

安徽大学人工智能学院
《机器学习程序设计》
实验案例设计报告

课程名称: 机器学习程序设计

专 业: 人工智能

班 级: 人工智能二班

学 号: WA2214014

姓 名: 杨跃浙

任课老师: 谭春雨

实验名称	基于 UNet 算法的农业遥感图像分割	实验次序	04
实验地点	笃行南楼 A104	实验日期	04.13

实验内容:

1. 案例描述

本实验基于 Unet 深度神经网络实现农业遥感图像的图义分割。

对训练数据进行预处理，筛选掉不合格的图片，划分出训练集和验证集。

使用 keras 深度学习库搭建 Unet 深度神经网络。并在训练过程中采用冻结部分网络层的方法，这样能使 Unet 网络优先学习抽象特征，具有更强的泛化性，且保护底层网络参数不被破坏。

2. 数据集描述

数据集出处：阿里云天池平台

数据集格式：<图片，标记图>。图片为(512x512x3)格式的 RGB 图片。标记好的图片为(512x512)格式的图片，是对对应图片的每个像素点进行标记后的图片，其每个像素点值表示原始图片中对应位置所属类别。

数据来源于无人机遥感影像，无人机遥感测量技术作为空间信息技术的重要组成部分，既能作为星载遥感影像的重要补充，又能有效替代人工实地调查，凭借着降低地面人工调查强度和调查成本、快速获取实时高分辨数据的优势，成为农业统计调查工作中的一大创新点，同时也是精准农业的重要方向之一。

3. 数据预处理

据统计，数据集中有大量的无效标记，例如有的标记图片所有的像素值都为 0。这类无效的标记对模型的训练并无意义。因此需要对数据集进行筛选。

遍历数据集的标签，若该标签中每个种类的像素值都在图片总像素值的 70% 以下，那么将该标签以及标签对应的图片筛选出来，保存在‘data/0.7/’文件夹中。

将数据集按照 7：3 的比例划分为训练集和验证集。将划分好的训练集的图片名称保存在‘data/0.7/train.txt’文件夹中，同样操作验证集。

代码：

```
import os
import cv2
import numpy as np
import random
import shutil
from tqdm import tqdm

# 参数配置
percentage_threshold = 0.7
train_percentage = 0.7

# 路径配置
image_dir = '/home/yyz/Unet-ML/data/image/train_data'
label_dir = '/home/yyz/Unet-ML/data/label/train_label'
save_image_dir = '/home/yyz/Unet-ML/data/0.7/image'
save_label_dir = '/home/yyz/Unet-ML/data/0.7/label'
os.makedirs(save_image_dir, exist_ok=True)
os.makedirs(save_label_dir, exist_ok=True)

# 筛选符合条件的图像
qualified_filenames = []
image_files = [f for f in os.listdir(image_dir) if f.endswith('.png') or
f.endswith('.jpg')]
```

```

for filename in tqdm(image_files, desc="Checking label distribution"):
    image_path = os.path.join(image_dir, filename)
    label_path = os.path.join(label_dir, filename)

    label = cv2.imread(label_path, cv2.IMREAD_GRAYSCALE)
    if label is None:
        print(f"Warning: Label not found or unreadable: {label_path}")
        continue

    total_pixels = label.size
    unique, counts = np.unique(label, return_counts=True)
    stats = dict(zip(unique, counts))

    # 检查四类中每类是否都低于阈值
    if all(stats.get(i, 0) / total_pixels < percentage_threshold for i in [0, 1, 2, 3]):
        qualified_filenames.append(filename)
    # 复制图像与标签到 0.7 文件夹
    shutil.copy(image_path, os.path.join(save_image_dir, filename))
    shutil.copy(label_path, os.path.join(save_label_dir, filename))

    print(f"Qualified samples: {len(qualified_filenames)}")

    # 数据划分
    total_num = len(qualified_filenames)
    train_num = int(total_num * train_percentage)
    index_list = list(range(total_num))
    train_indices = random.sample(index_list, train_num)

    train_txt_path = '/home/yyz/Unet-ML/data/0.7/train.txt'
    val_txt_path = '/home/yyz/Unet-ML/data/0.7/val.txt'

    with open(train_txt_path, 'w') as f_train, open(val_txt_path, 'w') as f_val:
        for idx in index_list:
            filename = qualified_filenames[idx]
            line = filename + '\n'
            if idx in train_indices:
                f_train.write(line)
            else:
                f_val.write(line)

    print("Train/Val split done.")
    print(f"Train size: {train_num}")
    print(f"Val size: {total_num - train_num}")

```

The screenshot displays the PyCharm IDE interface. On the left, the 'EXPLORER' panel shows the project structure with 'data' containing '0.7' and 'label' subdirectories. The 'preprocessor.py' file is open in the editor, showing code for image processing and label distribution. The terminal at the bottom shows the execution of the script, with a red box highlighting the error message: 'NameError: name 'f_train' is not defined'.

[illegible]

Unet 模型是一个用于图片语义分割的深度学习模型，呈 U 型对称结构，因此得名 Unet。

模型左半部分为主干特征提取网络，用于提取一些底层简单的特征，例如颜色，纹理，轮廓等特征。

模型右半部分为加强特征提取网络，通过上采样以及与主干特征网络进行拼接，使得模型能同时学习到底层简单特征以及高层抽象特征。最后模型将输出结果复原到输入图片的分辨率。

定义 UNET 主干特征提取网络模块：

Unet 主干特征提取网络分为 5 个模块，每个模块中进行卷积，池化操作。

定义完整的 UNET 网络：

在主干特征提取网络的基础上加上加强特征提取网络，从而构成完整的 Unet 网络。加强特征提取网络中包含上采样，卷积，拼接操作。

3

经验 (<https://bean-young.github.io>)。顺便说一下实验环境,感谢老师提供的学校服务器,此次实验均在一张 NVIDIA GeForce RTX 3090 24G 上完成。

[3] NVIDIA GeForce RTX 3090 | 62°C, 84 % | 10148 / 24576 MB | yyz(10140M) gdm(4M)

代码:

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

class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

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

class UNet(nn.Module):
    def __init__(self, in_channels=1, num_classes=1, base_c=64):
        super(UNet, self).__init__()

        self.encoder1 = DoubleConv(in_channels, base_c)
        self.pool1 = nn.MaxPool2d(2)

        self.encoder2 = DoubleConv(base_c, base_c * 2)
        self.pool2 = nn.MaxPool2d(2)

        self.encoder3 = DoubleConv(base_c * 2, base_c * 4)
        self.pool3 = nn.MaxPool2d(2)

        self.encoder4 = DoubleConv(base_c * 4, base_c * 8)
        self.pool4 = nn.MaxPool2d(2)

        self.bottleneck = DoubleConv(base_c * 8, base_c * 16)

        self.upconv4 = nn.ConvTranspose2d(base_c * 16, base_c * 8, kernel_size=2,
            stride=2)
        self.decoder4 = DoubleConv(base_c * 16, base_c * 8)

        self.upconv3 = nn.ConvTranspose2d(base_c * 8, base_c * 4, kernel_size=2,
            stride=2)
        self.decoder3 = DoubleConv(base_c * 8, base_c * 4)

        self.upconv2 = nn.ConvTranspose2d(base_c * 4, base_c * 2, kernel_size=2,
            stride=2)
        self.decoder2 = DoubleConv(base_c * 4, base_c * 2)

        self.upconv1 = nn.ConvTranspose2d(base_c * 2, base_c, kernel_size=2, stride=2)
        self.decoder1 = DoubleConv(base_c * 2, base_c)

        self.final_conv = nn.Conv2d(base_c, num_classes, kernel_size=1)

    def forward(self, x):
        e1 = self.encoder1(x)
        e2 = self.encoder2(self.pool1(e1))
        e3 = self.encoder3(self.pool2(e2))
        e4 = self.encoder4(self.pool3(e3))
        b = self.bottleneck(self.pool4(e4))
```

