问题，让神经网络的层数能够被设计地越来越深，从而提取更加复杂的语义特征。残差块与跳跃连接的结构如图 17 所示。

残差块 ResNet 通过引入残差快结构，将网络层之间的输入信号与输出信号之间的差异作为网络的学习目标，而不是直接学习完整的输出信号，学习方式如式 (15)所示。

$$F(x) = H(x) - x \tag{15}$$

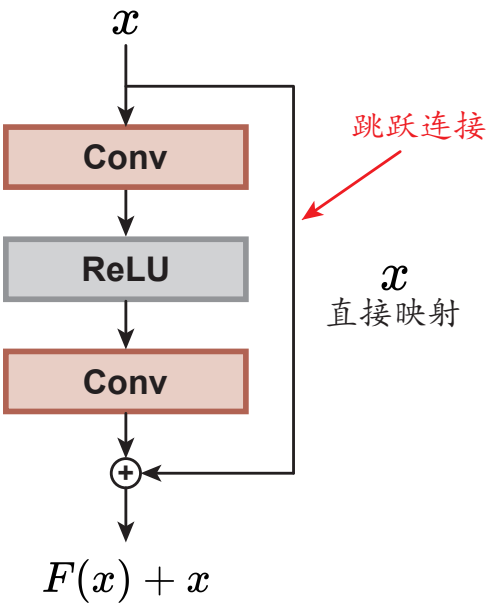

图 17　残差块与跳跃连接结构图

其中 x 为原始输入，$H(x)$ 为网络的输出映射，$F(x)$ 即为输入与输出映射之间的残差，是网络真正学习的内容。这使得网络更加容易训练，因为原始的输入信息被被保留了下来，网络只需要学习输入到输出的变化，而不是学习从输入到输出的完整映映射，很好地解决了梯度消失问题。

跳跃连接　ResNet 中的残差快还引入了跳跃连接，允许信号在模型的不同层之间传递，底层的信息可以直接传递到输出层，网络可以更轻松地学习恒等映射，而不会受到复杂的层次结构的阻碍。这有助于梯度的有效传播，降低了训练深度网络的难度。

5.1.6 迁移学习

本文为了更加准确地提取不规则坑洼边缘的深度特征，选取 ResNet 网络系列中较深的网络结构 ResNet-101 作为特征提取骨干网络。然而，训练一个深层次的网络模型需要大量的标签数据才能使其具有良好的泛化能力，否则网络极易陷入过拟合。在本文中，可用于训练的标签数据量较少，如果对 DeepLab v3+ 的所有网络参数进行随机初始化，然后使用较少的坑洼道路标签数据进行训练，那么模型容易对训练样本中的坑洼道路陷入过拟合，其语义分割能力无法繁泛化到特征分布更加复杂的测试样本上。

迁移学习（Transfer Learning）是一种机器学习技术，将在一个任务上学习的知识转移到不同但相关任务的模型上。先前被学习的数据成为源域，将要执行当任务的数据被称为目标域。迁移学习的目标是在源域上学习一个模型，这个模型被称为预训练模型，然后将此预训练模型应用在目标域上。

为了缓解训练数据过少带来的过拟合等问题，本文使用迁移学习将 ResNet-101 在

ImageNet 上的预训练权重加载到 DeepLab v3+ 内的特征提取骨干网络中，让网络参数从一开始就具备强大的特征提取能力。然后再使用坑洼道路的坑洼边缘标签数据微调 DeepLab v3+ 的参数，让网络具备对坑洼道路的语义分割能力。迁移学习过程如图 18 所示。

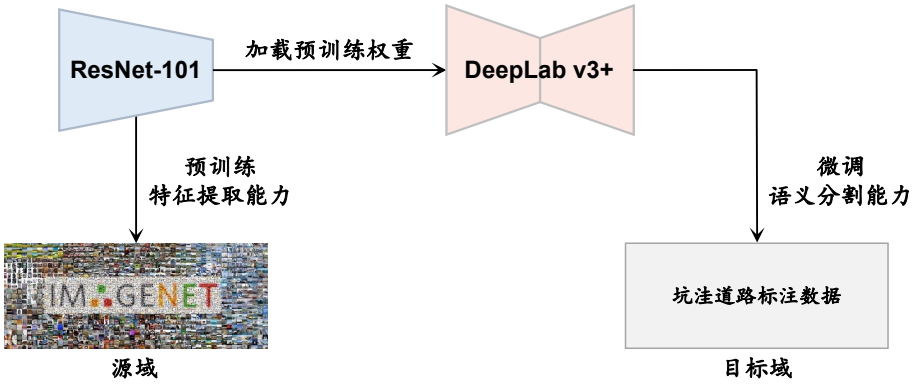

图 18　坑洼道路语义分割中的迁移学习

ImageNet 是一个大规模的图像数据库，是计算机视觉领域中的一个重要资源。它包含来自世界各地的数百万张图像，这些图像按照 WordNet 层次结构进行了标注，共涵盖数千个对象类别，使目前人工智能领域规模最大的数据集，许多经典的人工神经网络结构都有再 ImageNet 上的训练权重。

由于本文中目标域数据量较少，迁移学习可以利用源任务的大量数据，使得模型更容易泛化到目标任务。同时，预训练模型在大规模数据上进行了训练，具有更好的泛化能力。通过迁移学习，可以将这种泛化能力转移到目标任务上。通过使用坑洼道路语义标记数据进行微调，使得迁移学习后的 DeepLab v3+ 模型能更好地适应坑洼道路语义分割数据。

5.1.7 优化器设置

深度学习模型使用参数优化器来更新神经网络中的参数，使用学习率优化器来更新每一轮训练时的学习率大小。本文使用随机梯度下降（SGD）作为参数优化器，使用 PolyScheduler 作为学习率优化器。

随机梯度下降　梯度下降是优化模型参数最常用的方法，其参数更新如式 (16) 所示。其中 θ_t 表示 t 时刻的参数，$\mathcal{L}$ 表示损失函数值，η 表示学习率。

$$\theta_t = \theta_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial \theta_{t-1}} \tag{16}$$

在传统的梯度下降方法中，每一次迭代都要使用整个训练数据集来计算梯度并更新模型参数。而 SGD（Stochastic Gradient Descent）每次迭代只使用训练数据集中的一个随机子集来计算梯度和更新模型，使得算法的计算开销大幅度减小。

由于随机梯度下降每次只使用一小批样本，它的计算效率相对较高，并且在处理大规模数据集时具有一定的优势。然而，由于使用的样本是随机选择的，收敛过程可能会表现得更加嘈杂，但通常能够在相对短的时间内找到局部最优解。

PolyScheduler 传统固定调整学习率的优化算法通常会在训练过程的每个固定的时间步长调整学习率。这种静态学习率优化算法有一些潜在缺陷，特别是在处理复杂的、非均匀分布的数据集时。使用固定的学习率虽然在训练初期能使模型加快收敛速度，但在接近最优值时，学习率调整时的步长太大很可能导致跳过最优值，步长太小又可能使收敛速度变慢。

本文使用动态学习率优化算法 PolyScheduler 来对模型进行学习率更新，如式 (17) 所示。

$$\eta_{epoch} = \eta_{base} \times (\frac{epoch}{N_{epoch}})^{power} \tag{17}$$

其中，η_{epoch} 为当前训练伦次 $epoch$ 下的学习率大小，η_{base} 为初始学习率大小，$epoch$ 为当前训练轮次，N_{epoch} 为模型最大训练轮数，$power$ 为超参数，可用于控制学习率衰减程度。

动态更新学习率可以帮助算法更快地收敛到局部最优解。在训练初期，使用较大的学习率可以加速参数的更新，而在接近最优值时，逐渐减小学习率可以使模型更精确地调整参数，防止错过最优解。同时，动态学习率算法可以自适应地调整学习率，根据训练过程中的模型性能和数据分布的变化来灵活地调整学习率。这种适应性使得算法更具鲁棒性，可以更好地适应不同的数据和问题。

5.1.8 语义分割网络参数优化过程

DeepLab v3+ 等卷积神经网络使用误差反向传播算法来更新网络参数。在训练过程中，样本在网络中的输出值将与真实标签一起输入到损失函数中计算误差值，然后将误差值向前逐层传导，对每一层的权重求偏导计算梯度，让损失函数最小化。

语义分割本质为逐像素点的分类问题，坑洼道路语义分割可看作为逐像素点的坑洼道路与正常道路二分类问题，对此，我们采用如式 (18) 所示的二分类交叉熵函数作为 DeepLab v3+ 的损失函数。即对图像中每一个像素点计算分类损失函数。

$$\mathcal{L} = \frac{1}{N} \cdot \frac{1}{H \times W} \sum_{i}^{H \times W} \sum_{j}^{N} - \left[y_i^j \cdot \log\left(p_i^j\right) + \left(1 - y_i^j\right) \cdot \log\left(1 - p_i^j\right) \right] \tag{18}$$

其中 N 表示图像数量,H 与 W 分别为图像宽高，两者乘积即为像素点个数。y_i^j 表示样本 y^j 中像素点 i 的标签，p_i^j 表示像素点被预测为正常道路的概率。故 DeepLab v3+ 的训练流程如算法 2 所示。

Algorithm 2: 针对坑洼道路语义分割的 DeepLab v3 + 训练

Data: 训练样本与标注数据 $\mathcal{D}(X, X_{label})$；参数优化器 Optimizer；学习率优化器 Scheduler

Result: 训练后的模型参数 f'_ϕ

初始化网络模型参数 f_ϕ，设置 $\eta, N_{epoch}, N_{batch}$;

for $n = 1$ **to** N_{epoch} **do**

  随机打乱训练数据 $\mathcal{D}$;

  for 一个批次训练样本$(\mathbf{x}, \mathbf{x}_{label})$ **to** $\mathcal{D}$ **do**

    梯度清零：nograd();

    前向传播：$f_\phi(\mathbf{x}) = \hat{\mathbf{x}}$;

    计算损失函数：$\mathcal{L}$ = 损失函数$(\hat{\mathbf{x}}, \hat{\mathbf{x}}_{label})$;

    反向传播：计算损失函数关于模型参数的梯度;

    // 更新每一层模型参数;

    $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$;

    $b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(l)}}$;

  更新学习率：Scheduler.step();

return 训练后的模型参数 f'_ϕ

5.1.9 深度学习语义分割实验效果

实验设置 本文将增广后的 1600 张图像随机划分为训练集 $\mathcal{D}_{train}$ 与测试集 $\mathcal{D}_{test}$，为保证实验结果公正可靠，$\mathcal{D}_{train} \cap \mathcal{D}_{test} = \varnothing$ 其中 $\mathcal{D}_{train}$ 包含 1400 张图像，$\mathcal{D}_{test}$ 包含 200 张图像。对于参数优化器 SGD，设置基学习率为 0.07，用于加速 SGD 沿相关方向的收敛并减弱震荡的动量设置为 0.9，权重衰减的正则化项设置为 0.0005，并启用 Nesterov 动量，帮助模型更快收敛。所有实验在单张 Tesla A100 显卡上运行。

评价指标 本文选取像素准确率（Pixel Accuracy, PA），类别像素准确率（Class Pixel Accuracy, CPA），平均交并比（Mean Interesection over Union, MIoU），频权交并比（Frequency Weighted Intersection over Union, FWIoU）作为度量语义分割性能地指标 [4]。

坑洼道路语义分割问题即为像素水平的二分类问题，将图像中的每一个像素点分为坑洼与正常两个类别。在二分类问题中有四个常用指标：真正例 TP（True Postitive）、假正例 FP（False Positives）、假反例 FN（False Negatives）、真反例 TN（True Negatives），指标含义如下所示。

- **TP：** 表示模型将正类别样本正确地分类为正类别的数量。
- **FP：** 表示模型将负类别样本错误地分类为正类别的数量。

- **FN：**表示模型将正类别样本错误地分类为负类别的数量。
- **TN：**表示模型将正类别样本错误地分类为负类别的数量。

对于分割任务总共涉及到的 $k+1$ 类（含 k 个前景类和 1 个背景类），令 p_{ii} 表示正确分类的像素数；p_{ij} 式第 i 类被预测为第 j 类的像素数，像素准确率 PA 为分割类别正确的像素数占总像素数的比例，如式 (19) 所示。**像素准确率 PA** 常用于总体分割效果的估计，越接近 1 表明模型性能越好。

$$PA = \frac{TP+TN}{TP+TN+FN+TN} = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k}\sum_{j=0}^{k} p_{ij}} \tag{19}$$

像素精度所包含的信息有限，容易掩盖某一类别分割效果差的现象，对此可以使用**类别像素准确率 CPA** 来进行更加精确的评估。对第 i 类的分割结果来说，CPA 计算公式如式 (20) 所示。

$$CPA = \frac{TP}{TP+FP} = \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij}} \tag{20}$$

交并比 IoU 是当前语义分割研究中最常用的指标，表示分割掩码与标签像素的交叠率，如式 (21) 所示。

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP+FP+FN} = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k}\left(\sum_{j=0}^{k} p_{ji} + \sum_{j=0}^{k}(p_{ij}-p_{ii})\right)} \tag{21}$$

平均交并比 MIoU 是计算每个类别中 IoU 值的算数平均值，用于总体数据集的像素重叠情况，计算方式如式 (22) 所示。

$$\begin{aligned} MIoU &= \frac{1}{2}\left[\frac{TP}{TP+FP+FN} + \frac{TN}{TN+FN+FP}\right] \\ &= \frac{1}{k+1}\sum_{i=0}^{k}\frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k}(p_{ji}-p_{ii})} \end{aligned} \tag{22}$$

频率加权交并比 FWIoU 是对 MIoU 改进后的新评价标准，需对每个像素的类别按照其出现的频率设置权重，如式 (23) 所示。

$$\begin{aligned} FWIoU &= \frac{TP+FN}{TP+FP+TN+FN} \times \frac{TP}{TP+FP+FN} \\ &= \frac{1}{\sum_{i=0}^{k}\sum_{j=0}^{k} p_{ij}} \sum_{i=0}^{k}\left(\frac{\sum_{j=0}^{n} p_{ij} p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k}(p_{ji}-p_{ii})}\right) \end{aligned} \tag{23}$$

特征提取骨干网络对比　为了验证本文特征提取骨干网络的性能，本文分别选取在 ImageNet 上预训练过的 DRN，MobileNet，ResNet-101 作为 DeepLab-v3 的特征提取网络，SGD 作为参数优化器，PolyScheduler 作为学习率优化器，在相同的 $\mathcal{D}_{train}$ 上进行 50 轮次的训练，每轮训练后在标注数据中独立划分出的测试集上分别测试 AP、CAP、MIoU、FWIoU，如图 19 所示。

从图 19 中可以看出，在三个特征提取网络中，由于 MobileNet 的参数规模较小，性能明显低于另外两个网络。而 ResNet-101 的性能在 AP、CAP、MIoU、FWIoU 四个指标中都高于 DRN。模型在自主标注的测试数据 $\mathcal{D}_{test}$ 中性能数值如表 1 所示。

