队伍编号	MCB2302586
题号	A

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

摘要

本文构建一个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。使用**通道注意力机制**与**迁移学习**预训练的卷积神经网络实现坑洼图像分类任务，同时使用改进的**循环生成式对抗网络**进行**伪样本生成**来扩充训练数据，增强模型的鲁棒性与泛化能力。

对于问题一，首先使用直方图均衡化**突出图像中坑洼细节特征**，然后分别使用**水平翻转**、**高斯模糊**进行数据增广以扩充训练集。为了进一步丰富模型鲁棒性，本文使用**改进的循环生成式对抗网络**根据原始样本生成**相同语义信息的伪样本**，进一步增广训练数据。具体来说，将预训练好的坑洼道路分类器进行**参数冻结**来代替循环生成式对抗网络的**判别器**，再使用**周期一致性损失函数**约束图像的语义变化。对于骨干模型的选择，考虑到过拟合与训练速率等因素，本文选择 **ResNet-18** 作为骨干识别模型，并且使用**迁移学习**加载 ImageNet 数据集的预训练权重。考虑到训练数据中存在**类别不平衡**问题，本文在原始 ResNet-18 结构上添加**通道注意力机制**，让模型能有效提取图像通道信息。对于模型训练的配置，我们使用 Adam 作为网络参数优化器，并设置初始学习率为 0.0001，使用**余弦退火算法**来调整学习率。

对于问题二，首先对题给数据以 8：2 的比例划分训练集与测试集，为确保模型性能不依赖于特定数据分布，采用 **K− 折交叉验证方法**测试性能。由于数据样本整体呈现类别不平衡，为了全面评估模型性能，本文选取**平均准确率**、**正负样本召回率**、**F1 分数**、**AUC 指标**等性能指标，综合评价模型性能；在模型的具体评估中，将**传统机器学习模型和深度学习模型**进行对比实验，验证了深度学习模型的稳定性和准确性，再经过**模型速度评估**选择出 ResNet-18 作为骨干模型；最后，然后对图像增强方法与骨干模型中的通道注意力机制模块进行消融实验，检验其对分类性能提升的重要性。最后，对伪样本图像增广方法进行评估，强有力地证明了本文模型具备出色的鲁棒性和卓越的泛化能力。最终测试得到**平均准确率为 97.02%，正常与坑洼道路召回率分别为 98.67% 与 82.85%，F1-Score 为 0.86，AUC 为 0.90，在单张 RTX3060 显卡上训练用时为 262 秒**。

对于问题三，使用问题一建立通道注意力增强预训练过的 ResNet-18 在增广后数据集上训练 30 轮，对 4942 张测试集图像进行图像分类，结果保存在”test_result.csv”。

关键字：**坑洼道路分类　计算机视觉　伪样本生成　　通道注意力**

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：结合题目给出的图像文件，提取训练数据集中的图像特征，建立一个识别率高、速度快、分类准确的模型，用于识别图像中的道路是正常或坑洼。

问题二：对问题 1 中构建的模型进行训练，从不同维度进行模型评估。

问题三：利用已经训练好的模型来识别测试集中的坑洼图像，并将识别结果放到“test_result.csv”文件中。

二、 问题分析

对于问题一，题目要求建立识别率与速度兼具的模型，首先考虑对图像进行数据预处理，突出图像中的坑洼道路特征。由于题给数据量较少，还可以进行水平翻转垂直翻转等增广手段扩充数据样本。然后，考虑到深度学习与卷积神经网络在图像处理领域的高性能，可以使用 ResNet 等经典结构完成坑洼道路分类任务。为了加快训练速度，使用迁移学习技术加载 ImageNet 上的预训练权重，同时添加通道注意力机制让模型更加关注通道间的细节特征。

普通的数据增广效果有限，为了进一步增强模型的鲁棒性与泛化能力，可以考虑进行更为复杂的直接改变图像特征结构的数据增广，但需同时保持图像语义信息不变。于是考虑对循环生成式对抗网络进行改进，将其判别器改为在坑洼道路数据集上预训练过的网络，将其输出映射为概率值来充当判别器，促使生成器生成语义信息一致但特征空间有所改变的伪样本。将伪样本添加到原始样本中训练模型，以提高模型的鲁棒性。

对于问题二，题目要求根据给出的 301 张图像对问题一中构建的坑洼图像分类模型进行训练，并从不同维度对模型进行评估。为了有效验证模型性能，我们首先可以按照 8:2 的比例划分训练集与测试集，使用 $K-$ 折交叉验证的方法来保证验证结果的可靠性。为了应对训练样本类别不平衡的问题，可选取平均准确率、召回率、F1 分数、AUC 和作为模型性能评价指标，对模型进行详细全面的评价。

为了验证模型的有效性，我们可以从不同角度出发针对上一问建立的模型来对模型的内部模块进行评价，包括评估传统机器学习与深度学习性能、评估模型速度、评估通道注意力机制有效性、评估图像增强有效性。除此之外，我们对通道注意力机制模块和伪样本图像增广模块分别进行消融实验，以检验其能否有效提升模型的鲁棒性和泛化能力。

对于问题三，我们只需利用建立好的 ResNet-18 模型，加载预训练权重后添加 SE 模块，然后对图像进行预处理与增广，得到最后训练好的模型，对题目所给的测试数据进行测试即可。

三、 模型假设

- 假设题给数据中各个图像与标签对应正确，忽略错误标注的情况。
- 本文已经固定模型参数随机种子，对比实验与消融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
f_ϕ	特征提取器参数
f_W	分类器参数
$\mathbf{W}$	卷积核参数
b	神经网络中的偏置值
η	学习率
N_{epoch}	训练伦次
R_n	正常道路图像召回率
R_p	坑洼道路图像召回率

五、 模型建立与求解

5.1 问题一：建立道路坑洼分类模型

本文建立的坑洼道路图像分类模型主要流程包括图像预处理、通用数据增广策略、建立以卷积神经网络 ResNet-18 为骨干的分类模型。然后，考虑到训练数据的类别不平衡问题，对 ResNe-18 进行通道注意力增强并替换激活函数，以提高模型分类能力，然后改进循环生成式对抗网络来生成伪样本进行高层次的数据增广以改进模型的鲁棒性，具体方法流程如图 1所示。

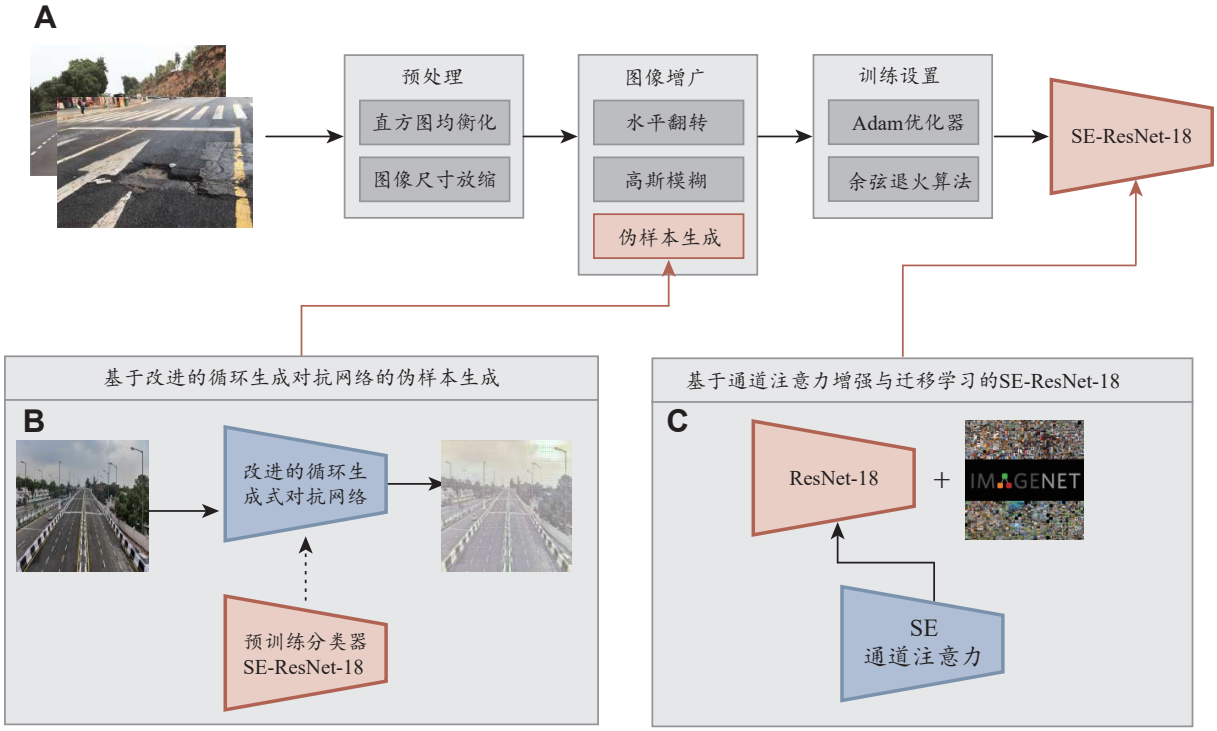


图 1　基于伪样本生成增广与通道注意力增强的坑洼道路图像分类模型框架图

5.1.1 图像预处理

原始路面图像数据质量欠佳，各种方式获取的正常路面图像与坑洼路面图像存在许多噪声，例如不均匀的光照、照片模糊、错综复杂的路面及周边环境等情况，如图 2 所示。过低的图像质量极易对后续的图像学习与分析建模过程造成影响，因此有必要对题目所给的原始路面图像进行图像预处理，有效提高图像质量。

直方图均衡化　直方图均衡化是一种简单有效的图像增强方法，通过改变图像的直方图分布来改变图像中各像素点的色彩值，适用于图像色彩特征较复杂的坑洼道路分类 [1]。主要用于增强动态范围偏小的图像对比度，图像增强效果如图 3 所示。

图 2　质量不均匀的路面图像，包括清晰度低、阴影遮挡、带水印噪声、光照不均衡等。

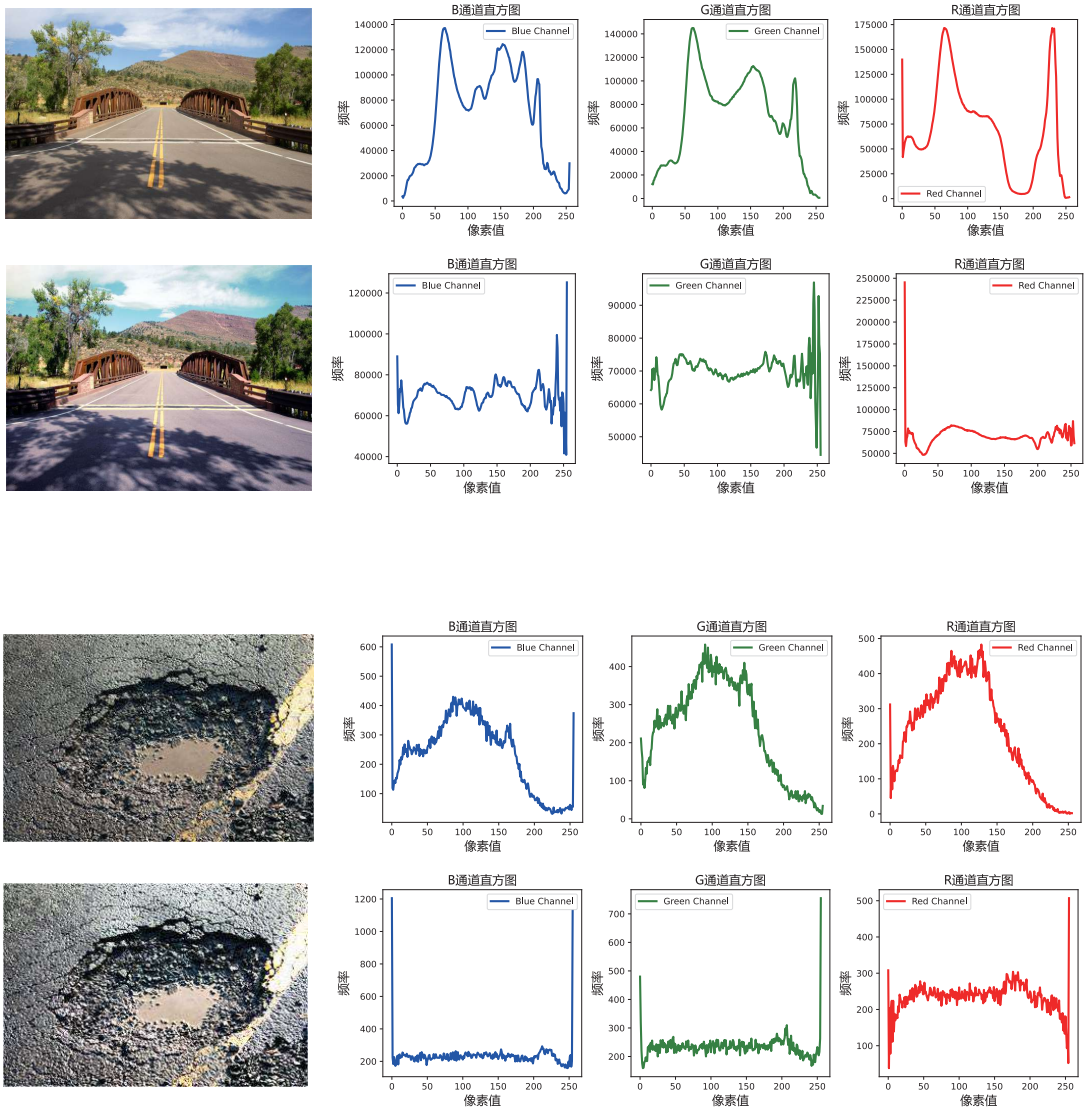


图 3　正常道路与坑洼道路 RGB 图像直方图均衡化示意图

对于灰度图像的直方图均衡化只需均衡灰度值即可，对于 RGB 彩色图像则需将三个颜色通道同时进行处理。彩色 RGB 图像的直方图均衡化流程如下算法 1 所示。

图像尺寸放缩　由于卷积神经网络中卷积核需要以固定步长在特征图中进行卷积，且需输出相同尺寸的特征图，因此在输入到网络之前需保证所有图像尺寸相同。为方便网络进行图像分析，本文将所有图像大小统一放缩为 224×224。

Algorithm 1: 针对坑洼道路分类的 RGB 直方图均衡化

Data: 原始训练样本 $\mathcal{D}(X)$

Result: 直方图均衡化后的样本 $\mathcal{D}'(X)$

```
// 从数据集选取训练样本;
for n = 1 to N(D) do
    所有通道像素转化为数组;
    for channel to (R, G, B) do
        统计每个色彩级别像素出现的频次;
        将原始图像中的像素频次归一化到 (0, 255) 之间;
    合并 3 个 channel 的像素值;
return 直方图均衡化后的样本 D'(X)
```

5.1.2 通用数据增广

图像增广是一种用于增加训练样本的技术特别适用于计算机视觉和深度学习任务。它的目的是通过对原始图像进行各种变换，生成多个不同版本的图像，从而扩展训练数据集，提高模型的性能和泛化能力。图像增广可以帮助模型更好地适应各种不同的视觉条件和变化。

为丰富训练样本的特征结构，同时扩充训练样本数量，本文对训练数据进行以下数据增广操作，包括水平翻转、高斯模糊，实际效果如图 4所示。

图 4　图像增广展示。（A）原图像。（B）水平翻转后图像。（C）高斯模糊后图像。

5.1.3 卷积神经网络

建模动机　计算机视觉（Computer Vision, CV）是人工智能领域的一个分支，旨在使计算机系统能够模拟人类视觉系统的功能，从图像和视频数据中获取、解释和理解信息。本题的坑洼道路分类问题属于计算机视觉领域的图像识别问题。