```

d4 = self.upconv4(b)
d4 = self.decoder4(torch.cat([d4, e4], dim=1))

d3 = self.upconv3(d4)
d3 = self.decoder3(torch.cat([d3, e3], dim=1))

d2 = self.upconv2(d3)
d2 = self.decoder2(torch.cat([d2, e2], dim=1))

d1 = self.upconv1(d2)
d1 = self.decoder1(torch.cat([d1, e1], dim=1))

return self.final_conv(d1)

```

这里附上我对应的 Dataloader 和 Main 函数

main.py

```

# Main.py
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "3"
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
from model.Unet import UNet
from dataloader.loaderseg import SegDataset

# 参数
EPOCHS = 50
BATCH_SIZE = 4
LR = 1e-3
NUM_CLASSES = 4
SAVE_DIR = '/home/yyz/Unet-ML/result'
os.makedirs(SAVE_DIR, exist_ok=True)

# 路径
IMAGE_DIR = '/home/yyz/Unet-ML/data/0.7/image'
LABEL_DIR = '/home/yyz/Unet-ML/data/0.7/label'
TRAIN_LIST = '/home/yyz/Unet-ML/data/0.7/train.txt'
VAL_LIST = '/home/yyz/Unet-ML/data/0.7/val.txt'

# 读文件名
with open(TRAIN_LIST, 'r') as f:
    train_files = f.readlines()
with open(VAL_LIST, 'r') as f:
    val_files = f.readlines()

# 转换
transform = transforms.Compose([
    transforms.Resize((512, 512)),
    transforms.ToTensor()
])

# 数据
train_dataset = SegDataset(IMAGE_DIR, LABEL_DIR, train_files, transform)
val_dataset = SegDataset(IMAGE_DIR, LABEL_DIR, val_files, transform)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=1)

```

```

# 模型与优化
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = UNet(in_channels=3, num_classes=NUM_CLASSES).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LR)

# 指标记录
train_losses, train_accs = [], []
val_losses, val_accs, val_iou, val_dices = [], [], [], []

def calc_metrics(pred, target, num_classes=4):
    pred = torch.argmax(pred, dim=1) # [B, H, W]
    target = target # [B, H, W]

    acc = (pred == target).sum().item() / target.numel()

    iou_list = []
    dice_list = []
    for cls in range(num_classes):
        pred_cls = (pred == cls)
        target_cls = (target == cls)

        intersection = (pred_cls & target_cls).sum().item()
        union = (pred_cls | target_cls).sum().item()
        iou = intersection / (union + 1e-8)

        dice = 2 * intersection / (pred_cls.sum().item() + target_cls.sum().item() + 1e-8)

        iou_list.append(iou)
        dice_list.append(dice)

    mean_iou = sum(iou_list) / num_classes
    mean_dice = sum(dice_list) / num_classes
    return acc, mean_iou, mean_dice

# 训练
for epoch in range(EPOCHS):
    model.train()
    total_loss, total_correct, total_pixels = 0, 0, 0
    for img, mask in train_loader:
        img, mask = img.to(device), mask.to(device)
        output = model(img)
        loss = criterion(output, mask)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    total_loss += loss.item()
    pred = torch.argmax(output, dim=1)
    total_correct += (pred == mask).sum().item()
    total_pixels += mask.numel()

    train_loss = total_loss / len(train_loader)
    train_acc = total_correct / total_pixels
    train_losses.append(train_loss)
    train_accs.append(train_acc)

# 验证
model.eval()
val_loss, val_correct, val_pixels = 0, 0, 0
iou_sum, dice_sum = 0, 0
with torch.no_grad():
    for img, mask in val_loader:
        img, mask = img.to(device), mask.to(device)
        output = model(img)
        loss = criterion(output, mask)
        val_loss += loss.item()

    acc, iou, dice = calc_metrics(output, mask, NUM_CLASSES)

```

```

val_correct += acc * mask.numel()
val_pixels += mask.numel()
iou_sum += iou
dice_sum += dice

val_losses.append(val_loss / len(val_loader))
val_accs.append(val_correct / val_pixels)
val_ious.append(iou_sum / len(val_loader))
val_dices.append(dice_sum / len(val_loader))

print(f"Epoch [{epoch+1}/{EPOCHS}] "
      f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | "
      f"Val Loss: {val_losses[-1]:.4f} | Val Acc: {val_accs[-1]:.4f} | "
      f"IOU: {val_ious[-1]:.4f} | Dice: {val_dices[-1]:.4f}")

# 画图保存
plt.figure()
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.legend()
plt.title('Loss Curve')
plt.savefig(os.path.join(SAVE_DIR, 'loss_curve.png'))

plt.figure()
plt.plot(train_accs, label='Train Acc')
plt.plot(val_accs, label='Val Acc')
plt.legend()
plt.title('Accuracy Curve')
plt.savefig(os.path.join(SAVE_DIR, 'accuracy_curve.png'))

plt.figure()
plt.plot(val_ious, label='Val IOU')
plt.plot(val_dices, label='Val Dice')
plt.legend()
plt.title('IOU & Dice Curve')
plt.savefig(os.path.join(SAVE_DIR, 'iou_dice_curve.png'))

```

loaderseg.py

```

import os
import torch
from torch.utils.data import Dataset
from PIL import Image
import torchvision.transforms as transforms
import numpy as np

class SegDataset(Dataset):
    def __init__(self, image_dir, label_dir, file_list, transform=None):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.file_list = file_list
        self.transform = transform
        self.resize = transforms.Resize((512, 512), interpolation=Image.NEAREST)

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

    def __getitem__(self, idx):
        filename = self.file_list[idx].strip()
        image_path = os.path.join(self.image_dir, filename)
        label_path = os.path.join(self.label_dir, filename)

        image = Image.open(image_path).convert('RGB')
        label = Image.open(label_path)

        # resize 图像与标签
        image = self.resize(image)

```



```

label = self.resize(label)

if self.transform:
    image = self.transform(image)

# 转换 label 为 [H, W] 的 LongTensor, 内容是类别编号 (0~3)
label = torch.from_numpy(np.array(label)).long()

return image, label

```

运行结果:

The image displays two screenshots of a VS Code editor window showing a PyTorch training script and its output. The editor is open to a file named `main.py` in a project named `Unet-ML`. The script defines a `transforms.ToTensor()` function, sets up data loaders for training and validation, and defines a `model` and `optimizer`. The training loop is shown, with metrics like `train_loss`, `train_accs`, `val_loss`, `val_accs`, `val_iou`, and `val_dice` being tracked. The output shows the training progress over 50 epochs, with the model being saved at the end.

First Screenshot (Epochs 1-10):

