哈爾濱工業大學

深度学习实验六报告

使用 U-Net 网络的语义分割

小组成员：李昌昊 刘天瑞 黄托朴森

1、选题说明及任务描述

1.1、选题说明

我们选择了语义分割这个题目。语义分割指的是一种计算机视觉任务，旨在将图像中的每个像素分配给特定的语义类别。

1.2、任务描述

语义分割的目标是将图像中的不同物体或区域分割开来，并为每个像素分配语义标签，例如人、车、道路、树等。

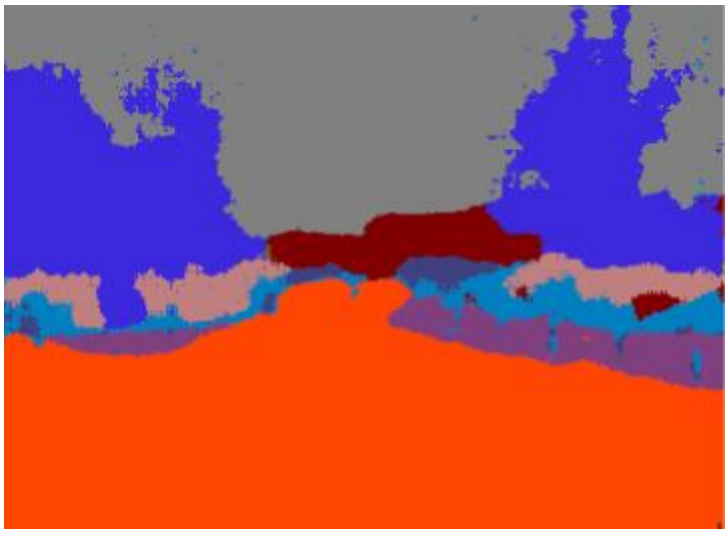

图 1 道路识别

传统的图像分割方法主要基于低级特征，例如边缘检测和纹理分析。然而，深度学习的出现为语义分割带来了显著的改进。深度学习模型可以从大量标注的图像数据中学习特征表示，以自动学习到更高级别的语义信息。常见的语义分割模型包括 U-Net、FCN、DeepLab 等。我们选择较为常用的 U-Net 来尝试实现语义分割任务。

2、数据集描述

U-Net 常用的数据集为 DRIVE，DRIVE（Digital Retinal Images for Vessel Extraction）数据集是由比利时鲁汶大学的计算机视觉中心创建的。该数据集专门用于评估眼底图像分割算法的性能，主要应用于视网膜血管分割任务。

DRIVE 数据集包含了来自 40 位不同患者的 40 幅彩色眼底图像，这些图像以分辨率为 565x584 像素。每幅图像都配有相应的手动标注，其中包括视网膜血管的像素级别标签。因为本身数据集需要和标注应该一一对应，导致其每个样例都要严格匹配其标注（数据增强的话需要同时以一样的操作对标注进行处理），所以并不需要对数据集进行数据增强处理。