在坑洼道路分类问题中，传统的特征工程往往无法达到令人满意的效果。首先，坑洼道路的路面情况非常复杂，具有较高的特征维度，而传统特征工程通常只能捕捉低级

像素信息，难以感知坑洼路面的高级语义特征。其次，使用传统特征工程去识别坑洼路面需要领域专家手工设计特征提取模板，依赖于代价高昂的人工成本。并且坑洼路面识别算法通常需要应用在各种各样的检测场景中，而传统特征工程方法对光照、尺度变换、图像拍摄角度等信息较为敏感，在实际应用场景中鲁棒性往往较差。

相比起传统特征工程，卷积神经网络（Convolution Neural Network, CNN）能够通过对图像与标签数据的监督学习自动从图像中完成特征提取与分类任务，无需手工设计特征，并且深层的卷积神经网络能够凭借其庞大的参数量从图像中提取高维语义信息，完成对诸如坑洼道路等高维特征识别任务。卷积神经网络的图像识别过程如图 5 所示。在本文中，道路坑洼的类型多种多样，手工设计模板较为复杂，且测试样本有几千张，模型需要面临很多不同的道路场景，对模型的鲁棒性要求较高，故本文采用卷积神经网络执行坑洼道路分类任务。

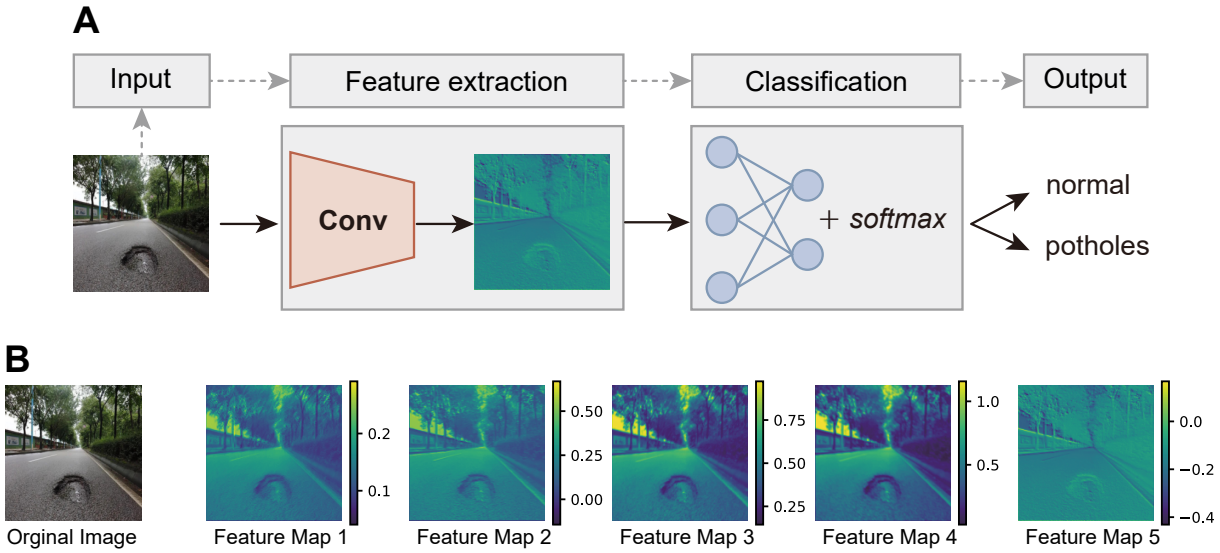


图 5　卷积神经网络识别图像整体框架。（A）使用卷积神经网络完成道路坑洼图像分类任务流程图；（B）卷积层提取图像特征的工作示意图。

卷积神经网络图像识别流程可概括为式 (1)。其中 $\mathbf{X}$ 表示输入样本，$f_\phi(.)$ 表示卷积神经网络的特征提取层，$f_\phi(\mathbf{X})$ 即为样本 $\mathbf{X}$ 经过特征提取层提取出的特征图，$f_\phi(W)$ 表示分类层，$f_W(f_\phi(\mathbf{X}))$ 即为特征图经分类器后输出的特征向量，σ 表示 Softmax 函数，$p(\mathbf{X})$ 表示样本 $\mathbf{X}$ 的最终类别概率。

$$\mathbf{X} \longrightarrow f_\phi(\mathbf{X}) \longrightarrow f_W(f_\phi(\mathbf{X})) \longrightarrow \sigma(f_W(f_\phi(\mathbf{X}))) \longrightarrow p(\mathbf{X}) \tag{1}$$

图像输入到卷积神经网络中，首先经过网络中的卷积层，以提取出原图像的高维特征图。然后，通过池化操作对其进行降维，引入激活函数增强数据的非线性特征，将高维特征图压缩为一个固定大小的特征向量。最后，将此特征向量输入到全连接层中，通过应用 softmax 函数映射为每个类别的概率分布，从而完成多类别分类任务。

卷积神经网络结构 卷积神经网络的完整结构如图 6 (a) 所示。一个能够完成端到端图像识别的卷积神经网络应该包含卷积层、池化层、激活函数、全连接层。

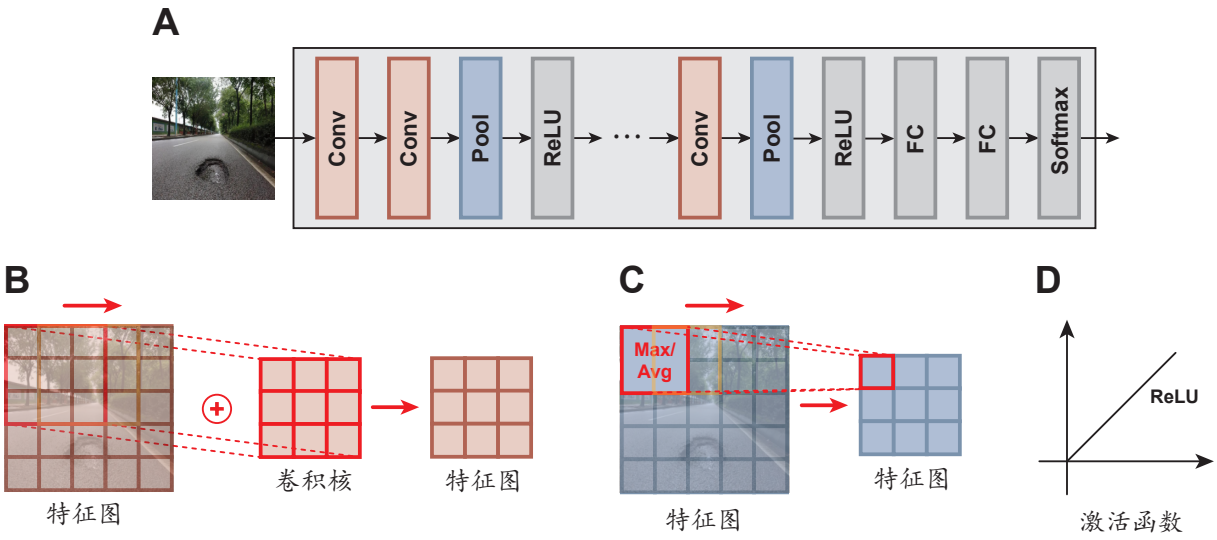


图 6 卷积神经网络工作示意图。(A) 卷积神经网络整体结构；(B) 卷积层结构；(C) 池化层结构；(D) 激活函数。

其中卷积层、池化层、激活函数完成特征提取，全连接层结合 Softmax 函数利用提取出的特征向量进行分类。每一层的功能如下所示：

- **卷积层：**在传统的全连接神经网络中，通常将输入图像直接拉伸成一维向量进行处理，不能有效提取图像的拓扑特征，从而导致图像的空间结构信息丢失。相比之下，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 6(b) 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 6(c) 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 6(d) 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。

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

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。
- **全连接层：**全连接层在卷积神经网络中通常出现在网络的末尾，负责将高层次的特征映射到输出空间。其主要作用是将卷积层和池化层的输出进行扁平化，并将它们连接到输出层以进行分类或回归等最终任务。
- **Softmax 函数：**Softmax 函数负责将全连接层输出的一维特征向量映射成概率，在图像分类任务中进行最后的类别判定。假设一个 N 分类的图像分类任务，对于一个输入样本 $\mathbf{X}$，其全连接层最后输出的特征向量表示为式 (7)。

$$\mathbf{X}^{'} = [x^{'}_1, x^{'}_2, \ldots, x^{'}_n] \tag{7}$$

其中 $x^{'}_i$ 表示为样本在第 i 个类上的特征值，通过 Softmax 函数将这些特征值被映射为每个类别的概率。softmax 函数如式 (8)所示。

$$\text{Softmax}(x^{'}_i) = \frac{\exp(x^{'}_i)}{\sum_{j=1}^{n} \exp(x^{'}_j)} \tag{8}$$

Softmax 函数将全连接层最后输出的特征值转化为概率分布，使每个类别的预测概率非负、相对权重准确表达，提高模型的可解释性和性能，确保分类结果合理。同时，提供有效的训练机制，有助于减小过拟合的风险，以平滑模型输出结果。

参数学习 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。

坑洼道路分类为二分类问题，即将所有图像分为正常道路与坑洼道路两类，对此，我们采用针对于如式 (9) 所示的二分类交叉熵函数作为模型的损失函数。

$$\mathcal{L} = \frac{1}{N}\sum_{i} -\left[y_i \cdot \log\left(p_i\right) + \left(1 - y_i\right) \cdot \log\left(1 - p_i\right)\right] \tag{9}$$

其中 y_i 表示样本 i 的标签，p_i 表示样本被预测为正常道路的概率。故卷积神经网络的训练流程如算法 2 所示。

Algorithm 2: 针对坑洼道路分类的卷积神经网络训练

Data: 训练样本与标注数据 $\mathcal{D}(X, Y)$

Result: 训练后的模型参数 f'_{ϕ}, f'_{W}

初始化网络模型参数 f_{ϕ}, f_{W}，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(x, y) **to** $\mathcal{D}$ **do**

    前向传播：计算模型的预测输出 $\hat{y}$;

    计算交叉熵损失函数：$\mathcal{L}$ = 损失函数$(y, \hat{y})$;

    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;

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

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

return 训练后的模型参数 f'_{ϕ}, f'_{W}

残差神经网路 卷积神经网络模型希望将模型设计地越来越深，因为随着层数加深模型可以拥有更大的参数量来拟合出性能更加优越的模型，因此早期卷积神经网络的研究一直在朝着加深模型深度的方向研究。

然而，随着网络层数的不断加深，由于梯度在网络每一层之间的传导会有信息损失，因此卷积神经网络的深层很难真正学习到完整的图像信息，极易发生梯度消失与梯度爆炸现象，发生网络退化。

为了解决上述的深层卷积神经网络的训练困难问题，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 7 所示。

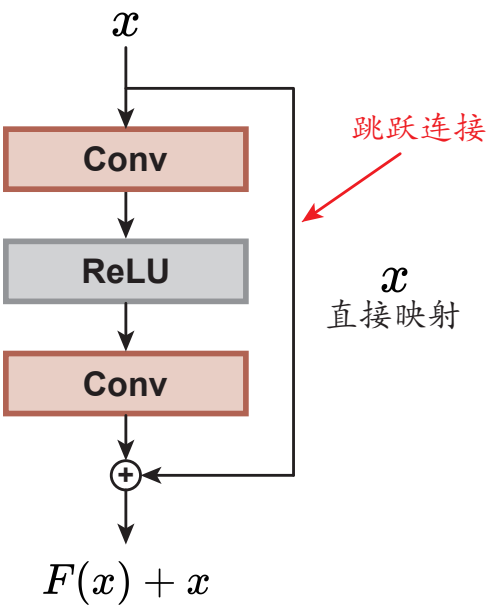


图 7　残差块与跳跃连接结构图

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 (10)所示。

$$F(x) = H(x) - x \tag{10}$$

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，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。

基于以上 ResNet 结构上的优势，同时考虑到题给数据量样本较少，为避免模型陷入过拟合，本文选择 ResNet-18 作为骨干模型。ResNet-34，ResNet-50 等模型层数过深，在本文较少的训练样本中容易陷入过拟合。

5.1.4 迁移学习

迁移学习（Transfer Learning）是一种机器学习技术，其核心思想是将一个领域中学到的知识（通常是模型的权重参数）应用到另一个相关领域的问题中。

ImageNet 是一个大规模的图像数据库，它包含来自各种类别的数百万张图像。ImageNet 数据库最知名的用途之一是作为深度学习和计算机视觉领域的基准数据集，特别是用于图像分类任务。

为加速模型训练，同时在数据有限的情况下提供更好的泛化能力，本文对网络模型加载在 ImageNet 上预训练的权重，然后再使用是给数据集信息进行微调。

5.1.5 通道注意力机制

卷积神经网络通常被看作是在局部感受野上，将空间信息与特征维度信息进行聚合的信息聚合体。如果能够通过学习的方式唉自动获取每个特征通道的重要程度，就可以更加有效已知对当前任务无用的特征。

SE 模块 [3] 能够有效地关注图像中各个通道特征之间的关系，它的结构与主要工作机制如图 8 所示。通道注意力机制主要分为三个步骤：压缩、激发、缩放。

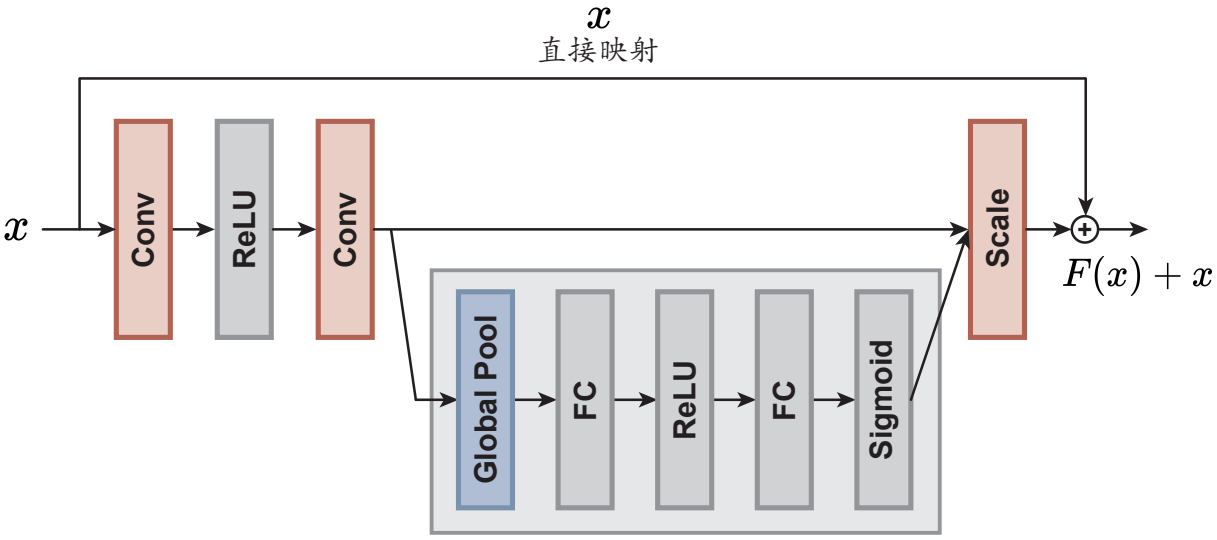


图 8　SE 模块工作流程图

在压缩阶段，SE 模块首先通过全局平均池化将每个特征值降到标量，形成 Squeeze 向量。在激发阶段，Squeeze 向量经过一个全连接网络学习通道权重，然后通过学习到的权重对每个通道进行重新加权。在缩放阶段，学习到的权重用来缩放特征图中的每个通道，从而产生经过调整的特征表示。

由于坑洼道路的图像数据较为复杂，网络很难提取到更丰富的语义信息，为了让网络更关注图像通道之间的联系与特征关系，本文在选取的 ResNet-18 模型中加入 SE 模块，形成 SE-ResNet-18 模型，为模型增添通道注意力机制，进一步优化模型性能。

