

基于深度学习的图像分割实验案例

一、 实验目的

通过本实验熟悉语义分割基本概念以及网络模型的一般训练、测试流程，掌握基于编解码方法的经典语义分割网络 U-Net 的基本网络架构和设计思想，能够完成 U-Net 网络编码器和解码器的构建，以及在所提供的细胞数据集上完成模型的训练和测试。

二、 实验内容

使用 Pytorch 完成 U-Net 的经典任务：医学图像分割（给定一张细胞结构图，把每个细胞互相分割开来），包括：从零开始搭建 U-Net 网络结构、实验数据处理、网络的训练与测试。

1. 实验环境与参考代码下载

```
1  ## Software Requirements
2      - Windows or Ubuntu16.04
3      - Python 3.7.4 or higher
4      - Pytorch 1.3.0 or higher
5      - CUDA 10.2 or higher
6      - cuDNN 7.6.02 or higher
7  ## Download from GitHub
8  ```shell
9      git clone https://github.com/1411102509/U-Net.git
10     cd U-Net
11  ```
```

所选用数据集非常简单（生物医学成像国际研讨会 ISBI 的开源数据集），容易上手，如图 1 所示，共 60 张电镜图，训练与测试数据分别有 30 张，训练数据由原始图像与标注的分割图像组成，分辨率为 512x512。参考代码也已经在上述 Github 中给出。

为方便后续测试使用，这里给出已经训练好的网络权重文件的下载链接：

https://drive.google.com/file/d/1vE1lAsZI3C1ihb1NAV_AIuDtGfsIN0f/view?usp=sharing

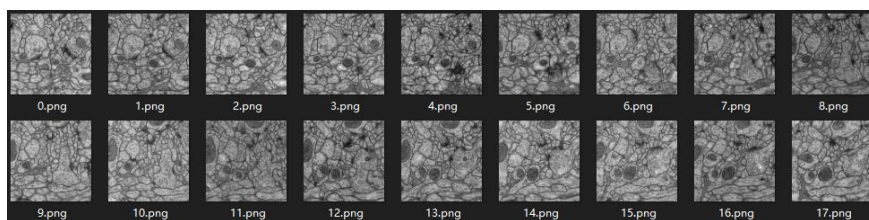


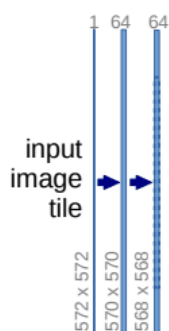
图 1. 数据集部分图像展示

2. 编程实现 U-Net 的网络结构

根据 U-Net 的网络结构特点，将网络的结构拆分为多个模块进行实现：

2.1 DoubleConv 模块：

从 U-Net 网络中可以看出，不管是下采样过程还是上采样过程，每一层都会连续进行两次卷积操作，这种操作在 U-Net 网络中重复很多次，可单独实现 DoubleConv 模块方便之后调用。



```
class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""
    def __init__(self, in_channels, out_channels):
        super().__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)
```

图 2. DoubleConv 模块结构及实现代码

其中：torch.nn.Sequential 是一个时序容器，Modules 会以它们传入的顺序被添加到容器中。比如上述代码的操作顺序：卷积→BN→ReLU→卷积→BN→ReLU。

DoubleConv 模块的 in_channels 和 out_channels 可以灵活设定，以便扩展使用，如上图所示的网络，in_channels 设为 1，out_channels 为 64。

输入图片大小为 572*572，经过步长为 1，padding 为 0 的 3*3 卷积，得到 570*570 的 feature map，再经过一次卷积得到 568*568 的 feature map。

计算公式： $O=(H-F+2\times P)/S+1$ ，H 为输入 feature map 的大小，O 为输出 feature map 的大小，F 为卷积核的大小，P 为 padding 的大小，S 为步长。