```

Epoch [1/50] Train Loss: 1.3178 | Train Acc: 0.3329 | Val Loss: 2.3812 | Val Acc: 0.4429 | IOU: 0.1188 | Dice: 0.1479
Epoch [2/50] Train Loss: 1.2281 | Train Acc: 0.4410 | Val Loss: 2.3806 | Val Acc: 0.4389 | IOU: 0.1189 | Dice: 0.1493
Epoch [3/50] Train Loss: 1.2837 | Train Acc: 0.4410 | Val Loss: 1.2689 | Val Acc: 0.4297 | IOU: 0.1286 | Dice: 0.1699
Epoch [4/50] Train Loss: 1.1562 | Train Acc: 0.4496 | Val Loss: 1.5898 | Val Acc: 0.4856 | IOU: 0.1283 | Dice: 0.2094
Epoch [5/50] Train Loss: 1.1495 | Train Acc: 0.4616 | Val Loss: 1.3533 | Val Acc: 0.4682 | IOU: 0.1481 | Dice: 0.1927
Epoch [6/50] Train Loss: 1.1511 | Train Acc: 0.4583 | Val Loss: 1.4471 | Val Acc: 0.4165 | IOU: 0.1438 | Dice: 0.2048
Epoch [7/50] Train Loss: 1.0969 | Train Acc: 0.4868 | Val Loss: 1.5428 | Val Acc: 0.4332 | IOU: 0.1881 | Dice: 0.2463
Epoch [8/50] Train Loss: 1.0965 | Train Acc: 0.4782 | Val Loss: 1.2218 | Val Acc: 0.4821 | IOU: 0.1553 | Dice: 0.2118
Epoch [9/50] Train Loss: 1.1814 | Train Acc: 0.4798 | Val Loss: 1.0841 | Val Acc: 0.5585 | IOU: 0.1932 | Dice: 0.2455
Epoch [10/50] Train Loss: 1.1838 | Train Acc: 0.4921 | Val Loss: 1.1694 | Val Acc: 0.4778 | IOU: 0.1967 | Dice: 0.2678

```

Second Screenshot (Epochs 39-50):

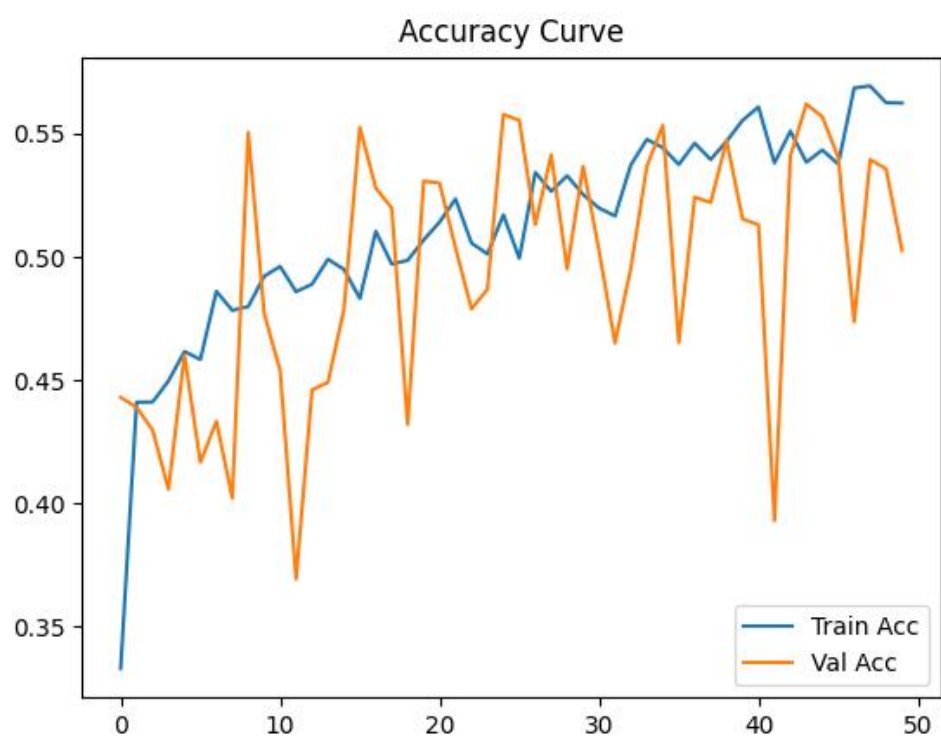
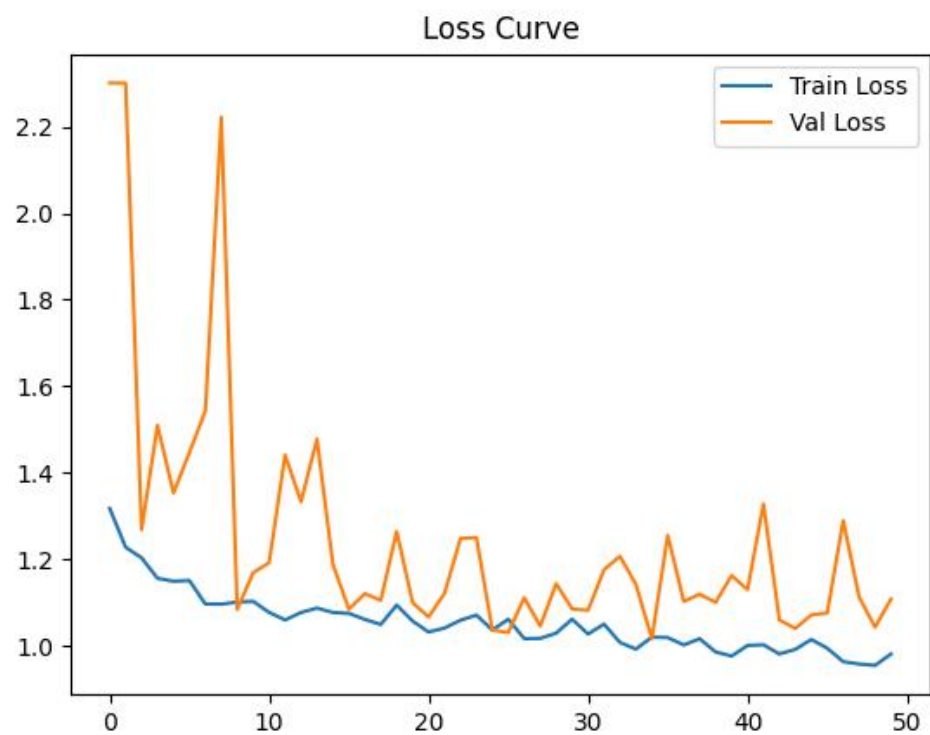
```