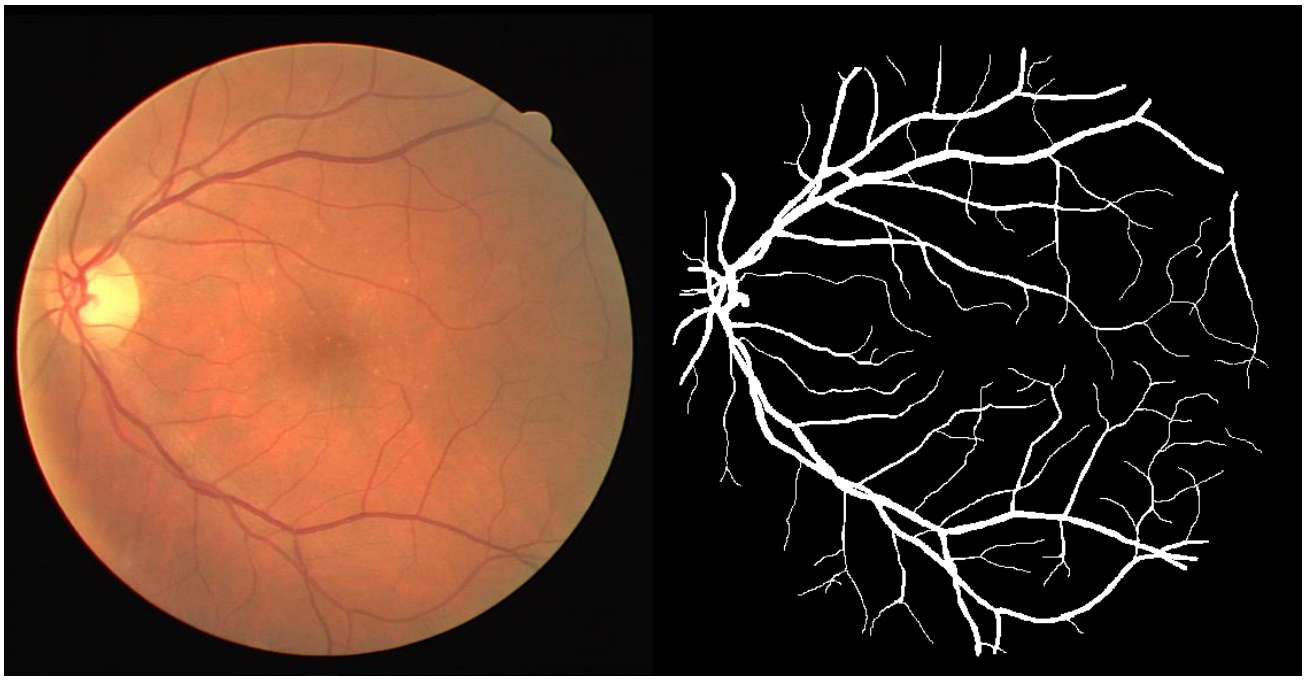

图 2 原图和标注

3 方案设计

3.1 损失函数

我们选择使用二元交叉熵 **BCEloss()**作为损失函数，根据模型输出结果和样本进行比较来更新模型。**BCELoss()** 通常将模型的输出视为概率值，并将目标标签视为二进制值。损失函数将计算每个样本的二元交叉熵，并返回平均损失。

3.2 网络结构

```python
class Conv(nn.Module):
    def __init__(self, C_in, C_out):
        super(Conv, self).__init__()
        self.layer = nn.Sequential(

            nn.Conv2d(C_in, C_out, 3, 1, 1),
            nn.BatchNorm2d(C_out),
            # 防止过拟合
            nn.Dropout(0.3),
            nn.LeakyReLU(),

            nn.Conv2d(C_out, C_out, 3, 1, 1),
            nn.BatchNorm2d(C_out),
            # 防止过拟合
            nn.Dropout(0.4),
            nn.LeakyReLU(),
        )

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

```python
# 下采样模块
class DownSampling(nn.Module):
    def __init__(self, C):
        super(DownSampling, self).__init__()
        self.Down = nn.Sequential(
            # 使用卷积进行2倍的下采样，通道数不变
            nn.Conv2d(C, C, 3, 2, 1),
            nn.LeakyReLU()
        )

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


# 上采样模块
class UpSampling(nn.Module):

    def __init__(self, C):
        super(UpSampling, self).__init__()
        # 特征图大小扩大2倍，通道数减半
        self.Up = nn.Conv2d(C, C // 2, 1, 1)

    def forward(self, x, r):
        # 使用邻近插值进行下采样
        up = F.interpolate(x, scale_factor=2, mode="nearest")
        x = self.Up(up)
        # 拼接，当前上采样的，和之前下采样过程中的
        return torch.cat((x, r), 1)
```

```python
class UNet(nn.Module):

    def __init__(self):
        super(UNet, self).__init__()

        # 4次下采样
        self.C1 = Conv(3, 64)
        self.D1 = DownSampling(64)
        self.C2 = Conv(64, 128)
        self.D2 = DownSampling(128)
        self.C3 = Conv(128, 256)
        self.D3 = DownSampling(256)
        self.C4 = Conv(256, 512)
        self.D4 = DownSampling(512)
        self.C5 = Conv(512, 1024)

        # 4次上采样
        self.U1 = UpSampling(1024)
        self.C6 = Conv(1024, 512)
        self.U2 = UpSampling(512)
        self.C7 = Conv(512, 256)
        self.U3 = UpSampling(256)
        self.C8 = Conv(256, 128)
        self.U4 = UpSampling(128)
        self.C9 = Conv(128, 64)

        self.Th = torch.nn.Sigmoid()
        self.pred = torch.nn.Conv2d(64, 3, 3, 1, 1)

    def forward(self, x):
        # 下采样部分
        R1 = self.C1(x)
        R2 = self.C2(self.D1(R1))
        R3 = self.C3(self.D2(R2))
        R4 = self.C4(self.D3(R3))
        Y1 = self.C5(self.D4(R4))

