

队伍编号	MCB2304072
赛道	A

基于改进 U-Net 的道路坑洼检测模型

摘 要

近年来，随着**深度学习**技术的迅速发展，为计算机视觉技术在道路坑洼检测中的应用提供了更多更强大的解决方案。为了检测不同场景下的道路坑洼图像的面积等参数，本文建立了基于**U-Net 语义分割检测模型**，并结合了**ResNet-50 模型**特征提取的优势和图像处理技术对模型进行改进，创新性的提出了**ResNet-Unet 道路坑洼检测模型**，实现了检测的精准、高效。

针对问题一，采用**U-Net** 作为主要的模型完成道路坑洼的提取识别任务。但是，传统的**U-Net 语义分割网络**存在一定检测上的疏漏，因而本文还结合了初赛中所用到的具有更加强大的特征提取能力的**ResNet-50** 网络模型来改进**U-Net** 网络模型以构建新的更加精准的**ResNet-50 道路坑洼检测模型**。首先利用**ResNet-50** 对输入图像进行特征提取，生成高级别的特征表示，然后，将这些特征传递到**U-Net** 网络中，进行进一步的特征编码和解码，最终生成图像的分割结果。通过这种方式，达到充分利用**ResNet-50** 在大规模图像分类任务上预训练得到的强大特征提取能力，同时又能够利用**U-Net** 网络在图像分割任务上的优势，从而达到更好的分割效果，提取出道路中坑洼的边缘。最后，我们计算出模型的**平均交并比 (mean Intersection over Union)** 达到 87.90%，**Recall** 达到 92.94%，**精确率 (Precision)** 达到 93.58%，**混淆矩阵 (confusion matrix)** 效果良好，四个指标来对模型进行评估，并不断优化模型。

针对问题二，在问题一建立的模型的基础上，我们首先对已识别出的坑洼区域的像素点进行数量的计算，计算的结果计为 N_a 。然后，我们计算整张图像所含有的像素点数目 N_s ，其中 N_s 表示为图像高度 x 与图像宽度 y 的乘积。通过 N_s 与 N_a 的比值来估计坑洼区域面积占整个图像面积的比例 DA (damage area)。

关键词：坑洼道路检测、U-Net 语义分割、ResNet-50、计算机视觉

目录

一、 问题重述	1
1.1 问题背景	1
二、 问题分析	1
2.1 问题 1 的分析	1
2.2 问题 2 的分析	2
三、 模型假设	2
四、 符号说明	2
五、 问题一模型的建立与求解	3
5.1 图像处理	3
5.1.1 数据增强	3
5.1.2 数据增广与平衡	4
5.1.3 数据集标注和划分	5
5.2 U-Net 语义分割模型的建立	5
5.2.1 卷积神经网络模型	5
5.2.2 U-Net 网络模型	6
5.2.3 U-Net 网络模型的代码实现	8
5.3 ResNet-50 特征提取模型的建立	8
5.3.1 残差神经网络（Resnet）模型	8
5.3.2 ResNet-50 网络模型	12
5.3.3 ResNet-50 模型的修改	13
5.3.4 冻结其他层	15
5.3.5 修改全连接层	15
5.3.6 训练过程以及结果	15
5.4 ResNet-Unet 道路坑洼检测模型	17
5.5 模型的求解	18
5.5.1 U-Net 模型提取结果	18
5.5.2 ResNet-Unet 模型提取结果	19
六、 问题二模型的建立与求解	20
七、 ResNet-Unet 模型评估	20
7.1 平均交并比（mIoU）	20
7.2 混淆矩阵（hist）	21
7.3 Recall	22
7.4 精确率	22
八、 参考文献	23
九、 附录	25
9.1 U-Net 模型	25
9.2 ResNet-50 结构（代码生成）	37
9.3 ResNet-50 模型结构示意图	45
9.4 数据集标注实例	46
9.5 ResNet-Unet 模型实现	46

一、问题重述

1.1 问题背景

道路坑洼是一种路面出现结构性崩坏的现象，会对车辆状况、道路安全构成巨大威胁，每年造成的大量车辆损坏和经济损失不计其数。随着我国交通运输业的迅速发展，公路里程数不断攀升，对公路的养护成本愈来愈庞大，如何更加有效的对道路缺陷进行检测与识别，成为了当前需要解决的突出问题。

当前，我国的道路坑洼识别与检测大多以人工检查和半自动检测为主，费时费力且结果良莠不齐，难以满足如今科学化、自动化的道路检查与养护需求，这促使人们一直致力于开发能够有效、准确和客观识别、定位路面坑洼的自动化检测系统。

随着图像处理和深度学习等技术的快速发展，为道路坑洼的检测提供了一定的理论支撑。针对道路坑洼图像，应用深度学习目标检测技术进行识别并提取参数以量化道路坑洼的情况，以更好的服务于道路避障、道路修复等工作，逐渐变得可行。

1.2 问题提出

问题 1：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，建立模型提取道路中坑洼的边缘。

问题 2：根据识别的坑洼边缘，估计坑洼区域的面积，并计算题目中的每个图片中坑洼面积占整个图像面积的比例。

二、问题分析

2.1 问题 1 的分析

建立模型提取道路中坑洼的边缘、面积等特性以便更好的完成道路避障、道路修复等工作，这实际上是一个计算机视觉领域中的语义识别任务。可以借助于深度学习的方法来解决。经查阅文献，U-Net 语义分割模型在图像分割领域应用广泛，本文将采用 U-Net 作为主要的模型完成道路坑洼的提取识别任务。但是，在实验过程中，发现传统的 U-Net 语义分割网络存在一定检测上的疏漏，经查阅文献和总结分析，我们判断是因为 U-Net 特征提取受到了不同场景的干扰所造成的，因而本文还结合了初赛中所用到的具有更加强大的特征提取能力的 ResNet-50 网络模型来改进 U-Net 网络模型以构建新的更加精准的 ReU-Net 道路坑洼检测模型。该模型将利用 ResNet-50 对输入图像进行特征提

取，生成高级别的特征表示，然后，这些特征表示被传递到 U-Net 网络中，U-Net 网络会利用这些特征进行进一步的特征编码和解码，最终生成图像的分割结果。通过这种方式，达到充分利用 ResNet-50 在大规模图像分类任务上预训练得到的强大特征提取能力，同时又能够利用 U-Net 网络在图像分割任务上的优势，从而达到更好的分割效果，提取出道路中坑洼的边缘。

最后，我们根据平均交并比（mean Intersection over Union），混淆矩阵（confusion matrix），PA_Recall，精确率（Precision）四个指标来对模型进行评估，并不断优化模型。



图 1 问题一建模过程

2.2 问题 2 的分析

在问题一建立的模型的基础上，计算图像坑洼区域的面积，其实就是计算图片坑洼区域所含的像素点。我们首先对已识别出的坑洼区域的像素点进行数量的计算，计算的结果计为 N_a 。然后，我们计算整张图像所含有的像素点数目 N_s ，其中 N_s 表示为图像高度 x 与图像宽度 y 的乘积。通过 N_s 与 N_a 的比值来估计坑洼区域面积占整个图像面积的比例 DA （damage area）。

三、模型假设

- 1. 假设道路坑洼区域与周围道路纹理存在明显的差异。
- 2. 假设道路坑洼通常具有一定的几何形状特征。
- 3. 假设数据集中标注的图像特征均可用

四、符号说明

符号	说明
$(f * g)(t)$	卷积的结果
$f(\tau)/g(t - \tau)$	输入函数

t	积的参数（通常表示时间或空间）
τ	积分变量
y	真实标签
p	模型的预测概率
γ	缩放参数
β	平移参数
TP	表示真正例数（模型正确预测为正例的数量）
TN	表示真反例数（模型正确预测为反例的数量）
FP	表示假正例数（模型错误地将负例预测为正例的数量）
FN	FN 表示假反例数（模型错误地将正例预测为负例的数量）
N_a	图片坑洼区域所含的像素点
N_s	整张图像所含有的像素点数目
DA	坑洼区域面积占整个图像面积的比 例

五、问题一模型的建立与求解

5.1 图像处理

5.1.1 数据增强

数据增强是在训练深度学习模型时常用的一种技术，通过对原始数据进行随机变换和处理，来生成更多多样化的训练样本，以达到增加模型的泛化能力、扩充训练集大小、提高模型的抗噪能力、减少过拟合的风险、增强模型的鲁棒性等作用。

基于 PyTorch，我们定义了一个数据增强的转换流程（`data_augmented_transforms`），包括以下几个步骤：

随机裁剪并调整大小（`RandomResizedCrop`）：随机选择图片的一部分区域，并将其

调整为指定的尺寸（这里是 224×224 像素）。这可以模拟不同尺度下的物体出现情况。

随机水平翻转 (RandomHorizontalFlip): 以一定的概率（默认为 0.5），对图片进行水平翻转，增加训练样本的多样性。

随机垂直翻转 (RandomVerticalFlip): 以一定的概率（默认为 0.5），对图片进行垂直翻转，进一步增加训练样本的多样性。

随机色彩调整 (ColorJitter): 对图片的亮度、对比度、饱和度和色调进行随机调整。这可以增加模型对不同光照条件和颜色变化的鲁棒性。

随机旋转 (RandomRotation): 随机将图片按照一定范围内的角度进行旋转（这里是最多 ± 30 度）。这可以模拟物体在不同方向上的出现情况。

随机动态模糊 (RandomGaussianBlur): 以一定的概率（这里是 0.1），对图片进行高斯模糊处理。模糊处理可以模拟运动模糊或者增加图片的噪声，从而增加模型对噪声的鲁棒性。

转换为张量 (ToTensor): 将图片数据转换为张量格式，便于在深度学习模型中进行处理。

归一化 (Normalize): 将图片的像素值进行标准化处理，使其均值为计算出的指定值 $[0.5083675664513708, 0.5444931348182125, 0.5449574978539466]$ ，标准差为计算出的指定值 $[0.052240807778822174, 0.03456613663032132, 0.045404974436703716]$ 。归一化可以加速模型的收敛，并提高训练的稳定性。

5.1.2 数据增广与平衡

数据平衡对于模型的性能和稳定性非常重要。当样本数量不均衡时，模型可能会倾向于预测数量较多的类别，而忽略数量较少的类别。这种情况下，模型的准确率可能会非常高，但是召回率和 F1 分数等指标会非常低。这会导致模型无法很好地识别数量较少的类别。因此，本文利用编写好的数据增强程序对数据集进行增广与平衡，将原有的 266 张图像扩充到 500 张，将原有的坑洼道路图像也扩展到 500 张。

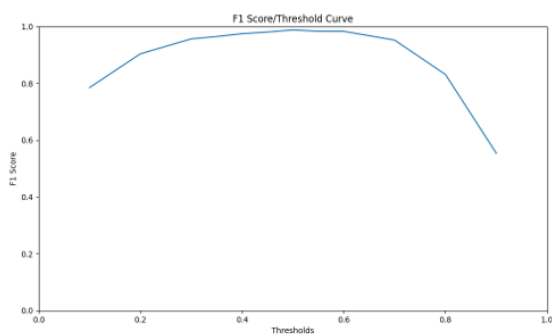
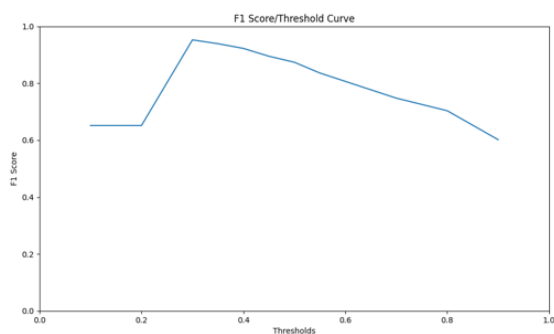


图 2 图为数据平衡（左）和数据平衡后（右）的 F1-score 指标对比

注：F1-score 是一种常用的评估分类模型性能的指标，它综合考虑了模型的精确率（precision）和召回率（recall）两个指标。F1-score 的取值范围在 0 到 1 之间，越接近 1 表示模型的性能越好。