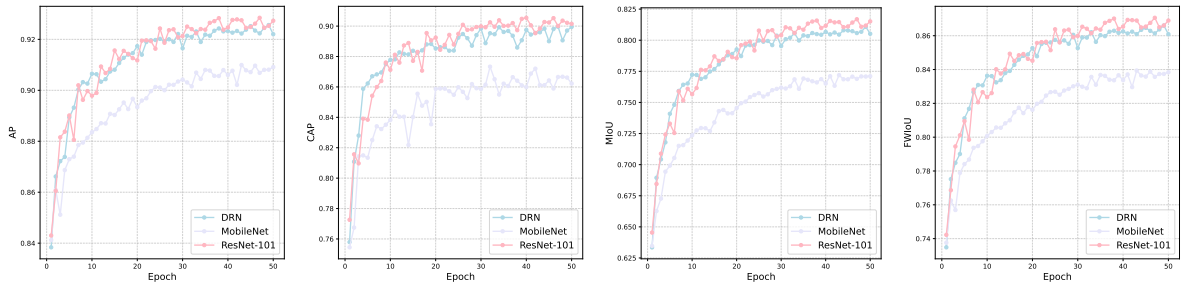

图 19　DeepLab v3+ 中不同特征提取骨干网络的性能

表 1　DeepLab v3+ 中不同特征提取骨干网络在自主标注测试集中的性能

特征提取骨干网络	AP	CAP	MIoU	FWIoU
DeepLab v3+ (DRN)	0.9220	0.8994	0.8051	0.8608
DeepLab v3+ (MobileNet)	0.9091	0.8622	0.7710	0.8383
DeepLab v3+ (ResNet-101)	**0.9272**	**0.9014**	**0.8151**	**0.8689**

语义分割模型对比　为了验证本文所选用语义分割模型 DeepLab v3+ 的性能，本文分别选取 U-Net、SegNet、DeepLab v3+ 两种网络，SGD 作为参数优化器，PolyScheduler 作为学习率优化器，在相同的 $\mathcal{D}_{train}$ 上进行 50 轮次的训练，每轮训练后在标注数据中独立划分出的测试集 $\mathcal{D}_{test}$ 上分别测试 AP、CAP、MIoU、FWIoU，如图 20 所示。

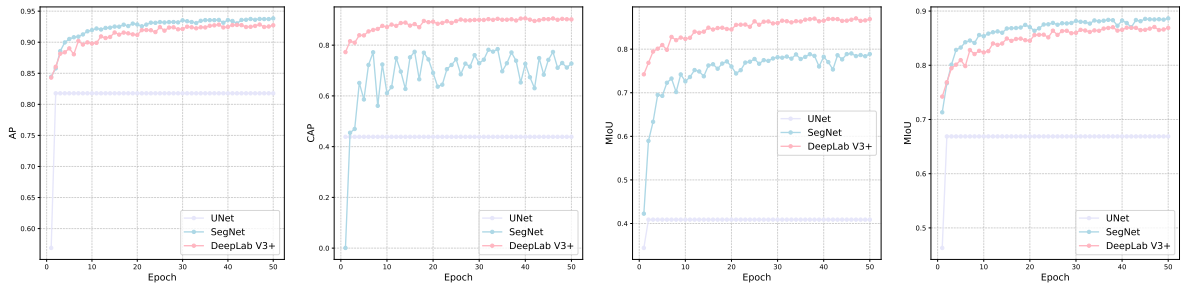

图 20　语义分割模型性能对比

从图 20 中可以在坑洼道路图像语义分割任务中，看出 UNet 比 DeepLab v3+ 模性能差距较远，且模型仅经过一轮训练后进入收敛状态。UNet 在训练过程中，CAP 持续为零，说明模型已经对坑洼语义信息失去了识别能力，将图像中所有像素均认为是正常道路图案。因为 UNet 模型的层次结构较为简单，不具备提取类似于坑洼图案的复杂高维语义特征，模型在使用损失函数进行训练时，为保证损失的最小化而直接放弃了对坑洼图案的识别。

而对于 SegNet，虽然在 AP 与 FWIoU 上比 DeepLab v3+ 略高，但观察到 CAP 与 MIoU 中仅有 0.7274 与 0.7888，远低于 0.9014 与 0.8151，说明其对坑洼像素点的分割能

力依然远低于 ResNet-101，其他两项指标高是因为坑洼图案占图像总体相对较少。实验的具体数值如表 2 所示。

表 2　不同语义分割模型在自主标注测试集中的性能

语义分割模型	AP	CAP	MIoU	FWIoU
UNet	0.8177	0.0000	0.4088	0.6687
SegNet	**0.9384**	0.7274	0.7888	**0.8865**
DeepLab v3+	0.9272	**0.9014**	**0.8151**	0.8689

模型最终训练　基于以上对比实验结果，本文选取 DeepLab v3+ 作为语义分割模型，ResNet-101 作为特征提取骨干网络，在 $\mathcal{D}_{train}$ 上进行 150 轮次的训练，每一轮训练后在 $\mathcal{D}_{test}$ 上进行测试，采用 AP、CAP、MIoU、FWIoU 四个指标进行性能验证，结果如图 21所示。

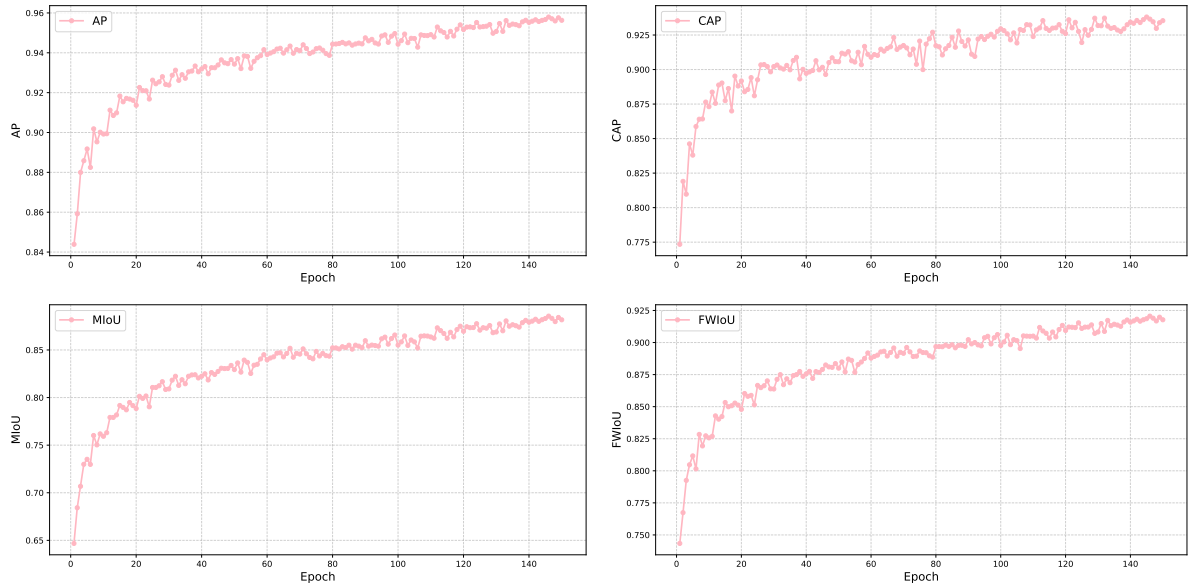

图 21　DeepLab v3+ 最终实验性能

DeepLab v3+ 在 D_{test} 上的 AP、CAP、MIoU、FWIoU 分别达到 0.9563、0.9353、0.8816、0.9177。

5.1.10 坑洼道路难样本处理效果

图 5 中展示了目标检测算法对坑洼道路图像中模糊图像、多坑图像、坑洼纹理复杂图像等难样本的检测效果，可以看出目标检测算法输出的回归框定位对于难样本的处理

是非常困难且不准确的。因为对于目标检测算法，标注信息没有精确到像素级，而图像中的坑洼特征非常复杂多样，目标检测算法无法在只有粗糙回归框标注的训练情况下对各种复杂的坑洼特征进行精准识别。

经过坑洼道路图像训练后的深度学习语义分割模型 DeepLab v3+，由于进行了像素级的语义信息标注，且通过垂直水平翻转、高斯模糊等数据增广的手段扩充了训练数据，因此具有较高的鲁棒性与坑洼特征识别能力，能够较为准确地识别各类难样本中的坑洼特征。图 22 展示了 DeepLab v3+ 对坑洼道路难样本的分割结果。

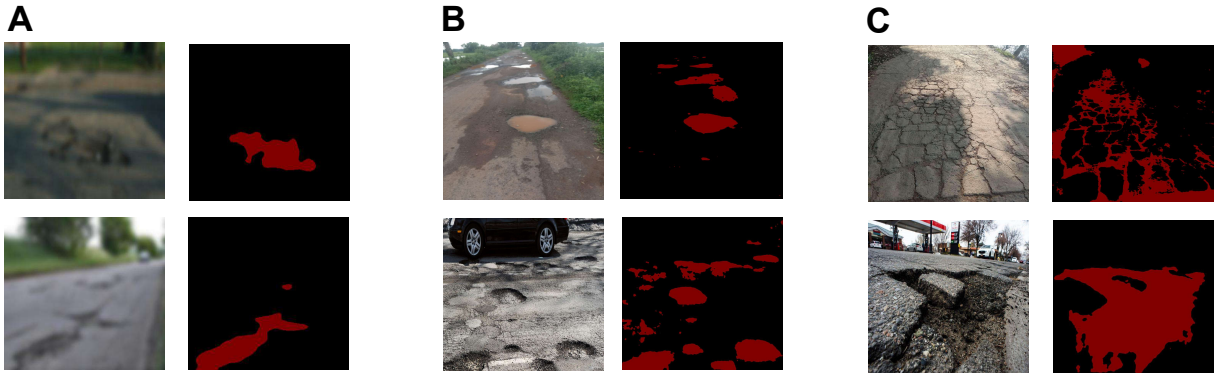

图 22　语义分割算法在坑洼道路图像难样本中的检测结果。（A）模糊图像。（B）多坑洼图像。（C）坑洼纹理复杂图像。

5.2 问题二：计算坑洼面积占比

在本文中，首先使用问题一建立的分类模型从题目所给数据中分离出坑洼路面图像，然后使用训练好的语义分割模型对坑洼道路图像进行语义分割并计算坑洼面积，将结果写入到结果文件中。本问建模流程如图 23 所示。

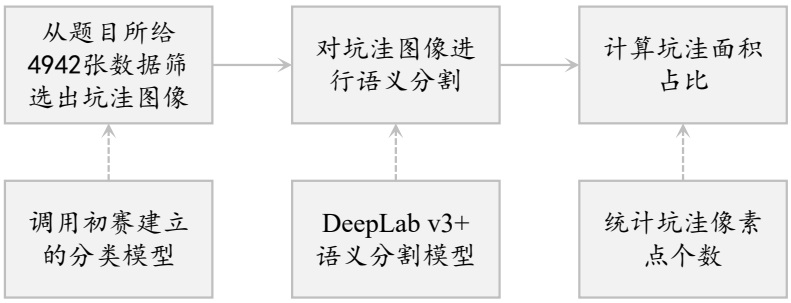

图 23　问题二建模流程

5.2.1 坑洼道路图像分类

对测试数据中所有 4942 张图像进行坑洼边缘特征提取并计算坑洼面积占比计算量较大，为简化计算，本文使用初赛问题建立的**基于伪样本生成增广与通道注意力增强的**

坑洼道路图像分类模型对 D_{test} 内所有图像进行分类，仅对分类为“坑洼道路”的图像进行坑洼道路语义分割。

经过分类模型进行分类后，$\mathcal{D}_{test}$4942 张图像内共有 3725 张图像被识别为“坑洼道路图像”，本文对上述图像提取坑洼边缘特征并计算坑洼面积占比。

5.2.2 面积占比计算

使用问题一建立的深度学习语义分割模型对分类出的坑洼道路图像进行语义分割，分割后的语义图像如图所示。部分具体语义分割实验效果见附录。

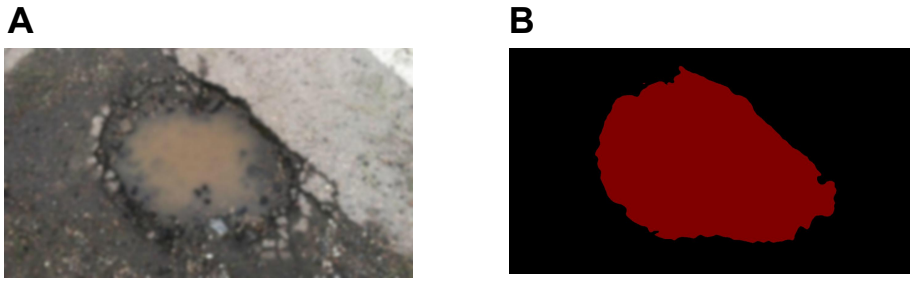

图 24　DeepLab v3+ 图像分割后的语义图

从图 24 中可以看出，在语义分割模型分割出的语义图中，被识别为坑洼图案的像素点为红色，被识别为背景图案的像素点为黑色。红色像素点在 RGB 通道的具体像素值为 $(128, 0, 0)$，黑色像素点为 $(0, 0, 0)$。因此，对于分割出的语义图 $\mathbf{x}_{seg}$，只需计算非零像素点占图像所有像素点的比值即可得出坑洼在原图中的面积占比。具体计算公式如式 (24) 所示。

$$S_{pothole} = \frac{N_{p_{pothole}}}{H_{\mathbf{x}_{seg}} \times W_{\mathbf{x}_{seg}}} \tag{24}$$

其中 $N_{p_{pothole}}$ 表示 $\mathbf{x}_{seg}$ 中非零像素点的个数，$H_{\mathbf{x}_{seg}}$ 和 $W_{\mathbf{x}_{seg}}$ 分别表示 $\mathbf{x}_{seg}$ 的高度与宽度，两者的乘积即为 $\mathbf{x}_{seg}$ 的像素点个数。

计算出每张图像的 $S_{pothole}$，将计算结果填入到 test_result2.csv 文件中。该问题完整处理流程如算法 3 所示。

六、 模型的评价与推广

6.1 模型优点

- 基于深度学习的语义分割模型将图像的语义信息精确到像素级，使图像能够精准地提特征复杂地不规则坑洼边缘，并且对模糊图像、多坑洼图像、纹理特征复杂图像等坑洼道路图像中的难样本也有较好的提取效果。