        # 上采样部分
        # 上采样的时候需要拼接起来
        O1 = self.C6(self.U1(Y1, R4))
        O2 = self.C7(self.U2(O1, R3))
        O3 = self.C8(self.U3(O2, R2))
        O4 = self.C9(self.U4(O3, R1))

        # 输出预测，这里大小跟输入是一致的
        # 可以把下采样时的中间抠出来再进行拼接，这样修改后输出就会更小
        return self.Th(self.pred(O4))
```

图 3 网络结构

1. **Conv** 模块：这是 U-Net 中基本的卷积块。它由两个连续的卷积层、批归一化、Dropout 和 LeakyReLU 激活函数组成。
2. **DownSampling** 模块：用于下采样的模块。通过一个卷积层进行 2 倍下采样，并使用 LeakyReLU 激活函数。
3. **UpSampling** 模块：用于上采样的模块。通过一个卷积层进行通道数减半，然后使用邻近插值对特征图进行 2 倍上采样。上采样后的特征图与之前下采样过程中的特征图进行拼接。
4. **UNet** 主干网络：主干网络由 4 次下采样和 4 次上采样组成。下采样部分依次进行卷积块的处理和下采样操作，得到一系列的特征图。上采样部分依次进行上采样操作和卷积块的处理，并将下采样过程中的特征图与对应的上采样结果进行拼接。最后，使用 Sigmoid 激活函数和卷积层输出预测结果

4、实验过程

4.1 调整数据

将数据和标注直接读入，转换为 RGB 模式确保颜色处理正确，然后转换为模型需要的 256*256 的正方形便于输入模型，转换为正方形的具体操作为置中然后使用黑色像素填充图片短边。

4.2 模型训练

测试集和训练集都是 20 张图片，故设置 batch_size 为 4，优化器使用 Adam，学习率设置为默认的 0.001，设置迭代次数为 300 次，每次训练完都将训练数据的第一张输出来查看训练效果，并将 loss 添加进入 tensorboard 的 summarywriter（）中，每隔 50 轮都保存一下暂时的训练模型。

4.3 模型测试

因为图片较多，将两张图片设置为一个输入来计算损失。我们对模型进行了多次训练来对比训练次数对模型的影响，分别是 300 次、600 次、和 900 次迭代。输出图片左边为原图，中间为模型输出，右边为原图对应标注。

```
[Running] set PYTHONIOENCODING=utf8 && python -u "d:\MyCode\python\DL_LAB6\test.py"
Loaded ./model.plt!
Batch Loss: 0.11357641220092773
Batch Loss: 0.09793677181005478
Batch Loss: 0.1312544196844101
Batch Loss: 0.09897303581237793
Batch Loss: 0.1226498931646347
Batch Loss: 0.08877606689929962
Batch Loss: 0.12392586469650269
Batch Loss: 0.1030554473400116
Batch Loss: 0.13240265846252441
Batch Loss: 0.0965053141117096
```

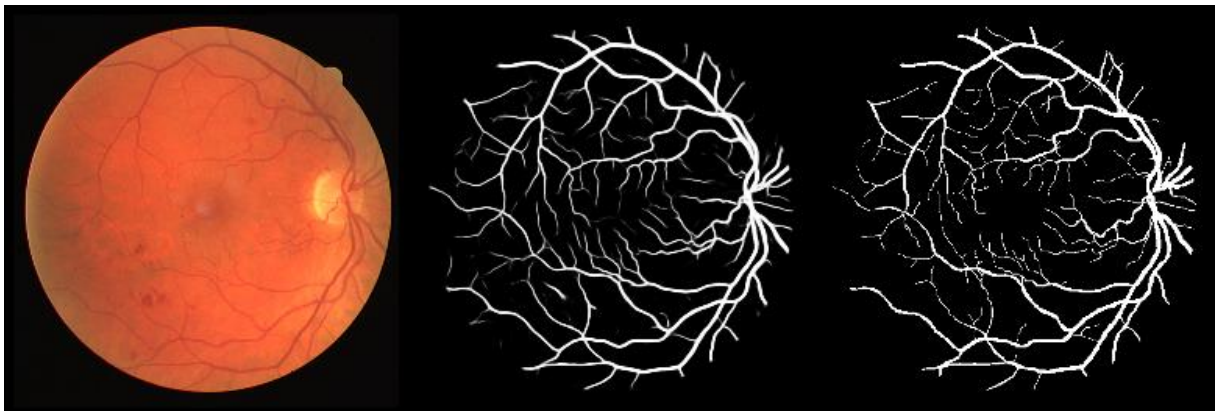

图 4 第一次训练结果