5.1.3 数据集标注和划分

对题目所给出的 5000 张图片，我们从中选出了 700 张道路坑洼图像作为用于训练的数据集，具体使用 Stratified K-Fold 交叉验证的方法划分训练集和测试集，利用 labeling 图像标注工具对其进行标注，过程如下图所示：



图 3 数据集标注

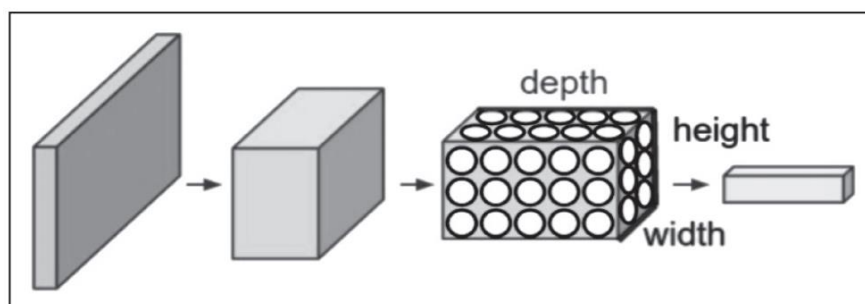
5.2 U-Net 语义分割模型的建立

5.2.1 卷积神经网络模型

卷积神经网络（CNN）是一种模仿生物神经网络人工构造的模型，是最常用也是最广泛的深度学习模型，具有很强的特征提取和表示能力。它可以从图像中自动提取出最重要的特征，用特征来表示原始数据，从而解决过去传统算法需手动提取特征带来的耗时、准确率低等问题，加之计算机性能的快速提升，使卷积神经网络得到了质的发展，因此在图像分类、目标识别以及其他领域，得到了广泛的应用，并且取得了许多显著的成就。

运用卷积神经网络模型可以更好的从复杂道路图片中提取出坑洼的轮廓、纹理、形态、边缘、局部特征等特征，从而更加准确、高效的实现坑洼道路检测的功能。

卷积神经网络一般是由卷积层、池化层、全连接层和激活层组成的。卷积层由多个



卷积核和对应的偏移值组成，通过卷积核与输入的图像进行卷积操作来提取特征；池化层的本质是下采样，从而减少参数量，降低过拟合的风险，其非线性池化函数有最大池化和平均池化等，特点是不改变神经网络深度，只改变其大小，减少最后全连接层中节点的数量；全连接层的主要功能是将卷积层和池化层提取到的特征进行整合，构建出高层次的特征表示，从而使模型能够更好地进行分类、识别等任务；激活层的作用是

图 4 卷积神经网络结构

增加模型的非线性特征，提高其拟合能力和表达能力，使模型可以更好地适应复杂的数据分布。用的激活函数有 Sigmoid、Tanh、ReLU、Leaky ReLU 和 SELU 等。

5.2.2 U-Net 网络模型

U-Net 模型是一种常用于图像分割任务的深度学习模型。它由 Olaf Ronneberger、Philipp Fischer 和 Thomas Brox 于 2015 年提出，被广泛应用于医学影像领域。UNet 模型的结构呈 U 字形，因此得名。它基于卷积神经网络的编码-解码结构，并在解码阶段通过跳跃连接将编码和解码路径中的特征图进行连接。这种设计使得 U-Net 能够在保持局部细节信息的同时，利用不同尺度的特征来进行更准确的分割。U-Net 模型在医学图像分割、自然图像分割等领域取得了很好的效果，成为图像分割任务中的经典模型之一。

U-Net 网络的主要结构包括了解码器、编码器、瓶颈层三个部分。

编码器：包括了四个程序块。每个程序块都包括 $3 \times 3 \times 3$ 的卷积（使用 Relu 激活函数），步长为 2 的 $2 \times 2 \times 2$ 的池化层（下采样）。每个程序块处理后，特征图逐步减小。

解码器：与编码器部分对称，也包括四个程序块，每个程序块包括步长为 2 的 $2 \times 2 \times 2$ 的上采样操作，然后与编码部分进行特征映射级联（Concatenate），即拼接，最后通过两个 $3 \times 3 \times 3$ 的卷积（Relu）。

瓶颈层：包含两个 $3 \times 3 \times 3$ 的卷积层。

最后经过一个 $1 \times 1 \times 1$ 的卷积层得到最后的输出。

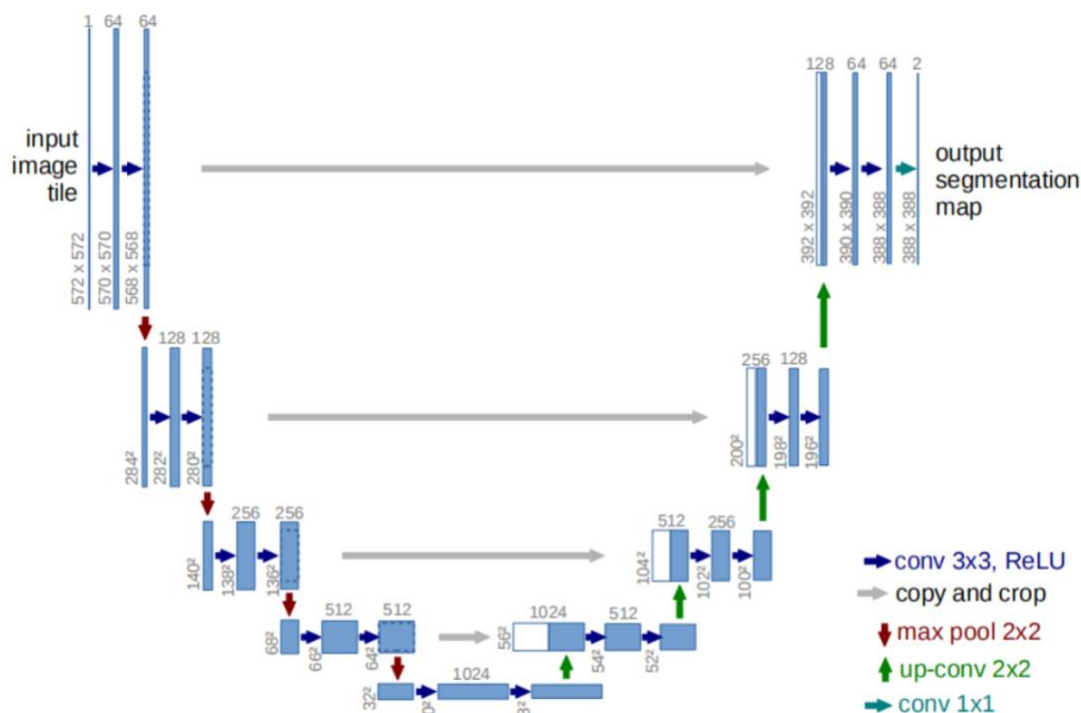


图 5 U-Net 网络模型结构图

如图所示，U-Net 网络结构的左半部分对应的是下采样特征提取部分，右半部分对应上采样恢复 Mask 掩码的部分，U-Net 网络整体上呈现 U 型结构，也因此而得名。U-Net 网络的左边为压缩通道(Contracting Path)，它重复采用 2 个卷积层和 1 个最大池化层的结构，每进行一次池化操作后特征图的维数就增加 1 倍。U-Net 网络的右边为扩张通道(ExpansivePath)，上采样时先进行一次反卷积运算，其特征图的尺寸扩大为原来的两倍，特征层的通道数减半。然后与第三次下采样得到的特征图进行特征融合，重新组成一个上采样 2 倍的特征图，之后再采用两个卷积层进行特征提取将特征图的信息，进一步融合并压缩特征层的通道数。图中(copy and crop)灰箭头是一种跳跃式的连接，这样可以最大化恢复原像素的位置信息以及解决像素的分类与定位问题。然后重复这一结构上采样 3 次直到恢复与原图尺寸大小相同的 Mask 掩码图像。

U-Net 的输入图片的大小可以是任意尺寸的图像，输出是一个 Mask 掩码图像，故 U-Net 网络模型是一个端到端的网络模型。相比于 FCN 网络模型 U-Net 网络模型的 copy and crop 拼接模式充分融合了各层的特征信息，即使其在训练数据很少的情况也能够得到很精确的分割结果。医学图像相对来说数据量较少且难以获得，所以 U-Net 网络模型常用于医学图像处理方面，并且可以取得比较好的分割效果。本文应用场景是基

于不同场景下题目所给出的道路坑洼检测（图像语义分割），也存在数据量相对较少和数据影像难以获得的特点，并且道路坑洼的形状与医学图像中病灶区域的形状都呈现不规则的特点。故本文选用 U-Net 作为本文的语义分割网络模型，并在其基础上结合 ResNet 网络模型进行一定的改进，使其能适应不同场景下道路坑洼检测与分割的任务。

5.2.3 U-Net 网络模型的代码实现

基于 PyTorch 框架实现 U-Net 模型的建立过程：

数据准备和预处理

通过 ISBI_Loader 类，读取训练数据集中的图片和标签，并进行预处理。预处理包括将图片转为灰度图、进行数据增强（如翻转、高斯模糊等），将标签的像素值归一化为 0-1 之间。

模型构建

使用 U-Net 网络模型（在 unet_model.py 中定义）来进行图像分割。

训练过程

使用 KFold 进行交叉验证，将训练集划分为多个 fold。

对每个 fold 进行训练和验证，每个 epoch 中，使用 DataLoader 将数据分批次加载，通过前向传播和反向传播更新模型参数。

在训练和验证过程中，计算损失函数（比如交叉熵损失）、准确率、F1-score 等指标，用于评估模型性能。

根据验证集上的 F1-score，保存最好的模型和最新的模型。

结果分析和可视化

使用自定义的 plot 库，绘制训练和验证集的损失函数曲线、准确率曲线、F1-score 曲线等，用于分析模型的训练过程和性能表现。

绘制混淆矩阵，分析模型在不同类别上的分类效果。

5.3 ResNet-50 特征提取模型的建立

5.3.1 残差神经网络（Resnet）模型

残差神经网络（Resnet）是卷积神经网络的一种改进和扩展。传统的卷积神经网络中，每个层都会对输入进行一些变换，然后将结果作为输出传递给下一层。但是，随着网络层数的增加，这种变换可能会导致信息的丢失或者变形，从而影响模型的性能。为了解决这个问题，ResNet 引入了残差块（Residual Block），通过添加跨层连接（shortcut connection）将输入直接添加到输出中，使得网络可以更加深入地学习特征。残差神经网络

络重构了深度卷积网络的学习过程，重新定向了深层卷积神经网络信息流，从而很好地解决了深层卷积神经网络存在的梯度消失、梯度爆炸和退化等问题。

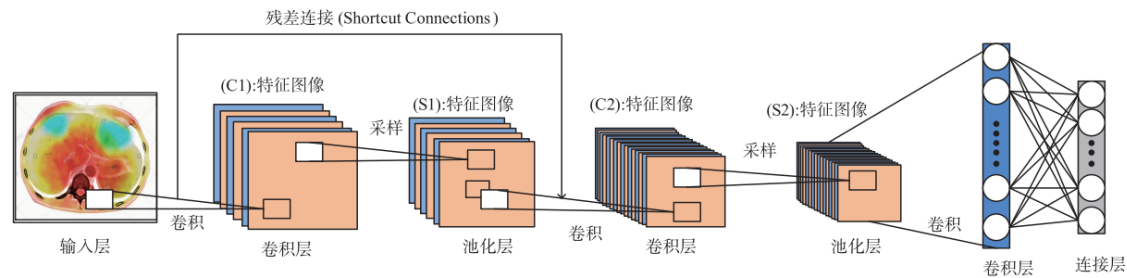


图 6 残差神经网络示意图

在深度学习神经网络中，卷积层是最重要的一层，它的作用是提取输入数据的特征，它内部的具体的操作是通过一个滤波器在输入数据上进行“扫描”操作，计算权重矩阵和扫描所得的数据矩阵的乘积，然后把结果汇总成一个输出像素。从而实现对于数据的局部特征的捕捉，最终合并成特征图，然后将特征图传递到后续的机器学习和分类等模块。卷积公式如下：