5.1.6 超参数配置

构建完卷积神经网络模型之后，要进行深度学习训练还需进行超参数配置。

本文选择 Adam 作为参数优化器，并设置学习率 $\eta = 0.0001$。Adam（Adaptive Moment Estimation）是一种常用的梯度下降优化算法，它结合了梯度下降和动量方法，同时使用了自适应学习率的技巧，以加快模型参数的更新过程。Adam 优化算法的主要思想是根据每个参数的自适应学习率来更新参数，以便更好地适应不同参数的特性。

为使学习率在训练过程中的变化更加平滑，本文使用余弦退火算法（CosineAnnealingLR）作为学习率调度器，如式 (11)所示。

$$\eta(t) = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\pi \frac{t}{T_{max}}\right)\right) \tag{11}$$

其中 $\eta(t)$ 表示 t 时刻的学习率具体数值，η_{min} 和 η_{max} 分别表示最小学习率和初始学习率，T_{max} 表示周期数。在余弦退火算法中，学习率在每个周期内从较大的值线性降低到较小的值，然后再开始下一个周期。这个过程通过余弦函数的周期性变化来实现。

该策略基于余弦函数的周期性变化来降低学习率，有助于防止模型在训练过程中陷入局部最小值，并在训练后期更加精细地调整学习率。

5.1.7 基于循环生成式对抗网络的伪样本图像增广

增广动机 题中所给数据量为小样本数据，仅 301 张图像，且存在严重的类别不平衡问题，网络难以精准拟合出坑洼道路图像之间的空间特征。

因此，除了计算机视觉领域通用的数据增广方法之外，为避免大幅度改动图像的空间结构特征，同时丰富训练样本的特征多样性，以便能让模型训练地更加鲁棒，本文使用循环生成式对抗网络（Cycle Generative Adversarial Network, GAN）[4] 进行图像生成，产生与正常样本语义信息相似但特征空间存在差异的伪样本，使用伪样本来扩充训练数据，以丰富数据的特征多样性。

循环生成式对抗网路结构 循环生成式对抗网络结构如图所示，主要由两个生成器 G_{A2B} 和 G_{B2A} 与两个判别器 D_A 和 D_B 组成。生成器负责生成图像，判别器负责判别领域信息。

循环生成式对抗网络的主要作用是完成图像内容的域迁移。假设有两张来自于不同领域信息的图像 X_A 与 X_B，CycleGAN 的目标是生成一张 X_B^{fake}，X_B^{fake} 的语义信息与 X_A 一致，领域信息却与 X_B 一致，如图 9 所示。

X_A 首先输入进生成器 G_{A2B} 生成 $G_{A2B}(\mathrm{X}_A)$，将输入的图像送入判别器 D_B 并使用对抗损失函数 $\mathcal{L}_{GAN}$ 进行训练，G_{A2B} 能够生成 B 域风格的图像。$\mathcal{L}_{GAN}$ 如式 (12) 所示。

$$\begin{aligned}
&\mathcal{L}_{GAN} = \mathcal{L}_{\mathrm{GAN}}\left(G_{A2B}, D_B\right) + \mathcal{L}_{\mathrm{GAN}}\left(G_{B2A}, D_A\right) \\
&= \mathbb{E}_{y \sim p_{\mathrm{data}}(y)}\left[\log D_B(y)\right] + \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}\left[\log\left(1 - D_B(G_{A2B}(x))\right)\right] \\
&+ \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}\left[\log D_A(x)\right] + \mathbb{E}_{y \sim p_{\mathrm{data}}(y)}\left[\log\left(1 - D_A(G_{B2A}(y))\right)\right]
\end{aligned} \tag{12}$$

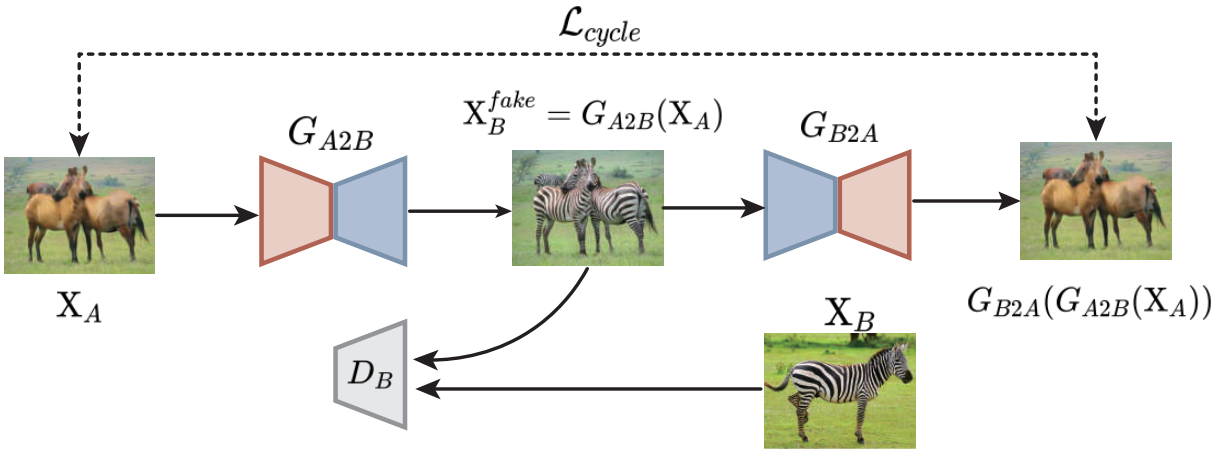


图 9　循环生成式对抗网络示意图

式 (12)表示了由域 A 到域 B 与域 B 到域 A 的两种过程，其中 x 与 y 是分别来自两个域的样本。生成器与判别器依此函数进行对抗训练。

但是仅使用 $\mathcal{L}_{GAN}$ 进行训练将会导致 X_A 虽然融入了域 B 的信息，但会丢失掉其原本的语义信息。因此，在循环生成式对抗神经网络中除了使用 $\mathcal{L}_{GAN}$ 约束领域信息转换之外还使用周期一致性损失函数 $\mathcal{L}_{cycle}$ 来约束样本的语义信息。$\mathcal{L}_{cycle}$ 如式 (13) 所示。

$$\begin{aligned}\mathcal{L}_{cycle} = &\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\|G_{B2A}(G_{A2B}(x)) - x\|_1\right] \\ &+ \mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\|G_{A2B}(G_{B2A}(y)) - y\|_1\right]\end{aligned} \tag{13}$$

为保持语义一致性，经 G_{A2B} 生成的 $G_{A2B}(\mathrm{X}_A)$ 再送入到生成器 G_{B2A} 中，将翻译后的域 B 图像重新翻译为原本的域 A，使用 L_1 距离来对比原始 X_A 与经过两次翻译后的 $G_{B2A}(G_{A2B}(x))$ 语义信息上的差异，约束语义信息的损失。

同时，循环生成式对抗网络使用 $\mathcal{L}_{identity}$ 来保证生成器仅完成领域信息的转换而不改变语义信息，如式 (14) 所示。

$$\mathcal{L}_{identity}(G_{A2B}, G_{B2A}) = \mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\|G_{A2B}(y) - y\|_1\right] + \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\|G_{B2A}(x) - x\|_1\right] \tag{14}$$

循环生成式对抗网络的整体损失函数如式 (15)所示：

$$\mathcal{L} = \mathcal{L}_{GAN} + \mathcal{L}_{cycle} + \mathcal{L}_{identity} \tag{15}$$

伪样本生成改进　原始的循环生成式对抗网络只能完成域迁移工作，即完成图像的领域翻译。而在本文的坑洼道路分类任务中，由于训练样本与测试样本较少，各项指标的高性能也很难反应模型的鲁棒性与泛化能力。因此本文希望借助循环生成式对抗网络来生成语义高度一致但模型却难以辨别的图像。

因此在本文中，循环生成式对抗网络要完成的不再是域迁移任务，而是在原有数据上生成模型难以辨别的伪样本，再借助伪样本来进行数据增广，提高模型的鲁棒性。

为此，本文首先在完整坑洼道路分类数据集 $\mathcal{D}$ 上预先训练一个 SE-ResNet-18，让此网络拥有基本地对正常道路与坑洼道路的分类能力，然后冻结 SE-ResNet-18 的参数，将其当做循环生成式对抗网络的判别器，以督促模型生成普通分类器难以辨别的样本，具体结构如图 10 所示。

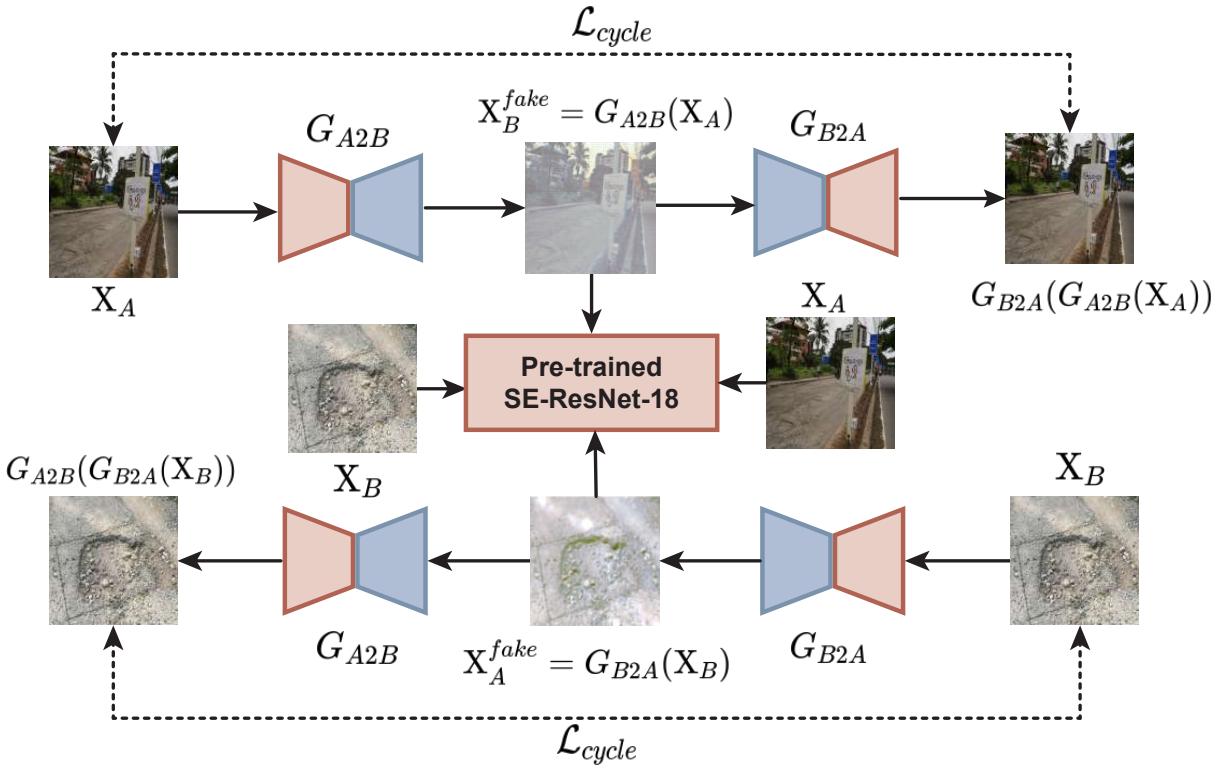


图 10　针对坑洼道路数据集伪样本生成改进的循环对抗网络结构图

依靠本文改进后的网络结构与训练方式，将预训练的 SE-ResNet-18 当做判别器，将网络的输出经过 Softmax 函数映射即可作为判别器分数，网络能够生成未经增广训练的一般分类器无法辨别的样本，并通过周期一致性损失函数又能有效地抑制住图像转译过程中的语义损失，从而生成语义信息高度一致但模型难以辨认的伪样本。本文基于改进循环生成式对抗网络的伪样本生成具体步骤如算法 3 和算法 4 所示。

从图 11 中可以看出，基于本文改进的循环生成式对抗网络生成的伪样本与原样本高度相似，但预训练的 SE-ResNet-18 分类网络却无法正确辨别，将伪样本扩充为训练样本后有助于增强模型鲁棒性。

5.2 问题二：多视角评价道路坑洼分类模型

面对训练样本类别不平衡问题，本文选取多个指标对模型进行性能评估，包括准确率，正负样本召回率，F1 分数，AUC。同时，为了验证模型中各模块是否能有效提升模型的鲁棒性和泛化能力，本文从多个视角对模型的各个模块进行大量的消融实验，验证模型的动机与有效性。具体步骤如图 12 所示。

Algorithm 3: 改进循环生成式对抗网络的训练过程

Data: 正道道路样本 $\mathcal{D}_A$ 与坑洼道路训练样本 $\mathcal{D}_B$

Result: 伪样本生成器 G'_{A2B}、G'_{B2A}

初始化网络模型参数 G_{A2B}, G_{B2A}，设置 $\eta, N_{epoch}, N_{batch}$;

将预训练的 SE-ResNet-18 设置为判别器 D;

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

  随机打乱训练数据 $\mathcal{D}_A, \mathcal{D}_B$;

  for 一个批次训练样本$(\mathrm{X}_A, \mathrm{X}_B)$ **to** $\mathcal{D}_A, \mathcal{D}_B$ **do**

    $\mathrm{X}_A \rightarrow G_{A2B} \rightarrow G_{A2B}(\mathrm{X}_A)$;

    $\mathrm{X}_B \rightarrow G_{B2A} \rightarrow G_{B2A}(\mathrm{X}_B)$;

    //具有坑洼道路分类能力的 SE-ResNet-18 进行判别;

    $G_{A2B}(\mathrm{X}_A) \rightarrow D \rightarrow D(G_{A2B}(\mathrm{X}_A))$;

    $G_{B2A}(\mathrm{X}_B) \rightarrow D \rightarrow D(G_{B2A}(\mathrm{X}_B)$;

    根据 D 分数分别计算 $\mathcal{L}_{GAN}$;

    $\mathcal{L}_{GAN}$ 反向传播更新 G_{A2B}, G_{B2A} 参数;

    //D 参数冻结，不进行参数更新;

    $G_{A2B}(\mathrm{X}_A) \rightarrow G_{B2A} \rightarrow G_{B2A}(G_{A2B}(\mathrm{X}_A))$;

    $G_{B2A}(\mathrm{X}_B) \rightarrow G_{A2B} \rightarrow G_{A2B}(G_{B2A}(\mathrm{X}_B))$;

    计算周期一致性损失函数 $\mathcal{L}_{cycle}(\mathrm{X}_A, G_{B2A}(G_{A2B}(\mathrm{X}_A))$;

    计算周期一致性损失函数 $\mathcal{L}_{cycle}(\mathrm{X}_B, G_{A2B}(G_{B2A}(\mathrm{X}_B))$;

    $\mathcal{L}_{cycle}$ 反向传播更新 G_{A2B}, G_{B2A} 参数;

return 训练后的模型参数 G'_{A2B}, G'_{B2A}

Algorithm 4: 正常道路与坑洼道路伪样本生成

Data: 正道道路样本 $\mathcal{D}_A$ 与坑洼道路训练样本 $\mathcal{D}_B$

Result: 伪样本数据集 $\mathcal{D}'_A$、$\mathcal{D}'_B$

加载训练好的 G'_{A2B}、G'_{B2A};

for 一个批次原始样本$(\mathrm{X}_A, \mathrm{X}_B)$ **to** $\mathcal{D}_A, \mathcal{D}_B$ **do**

  //使用原始样本生成伪样本;

  $\mathrm{X}_A \rightarrow \mathrm{G}_{A2B} \rightarrow \mathrm{X}_B^{fake}$;

  $\mathrm{X}_B \rightarrow \mathrm{G}_{B2A} \rightarrow \mathrm{X}_A^{fake}$;

  X_B^{fake} 添加到 $\mathcal{D}'_A$;