Algorithm 3: 处理问题二流程

Data: 题目所给数据 $\mathcal{D}_{test}$；初赛问题中训练好的坑洼道路图像分类模型 f_C；在坑洼道路图像数据集中训练好的语义分割模型 f'_ϕ；

for $i = 1$ **to** $\mathcal{D}_{test}$ **do**
  // $\mathbf{x}_i$ 为 $\mathcal{D}_{test}$ 中第 i 个图像;
  f_C 对图像进行分类;
  if $\mathbf{x}_i$ 为正常图像 **then**
    根据文件名在 test_result2.csv 中填入 0;
  else
    $\mathbf{x}_{seg} = f'_\phi(\mathbf{x}_i)$;
    统计 $\mathbf{x}_{seg}$ 中非零像素数量 $N_{p_{pothole}}$;
    计算 $\mathbf{x}_{seg}$ 高度 $H_{\mathbf{x}_{seg}}$ 与宽度 $W_{\mathbf{x}_{seg}}$;
    $S_{pothole} = \frac{N_{p_{pothole}}}{H_{\mathbf{x}_{seg}} \times W_{\mathbf{x}_{seg}}}$;
    根据文件名在 test_result10csv 中填入 $S_{pothole}$;
return 填写好面积占比结果的 test_result2.csv 文件

- 本文对公开数据集的数据自行进行标注后，使用垂直水平翻转，高斯模糊等图像增广手段，增加数据集中的空间特征复杂性，使模型更能适应场景中物体的不同姿态和方向变化。
- DeepLab v3+ 模型采用编码器-解码器结构来执行语义分割任务，能够将分隔后图像完全复原成原始输入图像的尺寸大小，抑制了语义信息的损失。同时，DeepLab v3+ 模型引入空洞卷积、空间特征金字塔池化，使得模型能更好地提取坑洼图像中的多尺度特征，对不同尺寸的坑洼道路图像均具备分割能力。

6.2 模型缺点

- 模型没有对学习率等超参数设置进行深入调优。
- 缺乏足够的训练数据。

6.3 模型推广

本文基于深度学习构建了用于坑洼道路语义分割的 DeepLab v3+ 语义分割模型。该模型不仅适用于坑洼道路语义分割，对于空间特征较为复杂、环境多样、成像条件参差不齐的图像语义分割任务均适用。同时，本文模型进行了充分的数据增广，对于样本量较小的语义分割任务同样能取得优越的性能。

参考文献

[1] 冯树杰. 基于深度学习的路面缺陷检测算法研究 [D]. 陕西科技大学, 2023. DOI: 10.27290/d.cnki.gxbqc. 2023. 000082.

[2] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[3] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3431-3440.

[4] 于营, 王春平, 付强, 等. 语义分割评价指标和评价方法综述 [J]. Journal of Computer Engineering & Applications, 2023, 59(6).