2.2 Down（下采样）模块：

U-Net 网络一共有 4 次下采样过程，这里的代码实现也很简单，只需一个 maxpool 池化层，进行下采样，然后接一个 DoubleConv 模块。



图 3. Down 模块结构及实现代码

至此，U-Net 网络的左半部分的下采样过程的代码已经完成，接下来是右半部分的上采样过程。

2.3 Up（上采样）模块：

上采样过程用到的最多的就是上采样，除了常规的上采样操作，还要注意有进行特征的融合。

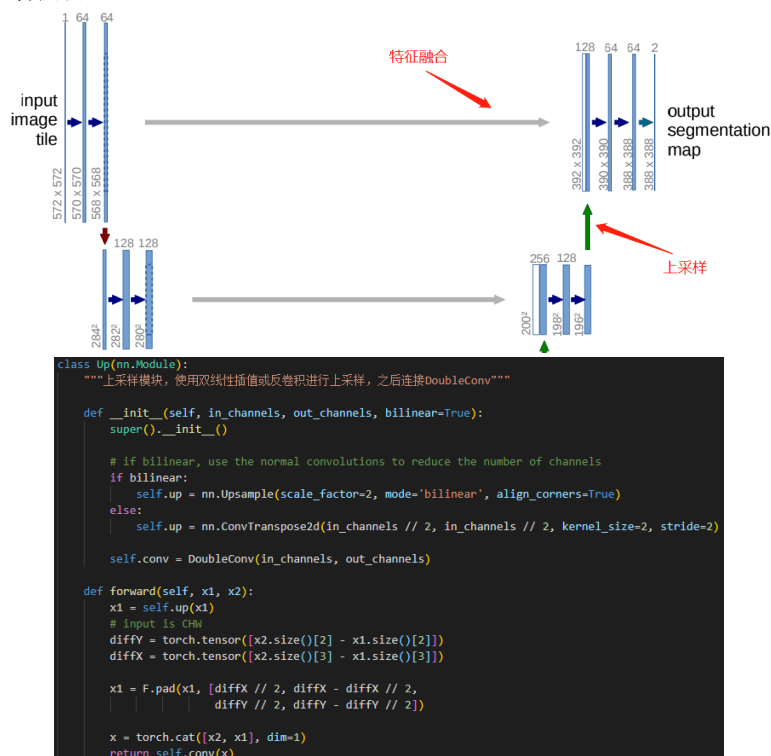
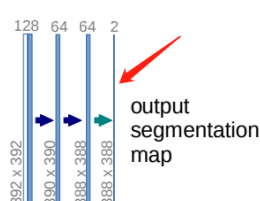


图 4. up 模块结构及实现代码

代码复杂一些，我们可以分开来看，首先是__init__初始化函数里定义的上采样方法以及卷积采用 DoubleConv。上采样，定义了两种方法：Upsample 和 ConvTranspose2d，也就是双线性插值和反卷积。在 forward 前向传播函数中，x1 接收的是上采样的数据，x2 接收的是特征融合的数据。特征融合方法就是先对小的 feature map 进行 padding，再进行 concat。

2.4 OutConv 模块：

用上述的 DoubleConv 模块、Down 模块、Up 模块就可以拼出 U-Net 的主体网络结构。U-Net 网络的输出需要根据分割数量，整合输出通道，操作很简单，就是 channel 的变换，下图展示的是分类为 2 的情况（通道为 2）。



```
class OutConv(nn.Module):
    '''卷积输出模块'''
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

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

图 5. OutConv 模块结构及实现代码

至此，U-Net 网络用到的模块都已写好，将上述的模块代码整合到一个 unet_parts.py 文件里，然后创建 unet_model.py，根据 U-Net 网络结构，设置每个模块的输入输出通道个数以及调用顺序，对各个模块进行整合，搭建完整的 U-Net 网络。（该部分参考代码在 U-Net/model/中已经给出）

运行代码，检查所编写的网络结构：