  X_A^{fake} 添加到 $\mathcal{D}'_B$;

return 伪样本数据集 $\mathcal{D}'_A$、$\mathcal{D}'_B$

图 11　本文改进的循环生成式对抗网络生成的伪样本

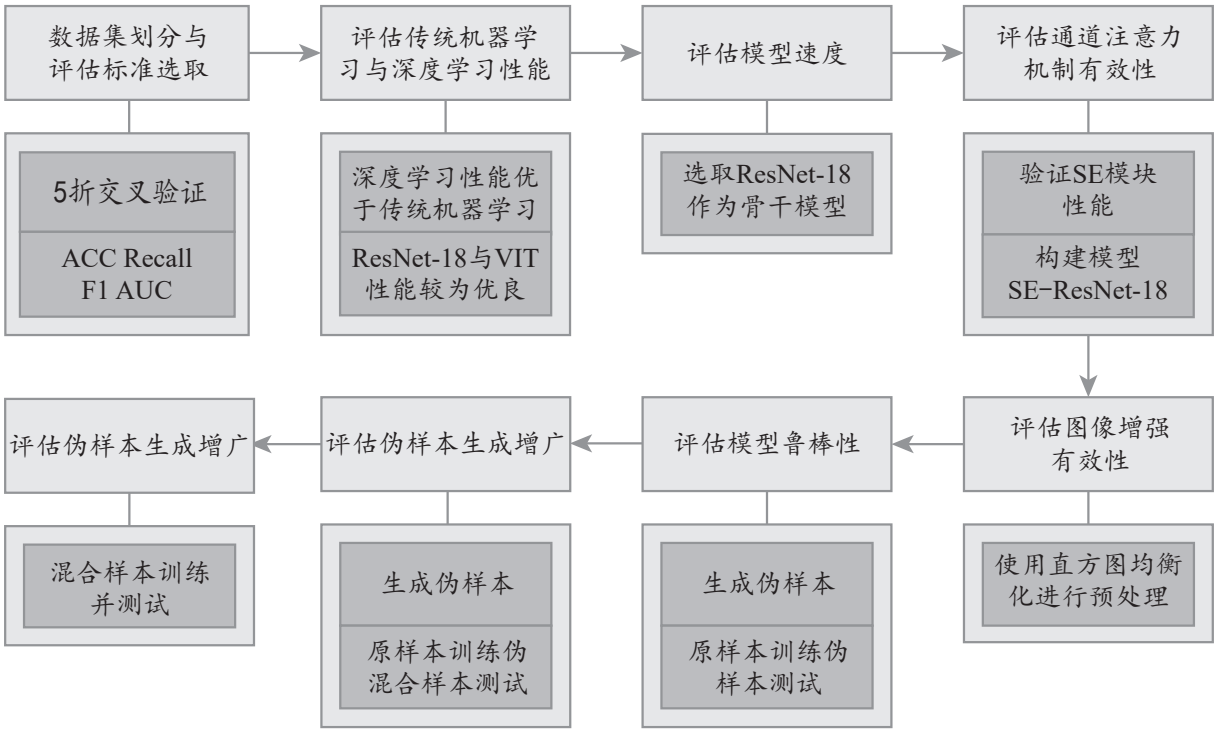
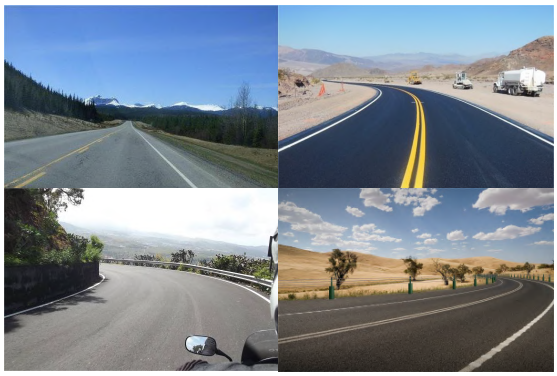


图 12　坑洼道路分类模型多视角评估工作流程图

5.2.1 数据集划分与评估标准选取

数据集划分　为了有效验证模型性能，需要对题给数据划分训练集 $\mathcal{D}_{train}$ 与测试集 $\mathcal{D}_{test}$ 且保证 $\mathcal{D}_{train} \cap \mathcal{D}_{test} = \varnothing$，让模型在从未见过的样本上进行测试以验证其泛化性能。题给数据共 301 张图像，共分为两类，即“正常道路”与“坑洼道路”，两类数据差异如图 13 所示。正常路面为平整光滑、路面情况较为良好的路面图像，坑洼路面为带有坑洼凹陷、粗糙凹坑、光滑水坑以及路面破损的路面图像。其中正常道路图像 266 张，坑洼道路图像 35 张。



正常道路

坑洼道路

图 13　正常道路和坑洼道路对比图

由于样本总量较少，如果仅仅将数据按照一定比例划分训练集与测试集，则划分数据好坏的影响将对模型的性能验证造成极大的随机性影响。为了保证模型性能验证的可靠性，我们选择 $K-$ 折交叉验证方法来划分数据。

$K-$ 折交叉验证（$K-$fold cross-validation）是一种常用于评估机器学习模型性能的技术，能够在有限的数据集上进行模型性能评估。它将数据随机分成 K 等份，对模型循环 K 次进行训练与测试，每次取其中一份数据作为测试集，其他 $K-1$ 份数据作为训练数据，最后计算模型在每个验证集上的性能指标的平均值，以获取对模型性能的更全面评估。

本文对数据 $\mathcal{D}$ 取 5 折交叉验证，即构造 5 个不同的数据集 $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5\}$，每个数据集的测试集互不相同。测试集与训练集图像总数计算如式 (16) 所示。

$$\begin{cases} N_{\mathcal{D}_i^{test}} &= \frac{1}{K} N_{\mathcal{D}} \\ N_{\mathcal{D}_i^{train}} &= \frac{K-1}{K} N_{\mathcal{D}} \end{cases} \tag{16}$$

其中 $\mathcal{D}_i^{test}, \mathcal{D}_i^{train}$ 分别表示第 i 个数据集的训练集与测试集的样本总量，$N_{\mathcal{D}}$ 为题给数据集的样本总量。同时，由于题给数据中有 266 张正常道路图像，而坑洼道路图像仅有 35 张，存在严重的类别不平衡问题。因此，为了保证模型在测试数据上具有足够的鲁棒性，必须让模型适应在此类别不平衡情况下进行学习。对于每一个 $\mathcal{D}_i$ 的划分，本文都严格按照原始数据集的类别比例进行数据划分，划分后的 $\mathcal{D}_i^{train}$ 与 $\mathcal{D}_i^{test}$ 均遵循原始数据中的类别不平衡条件。

除数量相等与类别一致以外，$K-$ 折交叉验证还需保证测试集互补交叉，且相加为总数据集，如式 (17) 所示。

$$\begin{cases} \mathcal{D}_1^{test} \cap \mathcal{D}_2^{test} \cap \cdots \cap \mathcal{D}_K^{test} &= \varnothing \\ \mathcal{D}_1^{test} \cup \mathcal{D}_2^{test} \cup \cdots \cup \mathcal{D}_K^{test} &= \mathcal{D} \end{cases} \tag{17}$$

详细数据划分数据如表 1 所示。

表 1　数据集详细划分情况

$\mathcal{D}$	$\mathcal{D}_i$	$\mathcal{D}_i^{train}$	$\mathcal{D}_i^{test}$	$\mathcal{D}_i^{train(n)}$	$\mathcal{D}_i^{train(p)}$	$\mathcal{D}_i^{test(n)}$	$\mathcal{D}_i^{test(p)}$
301	301	241	60	213	28	53	7

其中 $\mathcal{D}_i^{train(n)}$ 表示 $\mathcal{D}_i^{train}$ 中正常道路图像数量，$\mathcal{D}_i^{train(p)}$ 表示 $\mathcal{D}_i^{train}$ 中坑洼道路图像数量。$\mathcal{D}_i^{test(n)}$ 与 $\mathcal{D}_i^{test(p)}$ 分别表示 $\mathcal{D}_i^{test}$ 中正常道路图像与坑洼道路图像数量。

评估标准选取　评估标准选取对评估模型好坏至关重要，不同的模型标准能够反映出模型在不同方面的能力。由于在本文的坑洼道路分类问题中，存在严重的样本类别不平衡问题，因此仅采用准确率来衡量模型性能是不全面的。为了综合反应模型在样本类别不平衡条件下的各项性能，本文选取平均准确率（Average Accuracy, ACC）、召回率（Recall）、F1 分数（F1-score）、AUC（Area Under the Curve）作为模型评价指标，对模型进行详细全面的评价。

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。

ACC 表示模型正确分类的样本占总样本数的比例，计算公式如式 (18)所示。

$$\text{ACC} = \frac{TP + TN}{TP + FP + FN + TN} \tag{18}$$

Recall 表示实际正类别的样本中有多少被模型正确地分类为正类别, 计算公式如式 (19) 所示。

$$\text{Recall} = \frac{TP}{TP + FN} \tag{19}$$

F1-Score 是精确度和召回率的调和平均，它用于综合评估模型的性能，特别适用于不平衡的数据集，计算公式如式 (20) 所示。

$$\text{F1} - \text{Score} = \frac{2 \times \text{Precision} + \text{Recall}}{\text{Precision} + \text{Recall}} \tag{20}$$

其中 Precision 为精确度，表示模型预测为正类别的样本中有多少是真正的正类别，如式 (21) 所示。

$$\text{Precision} = \frac{TP}{TP + FP} \tag{21}$$

AUC 是一种用于评估二元分类模型性能的常见指标，它度量了模型的 Recall 与假正例率 FPR（False Positive Rate）之间的面积。其中 FPR 的计算方式如式 (22) 所示。

$$\mathrm{FPR} = \frac{FP}{FP + TN} \tag{22}$$

5.2.2 评估传统机器学习与深度学习性能

为了验证本文所使用的深度学习模型的有效性，首先评估传统机器学习方法与深度学习方法各自在 5 折交叉验证数据集上的性能。

传统机器学习 对于传统机器学习方法，我们使用直方图来提取图像特征，然后分别使用 SVM、决策树、$K-$ 最近邻、朴素贝叶斯来对图像进行分类，分别评估四种机器学习方法的性能。

直方图特征可以很好地反映道路图像的亮度分布、对比度、颜色分布等特征，供后续的机器学习分类模型进行分类。本文提取的直方图特征频谱图如图 14 所示。

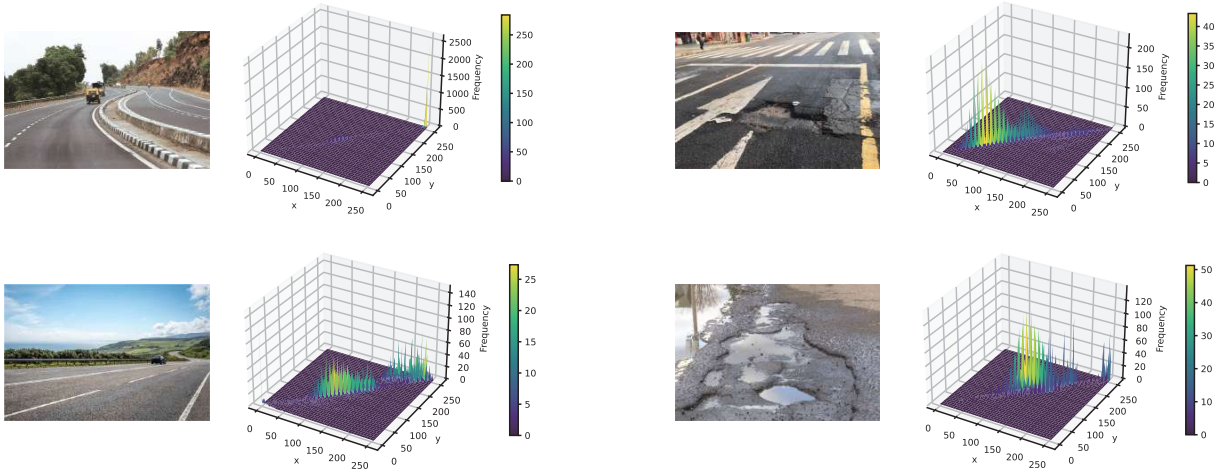


图 14　正常道路与坑洼道路的直方图特征频谱图

SVM、决策树、$K-$ 最近邻、朴素贝叶斯均为传统机器学习分类中的常用方法，使用于本文的坑洼道路分类问题。它们的算法特点如下所示：

- **SVM：** 适用于分类、回归、监督学习任务。其方法是找到一个决策边界（超平面），该边界将数据分为两个类别，同时使间隔最大化。SVM 可以使用不同的核函数来适应不同类型的数据。
- **决策树：** 适用于分类、回归、监督学习任务。在其方法中，叶节点代表道路类别，递归将数据集分割成更小的子集，直到达到停止条件。决策树易于解释，可以帮助识别特征的重要性，但容易过拟合。

- $K-$ **最近邻：**适用于分类、回归、监督学习任务。其方法是对一个新数据点，查找与该数据点最近的 K 个已知数据点，然后根据度量值进行预测。$K-$ 最近邻的性能高度依赖于选择的距离度量和 K 值。
- **朴素贝叶斯：**适用于分类任务。算法是基于贝叶斯定理的统计分类算法，假设特征之间是相互独立的。朴素贝叶斯适用于高维数据集，但由于独立性假设，某些情况下性能略低。

使用 5 折交叉验证对以上四种机器学习方法进行评估，取最终的平均指标作为结果，性能如表 2所示。

表 2　传统机器学习模型性能分析表

	ACC	R_n	R_p	F1-score	AUC
SVM	0.83	0.86	0.62	0.74	**0.47**
决策树	0.84	0.88	0.45	0.67	0.37
$K-$ 最近邻	**0.9**	**0.99**	0.11	0.55	0.18
朴素贝叶斯	0.77	0.77	**0.77**	**0.77**	0.42

结合表 2 中各传统机器学习模型的性能数据，绘制出如图 15所示的各模型的性能对比折线图。通过对比发现，四种机器学习方法在 ACC 上均取得合格的性能，其中 $K-$ 最近邻算法取得了 0.9 的准确率，但其在坑洼道路图像上的召回率仅为 0.11，意味着其无法正确判断坑洼道路，ACC 的高性能是由于测试数据的类别不平衡问题引起的。朴素贝叶斯虽然在两种图像中均取得 0.77 的召回率，但可能是由于其独立性假设造成的原因，且其总体预测准确率偏低。同时，在 AUC 指标中，所有机器学习算法均无法得到良好结果，最好结果的 SVM 仅为 0.47。综上所述，传统机器学习方法在坑洼道路分类问题中未能取得令人满意的性能。

深度学习骨干模型选择　为了验证各个骨干模型在坑洼道路分类问题中的性能，本文分别使用 $K-$ 折交叉验证方法与各个性能指标测算 AlexNet, VGGNet, ResNet，以及当下流行的 VIT（Vision Transformer）模型四种计算机视觉领域的常用骨干模型在坑洼道路数据集上的性能。其中 VGGNet 包含 VGG-16 与 VGG-19 两种结构，ResNet 包含 ResNet-18, ResNet-34 与 ResNet-50。

使用 5 折交叉验证，在训练轮次 N_{epoch} 分别为 20, 30, 40 的情况下分别计算模型的 ACC、正常道路 Recall、坑洼道路 Recall、F1-score、AUC 指标，实验结果如表 3 所示。