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau \tag{1}$$

其中， $(f * g)(t)$ 表示卷积的结果， $f(\tau)$ 和 $g(t - \tau)$ 是两个输入函数， $*$ 表示卷积操作， t 是卷积的参数（通常表示时间或空间）， τ 是积分变量。

在整个 ResNet 模型中，使用到两种池化层：平均池化层和最大池化层。

平均池化层

之所以用到平均池化层，是因为：首先，平均池化对输入特征图的平滑性更强，因为它考虑了整个池化区域内的平均值，而不是仅考虑最大值。平均池化保留了更多的输入信息，因为它对整个池化区域内的像素进行平均，而不会忽略掉次大的特征值。这对于学习更多的细节和语义信息是有益的，尤其是在深度网络中。这有助于减少信息的损失，特别是对于深层网络中的梯度流动而言。

其次，平均池化层因为它在池化区域内使用平均值，而不是选择最大值，从而减小了梯度的剧烈变化，可以实现缓解梯度消失的作用。

最后是因为 ResNet 中引入的残差连接允许跳跃连接前后层之间的特征，从而有助于传播梯度。平均池化与这种残差连接结合得更自然，因为它会保留更多的信息，并允许更容易地传递梯度。

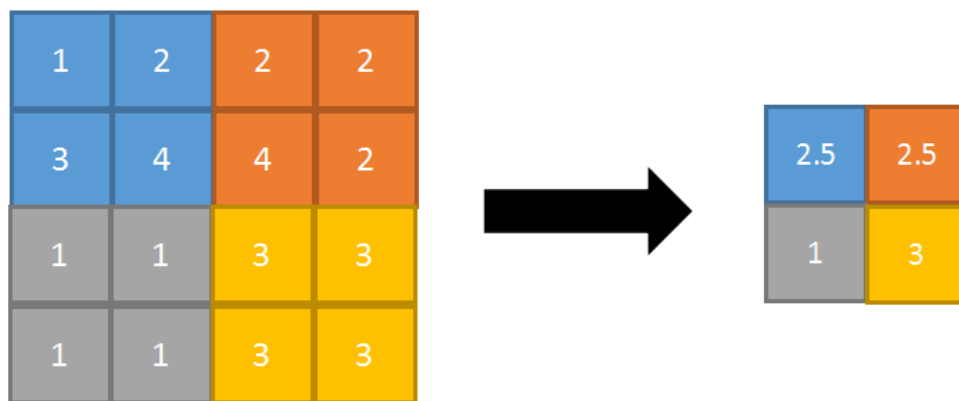


图 7 平均池化图像

最大池化层通过选择池化区域内的最大值来提取最显著的特征，这有助于突出输入数据中的主要特征。这对于图像分类和目标检测等任务非常有用，因为它有助于保留与任务相关的关键信息。

最大池化层具有一定程度的平移不变性，因为它选择池化区域内的最大值，而不关心其具体位置。这对于处理不同位置的特征非常有用，因为它允许模型更容易识别相同特征的不同实例。

最大池化对一些程度的噪声和小干扰具有抗性，因为即使池化区域中有一些小干扰，它仍然会选择最大值，从而减小了对噪声的敏感性。

最大池化层没有需要学习的参数，这有助于减小模型的参数数量，减少过拟合的风险，并提高模型的泛化能力。

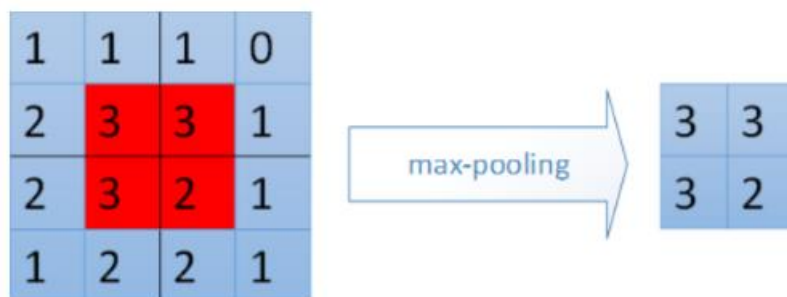


图 8 最大池化层

平均池化用于降低特征图的空间分辨率，减少参数数量，以及增加模型的感受野。在我们这个 ResNet-50 模型中，平均池化我们使用在每个残差块的最后一层，用于减小特征图的大小。最大池化用于提取特征图中的最显著特征，用于降低空间分辨率。在我们这个 ResNet-50 模型中，最大池化层使用在初始卷积层之后和每个残差块的第一个卷积层之后出现，用于逐渐减小特征图的大小。

随着网络不断加深，梯度消失或梯度爆炸会越来越明显。梯度消失现象是指每向前传播一层，都要乘以一个小于 1 的误差梯度，网络越来越深，乘以小于 1 的系数越来越

多，就越趋近于 0，梯度就会越来越小。梯度爆炸现象则是每向前传播一层，都要乘以一个大于 1 的误差梯度，网络越来越深，乘以大于 1 的系数越来越多，就越趋近于正无穷，梯度就会越来越大。

为了解决上述问题，采用了批量标准化（BN），针对路面坑洼检测的图片，它们通常以 RGB 格式呈现。BN 标准化是非常重要的一步，它的主要目的是对特征矩阵进行处理，确保每个维度的平均值为 0，方差为 1，即进行标准化处理。这样的标准化有助于减少和避免梯度消失和梯度爆炸的问题，因为它确保数据处于一个非饱和的状态。通过批量标准化处理，我们有效地控制了内部神经元的数值范围变化，降低了不同样本之间数值范围的不稳定性。

没有批量标准化时，每一层的输入分布在训练过程中可能会显著变化，导致网络收敛速度较慢并增加过拟合的风险。BN 通过规范化输入，有助于网络更快地学习并降低过拟合的风险。通过规范化激活，BN 使每次激活后神经元的分布更加显著，鼓励某些神经元变得不那么重要。

因此，减少了 Dropout 的需求，而 L2 正则化，通过对大权重进行惩罚来减轻过拟合的方法，也可以减小或消除，因为 BN 本身有助于控制由于不同输入分布而导致的参数增长。

对于这种 RGB 三通道的应用就非常有效，这有助于提高模型的训练效果。如下是具体实现的公式：

小批量均值：

$$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad (2)$$

小批量方差：

$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad (3)$$

标准化：

$$\hat{x} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (4)$$

缩放和平移：

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad (5)$$

其中 γ （缩放参数）和 β （平移参数）。

5.3.2 ResNet-50 网络模型

ResNet-50 是一种深度卷积神经网络（Convolutional Neural Network, CNN），它是残差网络（Residual Network）的一个具体实现。ResNet-50 由微软亚洲研究院的研究员提出，并在 2015 年的 ImageNet Large Scale Visual Recognition Challenge（ILSVRC）中取得了优异的结果。

ResNet-50 的名称中的"50"表示网络中的层数，它由 49 个卷积层和 1 个全连接层组成。与传统的 CNN 相比，ResNet-50 引入了残差单元（Residual Block）的概念，通过跨层连接（Skip Connection）的方式解决了深层网络训练中的梯度消失和梯度爆炸问题，使得网络更易于训练。

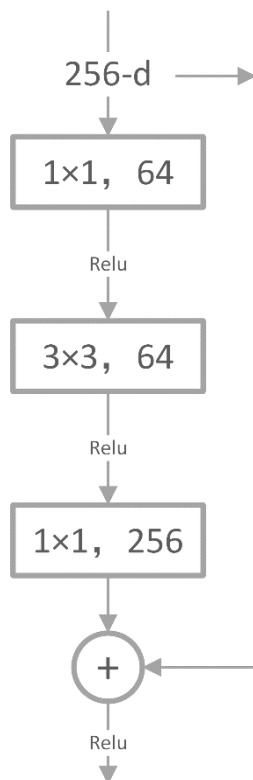


图 9 残差单元图

每个残差块内部包含了多个卷积层和批量归一化层（Batch Normalization），并且在每个残差块的输入上应用了一个恒等映射（identity mapping）或一个投影映射（projection mapping）。这些跨层连接可以将输入直接传递到输出，从而避免了信息的丢失。

ResNet-50 的架构使得它在图像分类、物体检测和语义分割等计算机视觉任务上表现出色。此外，ResNet-50 也成为了深度学习领域的一个里程碑，其思想和结构被广泛应用于其他网络的设计中，促进了深度学习的发展。ResNet-50 的基本结构如下（具体的预训练网络在附录列出）。

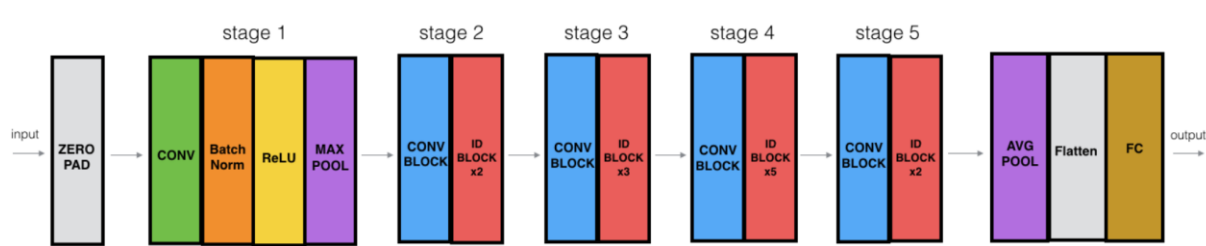
输入层: ResNet-50 的输入层接受图像数据作为输入, 通常为彩色图像, 其尺寸为 224x224 像素。

卷积层和池化层: ResNet-50 的第一个阶段包含一个卷积层和一个池化层。卷积层将输入图像进行卷积操作, 通常采用 7x7 大小的卷积核, 用于提取低级特征。接下来的池化层 (一般为最大池化) 将卷积层输出的特征图尺寸减小。

残差块: ResNet-50 有共 4 个残差块, 每个块包含若干个残差单元 (Residual Unit)。每个残差单元由两个相同尺寸的卷积层组成, 之间是一个跳跃连接 (shortcut connection)。这种跳跃连接使得网络能够更容易地学习残差函数, 并帮助解决梯度消失和梯度爆炸的问题。

全局平均池化层: 在 ResNet-50 的最后一个残差块后, 有一个全局平均池化层。它对特征图进行平均池化, 将每个特征图的尺寸缩减为 1x1, 保留了特征的整体空间信息。

全连接层: 最后, 通过一个或多个全连接层, 将提取的特征映射到具体的类别或输出值。通常, 最后一层的全连接层用于分类任务, 其中每个神经元代表一个类别。



ResNet-50 模型是一种强大的卷积神经网络架构, 具有很强的特征提取能力。ResNet-50 对输入图像进行特征提取, 生成高级别的特征表示, 然后, 这些特征表示被传递到 U-Net 网络中, U-Net 网络会利用这些特征进行进一步的特征编码和解码, 最终生成图像的分割结果。

通过这种方式, 我们可以充分利用 ResNet-50 在大规模图像分类任务上预训练得到的强大特征提取能力, 同时又能够利用 U-Net 网络在图像分割任务上的优势, 从而达到更好的分割效果。这种联合使用的方法在很多图像分割任务中都取得了较好的性能表现。

5.3.3 ResNet-50 模型的修改

本文基于 PyTorch 框架对预训练的 ResNet-50 的全连接层进行调参。即冻结其他层, 通过 Stratified K-Fold 交叉验证的方法训练全连接层 (分类器)。

K-Fold 交叉验证 (Cross-validation) 是一种用的模型评估方法, 它的原理是将数据集分成多个子集, 然后进行多轮的训练和测试, 以便更全面地评估模型的性能。交叉验证有助于减少模型在特定数据集上的过拟合或欠拟合问题, 并提供可靠的性能评估。