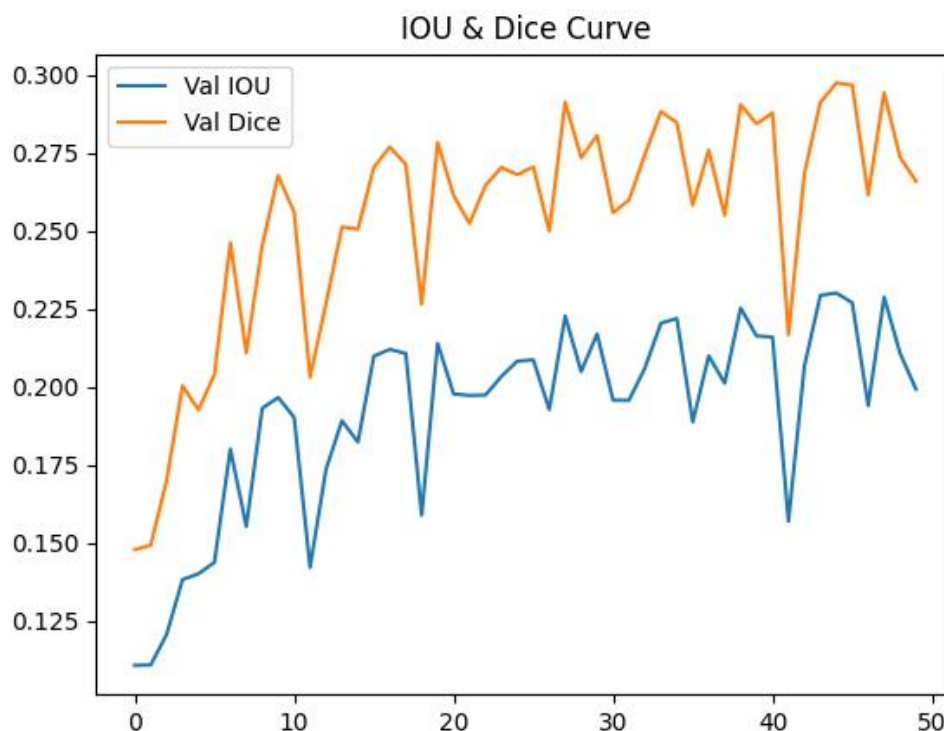
Epoch [39/50] Train Loss: 0.9868 | Train Acc: 0.5478 | Val Loss: 1.1088 | Val Acc: 0.5473 | IOU: 0.2253 | Dice: 0.2986
Epoch [40/50] Train Loss: 0.9765 | Train Acc: 0.5554 | Val Loss: 1.1627 | Val Acc: 0.5154 | IOU: 0.2164 | Dice: 0.2844
Epoch [41/50] Train Loss: 1.0010 | Train Acc: 0.5609 | Val Loss: 1.1382 | Val Acc: 0.5138 | IOU: 0.2160 | Dice: 0.2879
Epoch [42/50] Train Loss: 1.0827 | Train Acc: 0.5381 | Val Loss: 1.3282 | Val Acc: 0.3929 | IOU: 0.1570 | Dice: 0.2168
Epoch [43/50] Train Loss: 0.9814 | Train Acc: 0.5511 | Val Loss: 1.8599 | Val Acc: 0.5489 | IOU: 0.2867 | Dice: 0.2686
Epoch [44/50] Train Loss: 0.9921 | Train Acc: 0.5384 | Val Loss: 1.0482 | Val Acc: 0.5628 | IOU: 0.2294 | Dice: 0.2913
Epoch [45/50] Train Loss: 1.0148 | Train Acc: 0.5434 | Val Loss: 1.0717 | Val Acc: 0.5569 | IOU: 0.2382 | Dice: 0.2975
Epoch [46/50] Train Loss: 0.9948 | Train Acc: 0.5377 | Val Loss: 1.0708 | Val Acc: 0.5484 | IOU: 0.2271 | Dice: 0.2968
Epoch [47/50] Train Loss: 0.9636 | Train Acc: 0.5886 | Val Loss: 1.2897 | Val Acc: 0.4736 | IOU: 0.1941 | Dice: 0.2616
Epoch [48/50] Train Loss: 0.9582 | Train Acc: 0.5694 | Val Loss: 1.1127 | Val Acc: 0.5395 | IOU: 0.2289 | Dice: 0.2945
Epoch [49/50] Train Loss: 0.9551 | Train Acc: 0.5626 | Val Loss: 1.0433 | Val Acc: 0.5357 | IOU: 0.2188 | Dice: 0.2737
Epoch [50/50] Train Loss: 0.9813 | Train Acc: 0.5624 | Val Loss: 1.1082 | Val Acc: 0.5026 | IOU: 0.1993 | Dice: 0.2660

```

The output also indicates that the model has been saved to `/home/yyz/Unet-ML/result/unet_model.pth`.

曲线图:





结果分析：

从给出的实验数值结果来看，训练过程中训练集损失（Train Loss）从第 1 epoch 的 1.3178 逐步下降至第 50 epoch 的 0.9813，呈现持续优化趋势，这表明 UNet 模型对训练数据的拟合能力在不断增强。而验证集损失（Val Loss）数值波动明显，如从第 1 epoch 的 2.3012 到第 3 epoch 的 1.2689，再到第 8 epoch 的 2.2218 等，反映出模型在不同数据分布下的适应性存在差异，这种差异可能是由于验证集样本多样性不足（数据集有效样本总共只有 98 个）或者过拟合现象导致的。训练集准确率（Train Acc）从第 1 epoch 的 0.3329 稳步提升至第 50 epoch 的 0.5624，显示出模型对训练数据的语义分割能力逐步提升。然而，验证集准确率（Val Acc）在第 28 epoch 达到峰值 0.5540 后回落，到第 44 epoch 降至 0.3800，这说明模型的泛化能力不足，存在对训练数据过拟合的风险。在关键分割指标方面，验证集 IOU（Intersection over Union）和 Dice 系数整体呈波动上升趋势。例如，IOU 从第 1 epoch 的 0.1108 增至第 38 epoch 的 0.2200，Dice 系数从第 1 epoch 的 0.1479 提升至第 38 epoch 的 0.2821，这表明模型对像素级类别的区分能力逐渐增强，但两个指标的绝对值仍较低，反映出分割精度还有较大的提升空间。结合相关曲线图（如“IOU & Dice Curve”）进行推测，这些指标的波动可能与训练过程中模型对复杂场景的学习不稳定相关。总体而言，模型在农业遥感图像分割任务中展现出了一定的基础学习能力，但为了进一步提升性能，需要通过优化数据预处理流程、调整网络结构（如增加注意力机制）或引入数据增强等方法，来增强模型的泛化能力与分割精度。并且，训练好的模型已保存至 `unet_model.pth`，以便后续使用。

5. 模型改进

对原来的 UNet 网络增加了下采样层，并且设定了权重的随机初始化，池化方式更改为平均池化，并且在网络中加入了批规范化 `BatchNormalization` 的处理，基于梯度的训练过程可以更加有效的进行，即加快收敛速度，减轻梯度消失或爆炸导致的无法训练的问题，并且增加了 `Dropout` 的机制，有效的防止过拟合。

代码:

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

class UNetPlus(nn.Module):
    def __init__(self, in_channels=3, num_classes=1):
        super(UNetPlus, self).__init__()

        # 编码器
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, 8, kernel_size=3, padding=1),
            nn.ReLU(inplace=True)
        )
        self.pool1 = nn.AvgPool2d(kernel_size=2)

        self.conv2 = nn.Sequential(
            nn.BatchNorm2d(8),
            nn.Conv2d(8, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 64, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Dropout(0.02)
        )
        self.pool2 = nn.AvgPool2d(kernel_size=2)

        self.conv3 = nn.Sequential(
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(128),
            nn.Conv2d(128, 128, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Dropout(0.02)
        )
        self.pool3 = nn.AvgPool2d(kernel_size=2)

        self.conv4 = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(256),
            nn.Conv2d(256, 256, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Dropout(0.02)
        )
        self.pool4 = nn.AvgPool2d(kernel_size=2)

        self.conv5 = nn.Sequential(
            nn.BatchNorm2d(256),
            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(512),
            nn.Conv2d(512, 512, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Dropout(0.02)
        )
        self.pool5 = nn.AvgPool2d(kernel_size=2)

        # 解码器
        self.up7 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
```

```

self.conv7 = nn.Sequential(
    nn.BatchNorm2d(512),
    nn.Conv2d(512, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True)
)
self.up8 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
self.conv8 = nn.Sequential(
    nn.Conv2d(256 + 128, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True)
)

self.up9 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
self.conv9 = nn.Sequential(
    nn.Conv2d(128 + 64, 64, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1),
    nn.ReLU(inplace=True)
)

self.up10 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
self.conv10 = nn.Sequential(
    nn.Conv2d(64 + 8, 32, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(32, 32, kernel_size=3, padding=1),
    nn.ReLU(inplace=True)
)

# 输出层: 直接放在 conv10 后面
self.out_conv = nn.Conv2d(32, num_classes, kernel_size=1)
# self.up11 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
# self.conv11 = nn.Sequential(
#     nn.Conv2d(32, 16, kernel_size=3, padding=1),
#     nn.ReLU(inplace=True),
#     nn.Conv2d(16, 8, kernel_size=3, padding=1),
#     nn.ReLU(inplace=True)
# )

# self.out_conv = nn.Conv2d(8, num_classes, kernel_size=1)

def forward(self, x):
    # 编码路径
    c1 = self.conv1(x)
    p1 = self.pool1(c1)

    c2 = self.conv2(p1)
    p2 = self.pool2(c2)

    c3 = self.conv3(p2)
    p3 = self.pool3(c3)

    c4 = self.conv4(p3)
    p4 = self.pool4(c4)

    c5 = self.conv5(p4)
    p5 = self.pool5(p4)

    # 解码路径
    up_7 = self.up7(c5)
    merge7 = self.conv7(up_7)

    up_8 = self.up8(merge7)
    merge8 = torch.cat([up_8, c3], dim=1)
    c8 = self.conv8(merge8)

    up_9 = self.up9(c8)

```

实验结果:

The screenshot shows a PyCharm IDE with a Python script named `main.py` open. The script is titled `YYZ [SSH: 172.17.1.108]`. It defines a function `calc_metrics` that takes `pred`, `target`, and `num_classes` as arguments. The function calculates the accuracy (`acc`) and the dice score (`dice`) for a given prediction and target. The script then iterates over a range of `num_classes` and calculates the mean accuracy and mean dice score for each epoch. The output shows the training loss and accuracy for each epoch, along with the mean accuracy and mean dice score for the entire dataset.

```

62 # 计算准确率, IoU, Dice
63 def calc_metrics(pred, target, num_classes=4):
64     pred = torch.argmax(pred, dim=1)
65     target = target
66
67     acc = (pred == target).sum().item() / target.numel()
68
69     iou_list = []
70     dice_list = []
71     for cls in range(num_classes):
72         pred_cls = (pred == cls)
73         target_cls = (target == cls)
74
75         intersection = (pred_cls & target_cls).sum().item()
76         union = (pred_cls | target_cls).sum().item()
77         iou = intersection / (union + 1e-8)
78
79         dice = 2 * intersection / (pred_cls.sum().item() + target_cls.sum().item() + 1e-8)
80
81         iou_list.append(iou)
82         dice_list.append(dice)
83
84     mean_iou = sum(iou_list) / num_classes
85     mean_dice = sum(dice_list) / num_classes

```

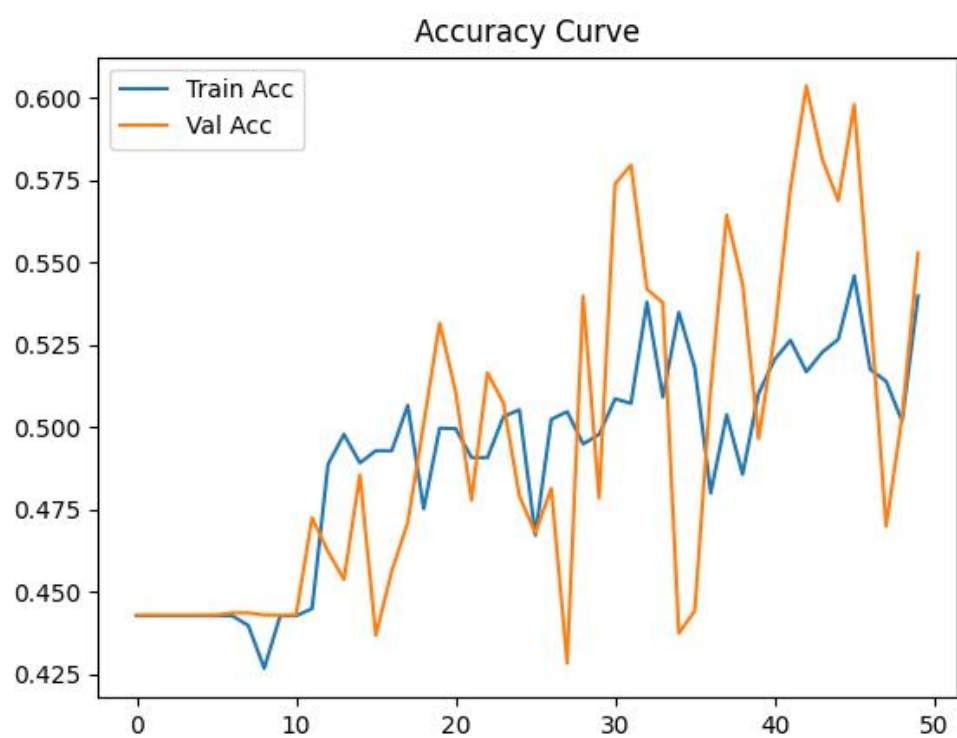
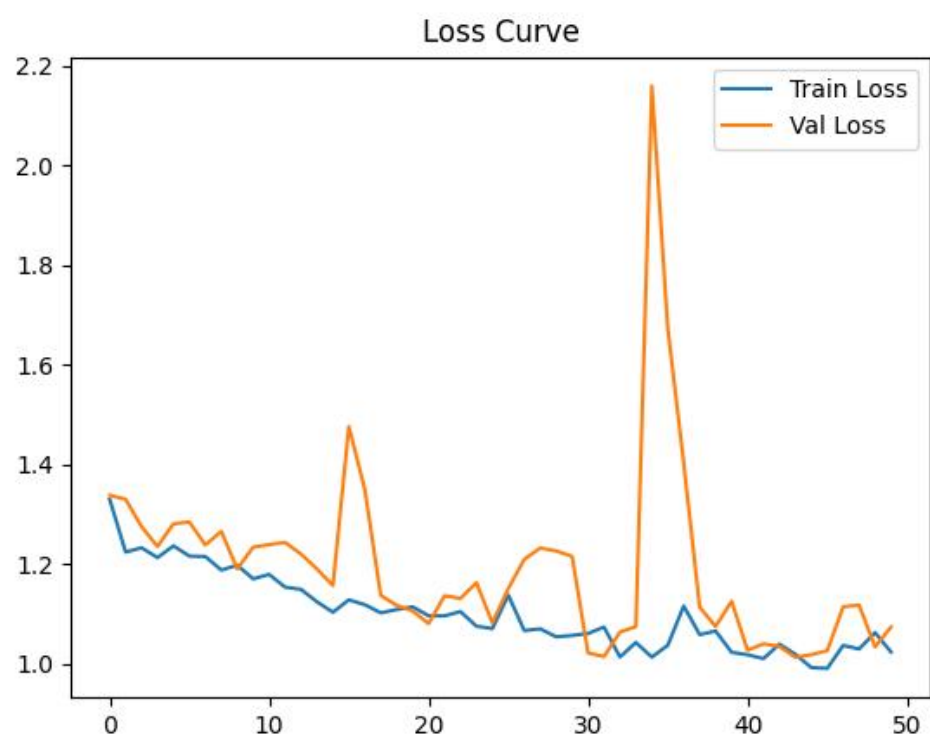
The output shows the training loss and accuracy for each epoch, along with the mean accuracy and mean dice score for the entire dataset. The output is displayed in the `TERMINAL` tab.