图 16 展示了以上 7 种深度学习模型在不同性能方面的差异情况。

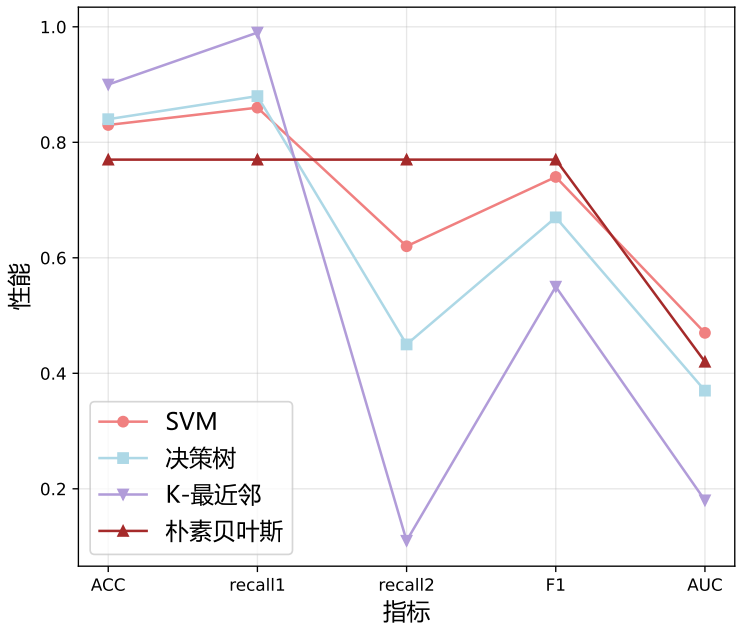


图 15　传统机器学习在道路坑洼分类问题上的性能对比图

表 3　基于深度学习的各道路坑洼分类模型在不同训练轮次中的性能分析表

网络模型	$N_{epoch}=20$					$N_{epoch}=30$					$N_{epoch}=40$				
	ACC	R_n	R_p	F1	AUC	ACC	R_n	R_p	F1	AUC	ACC	R_n	R_p	F1	AUC
AlexNet	0.88	1	0	0	0.5	0.88	0.98	0.02	0.02	0.5	0.89	0.98	0.17	0.22	0.57
VGG-16	0.88	1	0	0	0.5	0.88	1	0	0	0.5	0.89	0.97	0.05	0.05	0.51
VGG-19	0.88	1	0	0	0.5	0.88	1	0	0	0.5	0.88	1	0	0	0.5
ResNet-18	0.92	0.97	0.48	0.58	0.73	0.92	0.97	0.51	0.58	0.74	0.91	0.96	0.51	0.56	0.73
ResNet-34	0.89	0.96	0.37	0.45	0.66	0.90	0.96	0.39	0.47	0.68	0.90	0.96	0.34	0.43	0.65
ResNet-50	0.86	0.91	0.45	0.41	0.68	0.90	0.93	0.57	0.57	0.75	0.89	0.92	0.59	0.55	0.66
VIT	0.93	0.99	0.42	0.57	0.71	0.92	0.98	0.4	0.52	0.69	0.91	0.98	0.34	0.45	0.66

在 ACC 的评估中，ResNet-18 与 VIT 在训练 20 轮时分别达到 92% 与 93%，其他模型均在 90% 以下。ResNet 与 VIT 在道路坑洼分类的题给数据中性能最佳，但此时未考虑样本类别不平衡指标。

在 Recall 的评估中，对于正常道路图像的召回率，所有网络均在 0.9 以上，保持极高水准，其中 AlexNet、VGG-16、VGG19 在 $N_{epoch}=20$ 与 $N_{epoch}=30$ 时均为 1，意味着正常道路图像全部命中，但是在坑洼道路图像召回率中却为 0，意味着这些没有残差连接的深层网络无法完成样本类别不平衡的坑洼道路分类任务，无法从极少量的坑洼道

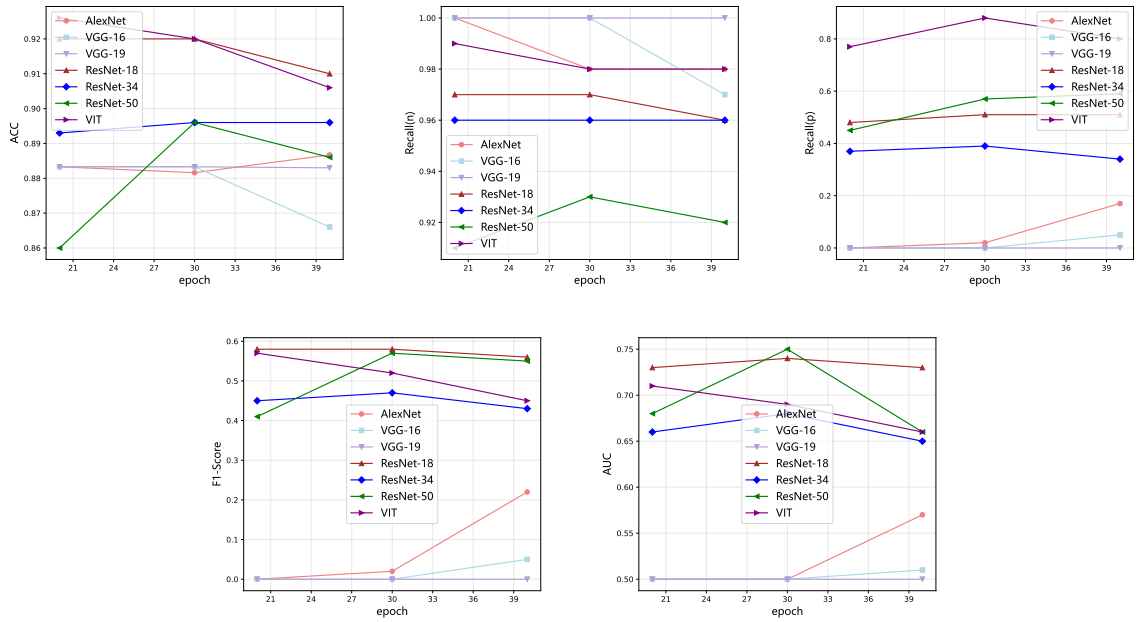


图 16　不同深度学习模型在坑洼道路分类任务中的性能对比图

路图像中提取特征，而 ACC 的极高值是由测试数据中样本的类别不平衡条件造成的。

而对于 ResNet 系列与 VIT 而言，虽然 R_n 略低，没有完美达到 1，但是依然有 0.96 以上的水平，其中 ResNet-18 与 VIT 基本维持在 0.96-0.99 的水平，在测试集中正常道路样本较多的情况下依然属于高数值。与 AlexNet、VGG-16、VGG19 相比，ResNet-18 在 R_p 性能方面始终能保持 0.5 左右的水平，VIT 也能保持在 0.3 到 0.4 左右的水平。

在 F1-Score 与 AUC 的评估中，AlexNet 与 VGG 系类由于无法正确判断坑洼道路图像，F1-Score 基本为零，AUC 仅为 0.5 左右，而 ResNet 系列与 VIT 的 F1-Score 在 0.5 左右，AUC 保持在 0.6 到 0.7 左右。其中 ResNet-18 为最高水平，当 $N_{epoch}=20$ 时，F1-Score 与 AUC 分别为 0.58 与 0.73，在本文训练样本与测试样本类别极度不平衡的问题情境下，已经取得较优结果。

综上所述，ResNet 系列中的 ResNet-18 模型与 VIT 模型在所有参与评估的深度学习模型中取得了较为优秀的结果。

传统机器学习与深度学习对比　当 N_{epoch} 分别为 20、30、40 时，深度学习模型 AlexNet、VGG、ResNet 与传统机器学习模型 SVM、决策树、$K-$ 最近邻、朴素贝叶斯在 ACC、Recall、F1-score、AUC 性能上的比较如图 17 所示。

由对比可知，SVM 与朴素贝叶斯在集中机器学习方法中性能较佳，R_p 与 $F-Score$ 甚至比 AlexNet 与 VGG-19 深度学习模型更高，但总体性能与所有的深度学习模型都有较大差距。除朴素贝叶斯以外，ResNet-18 在各项指标性能远远胜于机器学习方法，朴素贝叶斯虽然对坑洼道路分类能力较强，但总体分类准确率较低，仅为 0.77。

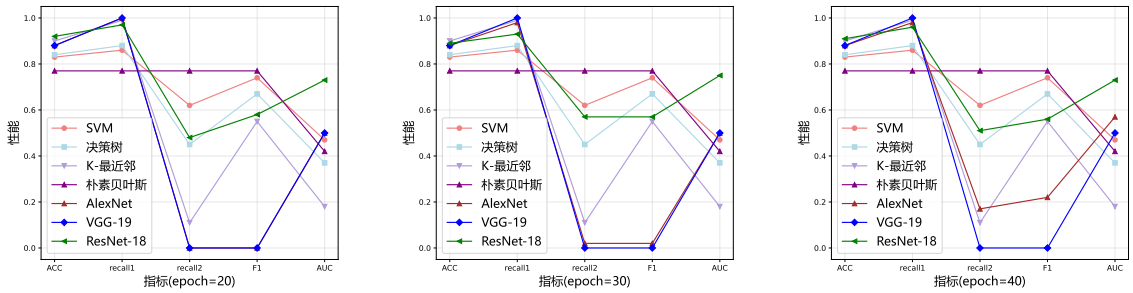


图 17　传统机器学习与深度学习性能对比图

综上所述，虽然由于训练数据类别不平衡问题，不依赖监督学习的机器学习方法在对极少样本的坑洼道路图像分类上能优于个别深度学习模型，但总体分类能力与深度学习模型有较大差距。因此，在坑洼道路分类问题中，基于深度学习的模型要优于传统机器学习模型。

5.2.3 评估模型速度

在深度学习骨干模型选择部分已经进行各个卷积神经网络综合性能的评估，其中 ResNet-18 模型与 VIT 模型的性能要远高于其他网络。下面对该两者进行训练速度评估，以构建出分类准确、速度快的模型。

由于评估训练速度不需要考虑模型精度、数据类别平衡等问题，本文在 $\mathcal{D}_1^{train}$ 上分别测算 $N_{epoch}=1$、$N_{epoch}=5$、$N_{epoch}=10$ 时 ResNet-18 模型与 VIT 模型的训练时间，结果如图 18 所示。

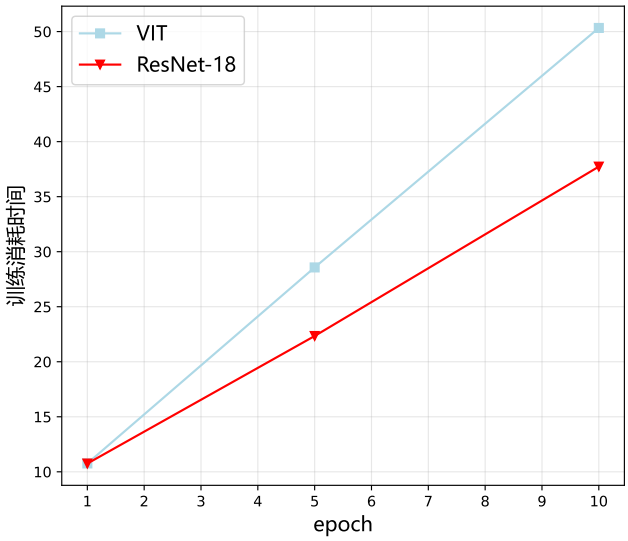


图 18　ResNet-18 模型与 VIT 模型训练时间对比图

由图 18 可知 ResNet-18 模型在训练数据上的训练时间明显低于 VIT 模型，说明 ResNet-18 模型拥有比 VIT 模型更高的识别速度，因此本文选择 ResNet-18 模型作为处理坑洼道路分类问题的骨干模型。

5.2.4 评估通道注意力机制有效性

为网络添加 SE 模块以增强网络的通道注意力有利于让模型更加关注特征通道之间的关系。本文分别在 $N_{epoch}=20$、$N_{epoch}=30$、$N_{epoch}=40$ 的情况下对 ResNet-18 模型和加入 SE 模块的 SE-ResNet-18 模型进行性能评估，结果如图 19所示。

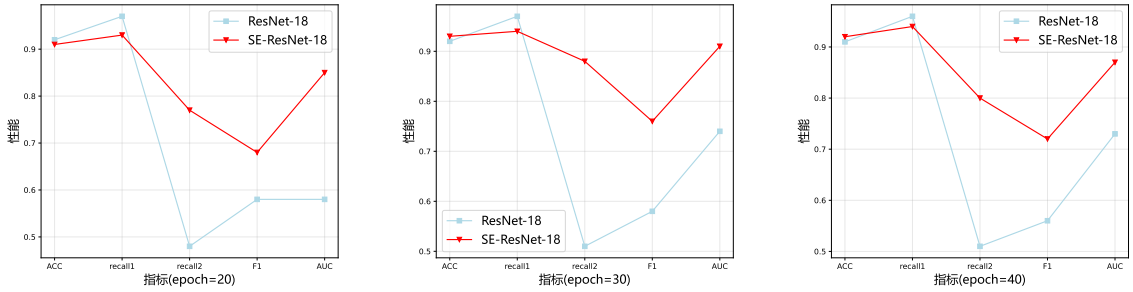


图 19　SE-ResNet-18 模型与 ResNet-18 模型性能对比图

通过对比可发现，经过通道注意力增强的 ResNet-18 模型除了 R_n 的值略低外，其他指标均显著高于原始的 ResNet-18 模型. 尤其是当 $N_{epoch}=30$ 时，SE-ResNet-18 模型的 ACC 为 93.6%，高于 ResNet-18 模型的 92%，R_p、F1-Score、AUC 分别为 0.88、0.76、0.91，显著高于 ResNet-18 模型的 0.51、0.58、0.74。具体性能数据如表 4 所示。

表 4　SE-ResNet-18 模型与 ResNet-18 模型性能对比

网络模型	$N_{epoch}=20$					$N_{epoch}=30$					$N_{epoch}=40$				
	ACC	R_n	R_p	F1	AUC	ACC	R_n	R_p	F1	AUC	ACC	R_n	R_p	F1	AUC
ResNet-18	**0.92**	**0.97**	0.48	0.58	0.73	0.92	**0.97**	0.51	0.58	0.74	0.91	**0.96**	0.51	0.56	0.73
SE-ResNet-18	0.91	0.93	**0.77**	**0.68**	**0.85**	**0.93**	0.94	**0.88**	**0.76**	**0.91**	**0.92**	0.94	**0.80**	**0.72**	**0.87**

综上所述，经过通道注意力增强的 SE-ResNet-18 模型比原始模型更加具备区分坑洼道路图像的能力，坑洼道路图像召回率达到 0.88，远高于其他深度学习模型，且总体准确率也超过原始 ResNet-18 模型，该实验论证了通道注意力机制增强的有效性。

5.2.5 评估图像增强有效性

本文对所有训练图像进行直方图均衡化，以优化数据全局色彩空间。使用 SE-ResNet-18 作为骨干模型，分别在使用原始数据与使用直方图均衡化后的数据上训练 30 轮次，

得到各个评价指标如表 5所示。

表 5　直方图均衡化性能对比

方法	ACC	R_n	R_p	F1-score	AUC
SE-ResNet-18	93.6	0.94	**0.88**	**0.76**	**0.91**
SE-ResNet-18 + 直方图均衡化	**94.33**	**0.96**	0.80	0.75	0.88

由表 5可知，加入直方图均衡化后，SE-ResNet-18 的 ACC 由 93.6% 提升到 94.33%，R_p 等指标略微下降，但依然维持在较高水平，说明直方图均衡化使得模型在样本类别不平衡环境中能够保持对少量坑洼道路样本的分类能力，同时在一定幅度上提升模型整体分类能力。

5.2.6 评估伪样本生成增广有效性与鲁棒性验证

为评估本位的伪样本生成策略有效性，验证模型的鲁棒新性能，首先构建完全伪样本数据集 $\mathcal{D}^{fake}$，让在 $\mathcal{D}$ 上训练过的 SE-ResNet-18 模型在 $\mathcal{D}^{fake}$ 上进行测试，以验证模型鲁棒性。