```
[Running] set PYTHONIOENCODING=utf8 && python -u "d:\MyCode\python\DL_LAB6\test.py"
Loaded ./model.plt!
Batch Loss: 0.15077440440654755
Batch Loss: 0.11725735664367676
Batch Loss: 0.13083547353744507
Batch Loss: 0.11602803319692612
Batch Loss: 0.1744539737701416
Batch Loss: 0.13961704075336456
Batch Loss: 0.19505926966667175
Batch Loss: 0.16453519463539124
Batch Loss: 0.18122737109661102
Batch Loss: 0.18593701720237732
```

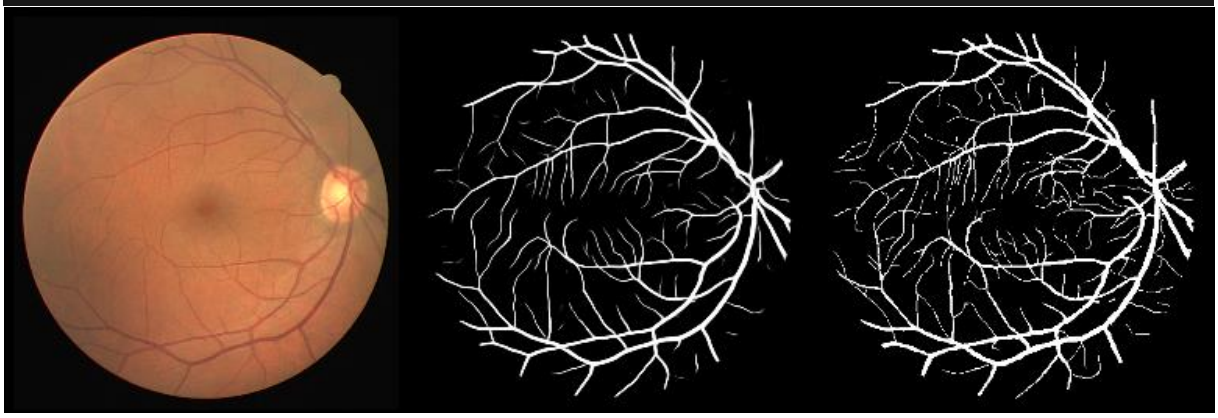

图 5 第二次训练结果