```
1  ```shell
2      python3 model/unet_model.py
3  ```
```

正常情况下，终端打印所构建的网络结构信息，检查无误后，网络搭建完成。

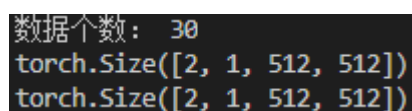
3. 加载训练数据

加载训练集中的所有数据，并进行数据增强。（该部分参考代码在 U-Net/utlis/dataset.py 中给出，并在参考代码中给出了代码的详细注释）。

运行代码，检查数据加载结果：

```
1  ```shell
2      python3 utlis/dataset.py
3  ```
```

正常情况下显示 30 组 torch.Size([2, 1, 512, 512]):



```
数据个数: 30
torch.Size([2, 1, 512, 512])
torch.Size([2, 1, 512, 512])
```

图 6. 数据检查的正确结果

4. 训练 U-Net 模型

选择合适的优化器(RMSProp 等)和损失函数(交叉熵等)，编写训练代码，指定硬件设备开始训练，并保存在测试集上 Loss 值最小的网络参数（由于实验基础，正常情况下需要根据验证集上的准确率选择保存的模型）。（该部分参考代码在 U-Net/train.py 中给出，并在参考代码中给出了代码的详细注释）

运行代码，开始训练：

```
1  ```shell
2      python3 train.py
3  ```
```

正常情况下训练完成会在根目录下生成 best_model.pth。

5. 测试训练好的 U-Net 模型

编写代码，加载模型参数(best_model.pth)，测试所有测试集图像。（该部分参考代码在 U-Net/predict.py 中给出，并在参考代码中给出了代码的详细注释）

运行代码，开始测试：