生成数据集　由于本文使用 $\mathcal{D}$ 进行 $K-$ 折交叉验证，构造了 $\{\mathcal{D}_1,\cdots,\mathcal{D}_K\}$ 序列数据集，为满足交叉验证续期，我们也分别将每一个子数据集的图像输入到生成器中生成伪样本，构成语义信息与原数据完全一致的 $\{\mathcal{D}_1^{fake},\cdots,\mathcal{D}_K^{fake}\}$。其中标签为正常道路的图像输入到 G_{A2B} 中进行生成，标签为坑洼道路的图像输入到 G_{B2A} 中进行生成。该数据集的特点是在保持 $\mathcal{D}$ 原有语义信息的情况下将 $\mathcal{D}$ 映射到了另一个边缘分布不同的特征空间中。

此外，我们还将 $\mathcal{D}$ 进行 $K-$ 折交叉验证，构造了 $\{\mathcal{D}_1,\cdots,\mathcal{D}_K\}$ 与 $\{\mathcal{D}_1^{fake},\cdots,\mathcal{D}_K^{fake}\}$ 对应序列中的每个数据集进行混合，生成 $\{\mathcal{D}_1^{fusion},\cdots,\mathcal{D}_K^{fusion}\}$ 数据集。$\mathcal{D}_i^{fusion}$ 中完全包含了 $\mathcal{D}_i$ 与 $\mathcal{D}_i^{fake}$ 的数据。该数据集的特点是混合了 $\mathcal{D}$ 与 $\mathcal{D}^{fake}$ 两种特征分布，有利于验证模型的鲁棒性。

验证模型鲁棒性性　首先将 SE-ResNet-18 在 $\mathcal{D}$ 上进行训练，然后测试其在 $\mathcal{D}^{fake}$ 与 $\mathcal{D}^{fusion}$ 上的表现性能，验证经过训练后的模型在特征空间扰动情况下的鲁棒性。

本次实验在 $N_{epoch}=30$ 情况下进行，其结果如表 6 所示，表中数据为 5 折交叉验证的平均值。由表 6 可知，让仅在原样本 $\mathcal{D}$ 上训练的深度学习模型去测试边缘分布略有改变的伪样本数据集 $\mathcal{D}^{fake}$ 时，模型完全失去对坑洼道路的识别能力。

表 6　基于伪样本的模型鲁棒性验证

数据	ACC	R_n	R_p	F1-Score	AUC
$\mathcal{D}$ 训练 +$\mathcal{D}$ 测试	**93.6**	0.94	**0.88**	**0.76**	**0.91**
$\mathcal{D}$ 训练 + $\mathcal{D}^{fake}$ 测试	87.8	**0.99**	0	0	0.49
$\mathcal{D}$ 训练 +$\mathcal{D}^{fusion}$ 测试	90.49	0.98	0.33	0.44	0.65

验证伪样本生成增广有效性　为验证本文的伪样本生成增广策略的有效性，将 $\mathcal{D}$ 与 $\mathcal{D}^{fake}$ 混合在一起的 $\mathcal{D}^{fusion}$ 作为增广训练集来训练 SE-ResNet-18，然后分别在 $\mathcal{D}^{fake}$ 与 $\mathcal{D}^{fusion}$ 进行测试。具体性能对比如表 7 所示。

表 7　伪样本生成增广有效性验证

数据	ACC	R_n	R_p	F1-Score	AUC
$\mathcal{D}$ 训练 + $\mathcal{D}^{fake}$ 测试	87.8	**0.99**	0	0	0.49
$\mathcal{D}^{fusion}$ 训练 + $\mathcal{D}^{fake}$ 测试	**96.92**	0.96	**0.96**	**0.88**	**0.96**
$\mathcal{D}$ 训练 +$\mathcal{D}^{fusion}$ 测试	90.49	**0.98**	0.33	0.44	0.65
$\mathcal{D}^{fusion}$ 训练 +$\mathcal{D}^{fusion}$ 测试	**91.99**	0.95	**0.68**	**0.67**	**0.81**
$\mathcal{D}$ 训练 +$\mathcal{D}$ 测试	93.6	0.94	**0.88**	0.76	**0.91**
$\mathcal{D}^{fusion}$ 训练 +$\mathcal{D}$ 测试	**97.02**	**0.98**	0.82	**0.86**	0.90

由表 7 可知，添加伪样本进行数据增广后，除了 R_n 略微有损失以外，在更为复杂的 $\mathcal{D}^{fake}$ 测试集上，增广后训练的模型性能明显有大幅度的提升。

将模型在混合数据 $\mathcal{D}^{fusion}$ 上训练后再与其在 $\mathcal{D}$ 上训练的性能进行对比，发现识别准确率由 93.6% 提升至 97.02%，F1-Score 由 0.76 提升到 0.86，且在单张 RTX3060 显卡上训练用时为 262 秒，充分说明本文所建立的模型拥有效率与准确性兼具的优越性能。

5.3 问题三：模型测试

本文首先使用 SE-ResNet-1 模型在 $\mathcal{D}$ 上训练一个能对 $\mathcal{D}$ 进行正确分类的模型，然后将此预训练模型当做判别器来训练循环生成式对抗网络。将训练好的循环生成式对抗网络当做伪样本生成器 G_{A2B} 与 G_{B2A}，$\mathcal{D}$ 中的正常道路样本输入到 G_{A2B} 中生成伪样本，坑洼道路样本输入到 G_{B2A} 中生成伪样本，将伪样本与原数据样本合成为增广数据集。

最后使用问题一建立的通道注意力增强 SE-ResNet-18 模型在增广数据集上训练 30 轮，使用 Adam 作为参数优化器，设置出示学习率为 0.0001，采用余弦退火算法来更新学习率。使用训练后的模型对 4942 张测试图像进行图像分类，结果记录在 “test_result.csv” 中。

六、 模型的评价与推广

6.1 模型优点

- 在原始的 ResNet-18 结构上进行通道注意力增强，使模型更加关注图像通道间特征，提升特征提取能力。
- 本文改进了原始循环生成式对抗网络的训练方式，将预训练的分类器进行参数冻结，然后代替循环生成式网络中原本的判别器，监督生成器生成伪样本，使其能生成语义信息高度一致但分类器难以分辨的伪样本，进行更加有效的数据增广。
- 本文对模型各部分的有效性进行了大量的对比实验与消融实验，论证了模型各部分的性能提升效果。

6.2 模型缺点

- 模型仅关注特征提取部分，没有充分解决类别不平衡问题。
- 伪样本的生成过程较粗糙，如在图像生成过程中没有进行精细的领域信息控制，在判别器的训练过程中没有进一步强化判别器的判别能力等。

6.3 模型推广

本文模型不仅适用于坑洼道路分类问题，对于数据内类别数据存在差异的正负样本分类问题均使用。同时，改进循环生成式网络的伪样本生成方法为提升分类模型的鲁棒性提供了新的解决方案。

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] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

[4] Zhu J Y, Park T, Isola P, et al. Unpaired image-to-image translation using cycle-consistent adversarial networks[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2223-2232.

附录 A　源代码

1.1　分类模型相关源代码

1.1.1　训练与测试源代码

main.py(5 折交叉验证训练代码)

```
import os
import random
import numpy as np
from torch.utils.data import DataLoader
from torchvision.transforms import transforms
from sklearn.metrics import recall_score, roc_curve, auc, f1_score
from models.resnet import *
from models.SEnet import *
from Dataloder import dataloader
import warnings
import cv2
from PIL import Image
# 忽略 UserWarning
warnings.filterwarnings("ignore", category=UserWarning)

# 设置随机种子
seed = 79
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# 定义ResNet-18模型

def apply_bilateral_filter(image):
    # 转换PIL图像为NumPy数组
    image = np.array(image)

    # 使用双边滤波
    image = cv2.bilateralFilter(image, d=9, sigmaColor=75, sigmaSpace=75)

    # 将处理后的图像转换回PIL图像
    image = Image.fromarray(image)

    return image

class EqualizeColorHistogram(object):
    def __call__(self, img):
```

```
        img_np = np.array(img)
        # 拆分通道
        b, g, r = cv2.split(img_np)
        # 对每个通道进行直方图均衡化
        equalized_b = cv2.equalizeHist(b)
        equalized_g = cv2.equalizeHist(g)
        equalized_r = cv2.equalizeHist(r)
        # 合并三个通道
        equalized_img = cv2.merge((equalized_b, equalized_g, equalized_r))
        return Image.fromarray(equalized_img)

# 定义数据预处理和转换
transform = transforms.Compose([
    transforms.ToPILImage(), # 将NumPy数组转换为PIL图像
    transforms.Resize((224, 224)), # 调整图像大小
    # transforms.RandomHorizontalFlip(),
    # transforms.RandomVerticalFlip(),
    # transforms.ColorJitter(),
    EqualizeColorHistogram(),
    transforms.ToTensor(), # 将图像转换为张量
])


acc = []
rc_normal = []
rc_potholes = []
auc_list = []
f1 = []

epoch = 30
lr = 0.0001
eta_min = 1e-8


def train(data_loader, epoch, model, optimizer, scheduler, lr, loss_fn):
    # 损失函数，你需要根据你的任务选择适当的损失函数
    criterion = loss_fn

    for e in range(epoch):
        model.train() # 设置模型为训练模式
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in data_loader:
            inputs = inputs.cuda()
            labels = labels.cuda()
```

```
            optimizer.zero_grad() # 梯度清零

            # 正向传播
            outputs = model(inputs)
            # outputs = outputs.logits
            loss = criterion(outputs, labels)

            # 反向传播和优化
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            # 计算正确率
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        accuracy = 100 * correct / total

        # 更新学习率（如果使用学习率调度器）
        if scheduler is not None:
            scheduler.step()
            current_lr = optimizer.param_groups[0]['lr']




# 计算recall
true_normal = []
true_potholes = []
def test(data_loader, model, loss_fn):
    model.eval() # 设置模型为评估模式
    running_loss = 0.0
    correct = 0
    total = 0
    true_labels = []
    predicted_labels = []

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs = inputs.cuda()
            labels = labels.cuda()

            outputs = model(inputs)
            loss = loss_fn(outputs, labels)
            running_loss += loss.item()
```

```
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # AUC
            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(predicted.cpu().numpy())

    accuracy = 100 * correct / total
    acc.append(accuracy)

    # 计算召回率
    recall_normal = recall_score(true_labels, predicted_labels, pos_label=0)
    recall_potholes = recall_score(true_labels, predicted_labels, pos_label=1)
    f1_sc = f1_score(true_labels, predicted_labels)

    # 计算AUC
    fpr, tpr, thresholds = roc_curve(true_labels, predicted_labels)
    auc_score = auc(fpr, tpr)
    rc_normal.append(recall_normal)
    rc_potholes.append(recall_potholes)
    auc_list.append(auc_score)
    f1.append(f1_sc)
    print(f"Acc = {accuracy:.2f}, recall_normal = {recall_normal:.4f}, recall_potholes =
        {recall_potholes:.4f}, auc = {auc_score:.4f}, f1-score = {f1_sc:.4f}")

import time

start_time = time.time()

for i in range(5):
    model = SE_ResNet18().cuda()

    loss = nn.CrossEntropyLoss()
    optim = torch.optim.Adam(model.parameters(), lr)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer=optim, T_max=int(epoch),
        eta_min=eta_min)

    data_root_train = './data_new_new/data' + str(i+1)
    train_dataloader = dataloader(data_root=os.path.join(data_root_train, 'train'),
        transform=transform)
    train_loader = DataLoader(dataset=train_dataloader, num_workers=12, batch_size=8,
        persistent_workers=False, drop_last=False)

    data_root_test = './data_test'
    test_dataloader = dataloader(data_root=data_root_test, transform=transform)
```

```
    test_loader = DataLoader(dataset=test_dataloader, num_workers=12, batch_size=4,
        persistent_workers=False)
    # 调用train函数
    train(data_loader=train_loader, epoch=epoch, model=model, optimizer=optim,
        scheduler=scheduler, loss_fn=loss, lr=lr)
    test(data_loader=test_loader, model=model, loss_fn=loss)
    torch.save(model.state_dict(), "SENet.pt")

end_time = time.time()
cost_time = end_time - start_time
print('Average Acc', sum(acc) / len(acc))
print('Average Recall Normal', sum(rc_normal) / len(rc_normal))
print('Average Recall Potholes', sum(rc_potholes) / len(rc_potholes))
print('Average AUC', sum(auc_list) / len(auc_list))
print('Average F1', sum(f1)/len(f1))
print('Cost Time', cost_time)
```

train.py（对测试集进行测试）

```
import os
import random
import numpy as np
import torch
from torch.utils.data import DataLoader
from torchvision.transforms import transforms
from sklearn.metrics import recall_score, roc_curve, auc, f1_score
from models.vit import *
from models.resnet import *
from models.alexnet import *
from models.VGG import *
from models.densenet import *
from models.SEnet import *
from Dataloder import dataloader
import warnings
import cv2
from PIL import Image
# 忽略 UserWarning
warnings.filterwarnings("ignore", category=UserWarning)

# 设置随机种子
seed = 79
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

```

```python
# 定义ResNet-18模型

def apply_bilateral_filter(image):
    # 转换PIL图像为NumPy数组
    image = np.array(image)

    # 使用双边滤波
    image = cv2.bilateralFilter(image, d=9, sigmaColor=75, sigmaSpace=75)

    # 将处理后的图像转换回PIL图像
    image = Image.fromarray(image)

    return image

class EqualizeColorHistogram(object):
    def __call__(self, img):
        img_np = np.array(img)
        # 拆分通道
        b, g, r = cv2.split(img_np)
        # 对每个通道进行直方图均衡化
        equalized_b = cv2.equalizeHist(b)
        equalized_g = cv2.equalizeHist(g)
        equalized_r = cv2.equalizeHist(r)
        # 合并三个通道
        equalized_img = cv2.merge((equalized_b, equalized_g, equalized_r))
        return Image.fromarray(equalized_img)

# 定义数据预处理和转换
transform = transforms.Compose([
    transforms.ToPILImage(), # 将NumPy数组转换为PIL图像
    transforms.Resize((224, 224)), # 调整图像大小
    transforms.RandomHorizontalFlip(),
    transforms.RandomApply([transforms.GaussianBlur(9)], p=0.5),
    transforms.ToTensor(), # 将图像转换为张量
])

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


acc = []
rc_normal = []
rc_potholes = []
auc_list = []
```

```python
f1 = []

epoch = 30
lr = 0.0001
eta_min = 1e-8

def train(data_loader, epoch, model, optimizer, scheduler, lr, loss_fn):
    # 损失函数，你需要根据你的任务选择适当的损失函数
    criterion = loss_fn

    for e in range(epoch):
        model.train() # 设置模型为训练模式
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in data_loader:
            inputs = inputs.cuda()
            labels = labels.cuda()
            optimizer.zero_grad() # 梯度清零

            # 正向传播
            outputs = model(inputs)
            # outputs = outputs.logits
            loss = criterion(outputs, labels)

            # 反向传播和优化
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            # 计算正确率
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        accuracy = 100 * correct / total

        # 更新学习率（如果使用学习率调度器）
        if scheduler is not None:
            scheduler.step()
            current_lr = optimizer.param_groups[0]['lr']



# 计算recall
```

```
true_normal = []
true_potholes = []
def test(data_loader, model, loss_fn):
    model.eval() # 设置模型为评估模式
    running_loss = 0.0
    correct = 0
    total = 0
    true_labels = []
    predicted_labels = []

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs = inputs.cuda()
            labels = labels.cuda()

            outputs = model(inputs)
            loss = loss_fn(outputs, labels)
            running_loss += loss.item()

            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # AUC
            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(predicted.cpu().numpy())

    accuracy = 100 * correct / total
    acc.append(accuracy)

    # 计算召回率
    recall_normal = recall_score(true_labels, predicted_labels, pos_label=0)
    recall_potholes = recall_score(true_labels, predicted_labels, pos_label=1)
    f1_sc = f1_score(true_labels, predicted_labels)

    # 计算AUC
    fpr, tpr, thresholds = roc_curve(true_labels, predicted_labels)
    auc_score = auc(fpr, tpr)
    rc_normal.append(recall_normal)
    rc_potholes.append(recall_potholes)
    auc_list.append(auc_score)
    f1.append(f1_sc)
    print(f"Acc = {accuracy:.2f}, recall_normal = {recall_normal:.4f}, recall_potholes =
        {recall_potholes:.4f}, auc = {auc_score:.4f}, f1-score = {f1_sc:.4f}")

import time