```
[Running] set PYTHONIOENCODING=utf8 && python -u "d:\MyCode\python\DL_LAB6\test.py"
Loaded ./model.plt!
Batch Loss: 0.23311710357666016
Batch Loss: 0.18540751934051514
Batch Loss: 0.2023126482963562
Batch Loss: 0.18372349441051483
Batch Loss: 0.16809263825416565
Batch Loss: 0.22035247087478638
Batch Loss: 0.18758289515972137
Batch Loss: 0.16387322545051575
Batch Loss: 0.1506401002407074
Batch Loss: 0.1797553300857544
```

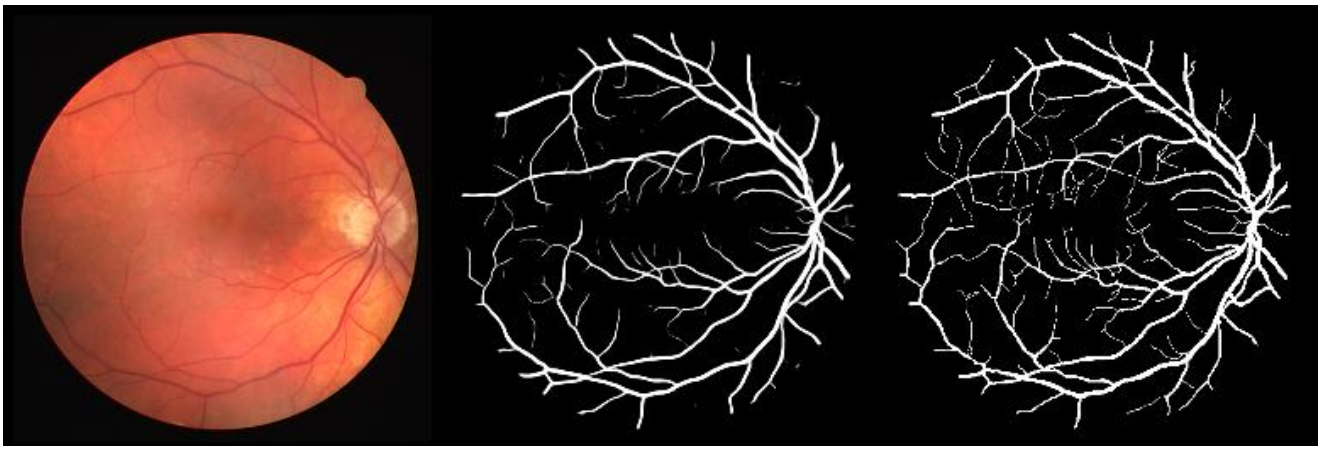

图 6 第三次训练结果

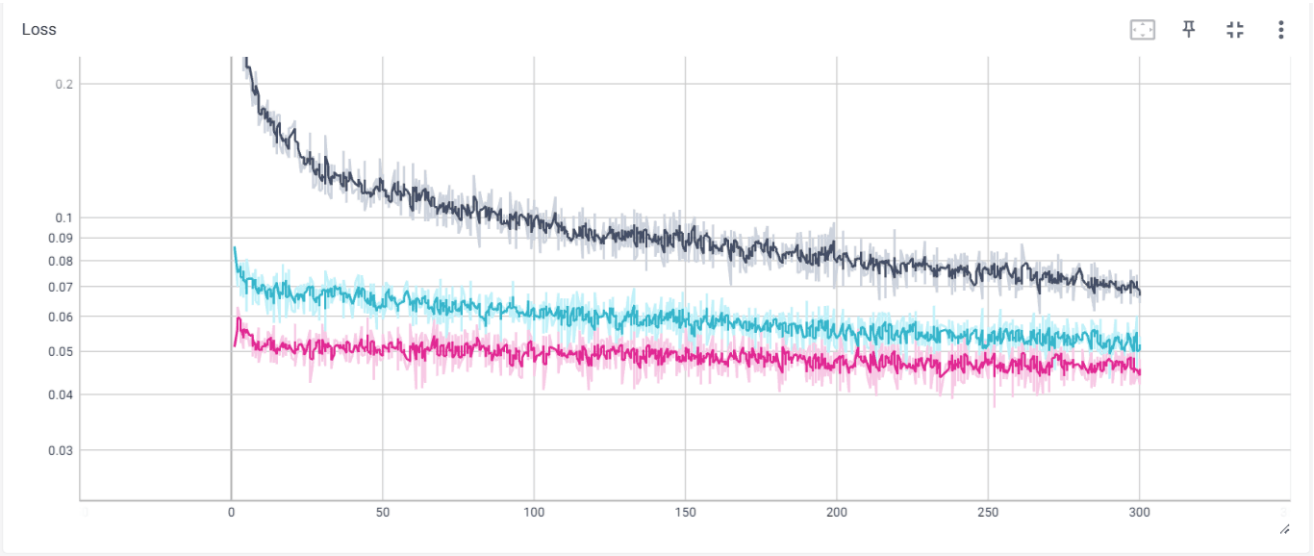

图 7 三次训练的训练集 loss（从上到下分别为第一次、第二次和第三次）

5、实验结果

本次实验测试了 300 次迭代、600 次迭代和 900 次迭代的 Unet 网络对 DRIVE 数据集的语义分割，300 次迭代的模型可以较好地提取出视网膜血管分布。越往后训练，虽然其在训练

集的 loss 在进一步降低，其在测试集上的表现却越来越糟，出现了明显的过拟合，其输出结果也表明其忽略了很多更加细小的血管。

6、方案评价

本次实验挑选的 Unet 网络在 DRIVE 数据集上表现较好，且网络的结构较为简单。在训练时其 loss 震荡较大，但整体训练结果良好。在多次训练后有比较大程度的过拟合表现，应该是损失函数使用较为简单导致，可以在这方面对模型进行进一步优化。

7、成员分工

李昌昊：优化模型搭建与训练迭代

刘天瑞：数据集处理，初步模型构建

黄托朴森：初步模型测试与最终结果评估、报告撰写