Stratified K-Fold 是在 K-Fold 交叉验证的基础上，保持每个折叠中的类别分布与整个数据集类别分布相似。下面是 K-Fold 交叉验证流程:

表 1 K-Fold 交叉验证流程

K-Fold 交叉验证流程	
Sept 1	将数据集分为 K 个相等的子集，其中 K 是预先确定的折数
Sept 2	对于每个折叠，选择其中一个子集作为验证集，而其他 K-1 个子集作为训练集
Sept 3	训练集对模型进行训练，并使用验证集评估模型的性能指标。
Sept 4	重复步骤 2 和 3，依次选择不同的子集作为验证集，直到每个子集都被用作验证集
Sept 5	对于每个折叠的模型，记录和收集性能指标，如准确率、精确度、召回率或 F1 分数
Sept 6	计算模型在 K 折交叉验证中的平均性能指标，作为对模型整体性能的评估。

将数据集使用 K-Fold 交叉验证方法，划分为 K 个子集，其中 $\frac{9K}{10}$ 个子集用于训练， $\frac{1K}{10}$ 个子集用于验证。这能为我们提供更准确的性能评估：，因为在机器学习中，我们通常希望了解模型在未见过的数据上的性能表现。使用 K-Fold 交叉验证可以提供多个独立的训练和验证集组合，从而减少对单个训练/验证集划分的依赖。通过对 K 个模型的性能进行平均，可以得到更准确可靠的性能评估。

此外，由于数据集往往是有限的资源，因此需要充分利用每个样本的信息。K-Fold 交叉验证可以确保每个样本都在验证集中出现过一次，从而充分利用数据。这样可以更好地评估模型在不同数据分布上的表现，提高模型的鲁棒性。

使用独立的验证集进行模型选择和调参也是非常重要的。通过将一部分数据保留用于验证，可以帮助检测和避免模型在训练集上的过拟合。使用不同的验证集进行验证，可以更好地估计模型的泛化能力。

除了用于评估模型性能外，验证集还可以用于选择模型的超参数，如学习率、正则化参数等。通过在每个验证集上训练和评估多个模型，可以比较它们在证集上的性能，并选择最佳的超参数配置。

5.3.4 冻结其他层

在模型建立时已经提到，冻结 ResNet 的参数使其在训练过程中不会被更新，这可以保留预训练权重的优势，这些预训练的权重对于许多常见的视觉任务都是有效的。通过冻结除全连接层以外的层，可以保留这些预训练的权重，从而避免破坏模型已学习到的有用特征。还可以避免过拟合，如果在初始阶段解冻所有层并进行训练，可能会导致在相对较小的训练集上过拟合。通过冻结大多数层并仅训练全连接层，可以减少模型的参数量，降低过拟合的风险，并帮助模型更好地泛化到新的数据。此外，还减少计算量，ResNet-50 具有数百万个参数，训练整个模型可能需要大量的计算资源和时间。由于全连接层是模型中参数数量最多的部分，冻结其他层可以显著减少需要计算的参数数量，加快模型训练速度。

5.3.5 修改全连接层

在模型建立时已经提到，我们对全连接层进行了修改，添加了几个 BN 和 Dropout 层。这可以做到加速和稳定训练，批量归一化层通过对每一批次的输入进行归一化，可以缩小输入的梯度范围，防止模型训练过程中的梯度消失或爆炸问题。因此，它有助于加速模型的训练，并可以使用更大的学习率。同时，BN 层通过减少内部协变量移位（Internal Covariate Shift）现象，可以使训练过程更加稳定。还可以提高模型的泛化能力：，批量归一化层对每一批次的输入进行归一化，有助于防止模型的过拟合现象。这是因为 BN 层通过规范化输入分布，减少了输入之间的依赖性，从而起到了一定的正则化效果，帮助模型更好地泛化到新的数据。还可以减少过拟合的风险：，Dropout 层是一种正则化技术，在训练过程中随机地将部分神经元置为 0，从而降低了神经元之间的依赖性。这样，模型被迫学习不同随机子集的特征，减少了神经元之间的共适应性，并减小了过拟合的风险。最后，还可以增强模型的鲁棒性：BN 层可以使模型对输入的缩放和平移更加鲁棒。这意味着，即使输入图像的亮度、对比度或颜色发生变化，模型的性能也能够保持稳定。

5.3.6 训练过程以及结果

首先定义一个交叉熵损失函数和一个 Adam 优化器用于模型的训练。

交叉熵损失函数（Cross-Entropy Loss）是深度学习中常用的一种损失函数，特别适用于分类问题。它衡量了模型的预测结果与真实标签之间的差异程度。我们定义的交叉

熵损失函数结合了 Sigmoid 激活函数和负对数似然损失，可以直接用于多类别分类任务。

交叉熵损失函数的计算公式如下：

$$L = -\sum(y * \log(p)) \quad (6)$$

其中， y 表示真实标签， p 表示模型的预测概率。交叉熵损失函数的目标是使得预测概率与真实标签尽可能接近，可以通过最小化该损失函数来优化模型参数。在反向传播过程中，交叉熵损失函数会根据模型的预测概率和真实标签计算梯度，从而更新模型的参数。

之后，对于每一个 `batch` 的数据进行前向传播、计算损失、反向传播、更新模型参数的操作，从而完成一次模型训练的过程。同时，使用 `tqdm` 库在训练时打印出进度条，便于观察训练的进度和效果。

在每次 `epoch` 结束后，使用验证集数据对模型进行验证，计算模型在验证集上的损失、准确率、精确度、召回率、F1 值等指标，并在进度条中显示出来。同时，将每次 `epoch` 的训练损失、验证损失、训练准确率、验证准确率、TPR、FPR、F1 值等数据存储到列表中备用。

在每次 `epoch` 结束后，更新并绘制训练过程中的损失曲线、准确率曲线、F1 值曲线、ROC 曲线等图表，并使用 `Qt5` 库将这些图表显示在 GUI 窗口中。

周期性地保存模型，以便在训练过程中出现错误或者需要继续训练时能够重新加载模型。同时，还根据最优的验证准确率和 F1 值保存最优模型。

每次 `epoch` 结束后，在控制台输出该次训练结果：TP、FP、TN、FN、F1 值和总样本数等信息。

采用 Stratified K-Fold 交叉验证方法对模型进行训练，在 `loss` 率显著降低以及准确率达到稳定后，对测试集进行预测，将错误的结果再次放入训练集使用交叉验证方法进行续训练。

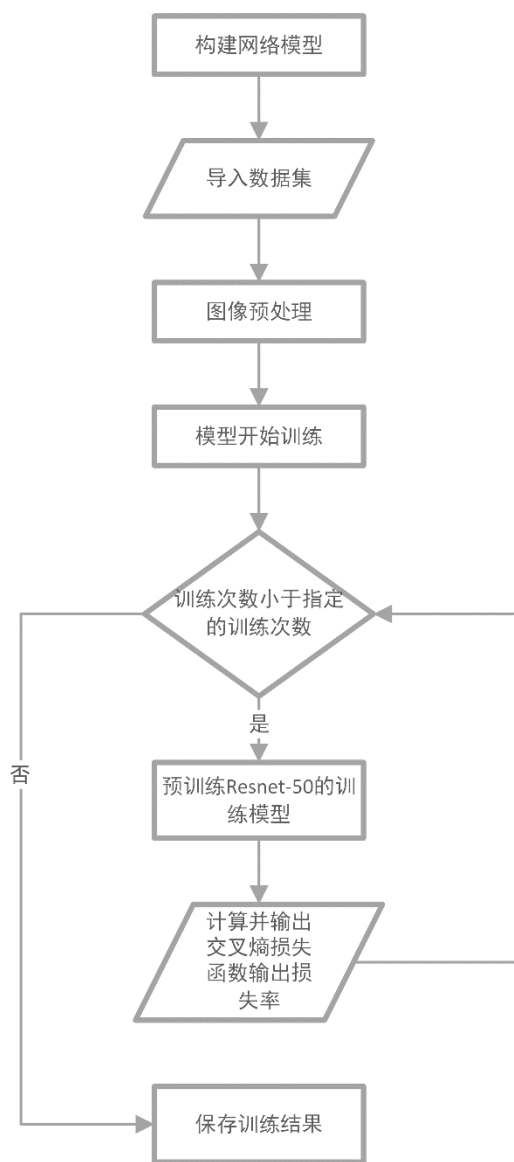


图 10 模型训练流程图

5.4 ResNet-Unet 道路坑洼检测模型

提取道路中坑洼的边缘，从而识别坑洼的边缘、面积等特性，实质上是计算机视觉中的语义分割应用的场景。

语义分割可以将图像中的每个像素分配到特定的语义类别。与普通的图像分类任务不同，语义分割要求对每个像素进行细粒度的分类，即为每个像素赋予一个标签，以表示它所属的语义类别。语义分割可以实现对图像中不同物体、场景和区域的精确分割，从而更好地理解图像内容。通过将图像划分成多个区域并为每个区域分配语义标签，语义分割为许多应用场景提供了关键信息，如自动驾驶中的道路和障碍物检测、医学影像中的病变分析、图像编辑中的对象分离等。因此也适用于本题对道路坑洼的检测场景。

语义分割通常使用深度学习方法，如卷积神经网络（CNN）和其变种，以提取图像特征并进行像素级别的分类。常见的语义分割模型包括 FCN、U-Net、DeepLab 等。这

些模型通过学习大量标注的图像数据，能够实现准确的语义分割结果。本文将采用 U-Net 作为主要模型来实现道路坑洼的语义分割任务。

但是，通过初步建模测试和查阅文献，发现道路坑洼语义分割在图像语义分割任务中相对较难实现，主要因为坑洼在路面中占比较小且坑洼部分的 RGB 值与路面的 RGB 值十分接近，提取特征时会造成干扰。再加上题目所给出的不同场景下复杂的道路图像特点增大了实际应用的难度，道路坑洼更不易被检出。所以，传统的 U-Net 语义分割网络模型不能满足题目的需要，因此本文将结合 ResNet-50 特征提取网络改进 U-Net 网络模型构建更深层次 D-UNet 网络语义分割网络。以便更好的应用于各种复杂的道路图像环境，进一步提高道路坑洼语义分割的分割精度。

5.5 模型的求解

5.5.1 U-Net 模型提取结果

传统的 U-Net 语义分割网络存在一定检测上的疏漏，部分结果如下图。经查阅文献和总结分析，我们判断是因为 U-Net 特征提取受到了不同场景的干扰所造成的，因而本文还结合了初赛中所用到的具有更加强大的特征提取能力的 ResNet-50 网络模型来改进 U-Net 网络模型以构建新的更加精准的 ReU-Net 道路坑洼检测模型。该模型将利用 ResNet-50 对输入图像进行特征提取，生成高级别的特征表示，然后，这些特征表示被传递到 U-Net 网络中，U-Net 网络会利用这些特征进行进一步的特征编码和解码，最终生成图像的分割结果。通过这种方式，达到充分利用 ResNet-50 在大规模图像分类任务上预训练得到的强大特征提取能力，同时又能够利用 U-Net 网络在图像分割任务上的优势，从而达到更好的分割效果，提取出道路中坑洼的边缘。