```
1  ````shell
2      python3 predict.py
3  ````
```

测试完成后，可以在 `data/test` 目录下，看到预测结果，如图 7 所示。

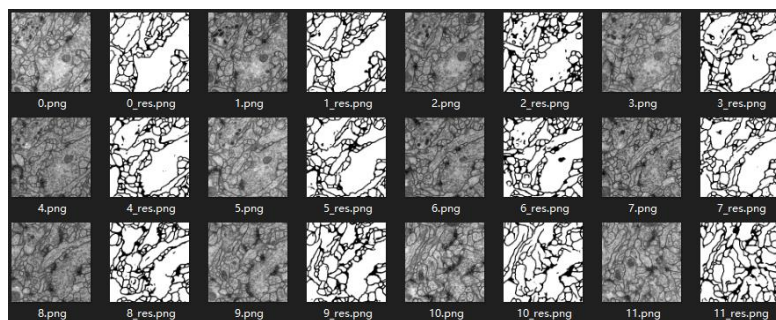


图 7. 经过网络预测后的分割结果

6. 拓展

上述给出的是最基础的例子，额外可自己编程实现 Loss 收敛曲线的绘制、计算 MIOU 等评价指标，还可尝试更换更大的数据集进行实验，如 `cityscapes` 等。

三、 知识要点

1. 深入理解语义基于编解码语义分割网络模型的基本结构和设计思想、以及 U-Net 网络设计的独特之处和优势。

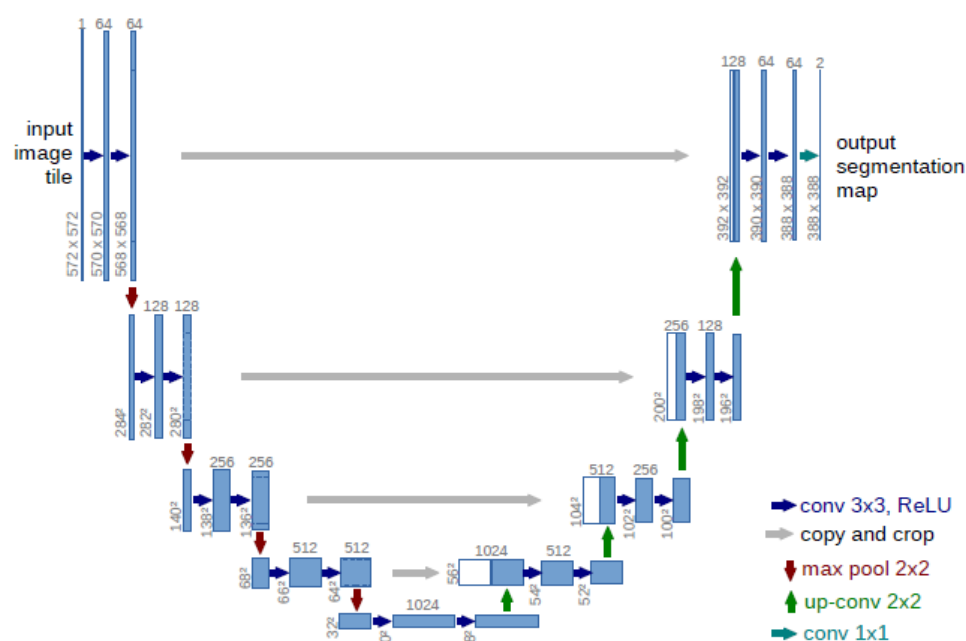
编码器-解码器结构解决了池化操作导致的图像分辨率降低、像素空间信息丢失问题。编码器通常由多个卷积层和池化层组成，作用是从原图中获取含有位置信息和语义信息的特征图。而解码器通常由反卷积层和反池化层构成，作用是恢复特征图中丢失的空间维度和位置信息，生成稠密的预测图。

U-Net 是原作者参加 ISBI Challenge 提出的一种分割网络，能够适应很小的训练集（大约 30 张图）。U-Net 与 FCN 都是很小的分割网络，没有使用空洞卷积，结构简单，主要应用于医学图像分析领域。

U-Net 的独特之处在于：将编码器中低分辨率特征图通过跳跃连接直接拼接到对应解码器上采样生成的特征图，从而有效融合了低层的细节信息和高

层的像素分类信息，实现更精确的分割。

2. 掌握 U-Net 模型的基本架构、每个基本模块的 Pytorch 编码实现，以及整体网络训练、测试过程。



整个 U-Net 网络结构如上图，类似于一个大大的 U 字母：首先进行 Conv+Pooling 下采样（编码）；然后 Deconv 反卷积进行上采样（解码），crop 之前的低层 feature map，进行融合；然后再次上采样。重复这个过程，直到获得输出 388x388x2 的 feature map，最后经过 softmax 获得分割图。总体来说与 FCN 思路非常类似。但与 FCN 逐点相加不同，U-Net 采用将特征在 channel 维度拼接在一起，形成更“厚”的特征。

四、 组织方式

教学组织方式，即为了对在课堂上如何就这一特定实验进行组织引导提出建议。

（一）、问题清单及提问顺序、资料发放顺序；

1. 基于编解码的语义分割网络有什么特定，如何设计？
2. 如何利用 Pytorch 构建 U-Net 网络结构，并进行训练测试？

(二)、课时分配（时间安排）；

- 1.课后自行阅读相关资料；
- 2.课堂小组讨论并提交分析提纲：2 课时；
- 3.课堂小组代表发言、进一步讨论：1 课时；
- 4.课堂讨论总结：1 课时。

(三)、讨论方式（情景模拟、小组式、辩论式等）；

建议小组式讨论，在讨论的过程中可以加深对于语义分割网络 U-Net 的理解与认识，小组式讨论可以增加学生的发言机会，使每个学生都可以参与进来，可以很好的提高学生的参与度。

(四)、课堂讨论总结。

课堂讨论总结的关键是：归纳发言者的主要观点；重申其重点及亮点；提醒大家对焦点 问题或有争议观点进行进一步思考；建议大家对案例素材进行扩展研究和深入分析。