附录 A　坑洼道路图像语义分割效果展示

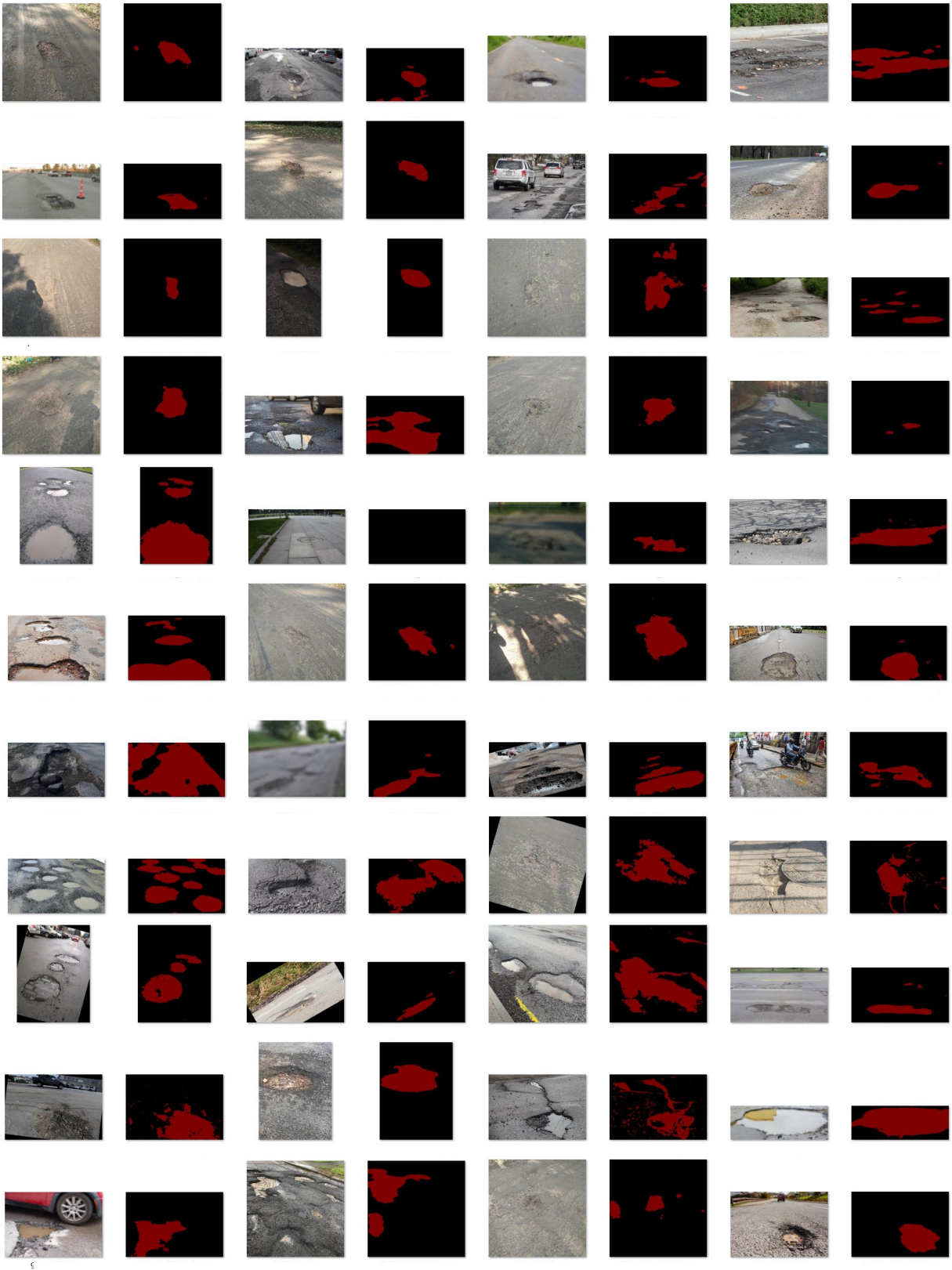

图 25　坑洼道路图像语义分割效果展示（1）

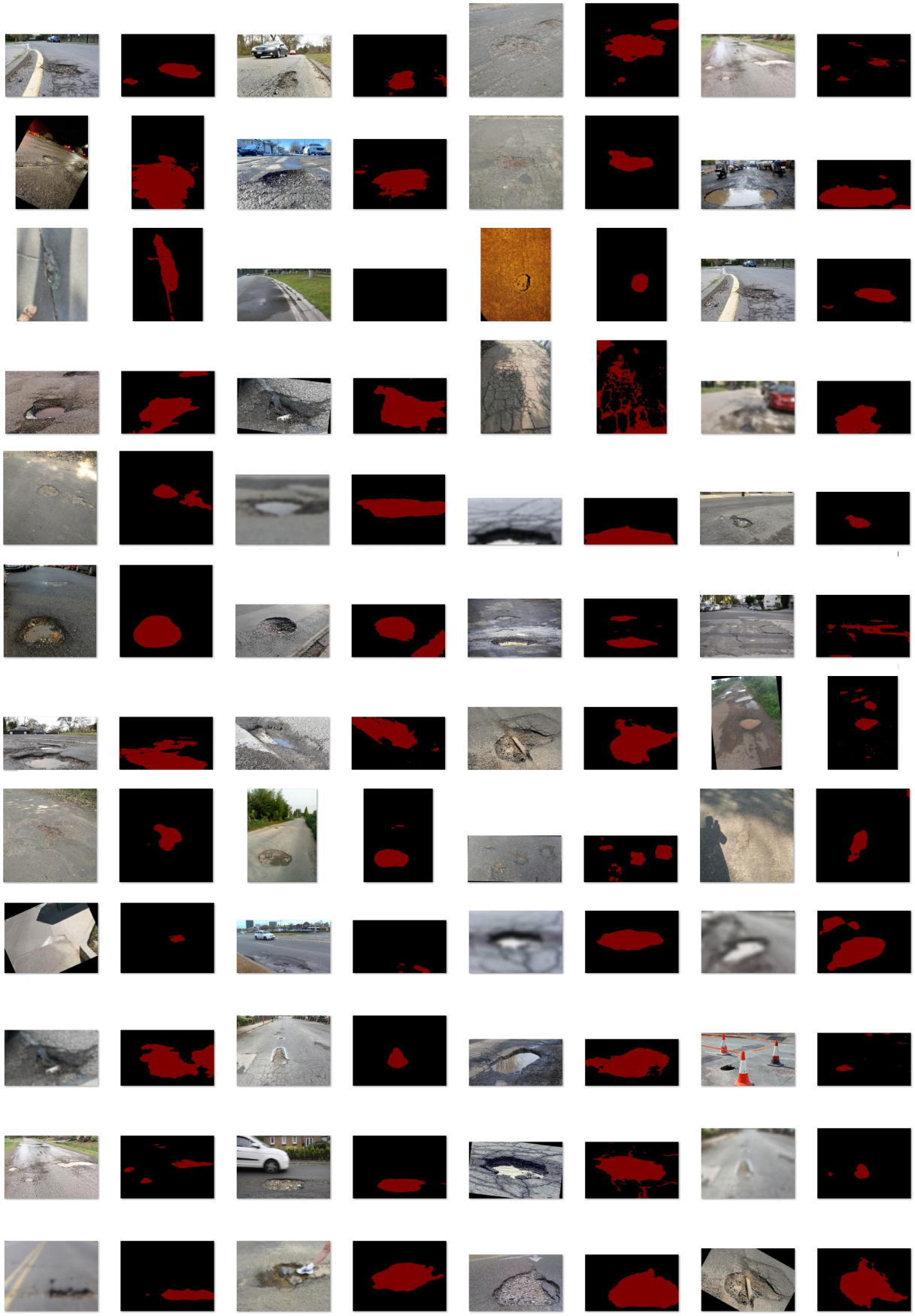

图 26　坑洼道路图像语义分割效果展示（2）

附录 B　源代码

2.1　问题一源代码

2.1.1　训练与测试源代码

train.py

```
import argparse
import csv
import os
import numpy as np
from tqdm import tqdm

from mypath import Path
from dataloaders import make_data_loader
from modeling.sync_batchnorm.replicate import patch_replication_callback
from modeling.deeplab import *
from utils.loss import SegmentationLosses
from utils.calculate_weights import calculate_weigths_labels
from utils.lr_scheduler import LR_Scheduler
from utils.saver import Saver
from utils.summaries import TensorboardSummary
from utils.metrics import Evaluator


class Trainer(object):
    def __init__(self, args):
        self.args = args
        self.epoch_data = []
        self.csv_filename = '../../output.csv'

        # Define Saver
        self.saver = Saver(args)
        self.saver.save_experiment_config()
        # Define Tensorboard Summary
        self.summary = TensorboardSummary(self.saver.experiment_dir)
        self.writer = self.summary.create_summary()

        # Define Dataloader
        kwargs = {'num_workers': args.workers, 'pin_memory': True}
        self.train_loader, self.val_loader, self.test_loader, self.nclass =
            make_data_loader(args, **kwargs)

        # Define network
        model = DeepLab(num_classes=self.nclass,
                        backbone=args.backbone,
```

```
                    output_stride=args.out_stride,
                    sync_bn=args.sync_bn,
                    freeze_bn=args.freeze_bn)

        train_params = [{'params': model.get_1x_lr_params(), 'lr': args.lr},
                    {'params': model.get_10x_lr_params(), 'lr': args.lr * 10}]

        # Define Optimizer
        optimizer = torch.optim.SGD(train_params, momentum=args.momentum,
                                    weight_decay=args.weight_decay, nesterov=args.nesterov)

        # Define Criterion
        # whether to use class balanced weights
        if args.use_balanced_weights:
            classes_weights_path = os.path.join(Path.db_root_dir(args.dataset),
                args.dataset+'_classes_weights.npy')
            if os.path.isfile(classes_weights_path):
                weight = np.load(classes_weights_path)
            else:
                weight = calculate_weigths_labels(args.dataset, self.train_loader, self.nclass)
            weight = torch.from_numpy(weight.astype(np.float32))
        else:
            weight = None
        self.criterion = SegmentationLosses(weight=weight,
            cuda=args.cuda).build_loss(mode=args.loss_type)
        self.model, self.optimizer = model, optimizer

        # Define Evaluator
        self.evaluator = Evaluator(self.nclass)
        # Define lr scheduler
        self.scheduler = LR_Scheduler(args.lr_scheduler, args.lr,
                                      args.epochs, len(self.train_loader))

        # Using cuda
        if args.cuda:
            self.model = torch.nn.DataParallel(self.model, device_ids=self.args.gpu_ids)
            patch_replication_callback(self.model)
            self.model = self.model.cuda()

        # Resuming checkpoint
        self.best_pred = 0.0
        if args.resume is not None:
            if not os.path.isfile(args.resume):
                raise RuntimeError("=> no checkpoint found at '{}'" .format(args.resume))
            checkpoint = torch.load(args.resume)
            args.start_epoch = checkpoint['epoch']
            if args.cuda:
```

```
            self.model.module.load_state_dict(checkpoint['state_dict'])
        else:
            self.model.load_state_dict(checkpoint['state_dict'])
        if not args.ft:
            self.optimizer.load_state_dict(checkpoint['optimizer'])
        self.best_pred = checkpoint['best_pred']
        print("=> loaded checkpoint '{}' (epoch {})"
              .format(args.resume, checkpoint['epoch']))

    # Clear start epoch if fine-tuning
    if args.ft:
        args.start_epoch = 0

def training(self, epoch):
    train_loss = 0.0
    self.model.train()
    tbar = tqdm(self.train_loader)
    num_img_tr = len(self.train_loader)
    for i, sample in enumerate(tbar):
        image, target = sample['image'], sample['label']
        if self.args.cuda:
            image, target = image.cuda(), target.cuda()
        self.scheduler(self.optimizer, i, epoch, self.best_pred)
        self.optimizer.zero_grad()
        output = self.model(image)
        loss = self.criterion(output, target)
        loss.backward()
        self.optimizer.step()
        train_loss += loss.item()
        tbar.set_description('Train loss: %.3f' % (train_loss / (i + 1)))
        self.writer.add_scalar('train/total_loss_iter', loss.item(), i + num_img_tr * epoch)

        # Show 10 * 3 inference results each epoch
        if i % (num_img_tr // 10) == 0:
            global_step = i + num_img_tr * epoch
            self.summary.visualize_image(self.writer, self.args.dataset, image, target,
                output, global_step)

    self.writer.add_scalar('train/total_loss_epoch', train_loss, epoch)
    print('[Epoch: %d, numImages: %5d]' % (epoch, i * self.args.batch_size +
        image.data.shape[0]))
    print('Loss: %.3f' % train_loss)

    if self.args.no_val:
        # save checkpoint every epoch
        is_best = False
        self.saver.save_checkpoint({
```

```
            'epoch': epoch + 1,
            'state_dict': self.model.module.state_dict(),
            'optimizer': self.optimizer.state_dict(),
            'best_pred': self.best_pred,
        }, is_best)


def validation(self, epoch):

    self.model.eval()
    self.evaluator.reset()
    tbar = tqdm(self.val_loader, desc='\r')
    test_loss = 0.0
    for i, sample in enumerate(tbar):
        image, target = sample['image'], sample['label']
        if self.args.cuda:
            image, target = image.cuda(), target.cuda()
        with torch.no_grad():
            output = self.model(image)
        loss = self.criterion(output, target)
        test_loss += loss.item()
        tbar.set_description('Test loss: %.3f' % (test_loss / (i + 1)))
        pred = output.data.cpu().numpy()
        target = target.cpu().numpy()
        pred = np.argmax(pred, axis=1)
        # Add batch sample into evaluator
        self.evaluator.add_batch(target, pred)

    # Fast test during the training
    Acc = self.evaluator.Pixel_Accuracy()
    Acc_class = self.evaluator.Pixel_Accuracy_Class()
    mIoU = self.evaluator.Mean_Intersection_over_Union()
    FWIoU = self.evaluator.Frequency_Weighted_Intersection_over_Union()
    self.writer.add_scalar('val/total_loss_epoch', test_loss, epoch)
    self.writer.add_scalar('val/mIoU', mIoU, epoch)
    self.writer.add_scalar('val/Acc', Acc, epoch)
    self.writer.add_scalar('val/Acc_class', Acc_class, epoch)
    self.writer.add_scalar('val/fwIoU', FWIoU, epoch)
    print('Validation:')
    print('[Epoch: %d, numImages: %5d]' % (epoch, i * self.args.batch_size +
        image.data.shape[0]))
    print("Acc:{}, Acc_class:{}, mIoU:{}, fwIoU: {}".format(Acc, Acc_class, mIoU, FWIoU))
    print('Loss: %.3f' % test_loss)
    self.epoch_data.append((epoch, Acc, Acc_class, mIoU, FWIoU, test_loss))

    new_pred = mIoU
    if new_pred > self.best_pred:
```

```
            is_best = True
            self.best_pred = new_pred
            self.saver.save_checkpoint({
                'epoch': epoch + 1,
                'state_dict': self.model.module.state_dict(),
                'optimizer': self.optimizer.state_dict(),
                'best_pred': self.best_pred,
            }, is_best)

    def record(self):
        with open(self.csv_filename, 'w', newline='') as csvfile:
            # 创建CSV写入对象
            csv_writer = csv.writer(csvfile)

            # 写入表头
            csv_writer.writerow(['Epoch', 'Acc', 'Acc_class', 'mIoU', 'FWIoU', 'Loss'])

            # 写入数据
            csv_writer.writerows(self.epoch_data)

def main():
    parser = argparse.ArgumentParser(description="PyTorch DeeplabV3Plus Training")
    parser.add_argument('--backbone', type=str, default='resnet',
                        choices=['resnet', 'xception', 'drn', 'mobilenet'],
                        help='backbone name (default: resnet)')
    parser.add_argument('--out-stride', type=int, default=16,
                        help='network output stride (default: 8)')
    parser.add_argument('--dataset', type=str, default='mydata',
                        choices=['pascal', 'coco', 'cityscapes', 'mydata'],
                        help='dataset name (default: pascal)')
    parser.add_argument('--use-sbd', action='store_true', default=False,
                        help='whether to use SBD dataset (default: True)')
    parser.add_argument('--workers', type=int, default=0,
                        metavar='N', help='dataloader threads')
    parser.add_argument('--base-size', type=int, default=513,
                        help='base image size')
    parser.add_argument('--crop-size', type=int, default=513,
                        help='crop image size')
    parser.add_argument('--sync-bn', type=bool, default=None,
                        help='whether to use sync bn (default: auto)')
    parser.add_argument('--freeze-bn', type=bool, default=False,
                        help='whether to freeze bn parameters (default: False)')
    parser.add_argument('--loss-type', type=str, default='ce',
                        choices=['ce', 'focal'],
                        help='loss func type (default: ce)')
    # training hyper params
    parser.add_argument('--epochs', type=int, default=150, metavar='N',
```

```
                    help='number of epochs to train (default: auto)')
    parser.add_argument('--start_epoch', type=int, default=0,
                    metavar='N', help='start epochs (default:0)')
    parser.add_argument('--batch-size', type=int, default=2,
                    metavar='N', help='input batch size for \
                            training (default: auto)')
    parser.add_argument('--test-batch-size', type=int, default=2,
                    metavar='N', help='input batch size for \
                            testing (default: auto)')
    parser.add_argument('--use-balanced-weights', action='store_true', default=False,
                    help='whether to use balanced weights (default: False)')
    # optimizer params
    parser.add_argument('--lr', type=float, default=0.007 , metavar='LR',
                    help='learning rate (default: auto)')
    parser.add_argument('--lr-scheduler', type=str, default='poly',
                    choices=['poly', 'step', 'cos'],
                    help='lr scheduler mode: (default: poly)')
    parser.add_argument('--momentum', type=float, default=0.9,
                    metavar='M', help='momentum (default: 0.9)')
    parser.add_argument('--weight-decay', type=float, default=5e-4,
                    metavar='M', help='w-decay (default: 5e-4)')
    parser.add_argument('--nesterov', action='store_true', default=False,
                    help='whether use nesterov (default: False)')
    # cuda, seed and logging
    parser.add_argument('--no-cuda', action='store_true', default=
                    False, help='disables CUDA training')
    parser.add_argument('--gpu-ids', type=str, default='0',
                    help='use which gpu to train, must be a \
                    comma-separated list of integers only (default=0)')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
    # checking point
    parser.add_argument('--resume', type=str, default=None,
                    help='put the path to resuming file if needed')
    parser.add_argument('--checkname', type=str, default='deeplab-resnet',
                    help='set the checkpoint name')
    # finetuning pre-trained models
    parser.add_argument('--ft', action='store_true', default=True,
                    help='finetuning on a different dataset')
    # evaluation option
    parser.add_argument('--eval-interval', type=int, default=1,
                    help='evaluuation interval (default: 1)')
    parser.add_argument('--no-val', action='store_true', default=False,
                    help='skip validation during training')

    args = parser.parse_args()
    args.cuda = not args.no_cuda and torch.cuda.is_available()
```

```
    if args.cuda:
        try:
            args.gpu_ids = [int(s) for s in args.gpu_ids.split(',')]
        except ValueError:
            raise ValueError('Argument --gpu_ids must be a comma-separated list of integers
                only')

    if args.sync_bn is None:
        if args.cuda and len(args.gpu_ids) > 1:
            args.sync_bn = True
        else:
            args.sync_bn = False

    # default settings for epochs, batch_size and lr
    if args.epochs is None:
        epoches = {
            'coco': 30,
            'cityscapes': 200,
            'pascal': 50,
        }
        args.epochs = epoches[args.dataset.lower()]

    if args.batch_size is None:
        args.batch_size = 4 * len(args.gpu_ids)

    if args.test_batch_size is None:
        args.test_batch_size = args.batch_size

    if args.lr is None:
        lrs = {
            'coco': 0.1,
            'cityscapes': 0.01,
            'pascal': 0.007,
        }
        args.lr = lrs[args.dataset.lower()] / (4 * len(args.gpu_ids)) * args.batch_size


    if args.checkname is None:
        args.checkname = 'deeplab-'+str(args.backbone)
    print(args)
    torch.manual_seed(args.seed)
    trainer = Trainer(args)
    print('Starting Epoch:', trainer.args.start_epoch)
    print('Total Epoches:', trainer.args.epochs)
    for epoch in range(trainer.args.start_epoch, trainer.args.epochs):
        trainer.training(epoch)
        if not trainer.args.no_val and epoch % args.eval_interval == (args.eval_interval - 1):
```

```
            trainer.validation(epoch)

    trainer.record()
    trainer.writer.close()

if __name__ == "__main__":
   main()
```

demo.py

```
#
# demo.py
#
import argparse
import os
import numpy as np
import time
import gc
from modeling.deeplab import *
from dataloaders import custom_transforms as tr
from PIL import Image
from torchvision import transforms
from dataloaders.utils import *
from torchvision.utils import make_grid, save_image

def main():

    parser = argparse.ArgumentParser(description="PyTorch DeeplabV3Plus Training")
    parser.add_argument('--in-path', type=str, required=False, help='image to test',
        default='./dataset/test_data/potholes')
    parser.add_argument('--out-path', type=str, required=False, help='mask image to save',
        default='./dataset/output')
    parser.add_argument('--backbone', type=str, default='resnet',
                        choices=['resnet', 'xception', 'drn', 'mobilenet'],
                        help='backbone name (default: resnet)')
    parser.add_argument('--ckpt', type=str,
        default='./run/mydata/deeplab-resnet/model_best.pth.tar',
                        help='saved model')
    parser.add_argument('--out-stride', type=int, default=16,
                        help='network output stride (default: 8)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--gpu-ids', type=str, default='0',
                        help='use which gpu to train, must be a \
                        comma-separated list of integers only (default=0)')
    parser.add_argument('--dataset', type=str, default='mydata',
                        choices=['pascal', 'coco', 'cityscapes','invoice', 'mydata'],
```

```
                    help='dataset name (default: pascal)')
    parser.add_argument('--crop-size', type=int, default=500,
                    help='crop image size')
    parser.add_argument('--num_classes', type=int, default=2,
                    help='crop image size')
    parser.add_argument('--sync-bn', type=bool, default=None,
                    help='whether to use sync bn (default: auto)')
    parser.add_argument('--freeze-bn', type=bool, default=False,
                    help='whether to freeze bn parameters (default: False)')

    args = parser.parse_args()
    args.cuda = not args.no_cuda and torch.cuda.is_available()
    if args.cuda:
        try:
            args.gpu_ids = [int(s) for s in args.gpu_ids.split(',')]
        except ValueError:
            raise ValueError('Argument --gpu_ids must be a comma-separated list of integers
                only')

    if args.sync_bn is None:
        if args.cuda and len(args.gpu_ids) > 1:
            args.sync_bn = True
        else:
            args.sync_bn = False
    model_s_time = time.time()
    model = DeepLab(num_classes=args.num_classes,
                    backbone=args.backbone,
                    output_stride=args.out_stride,
                    sync_bn=args.sync_bn,
                    freeze_bn=args.freeze_bn)

    ckpt = torch.load(args.ckpt, map_location='cpu')
    model.load_state_dict(ckpt['state_dict'])
    model = model.cuda()
    model.eval()
    model_u_time = time.time()
    model_load_time = model_u_time-model_s_time
    print("model load time is {}".format(model_load_time))

    composed_transforms = transforms.Compose([
        tr.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        tr.ToTensor()])
    for name in os.listdir(args.in_path):
        with torch.no_grad():
            gc.collect()
            torch.cuda.empty_cache()
            s_time = time.time()
```

```
            image = Image.open(os.path.join(args.in_path, name)).convert('RGB')
            target = Image.open(os.path.join(args.in_path, name)).convert('L')
            sample = {'image': image, 'label': target}

            tensor_in = composed_transforms(sample)['image'].unsqueeze(0)
            tensor_in = tensor_in.cuda()


            output = model(tensor_in)

            grid_image = make_grid(decode_seg_map_sequence(torch.max(output[:3],
                1)[1].detach().cpu().numpy()),
                            3, normalize=False, range=(0, 255))
            save_image(grid_image, os.path.join(args.in_path, "{}_mask.png".format(name[0:-4])))

        u_time = time.time()
        img_time = u_time - s_time
        print("image:{} time: {} ".format(name, img_time))


    print("image save in in_path.")
if __name__ == "__main__":
   main()

# python demo.py --in-path your_file --out-path your_dst_file
```

2.1.2 构建数据集源代码

json2dataset.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# 这段代码负责将labelme标注后的文件转换为正确的格式
# @Time : 2023/12/15 下午12:56
# @Author : SiHang Xu
import os
files=os.listdir('./')
files.remove('json2dataset.py') # 删除这个py文件本身
for i in range(len(files)):
    os.system('labelme_json_to_dataset '+files[i])
```

split_dataset.py

```
#随机生成trian.txt, trainval.txt, val.txt

import os
```

```python
import random

# 设置路径和文件名
path = "./JPEGImages" # 替换为你的图片所在的路径
train_file = "train.txt"
trainval_file = "trainval.txt"
val_file = "val.txt"

# 获取路径下的所有图片文件
image_files = [f for f in os.listdir(path) if f.endswith(".jpg") or f.endswith(".png")]

# 打乱文件列表
random.shuffle(image_files)

# 计算划分的索引
total_images = len(image_files)
train_split = 1400
trainval_split = 1500

# 划分文件列表
train_images = [os.path.splitext(f)[0] for f in image_files[:train_split]]
trainval_images = [os.path.splitext(f)[0] for f in image_files[train_split:trainval_split]]
val_images = [os.path.splitext(f)[0] for f in image_files[trainval_split:]]

# 写入文件名到train.txt
with open(train_file, "w") as f:
    f.write("\n".join(train_images))

# 写入文件名到trainval.txt
with open(trainval_file, "w") as f:
    f.write("\n".join(trainval_images))

# 写入文件名到val.txt
with open(val_file, "w") as f:
    f.write("\n".join(val_images))
```

custom_transform.py

```python
import torch
import random
import numpy as np

from PIL import Image, ImageOps, ImageFilter

class Normalize(object):
    """Normalize a tensor image with mean and standard deviation.
    Args:
```

```
        mean (tuple): means for each channel.
        std (tuple): standard deviations for each channel.
    """
    def __init__(self, mean=(0., 0., 0.), std=(1., 1., 1.)):
        self.mean = mean
        self.std = std

    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        img = np.array(img).astype(np.float32)
        mask = np.array(mask).astype(np.float32)
        img /= 255.0
        img -= self.mean
        img /= self.std

        return {'image': img,
                'label': mask}


class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        # swap color axis because
        # numpy image: H x W x C
        # torch image: C X H X W
        img = sample['image']
        mask = sample['label']
        img = np.array(img).astype(np.float32).transpose((2, 0, 1))
        mask = np.array(mask).astype(np.float32)

        img = torch.from_numpy(img).float()
        mask = torch.from_numpy(mask).float()

        return {'image': img,
                'label': mask}


class RandomHorizontalFlip(object):
    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        if random.random() < 0.5:
            img = img.transpose(Image.FLIP_LEFT_RIGHT)
            mask = mask.transpose(Image.FLIP_LEFT_RIGHT)

```