图 11 U-Net 分割结果示例 1

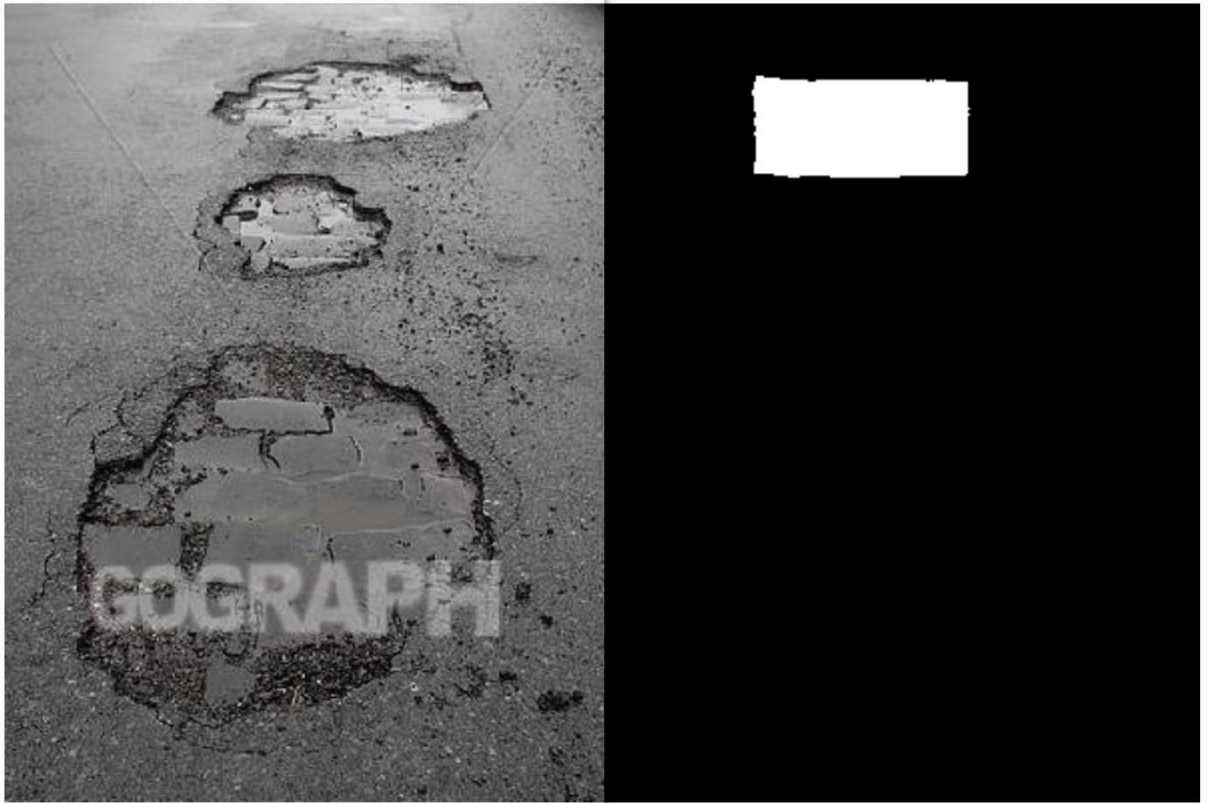


图 12 U-Net 分割结果示例 2

5.5.2 ResNet-Unet 模型提取结果

使用改进后的 ResNet-Unet 提取的图像结果如图所示：

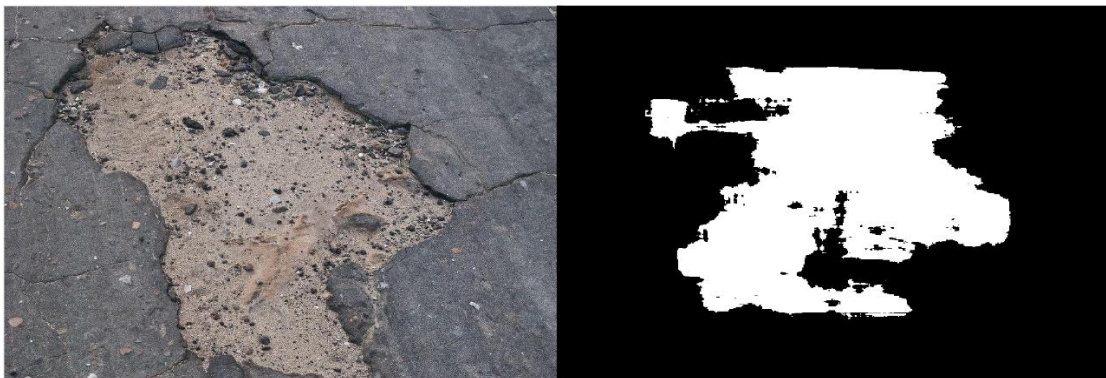


图 13 ResNet-Unet 分割结果 1



图 14 ResNet-Unet 分割结果 2

六、问题二模型的建立与求解

计算图像坑洼区域的面积，就是计算图片坑洼区域所含的像素点。我们首先对已识别出的坑洼区域的像素点进行数量的计算，计算的结果计为 N_a 。

然后，我们计算整张图像所含有的像素点数目 N_s ，其中 N_s 表示为图像高度 x 与图像宽度 y 的乘积。通过 N_s 与 N_a 的比值来估计坑洼区域面积占整个图像面积的比例 DA (damage area)。

$$DA = \frac{N_a}{N_s} \times 100\% \quad (7)$$

七、ResNet-Unet 模型评估

7.1 平均交并比 (mIoU)

平均交并比 (mean Intersection over Union)，简称 mIoU，是一种常用的语义分割性能指标。它度量预测的分割结果与真实的分割结果之间的重叠程度，即计算每个类别的 IoU 并对它们求平均。mIoU 的值越高，表示模型的性能越好。公式如下：

$$mIoU = (1/n) * \sum (IoU_i) \quad (8)$$

其中 n 表示类别数， IoU_i 表示第 i 个类别的 IoU。mIoU 可以衡量模型在所有类别上的分割性能，越高表示模型的性能越好。通常情况下，mIoU 值大于 0.5 才被认为是较好的分割结果。

计算结果如图所示：

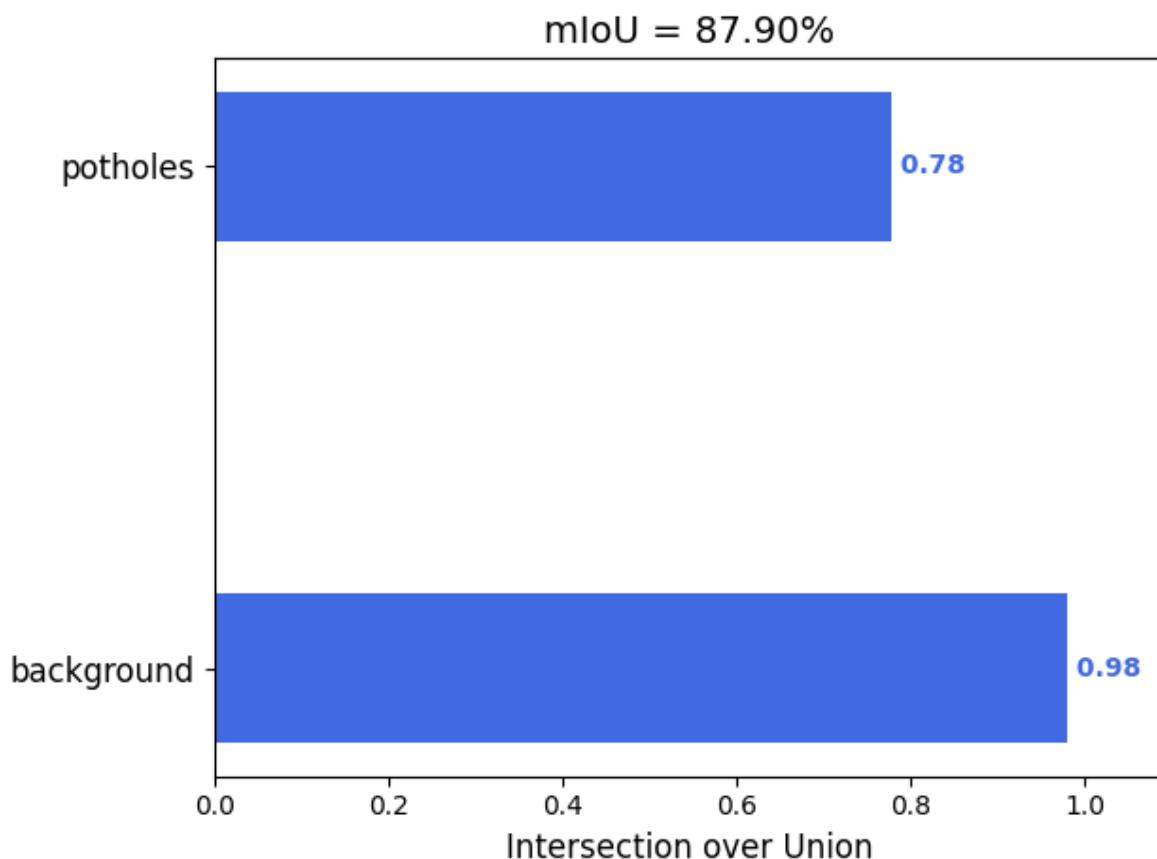


图 15 mIoU 结果

7.2 混淆矩阵 (hist)

混淆矩阵 (Confusion Matrix) 是机器学习领域中常用的评估分类模型性能的工具。它是一个二维矩阵，用于比较模型预测结果和真实标签之间的差异。混淆矩阵的四个基本术语如下：

真正例 (True Positive, TP)：模型将正例正确地预测为正例的数量。

假正例 (False Positive, FP)：模型将负例错误地预测为正例的数量。

假反例 (False Negative, FN)：模型将正例错误地预测为负例的数量。

真反例 (True Negative, TN)：模型将负例正确地预测为负例的数量。

使用这些术语，混淆矩阵可以表示为以下形式：

表 2 混淆矩阵表示

	预测正例	预测负例
真实正例	TP	FN
真实负例	FP	TN

混淆矩阵可用于计算多个分类指标，例如准确率 (Accuracy)、精确率 (Precision)、

召回率（Recall）和 F1 分数等，以评估模型在不同类别上的表现。

计算结果如下：


dataset > prepare > result > 	confusion_matrix.csv		
1		,background,	potholes
2	background,	183187269.0,	1820907.0
3	potholes,	2074818.0,	13711746.0