```

```
start_time = time.time()
# import torchvision.models as models
# resnet18 = models.resnet18(pretrained=True)
# state_dict = resnet18.state_dict()
model = ResNet18().cuda()
# model = torch.load('se_resnet101.pth.tar')

# se_resnet18_state_dict = model.state_dict()
# for key in state_dict:
#     if key in se_resnet18_state_dict:
#         # print(key)
#         se_resnet18_state_dict[key] = state_dict[key]

# model.load_state_dict(se_resnet18_state_dict)


loss = nn.CrossEntropyLoss()
optim = torch.optim.Adam(model.parameters(), lr)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer=optim, T_max=int(epoch),
    eta_min=eta_min)

data_root_train = './data'
train_dataloader = dataloader(data_root=data_root_train, transform=transform)
train_loader = DataLoader(dataset=train_dataloader, num_workers=12, batch_size=8,
    persistent_workers=False, drop_last=False, shuffle=True)


data_root_test = './data_test'
test_dataloader = dataloader(data_root=data_root_test, transform=transform_test)
test_loader = DataLoader(dataset=test_dataloader, num_workers=12, batch_size=8,
    persistent_workers=False)

train(data_loader=train_loader, epoch=epoch, model=model, optimizer=optim,
    scheduler=scheduler, loss_fn=loss, lr=lr)
test(data_loader=test_loader, model=model, loss_fn=loss)
torch.save(model, 'best.pt')


end_time = time.time()
cost_time = end_time - start_time
print('Average Acc', sum(acc) / len(acc))
print('Average Recall Normal', sum(rc_normal) / len(rc_normal))
print('Average Recall Potholes', sum(rc_potholes) / len(rc_potholes))
print('Average AUC', sum(auc_list) / len(auc_list))
print('Average F1', sum(f1)/len(f1))
print('Cost Time', cost_time)
```

ml.py（机器学习方法对比）

```python
# -*- coding: utf-8 -*-
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


acc = []
recall1 = []
recall2 = []
auc_score = []
f1score = []

def draw_3D(hist):
    hist = np.reshape(hist, (256, 256))
    # Create a mesh grid for X and Y values
    x = np.linspace(0, 255, 256)
    y = np.linspace(0, 255, 256)
    x, y = np.meshgrid(x, y)

    # Create a 3D plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the 3D histogram
    ax.plot_surface(x, y, hist, cmap='viridis')

    # Set labels
    ax.set_xlabel('Channel 0')
    ax.set_ylabel('Channel 1')
    ax.set_zlabel('Frequency')

    # Show the plot
    plt.show()

for j in range(1,6):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    data_root = './data_new_new/data'+str(j)
```

```
for i in os.listdir(data_root + '/train'):
    X_train.append(os.path.join(data_root + '/train', i))
    if i.startswith('normal'):
        y_train.append(0)
    else:
        y_train.append(1)

for i in os.listdir(data_root + '/test'):
    X_test.append(os.path.join(data_root + '/test', i))
    if i.startswith('normal'):
        y_test.append(0)
    else:
        y_test.append(1)
#
# print(len(X_train), len(X_test), len(y_train), len(y_test))


XX_train = []
for i in X_train:
    image = cv2.imdecode(np.fromfile(i, dtype=np.uint8), cv2.IMREAD_COLOR)
    img = cv2.resize(image, (256, 256),
                     interpolation=cv2.INTER_CUBIC)
    hist = cv2.calcHist([img], [0, 1], None,
                        [256, 256], [0.0, 255.0, 0.0, 255.0])

    XX_train.append(((hist / 255).flatten()))

# 测试集
XX_test = []
for i in X_test:
    image = cv2.imdecode(np.fromfile(i, dtype=np.uint8), cv2.IMREAD_COLOR)

    img = cv2.resize(image, (256, 256),
                     interpolation=cv2.INTER_CUBIC)
    hist = cv2.calcHist([img], [0, 1], None,
                        [256, 256], [0.0, 255.0, 0.0, 255.0])


    XX_test.append(((hist / 255).flatten()))



# ------------------------------------------------------------------------------
# 第三步 基于支持向量机的图像分类处理
# ------------------------------------------------------------------------------
# 0.5
# 常见核函数 'linear' , 'poly' , 'rbf' , 'sigmoid' , 'precomputed'
```

```
clf = SVC().fit(XX_train, y_train)
clf = SVC(kernel="linear").fit(XX_train, y_train)
predictions_labels = clf.predict(XX_test)

# ------------------------------------------------------------------------------
# 第三步 基于决策树的图像分类处理
# ------------------------------------------------------------------------------
# 0.36
# from sklearn.tree import DecisionTreeClassifier
# clf = DecisionTreeClassifier().fit(XX_train, y_train)
# predictions_labels = clf.predict(XX_test)

# ------------------------------------------------------------------------------
# 第三步 基于KNN的图像分类处理
# ------------------------------------------------------------------------------
# 0.11
# from sklearn.neighbors import KNeighborsClassifier
# clf = KNeighborsClassifier(n_neighbors=11).fit(XX_train, y_train)
# predictions_labels = clf.predict(XX_test)

# ------------------------------------------------------------------------------
# 第三步 基于朴素贝叶斯的图像分类处理0.
# ------------------------------------------------------------------------------
# 0.01
# from sklearn.naive_bayes import BernoulliNB
# clf = BernoulliNB().fit(XX_train, y_train)
# predictions_labels = clf.predict(XX_test)

# print(u'预测结果:')
# print(predictions_labels)
# print(u'算法评价:')
# print(classification_report(y_test, predictions_labels))
#


# labels = ['0', '1']



y_true = y_test # 正确标签
y_pred = predictions_labels # 预测标签

from sklearn.metrics import roc_auc_score, recall_score, accuracy_score, f1_score

auc = roc_auc_score(y_true, y_pred)
recall_class_0 = recall_score(y_true, y_pred, pos_label=0)
recall_class_1 = recall_score(y_true, y_pred, pos_label=1)
```

```
    acc_score = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    acc.append(acc_score)
    auc_score.append(auc)
    recall1.append(recall_class_0)
    recall2.append(recall_class_1)
    f1score.append(f1)

print('Acc', sum(acc)/len(acc))
print('Recall1', sum(recall1)/len(recall1))
print('Recall2', sum(recall2)/len(recall2))
print('AUC', sum(auc_score)/len(auc_score))
print('F1', sum(f1score)/len(f1score))
```

Dataloder..py（加载 Dataloder）

```
 import torch.utils.data as data
import os
import cv2

class dataloader(data.Dataset):
    def __init__(self, data_root, transform):
        self.data_root = data_root
        self.transform = transform
        self.items = []
        self.label = []
        for i in os.listdir(data_root):
            image = cv2.imread(os.path.join(self.data_root, i), 1)
            # image = hisEqulColor1(image)
            image = self.transform(image)
            self.items.append(image)
            if(i.startswith('normal')):
                self.label.append(0)
            else:
                self.label.append(1)

    def __getitem__(self, item):
        image = self.items[item]
        label = self.label[item]
        if self.transform:
            image = self.transform(image)

        return image, label

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

1.1.2 模型结构源代码

alexnet.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models
from torchvision import models

class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
        self.alexnet = torchvision.models.alexnet(pretrained = False)
        self.alexnet.classifier = nn.Sequential(
            nn.Linear(256 * 6 * 6, 4096), # 自定义全连接层
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 2)
        )

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

resnet.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models
from torchvision import models

class ResNet18(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet18, self).__init__()
        self.resnet = torchvision.models.resnet18(pretrained=True)
        num_ftrs = self.resnet.fc.in_features
        self.dropout = nn.Dropout(p=0.5)

        self.resnet.fc = nn.Sequential(
            self.dropout,
```

```
            nn.Linear(num_ftrs, num_classes)
        )

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


class ResNet34(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet34, self).__init__()
        self.resnet = torchvision.models.resnet34(pretrained=False)
        num_ftrs = self.resnet.fc.in_features
        self.dropout = nn.Dropout(p=0.5)


        self.resnet.fc = nn.Sequential(
            self.dropout,
            nn.Linear(num_ftrs, num_classes)
        )

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

class ResNet50(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet50, self).__init__()
        self.resnet = torchvision.models.resnet50(pretrained=False)
        num_ftrs = self.resnet.fc.in_features
        self.dropout = nn.Dropout(p=0.5)


        self.resnet.fc = nn.Sequential(
            self.dropout,
            nn.Linear(num_ftrs, num_classes)
        )

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

SEnet.py

```
# Here is the code :

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

```

```python
class SE_Block(nn.Module):                    # Squeeze-and-Excitation block
    def __init__(self, in_planes):
        super(SE_Block, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.conv1 = nn.Conv2d(in_planes, in_planes // 16, kernel_size=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(in_planes // 16, in_planes, kernel_size=1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.avgpool(x)
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        out = self.sigmoid(x)
        return out


class BasicBlock(nn.Module): # 左侧的 residual block 结构 (18-layer、34-layer)
    expansion = 1

    def __init__(self, in_planes, planes, stride=1): # 两层卷积 Conv2d + Shutcuts
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.relu = nn.ReLU(inplace=True)
        self.bn2 = nn.BatchNorm2d(planes)

        self.SE = SE_Block(planes)      # Squeeze-and-Excitation block

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes: # Shutcuts用于构建 Conv Block 和
            Identity Block
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        # out = F.relu(self.bn1(self.conv1(x)))
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        SE_out = self.SE(out)
```

```
        out = out * SE_out
        out += self.shortcut(x)
        # out = F.relu(out)
        out = self.relu(out)
        return out


class Bottleneck(nn.Module): # 右侧的 residual block 结构（50-layer、101-layer、152-layer）
    expansion = 4

    def __init__(self, in_planes, planes, stride=1): # 三层卷积 Conv2d + Shutcuts
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion*planes,
                               kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.SE = SE_Block(self.expansion*planes)    # Squeeze-and-Excitation block

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes: # Shutcuts用于构建 Conv Block 和
            Identity Block
           self.shortcut = nn.Sequential(
              nn.Conv2d(in_planes, self.expansion*planes,
                        kernel_size=1, stride=stride, bias=False),
              nn.BatchNorm2d(self.expansion*planes)
           )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        # out = F.leaky_relu(self.bn1(self.conv1(x)))
        # out = F.leaky_relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        SE_out = self.SE(out)
        out = out * SE_out
        out += self.shortcut(x)
        out = F.relu(out)
        return out


class SE_ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=2):
```

```
        super(SE_ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7,
                               stride=2, padding=3, bias=False)           # conv1
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1) # conv2_x
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2) # conv3_x
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2) # conv4_x
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2) # conv5_x
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(x)))
        # x = F.leaky_relu(self.bn1(self.conv1(x)))
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x4 = self.layer4(x3)

        x = self.avgpool(x4)
        x = torch.flatten(x, 1)
        out = self.linear(x)
        return out


def SE_ResNet18():
    return SE_ResNet(BasicBlock, [2, 2, 2, 2])


def SE_ResNet34():
    return SE_ResNet(BasicBlock, [3, 4, 6, 3])


def SE_ResNet50():
    return SE_ResNet(Bottleneck, [3, 4, 6, 3])

```

```

def SE_ResNet101():
    return SE_ResNet(Bottleneck, [3, 4, 23, 3])


def SE_ResNet152():
    return SE_ResNet(Bottleneck, [3, 8, 36, 3])

class SelfAttention(nn.Module):
    def __init__(self, in_channels):
        super(SelfAttention, self).__init__()
        self.in_channels = in_channels

        self.query = nn.Conv2d(in_channels, in_channels, kernel_size=(1, 1))
        self.key = nn.Conv2d(in_channels, in_channels, kernel_size=(1, 1))
        self.value = nn.Conv2d(in_channels, in_channels, kernel_size=(1, 1))
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        batch_size, C, H, W = x.size()

        query = self.query(x).view(batch_size, C, -1)
        key = self.key(x).view(batch_size, C, -1)
        value = self.value(x).view(batch_size, C, -1)

        attention_scores = torch.matmul(query, key.transpose(1, 2))
        attention_scores = attention_scores / (C ** 0.5)
        attention_weights = self.softmax(attention_scores)

        output = torch.matmul(attention_weights, value)

        output = output.view(batch_size, C, H, W)

        return output
```

VGG.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models
from torchvision import models


class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
```

```
        self.vgg = torchvision.models.vgg16(pretrained=False)

        self.vgg.classifier = nn.Sequential(
            nn.Linear(25088, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 2)
        )

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


class VGG19(nn.Module):
    def __init__(self):
        super(VGG19, self).__init__()
        self.vgg = torchvision.models.vgg19(pretrained=False)

        self.vgg.classifier = nn.Sequential(
            nn.Linear(25088, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 2)
        )
    def forward(self, x):
        return self.vgg(x)
```

vit.py

```
from vit_pytorch import ViT
import torch.nn as nn


class VIT(nn.Module):
    def __init__(self):
        super(VIT, self).__init__()
        self.vit = ViT(
            image_size=224,
            patch_size=28,
            num_classes=2,
```

```
            dim=1024,
            depth=6,
            heads=16,
            mlp_dim=2048,
            dropout=0.1,
            emb_dropout=0.1
        )
    def forward(self, x):
        return self.vit(x)
```