```
        return {'image': img,
                'label': mask}


class RandomRotate(object):
    def __init__(self, degree):
        self.degree = degree

    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        rotate_degree = random.uniform(-1*self.degree, self.degree)
        img = img.rotate(rotate_degree, Image.BILINEAR)
        mask = mask.rotate(rotate_degree, Image.NEAREST)

        return {'image': img,
                'label': mask}


class RandomGaussianBlur(object):
    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        if random.random() < 0.5:
            img = img.filter(ImageFilter.GaussianBlur(
                radius=random.random()))

        return {'image': img,
                'label': mask}


class RandomScaleCrop(object):
    def __init__(self, base_size, crop_size, fill=0):
        self.base_size = base_size
        self.crop_size = crop_size
        self.fill = fill

    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        # random scale (short edge)
        short_size = random.randint(int(self.base_size * 0.5), int(self.base_size * 2.0))
        w, h = img.size
        if h > w:
            ow = short_size
            oh = int(1.0 * h * ow / w)
        else:
```

```
            oh = short_size
            ow = int(1.0 * w * oh / h)
        img = img.resize((ow, oh), Image.BILINEAR)
        mask = mask.resize((ow, oh), Image.NEAREST)
        # pad crop
        if short_size < self.crop_size:
            padh = self.crop_size - oh if oh < self.crop_size else 0
            padw = self.crop_size - ow if ow < self.crop_size else 0
            img = ImageOps.expand(img, border=(0, 0, padw, padh), fill=0)
            mask = ImageOps.expand(mask, border=(0, 0, padw, padh), fill=self.fill)
        # random crop crop_size
        w, h = img.size
        x1 = random.randint(0, w - self.crop_size)
        y1 = random.randint(0, h - self.crop_size)
        img = img.crop((x1, y1, x1 + self.crop_size, y1 + self.crop_size))
        mask = mask.crop((x1, y1, x1 + self.crop_size, y1 + self.crop_size))

        return {'image': img,
                'label': mask}


class FixScaleCrop(object):
    def __init__(self, crop_size):
        self.crop_size = crop_size

    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']
        w, h = img.size
        if w > h:
            oh = self.crop_size
            ow = int(1.0 * w * oh / h)
        else:
            ow = self.crop_size
            oh = int(1.0 * h * ow / w)
        img = img.resize((ow, oh), Image.BILINEAR)
        mask = mask.resize((ow, oh), Image.NEAREST)
        # center crop
        w, h = img.size
        x1 = int(round((w - self.crop_size) / 2.))
        y1 = int(round((h - self.crop_size) / 2.))
        img = img.crop((x1, y1, x1 + self.crop_size, y1 + self.crop_size))
        mask = mask.crop((x1, y1, x1 + self.crop_size, y1 + self.crop_size))

        return {'image': img,
                'label': mask}

```

```python
class FixedResize(object):
    def __init__(self, size):
        self.size = (size, size) # size: (h, w)

    def __call__(self, sample):
        img = sample['image']
        mask = sample['label']

        assert img.size == mask.size

        img = img.resize(self.size, Image.BILINEAR)
        mask = mask.resize(self.size, Image.NEAREST)

        return {'image': img,
                'label': mask}
```

mydata.py

```python
from __future__ import print_function, division
import os
from PIL import Image
import numpy as np
from torch.utils.data import Dataset
from mypath import Path
from torchvision import transforms
from dataloaders import custom_transforms as tr

class VOCSegmentation(Dataset):
    """
    PascalVoc dataset
    """
    NUM_CLASSES = 2

    def __init__(self,
                 args,
                 base_dir=Path.db_root_dir('mydata'),
                 split='train',
                 ):
        """
        :param base_dir: path to VOC dataset directory
        :param split: train/val
        :param transform: transform to apply
        """
        super().__init__()
        self._base_dir = base_dir
        self._image_dir = os.path.join(self._base_dir, 'JPEGImages')
        self._cat_dir = os.path.join(self._base_dir, 'SegmentationClass')
```

```
        if isinstance(split, str):
            self.split = [split]
        else:
            split.sort()
            self.split = split

        self.args = args

        _splits_dir = os.path.join(self._base_dir, 'ImageSets', 'Segmentation')

        self.im_ids = []
        self.images = []
        self.categories = []

        for splt in self.split:
            with open(os.path.join(os.path.join(_splits_dir, splt + '.txt')), "r") as f:
                lines = f.read().splitlines()

            for ii, line in enumerate(lines):
                _image = os.path.join(self._image_dir, line + ".png")
                _cat = os.path.join(self._cat_dir, line + ".png")
                assert os.path.isfile(_image)
                assert os.path.isfile(_cat)
                self.im_ids.append(line)
                self.images.append(_image)
                self.categories.append(_cat)

        assert (len(self.images) == len(self.categories))

        # Display stats
        print('Number of images in {}: {:d}'.format(split, len(self.images)))

    def __len__(self):
        return len(self.images)


    def __getitem__(self, index):
        _img, _target = self._make_img_gt_point_pair(index)
        sample = {'image': _img, 'label': _target}

        for split in self.split:
            if split == "train":
                return self.transform_tr(sample)
            elif split == 'val':
                return self.transform_val(sample)

```

```
77
78      def _make_img_gt_point_pair(self, index):
79          _img = Image.open(self.images[index]).convert('RGB')
80          _target = Image.open(self.categories[index])
81
82          return _img, _target
83
84      def transform_tr(self, sample):
85          composed_transforms = transforms.Compose([
86              tr.RandomHorizontalFlip(),
87              tr.RandomScaleCrop(base_size=self.args.base_size, crop_size=self.args.crop_size),
88              tr.RandomGaussianBlur(),
89              tr.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
90              tr.ToTensor()])
91
92          return composed_transforms(sample)
93
94      def transform_val(self, sample):
95
96          composed_transforms = transforms.Compose([
97              tr.FixScaleCrop(crop_size=self.args.crop_size),
98              tr.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
99              tr.ToTensor()])
100
101         return composed_transforms(sample)
102
103     def __str__(self):
104         return 'VOC2012(split=' + str(self.split) + ')'
105
106
107 if __name__ == '__main__':
108     from dataloaders.utils import decode_segmap
109     from torch.utils.data import DataLoader
110     import matplotlib.pyplot as plt
111     import argparse
112
113     parser = argparse.ArgumentParser()
114     args = parser.parse_args()
115     args.base_size = 513
116     args.crop_size = 513
117
118     voc_train = VOCSegmentation(args, split='train')
119
120     dataloader = DataLoader(voc_train, batch_size=5, shuffle=True, num_workers=0)
121
122     for ii, sample in enumerate(dataloader):
123         for jj in range(sample["image"].size()[0]):
```

```
            img = sample['image'].numpy()
            gt = sample['label'].numpy()
            tmp = np.array(gt[jj]).astype(np.uint8)
            segmap = decode_segmap(tmp, dataset='pascal')
            img_tmp = np.transpose(img[jj], axes=[1, 2, 0])
            img_tmp *= (0.229, 0.224, 0.225)
            img_tmp += (0.485, 0.456, 0.406)
            img_tmp *= 255.0
            img_tmp = img_tmp.astype(np.uint8)
            plt.figure()
            plt.title('display')
            plt.subplot(211)
            plt.imshow(img_tmp)
            plt.subplot(212)
            plt.imshow(segmap)

        if ii == 1:
            break

    plt.show(block=True)
```

2.1.3 数据增广源代码

augment_image.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# 图像增广
# @Time : 2023/12/12 下午8:05
# @Author : SiHang Xu
from torchvision import transforms
from PIL import Image
import os
import random

count = 0


def augment_images(input_path, output_path, seed=42):
    global count
    # 设置随机数种子


    random.seed(seed)

    # 创建输出路径
```

```
    os.makedirs(output_path, exist_ok=True)

    # 获取两组图片的文件名列表
    files_group1 = os.listdir(input_path + '/JPEGImages')
    files_group2 = os.listdir(input_path + '/SegmentationClass')

    # 对每一对文件进行相同的增广操作
    for file1, file2 in zip(files_group1, files_group2):
        image1 = Image.open(os.path.join(input_path + '/JPEGImages', file1))
        image2 = Image.open(os.path.join(input_path + '/SegmentationClass', file2))

        # 对两组图像应用相同的随机数操作
        random.seed(seed)
        rand_val = random.random()

        augmentations = transforms.Compose([
            transforms.RandomHorizontalFlip(p=1),
            # 添加其他需要的增广操作
        ])
        # 设置相同的随机数种子，确保增广一致
        augmented_image1 = augmentations(image1)
        augmented_image2 = augmentations(image2)

        # 保存增广后的图片
        augmented_image1.save(os.path.join(output_path, 'JPEGImages', str(count) + '.png'))
        augmented_image2.save(os.path.join(output_path, 'SegmentationClass', str(count) +
            '.png'))
        count += 1


def aug_images(input_path, output_path, seed=42):
    global count
    # 设置随机数种子


    random.seed(seed)

    # 创建输出路径
    os.makedirs(output_path, exist_ok=True)

    # 获取两组图片的文件名列表
    files_group1 = os.listdir(input_path + '/JPEGImages')
    files_group2 = os.listdir(input_path + '/SegmentationClass')

    # 对每一对文件进行相同的增广操作
    for file1, file2 in zip(files_group1, files_group2):
        image1 = Image.open(os.path.join(input_path + '/JPEGImages', file1))
```

```
        image2 = Image.open(os.path.join(input_path + '/SegmentationClass', file2))

        # 对两组图像应用相同的随机数操作
        random.seed(seed)
        rand_val = random.random()

        augmentations = transforms.Compose([
            transforms.RandomVerticalFlip(p=1),
            # 添加其他需要的增广操作
        ])
        # 设置相同的随机数种子，确保增广一致
        augmented_image1 = augmentations(image1)
        augmented_image2 = augmentations(image2)

        # 保存增广后的图片
        augmented_image1.save(os.path.join(output_path, 'JPEGImages', str(count) + '.png'))
        augmented_image2.save(os.path.join(output_path, 'SegmentationClass', str(count) +
            '.png'))
        count += 1


def gauss_images(input_path, output_path, seed=42):
    global count
    # 设置随机数种子


    random.seed(seed)

    # 创建输出路径
    os.makedirs(output_path, exist_ok=True)

    # 获取两组图片的文件名列表
    files_group1 = os.listdir(input_path + '/JPEGImages')
    files_group2 = os.listdir(input_path + '/SegmentationClass')

    # 对每一对文件进行相同的增广操作
    for file1, file2 in zip(files_group1, files_group2):
        image1 = Image.open(os.path.join(input_path + '/JPEGImages', file1))
        image2 = Image.open(os.path.join(input_path + '/SegmentationClass', file2))

        # 对两组图像应用相同的随机数操作
        random.seed(seed)
        rand_val = random.random()

        augmentations = transforms.Compose([
            transforms.GaussianBlur(kernel_size=11, sigma=5)
            # 添加其他需要的增广操作
```

```
        ])
        # 设置相同的随机数种子，确保增广一致
        augmented_image1 = augmentations(image1)
        augmented_image2 = image2

        # 保存增广后的图片
        augmented_image1.save(os.path.join(output_path, 'JPEGImages', str(count) + '.png'))
        augmented_image2.save(os.path.join(output_path, 'SegmentationClass', str(count) +
            '.png'))
        count += 1

    # 用法示例
input_path = './test'
output_path = './output'

augment_images(input_path, output_path)
aug_images(input_path, output_path)
gauss_images(input_path, output_path)
```

2.1.4 DeepLab v3+ 网络结构源代码

aspp.py

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
from modeling.sync_batchnorm.batchnorm import SynchronizedBatchNorm2d

class _ASPPModule(nn.Module):
    def __init__(self, inplanes, planes, kernel_size, padding, dilation, BatchNorm):
        super(_ASPPModule, self).__init__()
        self.atrous_conv = nn.Conv2d(inplanes, planes, kernel_size=kernel_size,
                                     stride=1, padding=padding, dilation=dilation, bias=False)
        self.bn = BatchNorm(planes)
        self.relu = nn.ReLU()

        self._init_weight()

    def forward(self, x):
        x = self.atrous_conv(x)
        x = self.bn(x)

        return self.relu(x)

    def _init_weight(self):
```

```
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                torch.nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, SynchronizedBatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()

class ASPP(nn.Module):
    def __init__(self, backbone, output_stride, BatchNorm):
        super(ASPP, self).__init__()
        if backbone == 'drn':
            inplanes = 512
        elif backbone == 'mobilenet':
            inplanes = 320
        else:
            inplanes = 2048
        if output_stride == 16:
            dilations = [1, 6, 12, 18]
        elif output_stride == 8:
            dilations = [1, 12, 24, 36]
        else:
            raise NotImplementedError

        self.aspp1 = _ASPPModule(inplanes, 256, 1, padding=0, dilation=dilations[0],
            BatchNorm=BatchNorm)
        self.aspp2 = _ASPPModule(inplanes, 256, 3, padding=dilations[1], dilation=dilations[1],
            BatchNorm=BatchNorm)
        self.aspp3 = _ASPPModule(inplanes, 256, 3, padding=dilations[2], dilation=dilations[2],
            BatchNorm=BatchNorm)
        self.aspp4 = _ASPPModule(inplanes, 256, 3, padding=dilations[3], dilation=dilations[3],
            BatchNorm=BatchNorm)

        self.global_avg_pool = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                             nn.Conv2d(inplanes, 256, 1, stride=1, bias=False),
                                             BatchNorm(256),
                                             nn.ReLU())
        self.conv1 = nn.Conv2d(1280, 256, 1, bias=False)
        self.bn1 = BatchNorm(256)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self._init_weight()

    def forward(self, x):
        x1 = self.aspp1(x)
```

```
        x2 = self.aspp2(x)
        x3 = self.aspp3(x)
        x4 = self.aspp4(x)
        x5 = self.global_avg_pool(x)
        x5 = F.interpolate(x5, size=x4.size()[2:], mode='bilinear', align_corners=True)
        x = torch.cat((x1, x2, x3, x4, x5), dim=1)