图 16 混淆矩阵结果

7.3 Recall

像素召回率（Recall）是表示模型正确预测为正例的样本数占有所有真实正例的样本数的比例。计算公式为：

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN}) \quad (9)$$

计算结果如图所示：

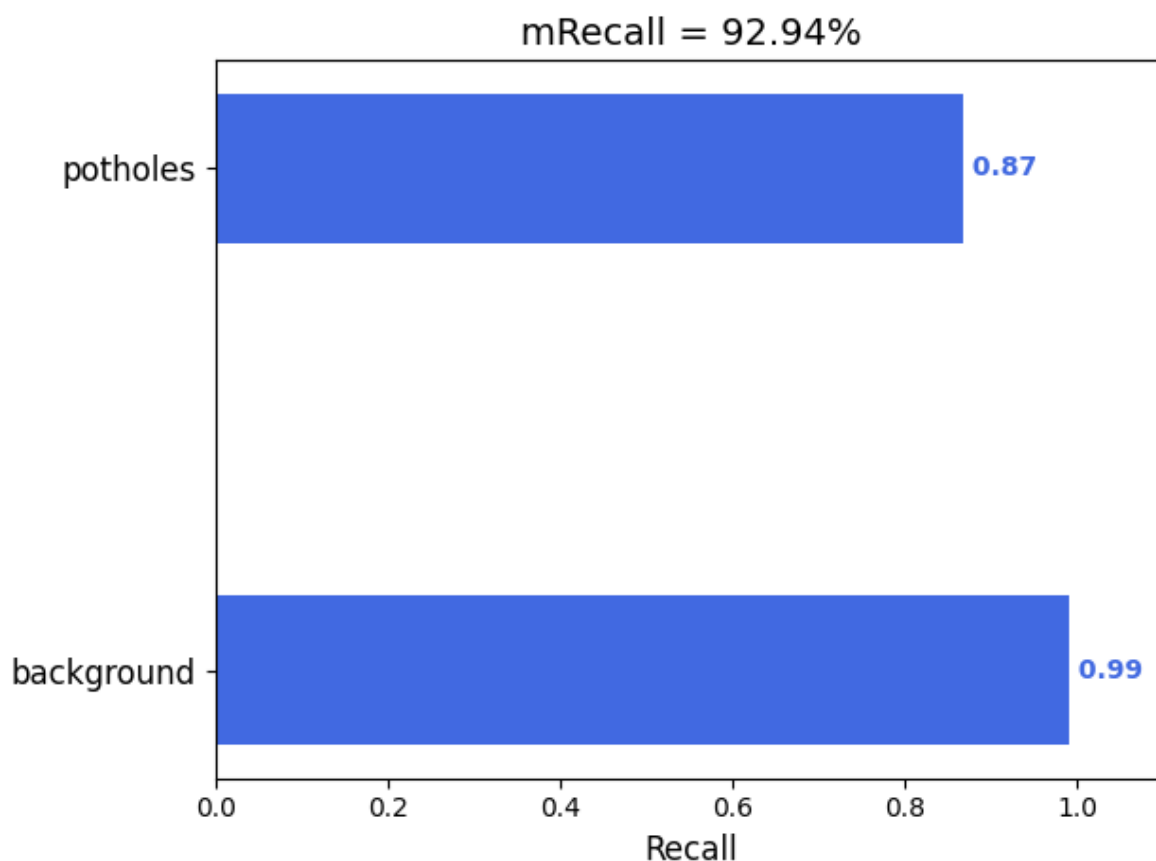


图 17 Recall 计算结果

7.4 精确率

精确率（Precision）是表示模型正确预测为正例的样本数占有所有预测为正例的样本数的比例。精确率越高，表示分类器产生的正类预测更加准确。计算公式为：

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP}) \quad (10)$$

计算结果如图：

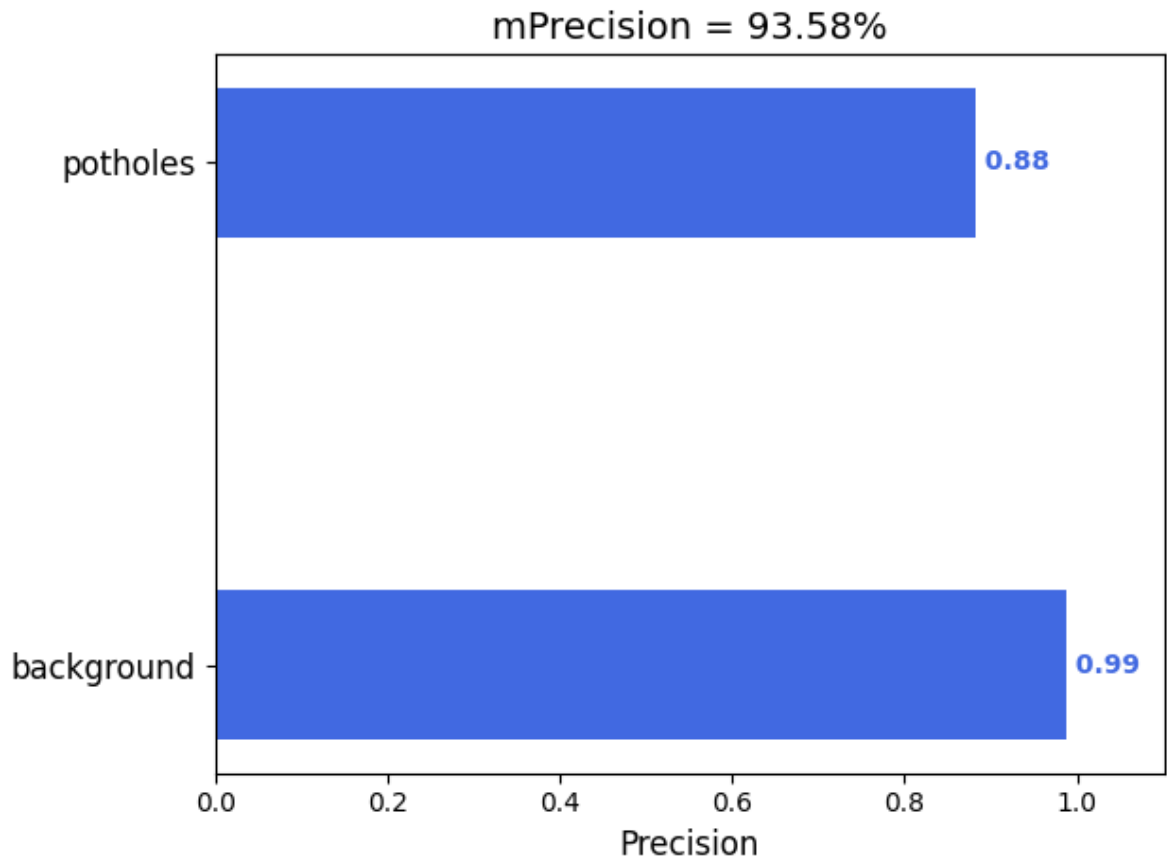


图 18 Precision 计算结果

八、参考文献

- [1] 陈鹏 . 基于深度学习的道路坑洼检测 [D]. 上海师范大学,2020.DOI:10.27312/d.cnki.gshsu.2020.001756.
- [2]徐奕哲.基于 Resnet-50 的猫狗图像识别[J].电子制作,2019(16):44-45+55.DOI:10.16589/j.cnki.cn11-3571/tn.2019.16.016.
- [3]白芮,徐杨,王彬等.基于改进 YOLOv5s 的道路坑洼检测算法[J].计算机与现代化,2023(06):69-75.
- [4]李锐,王鑫,杨威朋等.计算机视觉在道路成像和坑洞检测中的应用研究综述[J].汽车工程师,2023(09):1-8.DOI:10.20104/j.cnki.1674-6546.20230308.
- [5]周涛,霍兵强,陆惠玲等.残差神经网络及其在医学图像处理中的应用研究[J].电子学报,2020,48(07):1436-1447.
- [6] 贾大勇 . 基于深度学习的道路坑洼语义分割研究 [D]. 宁夏大学,2023.DOI:10.27257/d.cnki.gnxhc.2022.000932.
- [7] 李稳稳 . 基于机器视觉的道路缺陷检测与识别系统设计 [D]. 福建工程学

院,2023.DOI:10.27865/d.cnki.gfgxy.2023.000250.

[8] 赵潇. 基于深度学习的路面坑洼检测系统研究与实现 [D]. 宁夏大学,2021.DOI:10.27257/d.cnki.gnxhc.2020.000227.

九、附录

9.1 U-Net 模型

该程序基于 Python3 编写

```
# LINUX JUPYTER
# !apt-get update
# !apt-get install -y libgl1-mesa-glx

# !pip config set global.index-url https://mirrors.aliyun.com/pypi/simple
# !pip install scikit-learn
# !pip install opencv-python
# !pip install ipyml

# LINUX JUPYTER
# !apt-get update
# !apt-get install -y libgl1-mesa-glx

# !pip config set global.index-url https://mirrors.aliyun.com/pypi/simple
# !pip install scikit-learn
# !pip install opencv-python
# !pip install ipyml

import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from tqdm import tqdm
import cv2
import os
import glob
import random
from IPython.display import clear_output
from model.unet_model import UNet

from sklearn.model_selection import KFold
```

```

class ISBI_Loader(Dataset):
    def __init__(self, data_path=None, data_lst=None, use_grey=True, use_augment=True):
        if data_path is not None:
            # 初始化函数，读取所有 data_path 下的图片
            self.data_path = data_path
            self.imgs_path = glob.glob(os.path.join(data_path, "images/*.jpg"))

        if data_lst is not None:
            # 直接载入
            self.imgs_path = data_lst

        # 预处理方法
        self.use_grey = use_grey
        self.use_augment = use_augment

    def augment(self, image, flipCode):
        # 使用 cv2.flip 进行数据增强，flipCode 为 1 水平翻转，0 垂直翻转，-1 水平+垂直
        # 翻转
        flip = cv2.flip(image, flipCode)
        return flip

    def __getitem__(self, index):
        # 根据 index 读取图片
        image_path = self.imgs_path[index]

        # 根据 image_path 生成 label_path
        label_path = image_path.replace("images", "mask")
        label_path = label_path.replace(".jpg", ".png")

        # 读取训练图片和标签图片
        image = cv2.imread(image_path)
        label = cv2.imread(label_path)
        image = cv2.resize(image, (512, 512))
        label = cv2.resize(label, (512, 512), interpolation=cv2.INTER_NEAREST)

        # 将数据转为单通道的图片

```

```

if self.use_grey:
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    label = cv2.cvtColor(label, cv2.COLOR_BGR2GRAY)

# 处理标签，将像素值为 255 的改为 1
if label.max() > 1:
    label = label / 255

# 随机进行数据增强，为 3 时不做处理
if self.use_augment:
    flipCode = random.choice([-1, 0, 1, 2, 3])

    if flipCode != 2 or flipCode != 3:
        image = self.augment(image, flipCode)
        label = self.augment(label, flipCode)
    if flipCode == 2:
        kernel_size = (random.choice([1, 2, 3]), random.choice([1, 2, 3]))
        sigma = random.choice([1, 2, 3])
        cv2.GaussianBlur(image, kernel_size, sigma)

    image = image.reshape(1, image.shape[0], image.shape[1])
    label = label.reshape(1, label.shape[0], label.shape[1])

    return image, label

def __len__(self):
    # 返回训练集大小
    return len(self.imgs_path)

print("ok")

from utils_files import *

# 设置模型参数
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```



```

# 创建 unet 网络实例
model = UNet(n_channels=1, n_classes=1).to(device)

# 续训练(Fine-tuning)权重加载
fine_tuning_weights_dir = str(get_current_path() / "blost_0.13408492505550385.pth")
# model.load_state_dict(torch.load(fine_tuning_weights_dir, map_location=device))

# 定义损失函数和优化器
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.RMSprop(
    model.parameters(), lr=0.00001, weight_decay=1e-8, momentum=0.9
)

print(model)

# 初始化图表

# 使用 qt5(vscode)
%matplotlib qt5
from utils_plot import Myplot

plot = Myplot(2, 2)

# plot.axes[1, 1].remove()

plt_epochs = []

plt_trainloss = []
plt_valloss = []

plt_trainacc = []
plt_valacc = []

plt_tpr = []
plt_fpr = []

plt_f1 = []

```

```

plot.save("./result/figure.png")
print("ok")

# 训练参数
epochs = 50
batch_size = 6
dataset_path = get_current_path() / "dataset" / "prepare"

# 最优数值
best_loss = float("inf")
best_accuracy = 0
best_f1 = 0

# 累计 epochs
accumulate_epochs = 0

# 创建 KFold 对象，将数据集划分为 K 个折叠
kf = KFold(n_splits=10)

# 获取训练集列表
k_fold_lst = get_file_list(dataset_path / "images")
k_fold_lst = [str(file_pathobj) for file_pathobj in k_fold_lst]

# 循环进行 K 次交叉验证
for fold_index, (train_index, val_index) in enumerate(kf.split(k_fold_lst)):
    # 根据索引划分训练集和验证集
    train_lst = [k_fold_lst[i] for i in train_index]
    val_lst = [k_fold_lst[i] for i in val_index]

    # 加载训练集
    isbi_tra_dataset = ISBI_Loader(data_lst=train_lst)
    train_loader = DataLoader(
        dataset=isbi_tra_dataset, batch_size=batch_size, shuffle=True
    )

    isbi_val_dataset = ISBI_Loader(data_lst=val_lst)

```

```

val_loader = DataLoader(
    dataset=isbi_val_dataset, batch_size=batch_size, shuffle=False
)

print(
    "fold:{:.0f} train_cnt:{:.0f} val_cnt:{:.0f}".format(
        fold_index, len(train_loader), len(val_loader)
    )
)

# 开始训练
for epoch in range(epochs):
    print(
        "Epoch: {:.0f} Accumulate epochs: {:.0f}".format(epoch, accumulate_epochs)
    )

    # 训练模式
    model.train()

    train_loss = 0.0

    train_acc = 0.0
    train_tptn = 0.0
    train_samples = 0.0

    # 按照 batch_size 开始训练
    for image, label in tqdm(train_loader):
        optimizer.zero_grad()

        # 将数据拷贝到 device 中
        image = image.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.float32)

        # 使用网络参数，输出预测结果
        pred = model(image)

        # 计算 loss

```

```

loss = criterion(pred, label)
train_loss += loss.item()