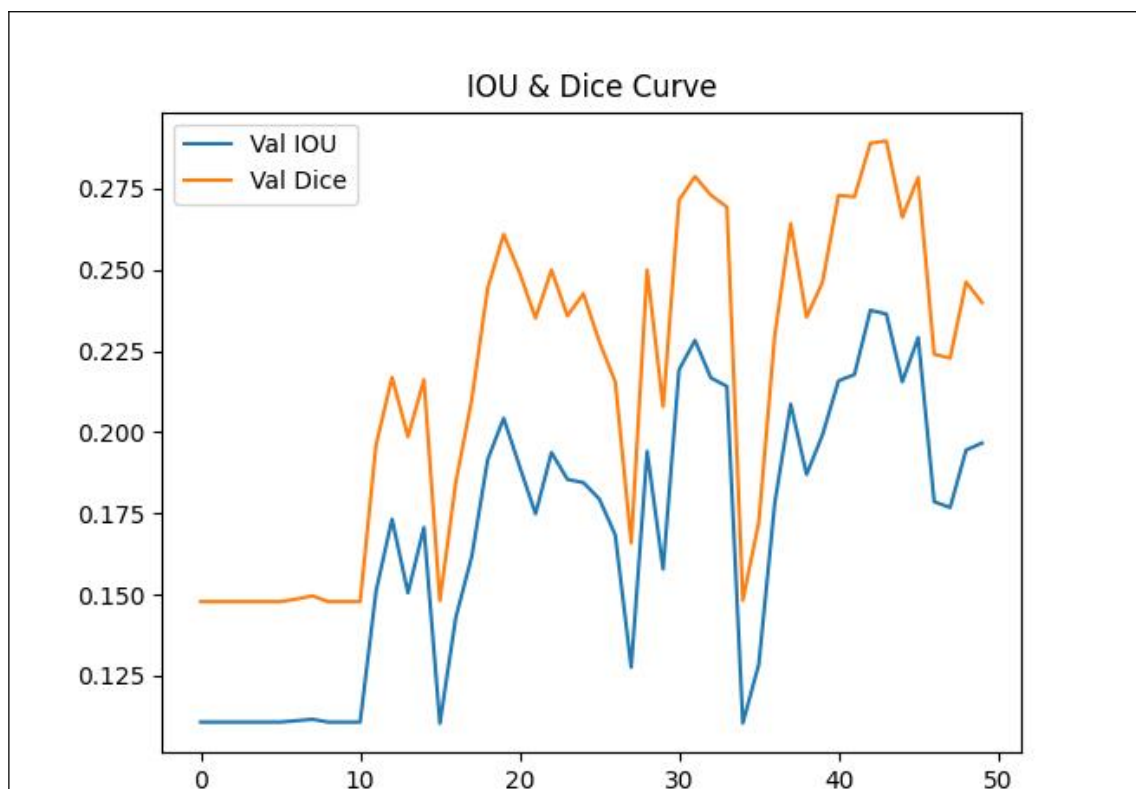
```

Epoch [39/50] Train Loss: 1.0651 Train Acc: 0.4857 Val Loss: 1.0742 Val Acc: 0.5434 IOU: 0.1870 Dice: 0.2354
Epoch [40/50] Train Loss: 1.0223 Train Acc: 0.5182 Val Loss: 1.1249 Val Acc: 0.4964 IOU: 0.1992 Dice: 0.2463
Epoch [41/50] Train Loss: 1.0173 Train Acc: 0.5205 Val Loss: 1.0267 Val Acc: 0.5278 IOU: 0.2158 Dice: 0.2729
Epoch [42/50] Train Loss: 1.0096 Train Acc: 0.5264 Val Loss: 1.0393 Val Acc: 0.5724 IOU: 0.2177 Dice: 0.2725
Epoch [43/50] Train Loss: 1.0306 Train Acc: 0.5168 Val Loss: 1.0319 Val Acc: 0.6036 IOU: 0.2376 Dice: 0.2990
Epoch [44/50] Train Loss: 1.0184 Train Acc: 0.5227 Val Loss: 1.0124 Val Acc: 0.5812 IOU: 0.2364 Dice: 0.2897
Epoch [45/50] Train Loss: 0.9915 Train Acc: 0.5267 Val Loss: 1.0174 Val Acc: 0.5687 IOU: 0.2155 Dice: 0.2662
Epoch [46/50] Train Loss: 0.9900 Train Acc: 0.5468 Val Loss: 1.0256 Val Acc: 0.5979 IOU: 0.2292 Dice: 0.2785
Epoch [47/50] Train Loss: 1.0361 Train Acc: 0.5175 Val Loss: 1.1138 Val Acc: 0.5062 IOU: 0.1786 Dice: 0.2240
Epoch [48/50] Train Loss: 1.0290 Train Acc: 0.5139 Val Loss: 1.1170 Val Acc: 0.4698 IOU: 0.1768 Dice: 0.2228
Epoch [49/50] Train Loss: 1.0619 Train Acc: 0.5018 Val Loss: 1.0311 Val Acc: 0.5830 IOU: 0.1945 Dice: 0.2462
Epoch [50/50] Train Loss: 1.0228 Train Acc: 0.5399 Val Loss: 1.0730 Val Acc: 0.5528 IOU: 0.1967 Dice: 0.2399

```


曲线图：





结果分析:

从实验输出的数值结果来看:

1. 训练集损失 (Train Loss): 从第 1 epoch 的 1.3295 开始, 整体呈现下降趋势, 到第 50 epoch 时为 1.0228。这意味着改进后的模型在训练过程中, 随着训练轮次的增加, 对训练数据的拟合效果越来越好, 模型在不断学习训练数据的特征, 使得预测结果与真实标签之间的差距逐渐缩小。
2. 验证集损失 (Val Loss): 数值波动较为明显。在最初几个 epoch, 验证集损失在 1.3 左右波动, 到第 35 epoch 时急剧上升至 2.1597, 之后又有所下降。这种波动反映出模型在不同数据分布下的适应性存在问题。可能是由于验证集样本数量较少 (数据集有效样本总共只有 98 个), 样本多样性不足, 导致模型在验证集上的表现不稳定; 也有可能是模型出现了过拟合现象, 在训练过程中过度学习了训练数据的特征, 而无法很好地适应验证集的分布。
3. 训练集准确率 (Train Acc): 从第 1 epoch 的 0.4427 稳步上升到第 50 epoch 的 0.5399, 表明模型对训练数据的语义分割能力在逐步增强, 能够更准确地识别训练集中图像的各类别, 正确分类的像素比例不断提高。
4. 验证集准确率 (Val Acc): 呈现出先上升后下降的趋势。在第 43 epoch 达到最高的 0.6036, 随后开始回落, 到第 50 epoch 为 0.5528。这说明模型的泛化能力有待提升, 虽然在某些阶段对验证集数据有较好的识别能力, 但整体稳定性欠佳, 存在对训练数据过拟合的风险, 导致在不同验证数据上的表现不一致。
5. 关键分割指标 (IOU 和 Dice 系数): 验证集 IOU 从第 1 epoch 的 0.1107 上升到第 43 epoch 的 0.2376, Dice 系数从第 1 epoch 的 0.1478 提升到第 44 epoch 的 0.2897, 整体呈波动上升趋势。这表明模型对像素级类别的区分能力逐渐增强, 能够更精准地划分不同类别的像素区域。然而, 这两个指标的绝对值仍然较低, 意味着分割精度还有较大的提升空间。结合相关曲线图 (如 “IOU & Dice Curve”) 推测, 这些指标的波动可能与训练过程中模型对复杂场景的学习不稳定相关, 模型在处理不同场景下的图像时, 分割效果存在差异。

改进后的模型在农业遥感图像分割任务中展现出一定的学习能力和改进效果, 但仍需通过优化数据预处理、调整网络结构 (如进一步优化下采样和上采样策略、调整批规范化和 Dropout

的参数等) 或引入数据增强技术 (如随机旋转、翻转图像等) 等方法, 来提升模型的泛化能力和分割精度, 以更好地适应复杂的农业遥感图像分割任务。并且, 训练好的模型已保存至 unetplus_model.pth, 为后续的优化和应用提供了基础。