        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        return self.dropout(x)

    def _init_weight(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                torch.nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, SynchronizedBatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()


def build_aspp(backbone, output_stride, BatchNorm):
    return ASPP(backbone, output_stride, BatchNorm)
```

decoder.py

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
from modeling.sync_batchnorm.batchnorm import SynchronizedBatchNorm2d

class Decoder(nn.Module):
    def __init__(self, num_classes, backbone, BatchNorm):
        super(Decoder, self).__init__()
        if backbone == 'resnet' or backbone == 'drn':
            low_level_inplanes = 256
        elif backbone == 'xception':
            low_level_inplanes = 128
        elif backbone == 'mobilenet':
            low_level_inplanes = 24
        else:
            raise NotImplementedError
```

```
        self.conv1 = nn.Conv2d(low_level_inplanes, 48, 1, bias=False)
        self.bn1 = BatchNorm(48)
        self.relu = nn.ReLU()
        self.last_conv = nn.Sequential(nn.Conv2d(304, 256, kernel_size=3, stride=1, padding=1,
            bias=False),
                                       BatchNorm(256),
                                       nn.ReLU(),
                                       nn.Dropout(0.5),
                                       nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
                                           bias=False),
                                       BatchNorm(256),
                                       nn.ReLU(),
                                       nn.Dropout(0.1),
                                       nn.Conv2d(256, num_classes, kernel_size=1, stride=1))
        self._init_weight()


    def forward(self, x, low_level_feat):
        low_level_feat = self.conv1(low_level_feat)
        low_level_feat = self.bn1(low_level_feat)
        low_level_feat = self.relu(low_level_feat)

        x = F.interpolate(x, size=low_level_feat.size()[2:], mode='bilinear', align_corners=True)
        x = torch.cat((x, low_level_feat), dim=1)
        x = self.last_conv(x)

        return x

    def _init_weight(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                torch.nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, SynchronizedBatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()

def build_decoder(num_classes, backbone, BatchNorm):
    return Decoder(num_classes, backbone, BatchNorm)
```

deeplab.py

```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F
from modeling.sync_batchnorm.batchnorm import SynchronizedBatchNorm2d
from modeling.aspp import build_aspp
from modeling.decoder import build_decoder
from modeling.backbone import build_backbone

class DeepLab(nn.Module):
    def __init__(self, backbone='resnet', output_stride=16, num_classes=21,
                 sync_bn=True, freeze_bn=False):
        super(DeepLab, self).__init__()
        if backbone == 'drn':
            output_stride = 8

        if sync_bn == True:
            BatchNorm = SynchronizedBatchNorm2d
        else:
            BatchNorm = nn.BatchNorm2d

        self.backbone = build_backbone(backbone, output_stride, BatchNorm)
        self.aspp = build_aspp(backbone, output_stride, BatchNorm)
        self.decoder = build_decoder(num_classes, backbone, BatchNorm)

        self.freeze_bn = freeze_bn

    def forward(self, input):
        x, low_level_feat = self.backbone(input)
        x = self.aspp(x)
        x = self.decoder(x, low_level_feat)
        x = F.interpolate(x, size=input.size()[2:], mode='bilinear', align_corners=True)

        return x

    def freeze_bn(self):
        for m in self.modules():
            if isinstance(m, SynchronizedBatchNorm2d):
                m.eval()
            elif isinstance(m, nn.BatchNorm2d):
                m.eval()

    def get_1x_lr_params(self):
        modules = [self.backbone]
        for i in range(len(modules)):
            for m in modules[i].named_modules():
                if self.freeze_bn:
                    if isinstance(m[1], nn.Conv2d):
                        for p in m[1].parameters():
                            if p.requires_grad:
```

```
                            yield p
            else:
                if isinstance(m[1], nn.Conv2d) or isinstance(m[1], SynchronizedBatchNorm2d) \
                        or isinstance(m[1], nn.BatchNorm2d):
                    for p in m[1].parameters():
                        if p.requires_grad:
                            yield p

    def get_10x_lr_params(self):
        modules = [self.aspp, self.decoder]
        for i in range(len(modules)):
            for m in modules[i].named_modules():
                if self.freeze_bn:
                    if isinstance(m[1], nn.Conv2d):
                        for p in m[1].parameters():
                            if p.requires_grad:
                                yield p
                else:
                    if isinstance(m[1], nn.Conv2d) or isinstance(m[1], SynchronizedBatchNorm2d) \
                            or isinstance(m[1], nn.BatchNorm2d):
                        for p in m[1].parameters():
                            if p.requires_grad:
                                yield p

if __name__ == "__main__":
    model = DeepLab(backbone='mobilenet', output_stride=16)
    model.eval()
    input = torch.rand(1, 3, 513, 513)
    output = model(input)
    print(output.size())
```

2.1.5 UNet 源代码

train.py

```
import argparse
import csv
import logging
import os
from pathlib import Path

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import wandb
```

```python
from torch import optim
from torch.utils.data import DataLoader, random_split
from tqdm import tqdm

from evaluate import evaluate
from unet import UNet
from utils.data_loading import BasicDataset, CarvanaDataset
from utils.dice_score import dice_loss

dir_img = Path('./data/subset1/folder1/')
dir_mask = Path('./data/subset1/folder2/')
dir_checkpoint = Path('./checkpoints/')


def calculate_iou(confusion_matrix):
    intersection = confusion_matrix.diagonal()
    union = confusion_matrix.sum(axis=1) + confusion_matrix.sum(axis=0) - intersection
    iou = intersection / union
    return iou


def calculate_miou(confusion_matrix):
    return np.mean(calculate_iou(confusion_matrix))


def calculate_fwiou(confusion_matrix):
    frequency = confusion_matrix.sum(axis=1) / confusion_matrix.sum()
    fwiou = np.sum(frequency * calculate_iou(confusion_matrix))
    return fwiou



def train_model(
        model,
        device,
        epochs: int = 5,
        batch_size: int = 1,
        learning_rate: float = 1e-5,
        val_percent: float = 0.1,
        save_checkpoint: bool = True,
        img_scale: float = 0.5,
        amp: bool = False,
        weight_decay: float = 1e-8,
        momentum: float = 0.999,
        gradient_clipping: float = 1.0,
):
    # 1. Create dataset
```

```
    try:
        dataset = CarvanaDataset(dir_img, dir_mask, img_scale)
    except (AssertionError, RuntimeError, IndexError):
        dataset = BasicDataset(dir_img, dir_mask, img_scale)

    # 2. Split into train / validation partitions
    n_val = int(len(dataset) * val_percent)
    n_train = len(dataset) - n_val
    train_set, val_set = random_split(dataset, [n_train, n_val],
         generator=torch.Generator().manual_seed(0))

    # 3. Create data loaders
    loader_args = dict(batch_size=batch_size, num_workers=0, pin_memory=False)
    train_loader = DataLoader(train_set, shuffle=True, **loader_args, drop_last=True)
    val_loader = DataLoader(val_set, shuffle=False, drop_last=True, **loader_args)

    # (Initialize logging)
    experiment = wandb.init(project='U-Net', resume='allow', anonymous='must')
    experiment.config.update(
        dict(epochs=epochs, batch_size=batch_size, learning_rate=learning_rate,
             val_percent=val_percent, save_checkpoint=save_checkpoint, img_scale=img_scale,
                 amp=amp)
    )

    logging.info(f'''Starting training:
        Epochs:          {epochs}
        Batch size:      {batch_size}
        Learning rate:   {learning_rate}
        Training size:   {n_train}
        Validation size: {n_val}
        Checkpoints:     {save_checkpoint}
        Device:          {device.type}
        Images scaling:  {img_scale}
        Mixed Precision: {amp}
    ''')

    # 4. Set up the optimizer, the loss, the learning rate scheduler and the loss scaling for
         AMP
    optimizer = optim.RMSprop(model.parameters(),
                              lr=learning_rate, weight_decay=weight_decay, momentum=momentum,
                                  foreach=True)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', patience=5) # goal:
         maximize Dice score
    grad_scaler = torch.cuda.amp.GradScaler(enabled=amp)
    criterion = nn.CrossEntropyLoss() if model.n_classes > 1 else nn.BCEWithLogitsLoss()
    global_step = 0

```

```
    # ... (前面的代码保持不变)

    # 5. Begin training
    validate_every = 1 # Validate every epoch, you can adjust this value
    for epoch in range(1, epochs + 1):
        model.train()
        epoch_loss = 0
        confusion_matrix = np.zeros((model.n_classes, model.n_classes), dtype=int)

        with tqdm(total=n_train, desc=f'Epoch {epoch}/{epochs}', unit='img') as pbar:
            for batch in train_loader:
                images, true_masks = batch['image'], batch['mask']

                assert images.shape[1] == model.n_channels, \
                    f'Network has been defined with {model.n_channels} input channels, ' \
                    f'but loaded images have {images.shape[1]} channels. Please check that ' \
                    'the images are loaded correctly.'

                images = images.to(device=device, dtype=torch.float32,
                     memory_format=torch.channels_last)
                true_masks = true_masks.to(device=device, dtype=torch.long)

                with torch.cuda.amp.autocast(enabled=True):
                    masks_pred = model(images)
                    if model.n_classes == 1:
                        loss = criterion(masks_pred.squeeze(1), true_masks.float())
                        loss += dice_loss(F.sigmoid(masks_pred.squeeze(1)), true_masks.float(),
                             multiclass=False)
                    else:
                        loss = criterion(masks_pred, true_masks)
                        loss += dice_loss(
                            F.softmax(masks_pred, dim=1).float(),
                            F.one_hot(true_masks, model.n_classes).permute(0, 3, 1, 2).float(),
                            multiclass=True
                        )

                optimizer.zero_grad(set_to_none=True)
                grad_scaler.scale(loss).backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), gradient_clipping)
                grad_scaler.step(optimizer)
                grad_scaler.update()

                pbar.update(images.shape[0])
                global_step += 1
                epoch_loss += loss.item()

        # Evaluation round
```

```
    if epoch % validate_every == 0:
        model.eval()
        confusion_matrix = np.zeros((model.n_classes, model.n_classes), dtype=int)

        for batch in val_loader:
            with torch.no_grad():
                images, true_masks = batch['image'], batch['mask']
                images = images.to(device=device, dtype=torch.float32,
                    memory_format=torch.channels_last)
                true_masks = true_masks.to(device=device, dtype=torch.long)

                masks_pred = model(images)
                predicted_masks = masks_pred.argmax(dim=1)

                confusion_matrix += np.bincount((model.n_classes * true_masks.flatten() +
                    predicted_masks.flatten()).cpu().numpy(),
                    minlength=model.n_classes**2).reshape(model.n_classes, model.n_classes)



        miou = calculate_miou(confusion_matrix)
        fwiou = calculate_fwiou(confusion_matrix)

        acc = (confusion_matrix.diagonal().sum() / confusion_matrix.sum())
        acc_class = confusion_matrix.diagonal() / confusion_matrix.sum(axis=1)

        logging.info('Epoch: {}, Acc: {:.4f}, Acc_class: {}, mIoU: {:.4f}, FWIoU:
            {:.4f}'.format(epoch, acc,
                                                                                acc_class,
                                                                                miou,
                                                                                fwiou))
        epoch_data = (epoch, acc, acc_class, miou, fwiou)


    if save_checkpoint:
        Path(dir_checkpoint).mkdir(parents=True, exist_ok=True)
        state_dict = model.state_dict()
        state_dict['mask_values'] = dataset.mask_values
        torch.save(state_dict, str(dir_checkpoint / 'checkpoint_epoch{}.pth'.format(epoch)))
        logging.info(f'Checkpoint {epoch} saved!')
        with open('output.csv', 'a', newline='') as csvfile:
            # 创建CSV写入对象
            csv_writer = csv.writer(csvfile)
            # 写入数据
            csv_writer.writerow(epoch_data)


```

```python
def get_args():
    parser = argparse.ArgumentParser(description='Train the UNet on images and target masks')
    parser.add_argument('--epochs', '-e', metavar='E', type=int, default=50, help='Number of epochs')
    parser.add_argument('--batch-size', '-b', dest='batch_size', metavar='B', type=int, default=8, help='Batch size')
    parser.add_argument('--learning-rate', '-l', metavar='LR', type=float, default=1e-5,
                        help='Learning rate', dest='lr')
    parser.add_argument('--load', '-f', type=str, default=False, help='Load model from a .pth file')
    parser.add_argument('--scale', '-s', type=float, default=0.5, help='Downscaling factor of the images')
    parser.add_argument('--validation', '-v', dest='val', type=float, default=10.0,
                        help='Percent of the data that is used as validation (0-100)')
    parser.add_argument('--amp', action='store_true', default=False, help='Use mixed precision')
    parser.add_argument('--bilinear', action='store_true', default=False, help='Use bilinear upsampling')
    parser.add_argument('--classes', '-c', type=int, default=2, help='Number of classes')

    return parser.parse_args()


if __name__ == '__main__':
    args = get_args()

    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    logging.info(f'Using device {device}')

    # Change here to adapt to your data
    # n_channels=3 for RGB images
    # n_classes is the number of probabilities you want to get per pixel
    model = UNet(n_channels=3, n_classes=args.classes, bilinear=args.bilinear)
    model = model.to(memory_format=torch.channels_last)

    logging.info(f'Network:\n'
                 f'\t{model.n_channels} input channels\n'
                 f'\t{model.n_classes} output channels (classes)\n'
                 f'\t{"Bilinear" if model.bilinear else "Transposed conv"} upscaling')

    if args.load:
        state_dict = torch.load(args.load, map_location=device)
        del state_dict['mask_values']
        model.load_state_dict(state_dict)
        logging.info(f'Model loaded from {args.load}')

    model.to(device=device)
```

```
    try:
        train_model(
            model=model,
            epochs=args.epochs,
            batch_size=args.batch_size,
            learning_rate=args.lr,
            device=device,
            img_scale=args.scale,
            val_percent=args.val / 100,
            amp=args.amp
        )
    except torch.cuda.OutOfMemoryError:
        logging.error('Detected OutOfMemoryError! '
                      'Enabling checkpointing to reduce memory usage, but this slows down
                          training. '
                      'Consider enabling AMP (--amp) for fast and memory efficient training')
        torch.cuda.empty_cache()
        model.use_checkpointing()
        train_model(
            model=model,
            epochs=args.epochs,
            batch_size=args.batch_size,
            learning_rate=args.lr,
            device=device,
            img_scale=args.scale,
            val_percent=args.val / 100,
            amp=args.amp
        )
```

unet_model.py

```
""" Full assembly of the parts to form the complete network """

from .unet_parts import *


class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=False):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = (DoubleConv(n_channels, 64))
        self.down1 = (Down(64, 128))
        self.down2 = (Down(128, 256))
        self.down3 = (Down(256, 512))
```

```
        factor = 2 if bilinear else 1
        self.down4 = (Down(512, 1024 // factor))
        self.up1 = (Up(1024, 512 // factor, bilinear))
        self.up2 = (Up(512, 256 // factor, bilinear))
        self.up3 = (Up(256, 128 // factor, bilinear))
        self.up4 = (Up(128, 64, bilinear))
        self.outc = (OutConv(64, n_classes))

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits

    def use_checkpointing(self):
        self.inc = torch.utils.checkpoint(self.inc)
        self.down1 = torch.utils.checkpoint(self.down1)
        self.down2 = torch.utils.checkpoint(self.down2)
        self.down3 = torch.utils.checkpoint(self.down3)
        self.down4 = torch.utils.checkpoint(self.down4)
        self.up1 = torch.utils.checkpoint(self.up1)
        self.up2 = torch.utils.checkpoint(self.up2)
        self.up3 = torch.utils.checkpoint(self.up3)
        self.up4 = torch.utils.checkpoint(self.up4)
        self.outc = torch.utils.checkpoint(self.outc)
```

unet_parts.py

```
""" Parts of the U-Net model """

import torch
import torch.nn as nn
import torch.nn.functional as F


class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
```

```
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)


class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)


class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
```

```
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        #
            https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854e247a9c615f1
        #
            https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)


class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)
```