1.2 改进的循环生成式对抗网络模型相关源代码

1.2.1 训练与测试源代码

train.py（改进 CycleGAN 的训练代码）

```

import argparse
import itertools

import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import torch
from models.SEnet import SE_ResNet18
from model import Generator
from utils import ReplayBuffer
from utils import LambdaLR
from utils import Logger
from datasets import ImageDataset
import torch.nn.functional as F

parser = argparse.ArgumentParser()
parser.add_argument('--epoch', type=int, default=0, help='starting epoch')
parser.add_argument('--n_epochs', type=int, default=200, help='number of epochs of training')
parser.add_argument('--batchSize', type=int, default=1, help='size of the batches')
parser.add_argument('--dataroot', type=str, default='datasets/road', help='root directory of
    the dataset')
parser.add_argument('--lr', type=float, default=0.0002, help='initial learning rate')
parser.add_argument('--decay_epoch', type=int, default=100, help='epoch to start linearly
    decaying the learning rate to 0')
parser.add_argument('--size', type=int, default=256, help='size of the data crop (squared
    assumed)')
parser.add_argument('--input_nc', type=int, default=3, help='number of channels of input data')
parser.add_argument('--output_nc', type=int, default=3, help='number of channels of output
    data')
```

```
parser.add_argument('--cuda', default='0', action='store_true', help='use GPU computation')
parser.add_argument('--n_cpu', type=int, default=8, help='number of cpu threads to use during batch generation')
opt = parser.parse_args()
print(opt)

if torch.cuda.is_available() and not opt.cuda:
    print("WARNING: You have a CUDA device, so you should probably run with --cuda")

###### Definition of variables ######
# Networks
netG_A2B = Generator(opt.input_nc, opt.output_nc)
netG_B2A = Generator(opt.output_nc, opt.input_nc)
netD_A = torch.load('SENet.pt')
netD_B = torch.load('SENet.pt')
netD_A = netD_A.eval()
netD_B = netD_B.eval()
if opt.cuda:
    netG_A2B.cuda()
    netG_B2A.cuda()
    netD_A.cuda()
    netD_B.cuda()

# Lossess
criterion_GAN = torch.nn.MSELoss()
criterion_cycle = torch.nn.L1Loss()
criterion_identity = torch.nn.L1Loss()

# Optimizers & LR schedulers
optimizer_G = torch.optim.Adam(itertools.chain(netG_A2B.parameters(), netG_B2A.parameters()),
                               lr=opt.lr, betas=(0.5, 0.999))
# optimizer_D_A = torch.optim.Adam(netD_A.parameters(), lr=opt.lr, betas=(0.5, 0.999))
# optimizer_D_B = torch.optim.Adam(netD_B.parameters(), lr=opt.lr, betas=(0.5, 0.999))

lr_scheduler_G = torch.optim.lr_scheduler.LambdaLR(optimizer_G,
    lr_lambda=LambdaLR(opt.n_epochs, opt.epoch, opt.decay_epoch).step)
# lr_scheduler_D_A = torch.optim.lr_scheduler.LambdaLR(optimizer_D_A,
    lr_lambda=LambdaLR(opt.n_epochs, opt.epoch, opt.decay_epoch).step)
# lr_scheduler_D_B = torch.optim.lr_scheduler.LambdaLR(optimizer_D_B,
    lr_lambda=LambdaLR(opt.n_epochs, opt.epoch, opt.decay_epoch).step)

# Inputs & targets memory allocation
Tensor = torch.cuda.FloatTensor if opt.cuda else torch.Tensor
input_A = Tensor(opt.batchSize, opt.input_nc, opt.size, opt.size)
input_B = Tensor(opt.batchSize, opt.output_nc, opt.size, opt.size)
target_real = Variable(Tensor(opt.batchSize).fill_(1.0), requires_grad=False)
target_fake = Variable(Tensor(opt.batchSize).fill_(0.0), requires_grad=False)
```

```
70
71  fake_A_buffer = ReplayBuffer()
72  fake_B_buffer = ReplayBuffer()
73
74  # Dataset loader
75  transforms_ = [
76      transforms.Resize((256,256)),
77      # transforms.RandomHorizontalFlip(),
78      # transforms.RandomVerticalFlip(),
79      # transforms.ColorJitter(),
80      transforms.ToTensor()
81  ]
82  dataloader = DataLoader(ImageDataset(opt.dataroot, transforms_=transforms_, unaligned=True),
83                          batch_size=opt.batchSize, shuffle=False, num_workers=opt.n_cpu)
84
85  # Loss plot
86  #logger = Logger(opt.n_epochs, len(dataloader))
87  ###################################
88
89  ###### Training ######
90  for epoch in range(opt.epoch, opt.n_epochs):
91      for i, batch in enumerate(dataloader):
92          # Set model input
93          real_A = Variable(input_A.copy_(batch['A']))
94          real_B = Variable(input_B.copy_(batch['B']))
95          ###### Generators A2B and B2A ######
96          optimizer_G.zero_grad()
97
98          # Identity loss
99          # G_A2B(B) should equal B if real B is fed
100         same_B = netG_A2B(real_B)
101         loss_identity_B = criterion_identity(same_B, real_B)*5.0
102         # G_B2A(A) should equal A if real A is fed
103         same_A = netG_B2A(real_A)
104         loss_identity_A = criterion_identity(same_A, real_A)*5.0
105
106         # GAN loss
107         fake_B = netG_A2B(real_A)
108         pred_fake = F.softmax(netD_B(fake_B), dim=1)[:, 1:]
109
110         loss_GAN_A2B = criterion_GAN(pred_fake, target_real)
111
112         fake_A = netG_B2A(real_B)
113         pred_fake = F.softmax(netD_B(fake_A), dim=1)[:, :1]
114         loss_GAN_B2A = criterion_GAN(pred_fake, target_real)
115
116         # Cycle loss
```

```
        recovered_A = netG_B2A(fake_B)
        loss_cycle_ABA = criterion_cycle(recovered_A, real_A)*10.0

        recovered_B = netG_A2B(fake_A)
        loss_cycle_BAB = criterion_cycle(recovered_B, real_B)*10.0

        # Total loss
        loss_G = loss_identity_A + loss_identity_B + loss_GAN_A2B + loss_GAN_B2A +
            loss_cycle_ABA + loss_cycle_BAB
        loss_G.backward()

        optimizer_G.step()

        print('epoch', str(epoch), 'loss_G:', loss_G.item(), 'loss_G_identity:',
            (loss_identity_A.item() + loss_identity_B.item()), 'loss_G_GAN:'
            ,(loss_GAN_A2B.item()+ loss_GAN_B2A).item(),'loss_G_cycle:', (loss_cycle_ABA.item()
            + loss_cycle_BAB.item()))


    lr_scheduler_G.step()

    # Save models checkpoints
    torch.save(netG_A2B.state_dict(), 'output/netG_A2B.pth')
    torch.save(netG_B2A.state_dict(), 'output/netG_B2A.pth')
    torch.save(netD_A.state_dict(), 'output/netD_A.pth')
    torch.save(netD_B.state_dict(), 'output/netD_B.pth')
###################################
```

test.py（生成伪样本）

```
#!/usr/bin/python3

import argparse
import sys
import os

import cv2
import torchvision.transforms as transforms
from torchvision.utils import save_image
from torch.utils.data import DataLoader
from torch.autograd import Variable
import torch
import torch.nn.functional as F
from model import Generator
from datasets import ImageDataset

parser = argparse.ArgumentParser()
```

```
parser.add_argument('--batchSize', type=int, default=1, help='size of the batches')
parser.add_argument('--dataroot', type=str, default='datasets/road', help='root directory of
    the dataset')
parser.add_argument('--input_nc', type=int, default=3, help='number of channels of input data')
parser.add_argument('--output_nc', type=int, default=3, help='number of channels of output
    data')
parser.add_argument('--size', type=int, default=256, help='size of the data (squared assumed)')
parser.add_argument('--cuda', action='store_true', help='use GPU computation')
parser.add_argument('--n_cpu', type=int, default=8, help='number of cpu threads to use during
    batch generation')
parser.add_argument('--generator_A2B', type=str, default='output/netG_A2B.pth', help='A2B
    generator checkpoint file')
parser.add_argument('--generator_B2A', type=str, default='output/netG_B2A.pth', help='B2A
    generator checkpoint file')
opt = parser.parse_args()
print(opt)

if torch.cuda.is_available() and not opt.cuda:
    print("WARNING: You have a CUDA device, so you should probably run with --cuda")

###### Definition of variables ######
# Networks
netG_A2B = Generator(opt.input_nc, opt.output_nc)
netG_B2A = Generator(opt.output_nc, opt.input_nc)
D = torch.load('SENet.pt')

if opt.cuda:
    netG_A2B.cuda()
    netG_B2A.cuda()
    D.cuda()

# Load state dicts
netG_A2B.load_state_dict(torch.load(opt.generator_A2B))
netG_B2A.load_state_dict(torch.load(opt.generator_B2A))

# Set model's test.py mode
netG_A2B.eval()
netG_B2A.eval()

# Inputs & targets memory allocation
Tensor = torch.cuda.FloatTensor if opt.cuda else torch.Tensor
input_A = Tensor(opt.batchSize, opt.input_nc, opt.size, opt.size)
input_B = Tensor(opt.batchSize, opt.output_nc, opt.size, opt.size)

# Dataset loader
transforms_ = transforms.Compose([transforms.ToTensor(),
              transforms.Resize([256, 256]),
```

```
            transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))])
dataloader = DataLoader(ImageDataset(opt.dataroot, transforms_=transforms_, mode='test'),
                        batch_size=opt.batchSize, shuffle=False, num_workers=opt.n_cpu)
###################################

###### Testing######

# Create output dirs if they don't exist
if not os.path.exists('output/A'):
    os.makedirs('output/A')
if not os.path.exists('output/B'):
    os.makedirs('output/B')
#
# for i, batch in enumerate(dataloader):
#     # Generate output
#     fake_B = 0.5*(netG_A2B(real_A).data + 1.0).cuda()
#     fake_A = 0.5*(netG_B2A(real_B).data + 1.0).cuda()
#
#     # Save image files
#     save_image(fake_A, 'output/A/normal_%04d.png' % (i+1))
#     save_image(fake_B, 'output/B/potholes_%04d.png' % (i+1))
#
#     sys.stdout.write('\rGenerated images %04d of %04d' % (i+1, len(dataloader)))
#
# sys.stdout.write('\n')
###################################

count_A = 0
count_B = 0
for i in os.listdir('./data_new_new/data5/train'):
    if i.startswith('normal'):
        image = cv2.imread(os.path.join('./data_new_new/data5/train', i))
        trans_image = transforms_(image)
        trans_image = trans_image.unsqueeze(0)
        fake_B = 0.5 * (netG_A2B(trans_image).data + 1.0).cuda()
        save_image(fake_B, 'output/A/normal_%04d.png' % (count_A + 1))
        count_A += 1
    else:
        image = cv2.imread(os.path.join('./data_new_new/data5/train', i))
        trans_image = transforms_(image)
        trans_image = trans_image.unsqueeze(0)
        fake_A = 0.5*(netG_B2A(trans_image).data + 1.0).cuda()
        save_image(fake_A, 'output/B/pothles_%04d.png' % (count_B + 1))
        count_B += 1
```

1.2.2 模型结构与辅助函数源代码

model.py（模型结构）

```
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models

class ResidualBlock(nn.Module):
    def __init__(self, in_features):
        super(ResidualBlock, self).__init__()

        conv_block = [ nn.ReflectionPad2d(1),
                       nn.Conv2d(in_features, in_features, 3),
                       nn.InstanceNorm2d(in_features),
                       nn.ReLU(inplace=True),
                       nn.ReflectionPad2d(1),
                       nn.Conv2d(in_features, in_features, 3),
                       nn.InstanceNorm2d(in_features) ]

        self.conv_block = nn.Sequential(*conv_block)

    def forward(self, x):
        return x + self.conv_block(x)

class Generator(nn.Module):
    def __init__(self, input_nc, output_nc, n_residual_blocks=9):
        super(Generator, self).__init__()

        # Initial convolution block
        model = [ nn.ReflectionPad2d(3),
                  nn.Conv2d(input_nc, 64, 7),
                  nn.InstanceNorm2d(64),
                  nn.ReLU(inplace=True) ]

        # Downsampling
        in_features = 64
        out_features = in_features*2
        for _ in range(2):
            model += [ nn.Conv2d(in_features, out_features, 3, stride=2, padding=1),
                       nn.InstanceNorm2d(out_features),
                       nn.ReLU(inplace=True) ]
            in_features = out_features
            out_features = in_features*2

        # Residual blocks
        for _ in range(n_residual_blocks):
```

```
            model += [ResidualBlock(in_features)]

        # Upsampling
        out_features = in_features//2
        for _ in range(2):
            model += [ nn.ConvTranspose2d(in_features, out_features, 3, stride=2, padding=1,
                output_padding=1),
                      nn.InstanceNorm2d(out_features),
                      nn.ReLU(inplace=True) ]
            in_features = out_features
            out_features = in_features//2

        # Output layer
        model += [ nn.ReflectionPad2d(3),
                   nn.Conv2d(64, output_nc, 7),
                   nn.Tanh() ]

        self.model = nn.Sequential(*model)

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


class ResNet18(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet18, self).__init__()
        self.resnet = models.resnet18(pretrained=True) # 使用预训练的权重
        # 替换最后一层全连接层，使其适应特定数量的类别
        num_ftrs = self.resnet.fc.in_features
        # 添加一个Dropout层
        self.dropout = nn.Dropout(p=0.5)

        # 替换全连接层，将Dropout层插入全连接层之前
        self.resnet.fc = nn.Sequential(
            self.dropout,
            nn.Linear(num_ftrs, num_classes)
        )

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

utils.py（辅助函数）

```
import random
import time
import datetime
import sys
```

```
from torch.autograd import Variable
import torch
from visdom import Visdom
import numpy as np

def tensor2image(tensor):
    image = 127.5*(tensor[0].cpu().float().numpy() + 1.0)
    if image.shape[0] == 1:
        image = np.tile(image, (3,1,1))
    return image.astype(np.uint8)

class Logger():
    def __init__(self, n_epochs, batches_epoch):
        self.viz = Visdom()
        self.n_epochs = n_epochs
        self.batches_epoch = batches_epoch
        self.epoch = 1
        self.batch = 1
        self.prev_time = time.time()
        self.mean_period = 0
        self.losses = {}
        self.loss_windows = {}
        self.image_windows = {}


    def log(self, losses=None, images=None):
        self.mean_period += (time.time() - self.prev_time)
        self.prev_time = time.time()

        sys.stdout.write('\rEpoch %03d/%03d [%04d/%04d] -- ' % (self.epoch, self.n_epochs,
            self.batch, self.batches_epoch))

        for i, loss_name in enumerate(losses.keys()):
            if loss_name not in self.losses:
                self.losses[loss_name] = losses[loss_name].item()
            else:
                self.losses[loss_name] += losses[loss_name].item()

            if (i+1) == len(losses.keys()):
                sys.stdout.write('%s: %.4f -- ' % (loss_name, self.losses[loss_name]/self.batch))
            else:
                sys.stdout.write('%s: %.4f | ' % (loss_name, self.losses[loss_name]/self.batch))

        batches_done = self.batches_epoch*(self.epoch - 1) + self.batch
        batches_left = self.batches_epoch*(self.n_epochs - self.epoch) + self.batches_epoch -
            self.batch
```

```
        sys.stdout.write('ETA: %s' %
            (datetime.timedelta(seconds=batches_left*self.mean_period/batches_done)))

        # Draw images
        for image_name, tensor in images.items():
            if image_name not in self.image_windows:
                self.image_windows[image_name] = self.viz.image(tensor2image(tensor.data),
                    opts={'title':image_name})
            else:
                self.viz.image(tensor2image(tensor.data), win=self.image_windows[image_name],
                    opts={'title':image_name})

        # End of epoch
        if (self.batch % self.batches_epoch) == 0:
            # Plot losses
            for loss_name, loss in self.losses.items():
                if loss_name not in self.loss_windows:
                    self.loss_windows[loss_name] = self.viz.line(X=np.array([self.epoch]),
                        Y=np.array([loss/self.batch]),
                                                                 opts={'xlabel': 'epochs', 'ylabel':
                                                                     loss_name, 'title': loss_name})
                else:
                    self.viz.line(X=np.array([self.epoch]), Y=np.array([loss/self.batch]),
                        win=self.loss_windows[loss_name], update='append')
                # Reset losses for next epoch
                self.losses[loss_name] = 0.0

            self.epoch += 1
            self.batch = 1
            sys.stdout.write('\n')
            self.batch += 1

class ReplayBuffer():
    def __init__(self, max_size=50):
        assert (max_size > 0), 'Empty buffer or trying to create a black hole. Be careful.'
        self.max_size = max_size
        self.data = []

    def push_and_pop(self, data):
        to_return = []
        for element in data.data:
            element = torch.unsqueeze(element, 0)
            if len(self.data) < self.max_size:
                self.data.append(element)
                to_return.append(element)
            else:
                if random.uniform(0,1) > 0.5:
```

```
                i = random.randint(0, self.max_size-1)
                to_return.append(self.data[i].clone())
                self.data[i] = element
            else:
                to_return.append(element)
        return Variable(torch.cat(to_return))

class LambdaLR():
    def __init__(self, n_epochs, offset, decay_start_epoch):
        assert ((n_epochs - decay_start_epoch) > 0), "Decay must start before the training
            session ends!"
        self.n_epochs = n_epochs
        self.offset = offset
        self.decay_start_epoch = decay_start_epoch

    def step(self, epoch):
        return 1.0 - max(0, epoch + self.offset - self.decay_start_epoch)/(self.n_epochs -
            self.decay_start_epoch)

def weights_init_normal(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        torch.nn.init.normal(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm2d') != -1:
        torch.nn.init.normal(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant(m.bias.data, 0.0)
```

1.3 模型推理填写结果表格相关源代码

fill_csv.py

```
import os
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 = './testdata_V2'

```

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

# 初始化一个CSV文件
csv_file = 'test_result.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})

print("CSV文件已创建并写入结果。")
```