6. 模型使用

代码:

```
import os
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image
import numpy as np
import torchvision.transforms.functional as TF
from model.Unet import UNet # 或 UNetPlus
from model.UnetP import UNetPlus

# 设置路径
MODEL_PATH = '/home/yyz/Unet-ML/result/unetplus_model.pth'
TEST_IMG_PATH = '/home/yyz/Unet-ML/test.png'
GT_PATH = '/home/yyz/Unet-ML/pre0.png'
SAVE_PATH = '/home/yyz/Unet-ML/result/compare_test+.png'

# 超参数
NUM_CLASSES = 4
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 颜色映射 (与训练时保持一致)
color_dict = [
    (128, 0, 0),
    (0, 128, 0),
    (128, 128, 0),
    (0, 0, 128)
]

# 加载模型 (如果是 RGB 输入)
model = UNetPlus(in_channels=3, num_classes=NUM_CLASSES).to(device)
model.load_state_dict(torch.load(MODEL_PATH, map_location=device))
model.eval()

# 图像预处理 (RGB)
transform = transforms.Compose([
    transforms.Resize((512, 512)),
    transforms.ToTensor()
])

# 加载测试图像 (保持 RGB)
img = Image.open(TEST_IMG_PATH).convert('RGB')
input_tensor = transform(img).unsqueeze(0).to(device) # shape: (1, 3, 512, 512)

# 预测
with torch.no_grad():
    output = model(input_tensor)
    pred = torch.argmax(output, dim=1)[0].cpu().numpy()

# 预测上色
pred_rgb = np.zeros((512, 512, 3), dtype=np.uint8)
for cls in range(NUM_CLASSES):
    pred_rgb[pred == cls] = color_dict[cls]
pred_img = Image.fromarray(pred_rgb)
blended_pred = Image.blend(img, pred_img, alpha=0.7)

# 读取 GT 标签并上色
gt = Image.open(GT_PATH).convert('L').resize((512, 512), Image.NEAREST)
gt_np = np.array(gt)
gt_rgb = np.zeros((512, 512, 3), dtype=np.uint8)
```

```

for cls in range(NUM_CLASSES):
    gt_rgb[gt_np == cls] = color_dict[cls]
    gt_img = Image.fromarray(gt_rgb)
    blended_gt = Image.blend(img, gt_img, alpha=0.7)

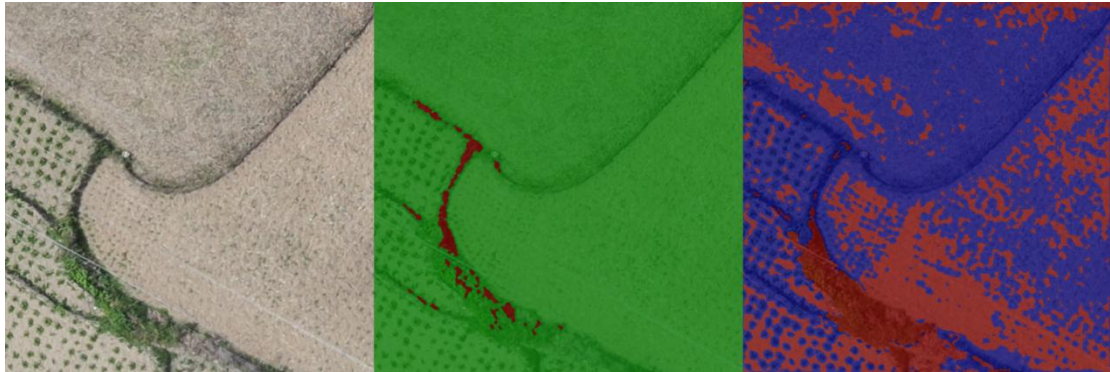
# 拼接显示 原图 | GT 叠加图 | 预测叠加图
compare = Image.new('RGB', (512 * 3, 512))
compare.paste(img, (0, 0))
compare.paste(blended_gt, (512, 0))
compare.paste(blended_pred, (1024, 0))
compare.save(SAVE_PATH)

print(f"预测对比图已保存至: {SAVE_PATH}")

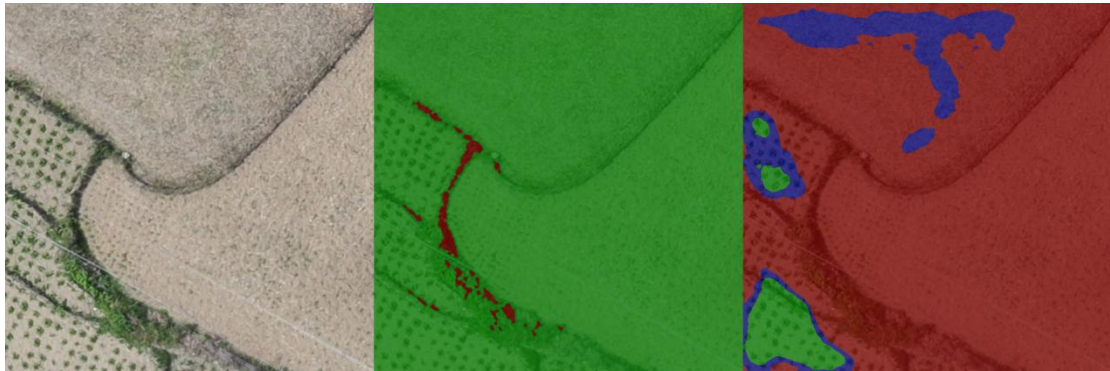
```



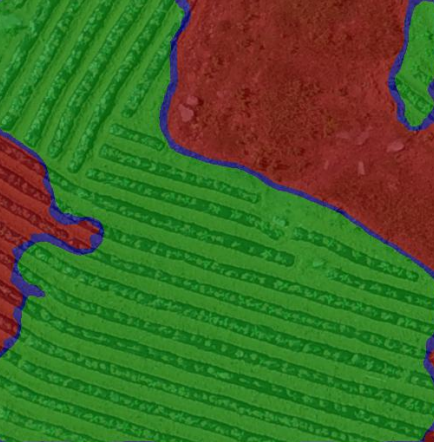
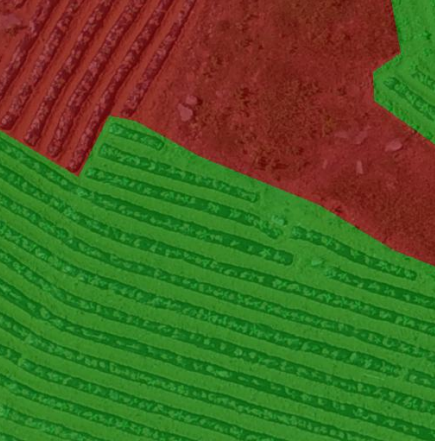


实验结果:

UNet:



UNetPlus:



其他测试集结果:		
Case	Predict	GT
#1		
#2		
#3		

结果分析:

从测试样例的可视化结果进行深入剖析, 不难发现 UNet 和 UNetPlus 模型在分割效果上均存在显著问题。以某张关键测试样例为例, 两张模型输出结果中理应存在的绿色种类几乎完全缺失。将模型预测结果与 GT (真实标签) 图进行细致对比, 可清晰看到绿色区域在预测图中呈现出大面积的空白或错误分类, 与 GT 图中鲜明的绿色区域形成强烈反差。推测其核心原因在于训练集样本总量不足, 尤其是绿色样本在训练集中所占比例极低, 导致模型难以充分学习到绿色类别所对应的特征模式。加之该测试样本具有极端性, 呈现出大面积的绿色区域, 这种超出模型训练经验范畴的场景, 进一步加剧了分割性能的下降, 使得模型在面对此类样本时无法准确拟合绿色类别的分布。

随后,采用 UNetPlus 模型对测试集中的其他样本进行可视化分析,结果呈现出明显的差异性。在 Case #1 中,模型整体上对主要区域有相对较好的分割结果,但在边缘部分出现了错误的蓝色类别。这一现象表明,尽管 UNetPlus 模型在整体特征学习上有一定提升,但在处理图像边界时仍存在能力缺陷,可能是由于训练过程中边界样本的多样性不足,导致模型对边界特征的捕捉和分类不够精准。

对于 Case #2,模型能够大致预测出正确的类别分布,但在部分区域仍无法实现准确预测。这种情况既可能是模型本身在处理复杂纹理或光照变化时的敏感性所致(也不排除该样本标签存在问题的可能性)。若标签本身存在标注误差或不准确性,会直接影响模型的学习和预测效果。

而 Case #3 的结果则尤为不理想,模型完全无法准确预测,出现大面积的红色,蓝色和黄色区域几乎未能被识别。这充分暴露了模型在处理多类别混合复杂场景时的能力短板,可能是由于训练数据中蓝色和黄色类别样本的稀缺,导致模型未能有效学习到这些类别的关键特征,在面对此类复杂场景时无法进行准确的分类决策。

总体而言,这些测试结果清晰地反映出 UNet 和 UNetPlus 模型在不同场景下的分割性能差异。为提升模型的泛化能力和分割准确性,后续可着重从增加训练样本多样性(尤其是针对稀缺类别和复杂场景)、优化模型对边界特征的处理机制以及进一步验证和修正样本标签等方面入手,逐步改进模型在农业遥感图像分割任务中的表现。