2.1.6 SegNet 源代码

train.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2023/12/15 下午2:11
# @Author : SiHang Xu
import csv
import logging
import time

import cv2
import numpy as np
from torch.utils import data as Data
from MyDataset import MyDataset
from SegNet import *


def calculate_iou(confusion_matrix):
    intersection = confusion_matrix.diagonal()
    union = confusion_matrix.sum(axis=1) + confusion_matrix.sum(axis=0) - intersection
    iou = intersection / union
    return iou

```

```
22
23  def calculate_miou(confusion_matrix):
24      return np.mean(calculate_iou(confusion_matrix))
25
26
27  def calculate_fwiou(confusion_matrix):
28      frequency = confusion_matrix.sum(axis=1) / confusion_matrix.sum()
29      fwiou = np.sum(frequency * calculate_iou(confusion_matrix))
30      return fwiou
31
32  epoch_data = []
33  def train_and_validate(SegNet, train_data, val_data, BATCH_SIZE, LR, MOMENTUM, EPOCH,
        CATE_WEIGHT, WEIGHTS, PRE_TRAINING):
34      SegNet = SegNet(3, 2).cuda()
35      SegNet.load_weights(PRE_TRAINING)
36
37      train_loader = torch.utils.data.DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True,
            drop_last=True, num_workers=12)
38      val_loader = torch.utils.data.DataLoader(val_data, batch_size=BATCH_SIZE, shuffle=False,
            drop_last=True, num_workers=12)
39
40      optimizer = torch.optim.SGD(SegNet.parameters(), lr=LR, momentum=MOMENTUM)
41
42      loss_func =
            nn.CrossEntropyLoss(weight=torch.from_numpy(np.array(CATE_WEIGHT)).float()).cuda()
43
44      for epoch in range(EPOCH):
45          # Training
46          SegNet.train()
47          for step, (b_x, b_y) in enumerate(train_loader):
48              b_x = b_x.cuda()
49              b_y = b_y.cuda()
50              b_y = b_y.view(BATCH_SIZE, 224, 224)
51              output = SegNet(b_x)
52              loss = loss_func(output, b_y.long())
53              loss = loss.cuda()
54              optimizer.zero_grad()
55              loss.backward()
56              optimizer.step()
57
58          print(f"Epoch: {epoch} || Training Loss: {loss:.4f}")
59
60          # Validation
61          SegNet.eval()
62          confusion_matrix = np.zeros((2, 2), dtype=int)
63          with torch.no_grad():
64              val_loss = 0.0
```

```
65
66          for step, (val_x, val_y) in enumerate(val_loader):
67              val_x = val_x.cuda()
68              val_y = val_y.cuda()
69              val_y = val_y.view(BATCH_SIZE, 224, 224)
70              val_output = SegNet(val_x)
71              val_loss += loss_func(val_output, val_y.long()).item()
72
73              val_output = torch.squeeze(val_output)
74
75              predict = val_output.argmax(dim=1)
76              target = val_y
77              confusion_matrix += np.bincount(
78                  (2 * target.flatten() + predict.flatten()).cpu().numpy(),
79                  minlength=2 ** 2).reshape(2, 2)
80
81          miou = calculate_miou(confusion_matrix)
82          fwiou = calculate_fwiou(confusion_matrix)
83
84          acc = (confusion_matrix.diagonal().sum() / confusion_matrix.sum())
85          acc_class = confusion_matrix.diagonal() / confusion_matrix.sum(axis=1)
86          print('Epoch: {}, Acc: {:.4f}, Acc_class: {}, mIoU: {:.4f}, FWIoU:
                {:.4f}'.format(epoch, acc,
87                                                                          acc_class,
88                                                                          miou,
89                                                                          fwiou))
90          epoch_data.append((epoch, acc, acc_class, miou, fwiou))
91
92      torch.save(SegNet.state_dict(), WEIGHTS + "SegNet_weights" + str(time.time()) + ".pth")
93      with open('output.csv', 'w', newline='') as csvfile:
94          # 创建CSV写入对象
95          csv_writer = csv.writer(csvfile)
96
97          # 写入表头
98          csv_writer.writerow(['Epoch', 'Acc', 'Acc_class', 'mIoU', 'FWIoU'])
99
100         # 写入数据
101         csv_writer.writerows(epoch_data)
102
103 # 示例调用
104 BATCH_SIZE = 8
105 LR = 0.001
106 MOMENTUM = 0.9
107 EPOCH = 50
108 CATE_WEIGHT = [1, 1] # 根据实际情况调整
109 WEIGHTS = "weights/"
110 PRE_TRAINING = "vgg16_bn-6c64b313.pth"
```

```
train_data = MyDataset(txt_path='train.txt')
val_data = MyDataset(txt_path='trainval.txt')

train_and_validate(SegNet, train_data, val_data, BATCH_SIZE, LR, MOMENTUM, EPOCH, CATE_WEIGHT,
    WEIGHTS, PRE_TRAINING)
```

MyDataset.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2023/12/15 下午2:09
# @Author : SiHang Xu
import numpy as np
import torch
from PIL import Image
from torch.utils.data import Dataset
import cv2

class MyDataset(Dataset):
    def __init__(self, txt_path):
        super(MyDataset, self).__init__()

        with open(txt_path, "r") as file:
            image_label = [line.strip() for line in file]

        self.image_label = image_label

    def __getitem__(self, item):
        root = self.image_label[item]
        image_path = f'./400dataset/JPEGImages/{root}.png'
        label_path = f'./400dataset/SegmentationClass/{root}.png'

        image = cv2.imread(image_path)
        image = cv2.resize(image, (224, 224))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert to RGB
        image = image / 255.0 # Normalize input
        image = torch.tensor(image, dtype=torch.float32)
        image = image.permute(2, 0, 1)

        label = Image.open(label_path)
        label = label.resize((224, 224), Image.NEAREST) # 使用最近邻插值
        label = np.array(label)
        label = torch.tensor(label, dtype=torch.long)
        # label = cv2.imread(label_path, cv2.IMREAD_GRAYSCALE)
        # label = cv2.resize(label, (224, 224))
        # label = torch.tensor(label, dtype=torch.long) # Assuming segmentation labels are
            integers
```

```
        return image, label

    def __len__(self):
        return len(self.image_label)
```

SegNet.pt

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2023/12/15 下午2:06
# @Author : SiHang Xu
import torch
from torch import nn
import torch.nn.functional as F


class Encoder(nn.Module):
    def __init__(self, input_channels):
        super(Encoder, self).__init__()

        self.enco1 = nn.Sequential(
            nn.Conv2d(input_channels, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.enco2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU()
        )
        self.enco3 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU()
```

```
        )
        self.enco4 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU()
        )
        self.enco5 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU()
        )

    def forward(self, x):
        id = []

        x = self.enco1(x)
        x, id1 = F.max_pool2d(x, kernel_size=2, stride=2, return_indices=True) # 保留最大值的位置
        id.append(id1)
        x = self.enco2(x)
        x, id2 = F.max_pool2d(x, kernel_size=2, stride=2, return_indices=True)
        id.append(id2)
        x = self.enco3(x)
        x, id3 = F.max_pool2d(x, kernel_size=2, stride=2, return_indices=True)
        id.append(id3)
        x = self.enco4(x)
        x, id4 = F.max_pool2d(x, kernel_size=2, stride=2, return_indices=True)
        id.append(id4)
        x = self.enco5(x)
        x, id5 = F.max_pool2d(x, kernel_size=2, stride=2, return_indices=True)
        id.append(id5)

        return x, id


class SegNet(nn.Module):
```

```
    def __init__(self, input_channels, output_channels):
        super(SegNet, self).__init__()

        self.weights_new = self.state_dict()
        self.encoder = Encoder(input_channels)

        self.deco1 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU()
        )
        self.deco2 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU()
        )
        self.deco3 = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU()
        )
        self.deco4 = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
```

```
        self.deco5 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, output_channels, kernel_size=3, stride=1, padding=1),
        )

    def forward(self, x):
        x, id = self.encoder(x)

        x = F.max_unpool2d(x, id[4], kernel_size=2, stride=2)
        x = self.deco1(x)
        x = F.max_unpool2d(x, id[3], kernel_size=2, stride=2)
        x = self.deco2(x)
        x = F.max_unpool2d(x, id[2], kernel_size=2, stride=2)
        x = self.deco3(x)
        x = F.max_unpool2d(x, id[1], kernel_size=2, stride=2)
        x = self.deco4(x)
        x = F.max_unpool2d(x, id[0], kernel_size=2, stride=2)
        x = self.deco5(x)

        return x

    def load_weights(self, weights_path):
        weights = torch.load(weights_path)
        del weights["classifier.0.weight"]
        del weights["classifier.0.bias"]
        del weights["classifier.3.weight"]
        del weights["classifier.3.bias"]
        del weights["classifier.6.weight"]
        del weights["classifier.6.bias"]

        names = []
        for key, value in self.encoder.state_dict().items():
            if "num_batches_tracked" in key:
                continue
            names.append(key)

        for name, dict in zip(names, weights.items()):
            self.weights_new[name] = dict[1]

        self.encoder.load_state_dict(self.weights_new)
```

2.1.7 边缘检测与目标检测

edge.py

```python
import cv2
import os
import numpy as np

def laplacian_edge_detection(image_path, alpha=3, beta=30):
    # Read the image
    original_image = cv2.imread("images/" + image_path, cv2.IMREAD_GRAYSCALE)
    original_images = cv2.imread("images/" + image_path)

    # Apply GaussianBlur to reduce noise and help with edge detection
    blurred_image = cv2.GaussianBlur(original_image, (5, 5), 0)

    # Apply Laplacian edge detection
    laplacian = cv2.Laplacian(blurred_image, cv2.CV_64F)

    # Convert the Laplacian result to uint8 (8-bit) for display
    laplacian = np.uint8(np.absolute(laplacian))

    # Adjust brightness and contrast
    laplacian_adjusted = cv2.convertScaleAbs(laplacian, alpha=alpha, beta=beta)

    # Display the original and adjusted Laplacian edge-detected images
    cv2.imwrite(os.path.join('output', image_path), laplacian_adjusted)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Replace 'path/to/your/image.jpg' with the actual path to your image file
for image in os.listdir("images"):
    laplacian_edge_detection(image)
```

detect.py

```python
import argparse
import os
import platform
import shutil
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
import torchvision.utils
```

```python
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import (
    check_img_size, non_max_suppression, apply_classifier, scale_coords,
    xyxy2xywh, plot_one_box, strip_optimizer, set_logging)
from utils.torch_utils import select_device, load_classifier, time_synchronized


save_csv = True
csv_f = open("results.csv","w")


def detect(save_img=False):
    out, source, weights, view_img, save_txt, imgsz = \
        opt.output, opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size
    webcam = source.isnumeric() or source.startswith(('rtsp://', 'rtmp://', 'http://')) or
         source.endswith('.txt')

    # Initialize
    set_logging()
    device = select_device(opt.device)

    if os.path.exists(out):
        shutil.rmtree(out) # delete output folder
    os.makedirs(out) # make new output folder
    half = device.type != 'cpu' # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device) # load FP32 model
    imgsz = check_img_size(imgsz, s=model.stride.max()) # check img_size
    if half:
        model.half() # to FP16

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2) # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model'])
            # load weights
        modelc.to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = True
```

```
        cudnn.benchmark = True # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz)
    else:
        save_img = True
        dataset = LoadImages(source, img_size=imgsz)

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in range(len(names))]

    # Run inference
    t0 = time.time()
    img = torch.zeros((1, 3, imgsz, imgsz), device=device) # init img
    _ = model(img.half() if half else img) if device.type != 'cpu' else None # run once
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float() # uint8 to fp16/32
        img /= 255.0 # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Inference
        t1 = time_synchronized()
        pred = model(img, augment=opt.augment)[0]

        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes,
            agnostic=opt.agnostic_nms)
        t2 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)

        # Process detections
        for i, det in enumerate(pred): # detections per image
            if save_csv:
                csv_f.write(os.path.basename(path)+",")
            if webcam: # batch_size >= 1
                p, s, im0 = path[i], '%g: ' % i, im0s[i].copy()
            else:
                p, s, im0 = path, '', im0s

            save_path = str(Path(out) / Path(p).name)
            txt_path = str(Path(out) / Path(p).stem) + ('_%g' % dataset.frame if dataset.mode ==
                'video' else '')
            s += '%gx%g ' % img.shape[2:] # print string
```

```
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh
            if det is not None and len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum() # detections per class
                    s += '%g %ss, ' % (n, names[int(c)]) # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt: # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist() #
                            normalized xywh
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * 5 + '\n') % (cls, *xywh)) # label format

                    if save_img or view_img: # Add bbox to image
                        label = '%s %.2f' % (names[int(cls)], conf)
                        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)],
                            line_thickness=3)

                    if save_csv:
                        csv_f.write("{} {} {} {} {}
                            ".format(str(int(cls.detach().cpu().numpy())+1),
                            str(int(xyxy[0].detach().cpu().numpy())),
                            str(int(xyxy[1].detach().cpu().numpy())),
                            str(int(xyxy[2].detach().cpu().numpy())),
                            str(int(xyxy[3].detach().cpu().numpy()))))
            csv_f.write("\n")

            # Print time (inference + NMS)
            print('%sDone. (%.3fs)' % (s, t2 - t1))

            # Stream results
            if view_img:
                cv2.imshow(p, im0)
                if cv2.waitKey(1) == ord('q'): # q to quit
                    raise StopIteration

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'images':
                    cv2.imwrite(save_path, im0)
                else:
                    if vid_path != save_path: # new video
```

```
                    vid_path = save_path
                    if isinstance(vid_writer, cv2.VideoWriter):
                        vid_writer.release() # release previous video writer

                    fourcc = 'mp4v' # output video codec
                    fps = vid_cap.get(cv2.CAP_PROP_FPS)
                    w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                    h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                    vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*fourcc),
                        fps, (w, h))
                vid_writer.write(im0)

    if save_txt or save_img:
        print('Results saved to %s' % Path(out))
        if platform.system() == 'Darwin' and not opt.update: # MacOS
            os.system('open ' + save_path)

    print('Done. (%.3fs)' % (time.time() - t0))


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str,
        default='weights/IMSC/last_120_640_32_aug2.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='inference/images', help='source') #
        file/folder, 0 for webcam
    parser.add_argument('--output', type=str, default='inference/output', help='output folder')
        # output folder
    parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.4, help='object confidence
        threshold')
    parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or
        --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    opt = parser.parse_args()
    print(opt)

    with torch.no_grad():
        if opt.update: # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
                detect()
```