# 二值化
pred_binary = torch.zeros_like(pred)
pred_binary = torch.where(pred >= 0.5, 1, 0)

# 计算 ACC
train_tptn += (pred_binary == label).sum().item()
train_samples += (pred_binary == 1).sum().item() + (
    pred_binary == 0
).sum().item()
train_acc = train_tptn / train_samples

# 更新参数
loss.backward()
optimizer.step()

# 求 epoch 平均 loss
train_loss /= len(train_loader)

print(
    "[{:.0f}/{:.0f}] LOSS: {:.3f} ACC:{:.3f} TP:{:.0f} TOTAL:{:.0f}".format(
        epoch + 1,
        epochs,
        train_loss,
        train_acc,
        train_tptn,
        train_samples,
    )
)

# 验证模式
model.eval()

val_loss = 0.0

```

```

val_acc = 0.0
val_tptn = 0.0
val_samples = 0.0

tp = 0.0
fp = 0.0
tn = 0.0
fn = 0.0

with torch.no_grad():
    for val_image, val_label in tqdm(val_loader):
        val_image = val_image.to(device=device, dtype=torch.float32)
        val_label = val_label.to(device=device, dtype=torch.float32)

        # 预测
        val_pred = model(val_image)

        # 计算 loss
        loss = criterion(val_pred, val_label)
        val_loss += loss.item()

        # 二值化
        val_pred[val_pred >= 0.5] = 1
        val_pred[val_pred < 0.5] = 0

        # 计算 TP、FP、TN、FN
        tp += ((val_pred == 1) & (val_label == 1)).sum().item()
        fp += ((val_pred == 1) & (val_label == 0)).sum().item()
        tn += ((val_pred == 0) & (val_label == 0)).sum().item()
        fn += ((val_pred == 0) & (val_label == 1)).sum().item()

    # 计算准确率
    val_acc = (tp + tn) / (tp + fp + tn + fn + 1e-8)

    # 计算精确率
    val_prec = tp / (tp + fp + 1e-8)

```

```

# 计算召回率
val_rec = tp / (tp + fn + 1e-8)

# 计算 F1 分数
val_f1 = 2 * (val_prec * val_rec) / (val_prec + val_rec + 1e-8)

# 求 epoch 平均 loss
val_loss /= len(val_loader)

print("vLoss:{:.3f}, vAccuracy:{:.3f}".format(val_loss, val_acc))

# 保存最优模型
if val_loss < best_loss:
    best_loss = val_loss
    torch.save(model.state_dict(), "./result/lost_{:.3f}.pth".format(val_loss))

if val_acc > best_accuracy:
    best_accuracy = val_acc
    torch.save(model.state_dict(), "./result/acc_{:.3f}.pth".format(val_acc))

if val_f1 > best_f1:
    best_f1 = val_f1
    torch.save(model.state_dict(), "./result/acc_{:.3f}.pth".format(val_f1))

torch.save(model.state_dict(), "./result/latest.pth")

# 画图
plt_trainloss.append(train_loss)
plt_trainacc.append(train_acc)
plt_valloss.append(val_loss)
plt_valacc.append(val_acc)
plt_f1.append(val_f1)
plt_epochs.append(accumulate_epochs)
plot.plot_learning_curve(0, 0, plt_epochs, plt_trainloss, plt_valloss)
plot.plot_accuracy_curve(0, 1, plt_epochs, plt_trainacc, plt_valacc)
plot.plot_f1_epoch_score_curve(1, 0, plt_epochs, plt_f1)
plot.plot_confusion_matrix(1, 1, tp, fp, tn, fn)

```

```

        plot.fresh()
        plot.save("./result/figure.png")

        accumulate_epochs += 1

import os
from tqdm import tqdm
from utils_metrics import compute_mIoU, show_results
import glob
import numpy as np
import torch
import os
import cv2
from model.unet_model import UNet

from utils_files import *

def unet_predict(test_dir, pred_dir):
    if not os.path.exists(pred_dir):
        os.makedirs(pred_dir)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # 加载网络，图片单通道，分类为 1。
    net = UNet(n_channels=1, n_classes=1)

    # 将网络拷贝到 device 中
    net.to(device=device)

    # 加载模型参数
    net.load_state_dict(torch.load("best_model.pth", map_location=device))

    # 测试模式
    net.eval()
    print("Load model done")

    img_names = os.listdir(test_dir)

```



```

image_ids = [image_name.split(".")[0] for image_name in img_names]

for image_id in tqdm(image_ids):
    # 获取文件列表
    image_path = os.path.join(test_dir, image_id + ".jpg")
    img = cv2.imread(image_path)
    origin_shape = img.shape

    # 转为灰度图
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, (512, 512))

    # 转为 batch 为 1，通道为 1，大小为 512*512 的数组
    img = img.reshape(1, 1, img.shape[0], img.shape[1])

    # 转为 tensor
    img_tensor = torch.from_numpy(img)

    # 将 tensor 拷贝到 device 中
    img_tensor = img_tensor.to(device=device, dtype=torch.float32)

    # 预测
    pred = net(img_tensor)

    # 提取结果
    pred[pred >= 0.5] = 255
    pred[pred < 0.5] = 0

    pred = np.array(pred.data.cpu()[0])[0]
    pred = cv2.resize(
        pred,
        (origin_shape[1], origin_shape[0]),
        interpolation=cv2.INTER_NEAREST,
    )

    cv2.imwrite(os.path.join(pred_dir, image_id + ".png"), pred)

```

```

print("Get predict result done")

dataset_path = get_current_path() / "dataset" / "test"

UNET_predict(test_dir=str(dataset_path / "images"), pred_dir=str(dataset_path / "pred"))

from utils_metrics import compute_mIoU, show_results
from utils_files import *
dataset_path = get_current_path() / "dataset" / "prepare"

gt_dir = str(dataset_path / "mask")
pred_dir = str(dataset_path / "pred")
test_dir = str(dataset_path / "images")
result_dir = str(dataset_path / "result")

img_names = os.listdir(test_dir)
image_ids = [image_name.split(".")[0] for image_name in img_names]

num_classes = 2
name_classes = ["background", "potholes"]

print("Get mIoU")
print(gt_dir)
print(pred_dir)
print(num_classes)
print(name_classes)
hist, IoUs, PA_Recall, Precision = compute_mIoU(
    gt_dir, pred_dir, image_ids, num_classes, name_classes
)
print("Get mIoU done.")
show_results(result_dir, hist, IoUs, PA_Recall, Precision, name_classes)

```

9.2 ResNet-50 结构（代码生成）

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```

```

track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(2): Bottleneck(
    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
)
(layer2): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(

```

```

        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
)
(1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,

```

```

track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
)
(1): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

```

```

        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )

```

```

    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),

```



```

bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

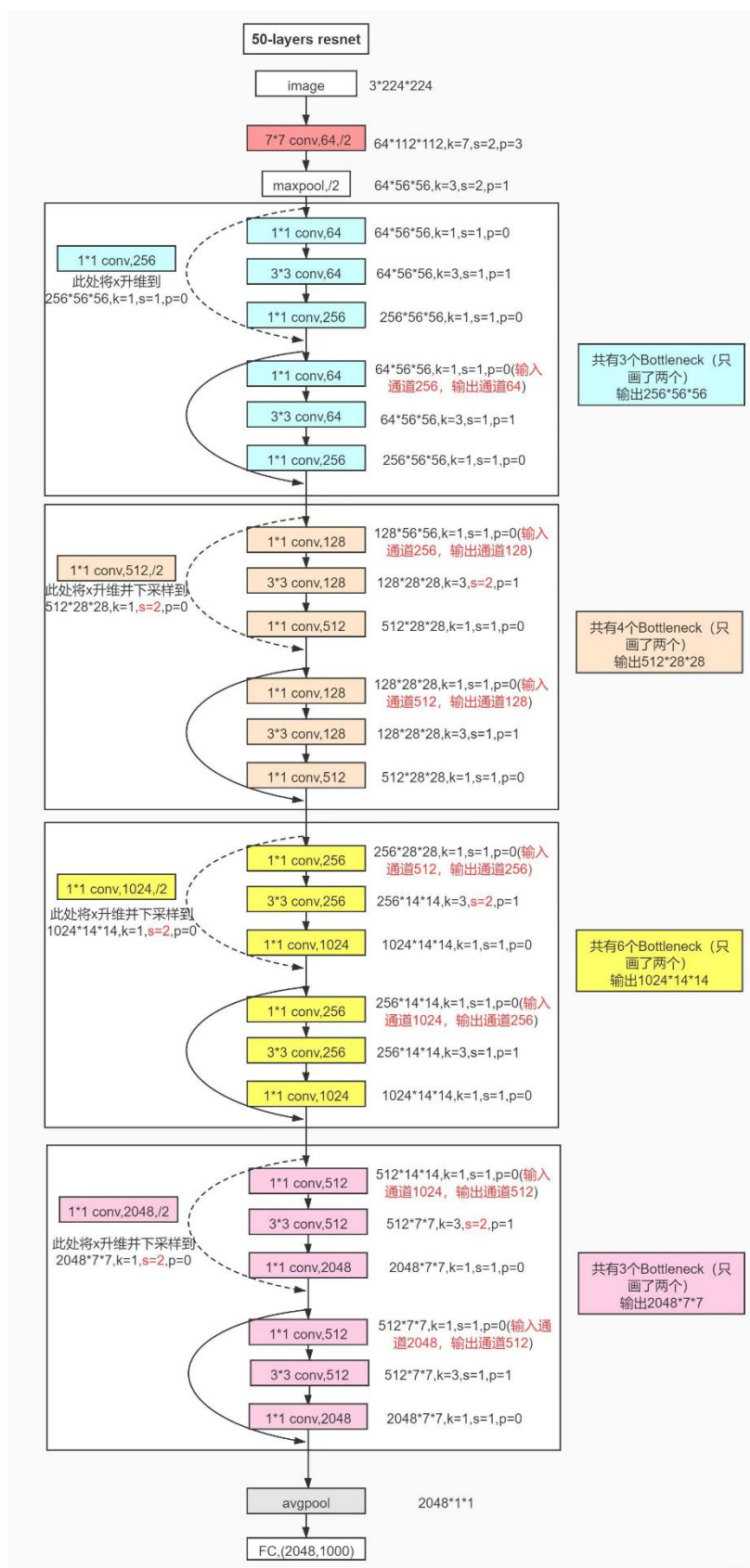
```

```

        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Sequential(
  (0): Linear(in_features=2048, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=256, out_features=128, bias=True)
  (4): ReLU()
  (5): Dropout(p=0.5, inplace=False)
  (6): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (7): Linear(in_features=128, out_features=64, bias=True)
  (8): ReLU()
  (9): Dropout(p=0.5, inplace=False)
  (10): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (11): Linear(in_features=64, out_features=2, bias=True)
)
)

```

9.3 ResNet-50 模型结构示意图



9.4 数据集标注实例

使用 labeling 标注的其中一张图像的 XML 实例

```
<annotation>
  <folder>images</folder>
  <filename>02oxhx7t.jpg</filename>
  <path>C:\Users\19156\Desktop\MATHORCUP2\code\dataset\test\images\02oxhx7t.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>420</width>
    <height>240</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>potholes</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>221</xmin>
      <ymin>128</ymin>
      <xmax>360</xmax>
      <ymax>220</ymax>
    </bndbox>
  </object>
</annotation>
```

9.5 ResNet-Unet 模型实现

该程序基于 Python3 编写

```
import torch
from torch import nn
```

```

from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from tqdm import tqdm
import numpy as np
import cv2
import os
import glob
import random
from IPython.display import clear_output
from model.resnet_model import Resnet_Unet

from sklearn.model_selection import KFold

class ISBI_Loader(Dataset):
    def __init__(self, data_path=None, data_lst=None, use_augment=True):
        if data_path is not None:
            # 初始化函数，读取所有 data_path 下的图片
            self.data_path = data_path
            self.imgs_path = glob.glob(os.path.join(data_path, "images/*.jpg"))

        if data_lst is not None:
            # 直接载入
            self.imgs_path = data_lst