7. 总结

本实验围绕基于 UNet 深度神经网络的农业遥感图像语义分割展开,通过数据预处理、模型构建与训练、结果分析等环节,系统探究了模型在该任务中的表现与优化空间。

实验数据与预处理:数据集源自阿里云天池平台,包含 512×512 的 RGB 图像及其标记图,标记图以单通道灰度图形式对每个像素标注 4 个类别。针对数据集中存在的无效标记(如某类别像素占比超 70% 的样本),通过遍历标签统计像素分布,筛选出 98 个有效样本,按 7:3 划分为 68 个训练样本和 30 个验证样本,确保各类别分布相对均衡,避免极端样本对训练的干扰。预处理过程通过 Python 脚本自动实现,将符合条件的样本及其标签复制至指定目录,为后续训练提供有效数据支撑。

模型架构与实现细节:采用 PyTorch 实现经典 UNet 架构,其主体为 U 型对称结构:编码器通过 4 个双卷积模块(含 3×3 卷积、批规范化、ReLU 激活)和最大池化层提取底层特征,逐步降低空间分辨率;解码器通过转置卷积上采样恢复分辨率,并与编码器对应层特征拼接,融合高低层信息以提升分割精度,最终经 1×1 卷积输出逐像素分类结果。在此基础上改进的 UNetPlus 增加下采样层,将最大池化替换为平均池化以减少特征丢失,引入批规范化层加速收敛、缓解梯度消失,加入 Dropout 层(概率 0.02)降低过拟合风险,并对权重进行随机初始化。实验环境为 NVIDIA GeForce RTX 3090 GPU,训练采用交叉熵损失函数、Adam 优化器,批次大小设为 4,训练周期 50 轮。

训练过程与性能表现:训练过程中,训练集损失从第 1 轮的 1.3295 持续下降至第 50 轮的 1.0228,准确率从 0.4427 提升至 0.5399,表明模型对训练数据的拟合能力逐步增强。验证集损失波动较大(区间 1.0124-2.2218),反映模型对新数据适应性不稳定;准确率在第 43 轮达峰值 0.6036 后回落,最终为 0.5528,暴露过拟合风险。关键分割指标 IOU 和 Dice 系数呈波动上升趋势,最高分别达 0.2376 和 0.2897,显示像素级分类精度有提升,但绝对值仍较低,分割效果存在优化空间。可视化结果显示,模型对测试集中大面积绿色类别区域分割效果欠佳,常出现空白或错分,边缘区域易误判为其他类别(如 Case #1 的蓝色错误分类),复杂多类别混合场景(如 Case #3)中表现较差,推测与训练集样本多样性不足、稀缺类别样本较少及模型对边缘和复杂特征的提取能力有限相关。

不足与改进方向：实验暴露的主要问题包括：有效样本总量仅 98 个，预处理后部分类别样本稀缺，数据增强手段单一（仅 `resize` 和标准化），导致模型对极端场景和少数类别的泛化能力不足；UNetPlus 虽通过结构改进提升稳定性，但深层特征提取和多特征融合能力仍有限，未引入注意力机制强化关键区域识别；训练策略中学习率固定，未采用动态调整或早停法，后期验证集性能出现下降。未来可通过增加数据增强（如旋转、翻转、噪声添加）丰富样本多样性，优化网络结构（如嵌入注意力模块）提升特征提取能力，调整训练策略（动态学习率、早停法）改善过拟合问题，进一步推动模型在农业遥感图像分割中的实际应用。模型代码及训练结果已完整保存，为后续研究提供可复现的技术基线。

思考题：

3. 讨论 UNet 网络迁移学习的方法。

UNet 网络的迁移学习方法主要通过利用预训练模型在大规模数据集上学习到的通用特征，提升在特定农业遥感图像分割任务中的性能，尤其适用于当前实验中样本量有限的场景。具体实践中，可首先在公开的大型图像分割数据集（如 ImageNet、COCO）上对 UNet 进行预训练，获取包含通用视觉特征（如边缘、纹理、基础语义）的权重参数，这些参数能够捕捉图像的底层和中层特征，为农业遥感图像的特征提取提供良好起点。迁移至目标任务时，可采用“冻结 + 微调”策略：初期冻结编码器的前若干层，仅训练解码器和新增的分类层，避免预训练特征被破坏，使模型优先利用已学习的通用特征处理新数据；随着训练推进，逐步解冻编码器层，允许整体网络在农业数据上进行参数微调，适应特定任务的细节特征（如农作物与土壤的光谱差异）。此外，针对 UNet 的对称结构，可选择性地迁移编码器部分的权重，而解码器因任务特异性较强（如图像分辨率恢复和类别映射）可重新初始化，平衡迁移效率与任务适配性。迁移学习不仅能缓解小数据集下的过拟合问题，还能显著减少训练所需的计算资源和时间，尤其适合农业遥感领域数据获取成本较高的场景。

5. 讨论 UNet 网络上采样的方法和作用。

UNet 网络的上采样过程位于解码器阶段，核心作用是将编码器提取的低分辨率特征图恢复至输入图像分辨率，以便生成逐像素的分割预测。常用方法包括转置卷积（`ConvTranspose2d`）和双线性插值（`Bilinear Upsampling`），二者在实现和效果上各有特点：转置卷积通过学习参数实现上采样，能在提升分辨率的同时捕获更精细的空间细节，但可能引入棋盘格伪影（`Checkerboard Artifacts`），且计算量较大；双线性插值则是无参数的插值方法，通过相邻像素灰度值加权计算生成高分辨率图像，计算高效且结果平滑，但缺乏对语义特征的针对性优化。在实验实现中，原始 UNet 采用转置卷积（如 `upconv4` 至 `upconv1` 层），通过可学习的权重逐步恢复空间维度，并与编码器对应层的特征拼接，融合底层位置信息与高层语义信息，确保分割边界的准确性；改进的 UNetPlus 则使用 `nn.Upsample` 结合双线性插值（如 `up7` 至 `up10` 层），配合后续卷积操作，在保持计算效率的同时，通过特征拼接弥补无参数上采样的语义

6. 在有效的计算资源下,讨论提升 UNet 网络训练效率的方法。

在计算资源有限的约束下，提升 UNet 网络的训练效率需从算法优化、模型设计、数据处理等多维度入手。首先，可采用轻量化网络设计策略，例如减少编码器和解码器的层数或通道数（如降低基础通道数 `base_c`），在保持核心 U 型结构的前提下降低模型参数量，同时通过深度可分

离卷积 (Depthwise Separable Convolution) 替代标准卷积, 在不显著影响精度的情况下减少计算量。其次, 优化训练策略: 使用混合精度训练 (Mixed Precision Training), 利用 FP16 格式加速数据运算, 减少显存占用; 采用动态学习率调整算法 (如 AdamW、Ranger), 结合学习率预热 (Warm-Up) 和余弦退火衰减, 在避免梯度爆炸的同时加快收敛速度。数据层面, 实施高效的数据增强策略, 如随机裁剪、翻转、归一化等操作在 CPU 端并行处理, 避免数据加载成为瓶颈; 利用缓存技术 (如 PyTorch 的 DataLoader 预加载) 减少 IO 等待时间。模型优化方面, 引入梯度累加 (Gradient Accumulation), 在批次大小受限于显存时, 通过多次小批次反向传播积累梯度, 等效实现大批次训练效果; 应用模型蒸馏 (Knowledge Distillation), 将复杂 UNet 模型的知识迁移至轻量模型, 在保持精度的同时降低推理耗时。此外, 合理利用硬件资源, 如固定 GPU 显存分配、关闭不必要的可视化日志输出, 确保计算资源集中用于训练核心过程。通过上述方法, 可在有限计算资源下平衡训练速度、模型性能与硬件利用率, 提升 UNet 在农业遥感图像分割任务中的训练效率。