```
            strip_optimizer(opt.weights)
        else:
            detect()
    csv_f.close()
```

yolo.py

```
import argparse
import logging
import math
from copy import deepcopy
from pathlib import Path

import torch
import torch.nn as nn

from models.common import Conv, Bottleneck, SPP, DWConv, Focus, BottleneckCSP, Concat, NMS
from models.experimental import MixConv2d, CrossConv, C3
from utils.general import check_anchor_order, make_divisible, check_file, set_logging
from utils.torch_utils import (
    time_synchronized, fuse_conv_and_bn, model_info, scale_img, initialize_weights,
        select_device)

logger = logging.getLogger(__name__)


class Detect(nn.Module):
    stride = None # strides computed during build
    export = False # onnx export

    def __init__(self, nc=80, anchors=(), ch=()): # detection layer
        super(Detect, self).__init__()
        self.nc = nc # number of classes
        self.no = nc + 5 # number of outputs per anchor
        self.nl = len(anchors) # number of detection layers
        self.na = len(anchors[0]) // 2 # number of anchors
        self.grid = [torch.zeros(1)] * self.nl # init grid
        a = torch.tensor(anchors).float().view(self.nl, -1, 2)
        self.register_buffer('anchors', a) # shape(nl,na,2)
        self.register_buffer('anchor_grid', a.clone().view(self.nl, 1, -1, 1, 1, 2)) #
            shape(nl,1,na,1,1,2)
        self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch) # output conv

    def forward(self, x):
        # x = x.copy() # for profiling
        z = [] # inference output
        self.training |= self.export
```

```
        for i in range(self.nl):
            x[i] = self.m[i](x[i]) # conv
            bs, _, ny, nx = x[i].shape # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training: # inference
                if self.grid[i].shape[2:4] != x[i].shape[2:4]:
                    self.grid[i] = self._make_grid(nx, ny).to(x[i].device)

                y = x[i].sigmoid()
                y[..., 0:2] = (y[..., 0:2] * 2. - 0.5 + self.grid[i].to(x[i].device)) *
                    self.stride[i] # xy
                y[..., 2:4] = (y[..., 2:4] * 2) ** 2 * self.anchor_grid[i] # wh
                z.append(y.view(bs, -1, self.no))

        return x if self.training else (torch.cat(z, 1), x)

    @staticmethod
    def _make_grid(nx=20, ny=20):
        yv, xv = torch.meshgrid([torch.arange(ny), torch.arange(nx)])
        return torch.stack((xv, yv), 2).view((1, 1, ny, nx, 2)).float()


class Model(nn.Module):
    def __init__(self, cfg='yolov5s.yaml', ch=3, nc=None): # model, input channels, number of
        classes
        super(Model, self).__init__()
        if isinstance(cfg, dict):
            self.yaml = cfg # model dict
        else: # is *.yaml
            import yaml # for torch hub
            self.yaml_file = Path(cfg).name
            with open(cfg) as f:
                self.yaml = yaml.load(f, Loader=yaml.FullLoader) # model dict

        # Define model
        if nc and nc != self.yaml['nc']:
            print('Overriding model.yaml nc=%g with nc=%g' % (self.yaml['nc'], nc))
            self.yaml['nc'] = nc # override yaml value
        self.model, self.save = parse_model(deepcopy(self.yaml), ch=[ch]) # model, savelist,
            ch_out
        # print([x.shape for x in self.forward(torch.zeros(1, ch, 64, 64))])

        # Build strides, anchors
        m = self.model[-1] # Detect()
        if isinstance(m, Detect):
            s = 128 # 2x min stride
```

```
            m.stride = torch.tensor([s / x.shape[-2] for x in self.forward(torch.zeros(1, ch, s, s))]) # forward
            m.anchors /= m.stride.view(-1, 1, 1)
            check_anchor_order(m)
            self.stride = m.stride
            self._initialize_biases() # only run once
            # print('Strides: %s' % m.stride.tolist())

        # Init weights, biases
        initialize_weights(self)
        self.info()
        print('')

    def forward(self, x, augment=False, profile=False):
        if augment:
            img_size = x.shape[-2:] # height, width
            s = [1, 0.83, 0.67] # scales
            f = [None, 3, None] # flips (2-ud, 3-lr)
            y = [] # outputs
            for si, fi in zip(s, f):
                xi = scale_img(x.flip(fi) if fi else x, si)
                yi = self.forward_once(xi)[0] # forward
                # cv2.imwrite('img%g.jpg' % s, 255 * xi[0].numpy().transpose((1, 2, 0))[:, :, ::-1]) # save
                yi[..., :4] /= si # de-scale
                if fi == 2:
                    yi[..., 1] = img_size[0] - yi[..., 1] # de-flip ud
                elif fi == 3:
                    yi[..., 0] = img_size[1] - yi[..., 0] # de-flip lr
                y.append(yi)
            return torch.cat(y, 1), None # augmented inference, train
        else:
            return self.forward_once(x, profile) # single-scale inference, train

    def forward_once(self, x, profile=False):
        y, dt = [], [] # outputs
        for m in self.model:
            if m.f != -1: # if not from previous layer
                x = y[m.f] if isinstance(m.f, int) else [x if j == -1 else y[j] for j in m.f] # from earlier layers

            if profile:
                try:
                    import thop
                    o = thop.profile(m, inputs=(x,), verbose=False)[0] / 1E9 * 2 # FLOPS
                except:
                    o = 0
```

```
            t = time_synchronized()
            for _ in range(10):
                _ = m(x)
            dt.append((time_synchronized() - t) * 100)
            print('%10.1f%10.0f%10.1fms %-40s' % (o, m.np, dt[-1], m.type))

        x = m(x) # run
        y.append(x if m.i in self.save else None) # save output

    if profile:
        print('%.1fms total' % sum(dt))
    return x

def _initialize_biases(self, cf=None): # initialize biases into Detect(), cf is class frequency
    # cf = torch.bincount(torch.tensor(np.concatenate(dataset.labels, 0)[:, 0]).long(), minlength=nc) + 1.
    m = self.model[-1] # Detect() module
    for mi, s in zip(m.m, m.stride): # from
        b = mi.bias.view(m.na, -1) # conv.bias(255) to (3,85)
        b[:, 4] += math.log(8 / (640 / s) ** 2) # obj (8 objects per 640 image)
        b[:, 5:] += math.log(0.6 / (m.nc - 0.99)) if cf is None else torch.log(cf / cf.sum()) # cls
        mi.bias = torch.nn.Parameter(b.view(-1), requires_grad=True)

def _print_biases(self):
    m = self.model[-1] # Detect() module
    for mi in m.m: # from
        b = mi.bias.detach().view(m.na, -1).T # conv.bias(255) to (3,85)
        print(('%6g Conv2d.bias:' + '%10.3g' * 6) % (mi.weight.shape[1], *b[:5].mean(1).tolist(), b[5:].mean()))

def fuse(self): # fuse model Conv2d() + BatchNorm2d() layers
    print('Fusing layers... ')
    for m in self.model.modules():
        if type(m) is Conv and hasattr(Conv, 'bn'):
            m._non_persistent_buffers_set = set() # pytorch 1.6.0 compatability
            m.conv = fuse_conv_and_bn(m.conv, m.bn) # update conv
            delattr(m, 'bn') # remove batchnorm
            m.forward = m.fuseforward # update forward
    self.info()
    return self

def add_nms(self): # fuse model Conv2d() + BatchNorm2d() layers
    if type(self.model[-1]) is not NMS: # if missing NMS
        print('Adding NMS module... ')
        m = NMS() # module
```

```
            m.f = -1 # from
            m.i = self.model[-1].i + 1 # index
            self.model.add_module(name='%s' % m.i, module=m) # add
        return self

    def info(self, verbose=False): # print model information
        model_info(self, verbose)


def parse_model(d, ch): # model_dict, input_channels(3)
    logger.info('\n%3s%18s%3s%10s %-40s%-30s' % ('', 'from', 'n', 'params', 'module',
        'arguments'))
    anchors, nc, gd, gw = d['anchors'], d['nc'], d['depth_multiple'], d['width_multiple']
    na = (len(anchors[0]) // 2) if isinstance(anchors, list) else anchors # number of anchors
    no = na * (nc + 5) # number of outputs = anchors * (classes + 5)

    layers, save, c2 = [], [], ch[-1] # layers, savelist, ch out
    for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']): # from, number, module, args
        m = eval(m) if isinstance(m, str) else m # eval strings
        for j, a in enumerate(args):
            try:
                args[j] = eval(a) if isinstance(a, str) else a # eval strings
            except:
                pass

        n = max(round(n * gd), 1) if n > 1 else n # depth gain
        if m in [Conv, Bottleneck, SPP, DWConv, MixConv2d, Focus, CrossConv, BottleneckCSP, C3]:
            c1, c2 = ch[f], args[0]

            # Normal
            # if i > 0 and args[0] != no: # channel expansion factor
            #     ex = 1.75 # exponential (default 2.0)
            #     e = math.log(c2 / ch[1]) / math.log(2)
            #     c2 = int(ch[1] * ex ** e)
            # if m != Focus:

            c2 = make_divisible(c2 * gw, 8) if c2 != no else c2

            # Experimental
            # if i > 0 and args[0] != no: # channel expansion factor
            #     ex = 1 + gw # exponential (default 2.0)
            #     ch1 = 32 # ch[1]
            #     e = math.log(c2 / ch1) / math.log(2) # level 1-n
            #     c2 = int(ch1 * ex ** e)
            # if m != Focus:
            #     c2 = make_divisible(c2, 8) if c2 != no else c2

```

```
            args = [c1, c2, *args[1:]]
            if m in [BottleneckCSP, C3]:
                args.insert(2, n)
                n = 1
        elif m is nn.BatchNorm2d:
            args = [ch[f]]
        elif m is Concat:
            c2 = sum([ch[-1 if x == -1 else x + 1] for x in f])
        elif m is Detect:
            args.append([ch[x + 1] for x in f])
            if isinstance(args[1], int): # number of anchors
                args[1] = [list(range(args[1] * 2))] * len(f)
        else:
            c2 = ch[f]

        m_ = nn.Sequential(*[m(*args) for _ in range(n)]) if n > 1 else m(*args) # module
        t = str(m)[8:-2].replace('__main__.', '') # module type
        np = sum([x.numel() for x in m_.parameters()]) # number params
        m_.i, m_.f, m_.type, m_.np = i, f, t, np # attach index, 'from' index, type, number
            params
        logger.info('%3s%18s%3s%10.0f %-40s%-30s' % (i, f, n, np, t, args)) # print
        save.extend(x % i for x in ([f] if isinstance(f, int) else f) if x != -1) # append to
            savelist
        layers.append(m_)
        ch.append(c2)
    return nn.Sequential(*layers), sorted(save)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--cfg', type=str, default='yolov5s.yaml', help='model.yaml')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    opt = parser.parse_args()
    opt.cfg = check_file(opt.cfg) # check file
    set_logging()
    device = select_device(opt.device)

    # Create model
    model = Model(opt.cfg).to(device)
    model.train()
```

2.2 问题二源代码

2.2.1 坑洼图像筛选源代码

classification.py

```python
import os
import shutil

import torch
from torchvision.transforms import transforms
import cv2
from models.SEnet import *
import csv

# model = SE_ResNet18().cuda().eval()
#
# model.load_state_dict(torch.load('SENet.pt'))

model = torch.load('best.pt').cuda().eval()

data_root = './data_test_V2'

transform_test = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])


# 初始化一个CSV文件
csv_file = 'test_result1.csv'

# 打开CSV文件以写入结果
with open(csv_file, mode='w', newline='') as file:
    fieldnames = ['fname', 'label']
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()

    # 遍历数据根目录中的文件/子目录
    for i in os.listdir(data_root):
        image_root = os.path.join(data_root, i)
        image = cv2.imread(image_root)
        image = transform_test(image)
        image = image.unsqueeze(0)
        image = image.cuda()
        output = model(image)
        _, predicted = torch.max(output, 1)

        # 设置label值
        label = 1 if predicted.item() == 0 else 0

```

```
        # 写入CSV文件
        writer.writerow({'fname': i, 'label': label})
        if label == 1:
            shutil.copyfile(os.path.join(data_root, i), os.path.join('./normal/',i))
        else:
            shutil.copyfile(os.path.join(data_root, i), os.path.join('./potholes/',i))
print("CSV文件已创建并写入结果。")
```

2.2.2 填写结果文件源代码

fill_normal.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2023/12/13 下午7:23
# @Author : SiHang Xu
import csv
import os

file_name = 'test_result2.csv'
# 指定结果文件的路径
csv_file_path = "test_result2.csv"

# 打开CSV文件，使用 'a' 模式
with open(csv_file_path, 'a', newline='') as csvfile:
    # 创建CSV写入对象
    csv_writer = csv.writer(csvfile)


    # 遍历目录中的文件
    for i in os.listdir('./dataset/test_data/normal'):
        image_path = './dataset/test_data/normal/' + i
        result = 0
        image_name = i

        # 将结果写入CSV文件
        if result is not None:
            csv_writer.writerow([image_name, int(result * 100)])
            print(image_name, result*100)
        else:
            print(f"无法处理 {image_name}")
```

calculate.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```python
# @Time : 2023/12/13 下午7:10
# @Author : SiHang Xu
import csv
import os

from PIL import Image

def calculate_non_black_pixel_ratio(image_path):
    try:
        # 打开图片
        image = Image.open(image_path)

        # 获取图片大小
        width, height = image.size

        # 初始化非黑色像素计数
        non_black_count = 0

        # 遍历图片像素
        for y in range(height):
            for x in range(width):
                # 获取像素颜色
                pixel = image.getpixel((x, y))

                # 检查是否为纯黑色
                if pixel != (0, 0, 0):
                    non_black_count += 1

        # 计算非黑色像素占比
        total_pixels = width * height
        non_black_ratio = non_black_count / total_pixels

        return non_black_ratio

    except Exception as e:
        print(f"Error: {e}")
        return None



# 指定结果文件的路径
csv_file_path = "test_result2.csv"

# 打开CSV文件，使用 'w' 模式
with open(csv_file_path, 'w', newline='') as csvfile:
    # 创建CSV写入对象
    csv_writer = csv.writer(csvfile)
```

```
    # 写入表头
    csv_writer.writerow(['fnames', 'pothole %'])

    # 遍历目录中的文件
    for i in os.listdir('./result_mask'):
        image_path = './result_mask/' + i
        result = calculate_non_black_pixel_ratio(image_path)
        image_name = i.split('_')[0] + '.jpg'

        # 将结果写入CSV文件
        if result is not None:
            csv_writer.writerow([image_name, int(result * 100)])
            print(image_name, result*100)
        else:
            print(f"无法处理 {image_name}")

print(f"数据已保存到 {csv_file_path}")
```