        # 预处理方法
        self.use_augment = use_augment

    def augment(self, image, flipCode):
        # 使用 cv2.flip 进行数据增强，flipCode 为 1 水平翻转，0 垂直翻转，-1 水平+垂直
        # 翻转
        flip = cv2.flip(image, flipCode)
        return flip

    def __getitem__(self, index):
        # 根据 index 读取图片
        image_path = self.imgs_path[index]

```

```

# 根据 image_path 生成 label_path
label_path = image_path.replace("images", "mask")
label_path = label_path.replace(".jpg", ".png")

# 读取训练图片和标签图片
image = cv2.imread(image_path)
label = cv2.imread(label_path)
image = cv2.resize(image, (512, 512))
label = cv2.resize(label, (512, 512), interpolation=cv2.INTER_NEAREST)

# label 转为单通道
label = cv2.cvtColor(label, cv2.COLOR_BGR2GRAY)

# 处理标签，将像素值为 255 的改为 1
if label.max() > 1:
    label = label / 255

# 随机进行数据增强，为 3 时不做处理
if self.use_augment:
    flipCode = random.choice([-1, 0, 1])

    if flipCode != 2:
        image = self.augment(image, flipCode)
        label = self.augment(label, flipCode)
    elif flipCode == 2:
        kernel_size = (random.choice([1, 3]), random.choice([1, 3]))
        sigma = random.choice([1, 2, 3])
        image = cv2.GaussianBlur(image, kernel_size, sigma)

# 转换为 tensor 并标准化
transform = transforms.Compose(
    [
        transforms.ToTensor(),
        # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)

```

```

        image_tensor = transform(image)
        # label_tensor = transform(label)

        label_tensor = label.reshape(1, label.shape[0], label.shape[1])

        return image_tensor, label_tensor

    def __len__(self):
        # 返回训练集大小
        return len(self.imgs_path)

print("ok")

from utils_files import *

# 设置模型参数
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 创建 resnet 网络实例
model = Resnet_Unet(BN_enable=True, resnet_pretrain=True).to(device)

# 续训练(Fine-tuning)权重加载
fine_tuning_weights_dir = str(get_current_path() / "blost_0.13408492505550385.pth")
# model.load_state_dict(torch.load(fine_tuning_weights_dir, map_location=device))

# 定义损失函数和优化器
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.RMSprop(
    model.parameters(), lr=0.00001, weight_decay=1e-8, momentum=0.9
)

print(model)

# 初始化图表

# 使用 qt5(vscode)

```

```
# %matplotlib qt5

# 使用 widget(linux jupyter)
%matplotlib widget

from utils_plot import Myplot

plot = Myplot(2, 2)

# plot.axes[1, 1].remove()

plt_epochs = []

plt_trainloss = []
plt_valloss = []

plt_trainacc = []
plt_valacc = []

plt_tpr = []
plt_fpr = []

plt_f1 = []

plot.save("./result/figure.png")
print("ok")

# 训练参数
epochs = 50
batch_size = 4
dataset_path = get_current_path() / "dataset" / "prepare"

# 最优数值
best_loss = float("inf")
best_accuracy = 0
best_f1 = 0
```



```

# 累计 epochs
accumulate_epochs = 0

# 创建 KFold 对象，将数据集划分为 K 个折叠
kf = KFold(n_splits=10)

# 获取训练集列表
k_fold_lst = get_file_list(dataset_path / "images")
k_fold_lst = [str(file_pathobj) for file_pathobj in k_fold_lst]

# 循环进行 K 次交叉验证
for fold_index, (train_index, val_index) in enumerate(kf.split(k_fold_lst)):
    # 根据索引划分训练集和验证集
    train_lst = [k_fold_lst[i] for i in train_index]
    val_lst = [k_fold_lst[i] for i in val_index]

    # 加载训练集
    isbi_tra_dataset = ISBI_Loader(data_lst=train_lst)
    train_loader = DataLoader(
        dataset=isbi_tra_dataset, batch_size=batch_size, shuffle=True
    )

    isbi_val_dataset = ISBI_Loader(data_lst=val_lst)
    val_loader = DataLoader(
        dataset=isbi_val_dataset, batch_size=batch_size, shuffle=False
    )

    print(
        "fold:{:.0f} train_cnt:{:.0f} val_cnt:{:.0f}".format(
            fold_index, len(train_loader), len(val_loader)
        )
    )

# 开始训练
for epoch in range(epochs):
    print(
        "Epoch: {:.0f} Accumulate epochs: {:.0f}".format(epoch, accumulate_epochs)
    )

```

```

)

# 训练模式
model.train()

train_loss = 0.0

train_acc = 0.0
train_tptn = 0.0
train_samples = 0.0

# 按照 batch_size 开始训练
for image, label in tqdm(train_loader):
    optimizer.zero_grad()

    # 将数据拷贝到 device 中
    image = image.to(device=device, dtype=torch.float32)
    label = label.to(device=device, dtype=torch.float32)

    # print("image:", image.shape)

    # 使用网络参数，输出预测结果
    pred = model(image)

    # print("pred:", pred.shape, " label:", label.shape)

    # 计算 loss
    loss = criterion(pred, label)
    train_loss += loss.item()

    # 二值化
    pred_binary = torch.zeros_like(pred)
    pred_binary = torch.where(pred >= 0.5, 1, 0)

    # 计算 ACC
    train_tptn += (pred_binary == label).sum().item()
    train_samples += (pred_binary == 1).sum().item() + (

```

```

        pred_binary == 0
    ).sum().item()
    train_acc = train_tptn / train_samples

    # 更新参数
    loss.backward()
    optimizer.step()

# 求 epoch 平均 loss
train_loss /= len(train_loader)

print(
    "[{:.0f}/{:.0f}] LOSS: {:.3f} ACC:{:.3f} TP:{:.0f} TOTAL:{:.0f}".format(
        epoch + 1,
        epochs,
        train_loss,
        train_acc,
        train_tptn,
        train_samples,
    )
)

# 验证模式
model.eval()

val_loss = 0.0

val_acc = 0.0
val_tptn = 0.0
val_samples = 0.0

tp = 0.0
fp = 0.0
tn = 0.0
fn = 0.0

with torch.no_grad():

```

```

for val_image, val_label in tqdm(val_loader):

    val_image = val_image.to(device=device, dtype=torch.float32)
    val_label = val_label.to(device=device, dtype=torch.float32)

    # 预测
    val_pred = model(val_image)

    # 计算 loss
    loss = criterion(val_pred, val_label)
    val_loss += loss.item()

    # 二值化
    val_pred[val_pred >= 0.5] = 1
    val_pred[val_pred < 0.5] = 0

    # 计算 TP、FP、TN、FN
    tp += ((val_pred == 1) & (val_label == 1)).sum().item()
    fp += ((val_pred == 1) & (val_label == 0)).sum().item()
    tn += ((val_pred == 0) & (val_label == 0)).sum().item()
    fn += ((val_pred == 0) & (val_label == 1)).sum().item()

    # 计算准确率
    val_acc = (tp + tn) / (tp + fp + tn + fn + 1e-8)

    # 计算精确率
    val_prec = tp / (tp + fp + 1e-8)

    # 计算召回率
    val_rec = tp / (tp + fn + 1e-8)

    # 计算 F1 分数
    val_f1 = 2 * (val_prec * val_rec) / (val_prec + val_rec + 1e-8)

    # 求 epoch 平均 loss
    val_loss /= len(val_loader)

    print("vLoss:{:.3f}, vAccuracy:{:.3f}".format(val_loss, val_acc))

```

```

# 保存最优模型
if val_loss < best_loss - 0.05:
    best_loss = loss
    torch.save(
        model.state_dict(),
        "./result/lost_{:.3f}_k{:.0f}_e{:.0f}.pth".format(
            val_loss, fold_index, accumulate_epochs
        ),
    )

# if val_acc > best_accuracy:
#     best_accuracy = val_acc
#     torch.save(
#         model.state_dict(),
#         "./result/acc_{:.3f}_k{:.0f}_e{:.0f}.pth".format(
#             val_acc, fold_index, accumulate_epochs
#         ),
#     )

if val_f1 > best_f1:
    best_f1 = val_f1
    torch.save(
        model.state_dict(),
        "./result/f1_{:.3f}_k{:.0f}_e{:.0f}.pth".format(
            val_f1, fold_index, accumulate_epochs
        ),
    )

torch.save(model.state_dict(), "./result/latest.pth")

# 画图
plt_trainloss.append(train_loss)
plt_trainacc.append(train_acc)
plt_valloss.append(val_loss)
plt_valacc.append(val_acc)
plt_f1.append(val_f1)

```

```

plt_epochs.append(accumulate_epochs)
plot.plot_learning_curve(0, 0, plt_epochs, plt_trainloss, plt_valloss)
plot.plot_accuracy_curve(0, 1, plt_epochs, plt_trainacc, plt_valacc)
plot.plot_f1_epoch_score_curve(1, 0, plt_epochs, plt_f1)
plot.plot_confusion_matrix(1, 1, tp, fp, tn, fn)
plot.fresh()
plot.save("./result/figure.png")
accumulate_epochs += 1

import os
from tqdm import tqdm
from utils_metrics import compute_mIoU, show_results
import glob
import numpy as np
import torch
import os
import cv2
from model.unet_model import UNet

from utils_files import *

def unet_predict(test_dir, pred_dir):
    if not os.path.exists(pred_dir):
        os.makedirs(pred_dir)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # 加载网络，图片单通道，分类为 1。
    net = UNet(n_channels=1, n_classes=1)

    # 将网络拷贝到 device 中
    net.to(device=device)

    # 加载模型参数
    net.load_state_dict(torch.load("best_model.pth", map_location=device))

    # 测试模式
    net.eval()

```

```

print("Load model done")

img_names = os.listdir(test_dir)
image_ids = [image_name.split(".")[0] for image_name in img_names]

for image_id in tqdm(image_ids):
    # 获取文件列表
    image_path = os.path.join(test_dir, image_id + ".jpg")
    img = cv2.imread(image_path)
    origin_shape = img.shape

    # 转为灰度图
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, (512, 512))

    # 转为 batch 为 1，通道为 1，大小为 512*512 的数组
    img = img.reshape(1, 1, img.shape[0], img.shape[1])

    # 转为 tensor
    img_tensor = torch.from_numpy(img)

    # 将 tensor 拷贝到 device 中
    img_tensor = img_tensor.to(device=device, dtype=torch.float32)

    # 预测
    pred = net(img_tensor)

    # 提取结果
    pred[pred >= 0.5] = 255
    pred[pred < 0.5] = 0

    pred = np.array(pred.data.cpu()[0])[0]
    pred = cv2.resize(
        pred,
        (origin_shape[1], origin_shape[0]),
        interpolation=cv2.INTER_NEAREST,
    )

```

```

        cv2.imwrite(os.path.join(pred_dir, image_id + ".png"), pred)

    print("Get predict result done")

dataset_path = get_current_path() / "dataset" / "test"

UNET_predict(test_dir=str(dataset_path / "images"), pred_dir=str(dataset_path / "pred"))

from utils_metrics import compute_mIoU, show_results
from utils_files import *
dataset_path = get_current_path() / "dataset" / "prepare"

gt_dir = str(dataset_path / "mask")
pred_dir = str(dataset_path / "pred")
test_dir = str(dataset_path / "images")
result_dir = str(dataset_path / "result")

img_names = os.listdir(test_dir)
image_ids = [image_name.split(".")[0] for image_name in img_names]

num_classes = 2
name_classes = ["background", "potholes"]

print("Get mIoU")
print(gt_dir)
print(pred_dir)
print(num_classes)
print(name_classes)
hist, IoUs, PA_Recall, Precision = compute_mIoU(
    gt_dir, pred_dir, image_ids, num_classes, name_classes
)
print("Get mIoU done.")
show_results(result_dir, hist, IoUs, PA_Recall, Precision, name_classes)

```