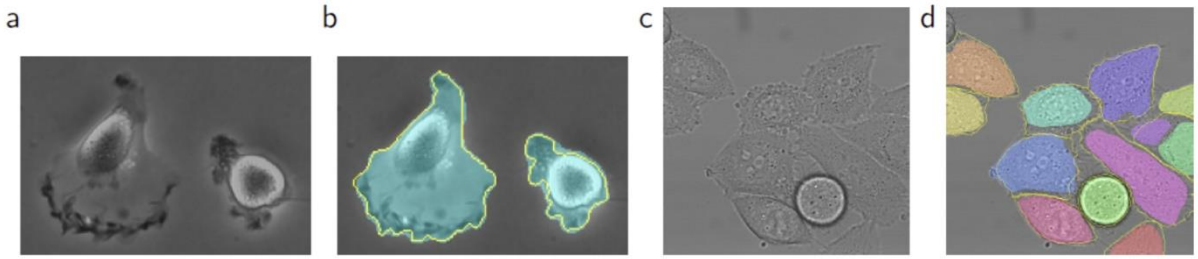
目录

一、U-Net 产生的原因以及简单介绍

🎈产生原因及背景：

U-Net 是为了解决生物医学图像分割问题而产生的。因为它的效果很好，所以后来被广泛应用于语义分割的各个方向：比如卫星图像分割等等。



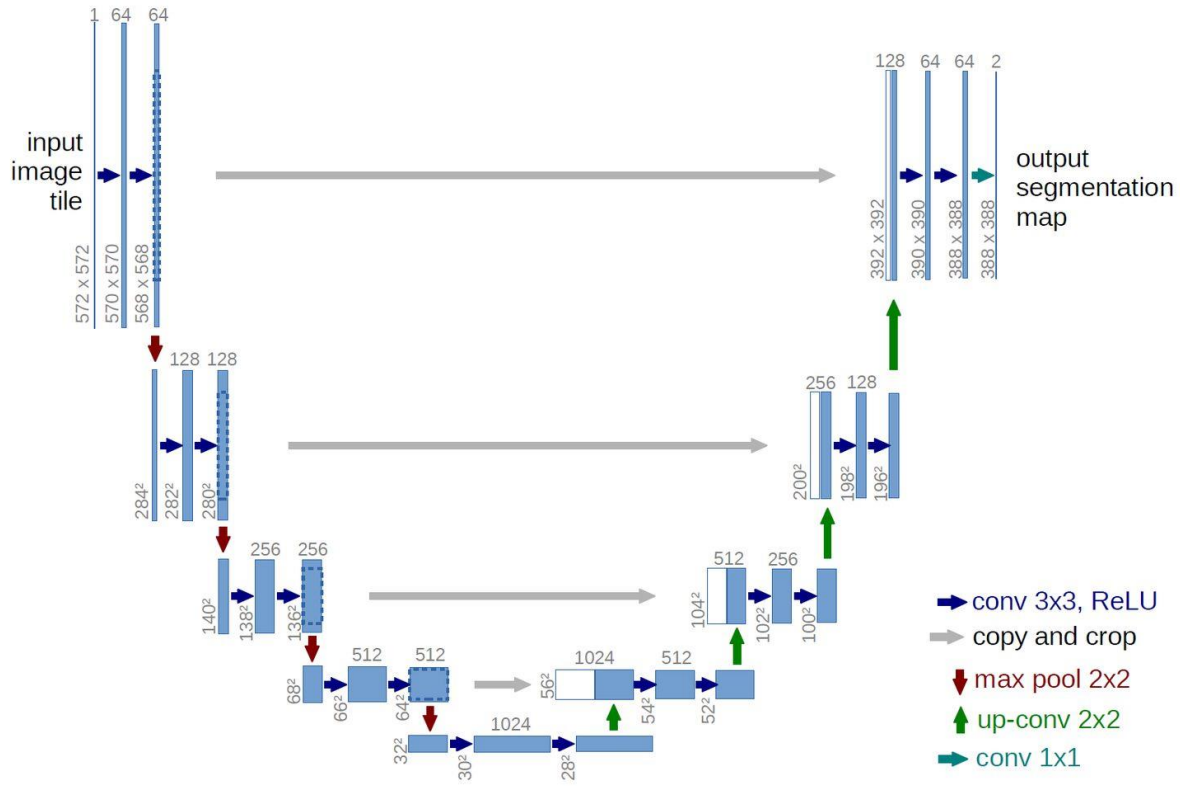
🎈变体：

U-Net 是由 FCN 衍生而来的，都是 Encoder-Decoder 结构，结构比较简单。

想了解 FCN 可以看博主往期文章全卷积网络 FCN 详解_tt 丫的博客-CSDN 博客

🎈优势和解决的问题：

因为在医学方面，样本收集比较困难，数据量难以达到那么多。为了解决这个问题，U-Net 应用了图像增强的方法，在数据集有限的情况下获得了不错的精度。



二、U-Net 网络结构分析

1、U-Net 网络结构图

🌳数字解释:

其中那些长条的上方（64，128 等等）都是通道数；像 572 * 572 这些是尺寸大小；

蓝白相间上的通道数是两份的总和，即白的和蓝的通道数各是那个通道数的一半

（比如下面这个蓝白相间通道数为 128，即代表白的 64，蓝的也是 64）

🌳框框和箭头解释:

蓝/白色框表示 feature map

蓝色箭头表示 3x3 卷积，用于特征提取；

灰色箭头表示 skip-connection（跳跃连接），用于特征融合；

红色箭头表示最大池化，用于降低维度；

绿色箭头表示上采样，用于恢复维度；

天蓝色（emmm 这个颜色是这么描述吧）箭头表示 1x1 卷积，用于输出结果。

2、U-Net 的 Encoder（收缩路径）
它是由卷积操作和下采样操作组成。

🌳卷积：

文中所用的卷积结构统一为 3x3 的卷积核，padding 为 0 ，striding 为 1，所以由公式
$n_{\text{out}} = \left[\frac{n_{\text{in}}\ 2p-k}{s}\right] 1$
得
$n_{\text{out}} = n_{\text{in}} - 2$
可以看到第 1~5 层卷积层都分别是由 3 个 3*3 卷积组成，每通过一个 3*3 卷积尺寸都减少 2

🌳池化（下采样）：

而前 4 层卷积层都通过最大池化进入下一层，各池化层的核大小均为 k=2，填充均为 p=0，步长均为 s=2，所以
$n_{\text{out}} = n_{\text{in}} /2$

🌳第 5 层没有 max-pooling，而是直接将得到的 feature map 送入 Decoder

3、U-Net 的 Decoder（扩展路径）
feature map 经过 Decoder 恢复原始尺寸，该过程由卷积，上采样和跳级结构组成。

🌳上采样：插值法

补充：

上采样一般有两种方法：

（1）反卷积（详见之前的博文 FCN 的介绍中全卷积网络 FCN 详解_tt 丫的博客-CSDN 博客）（2）插值（bilinear 双线性插值较为常见）

（原来的矩阵称为源矩阵；插值后的矩阵是目标矩阵）

举个栗子：我们要把以下 2*2 的矩阵插值成 4*4

1 2
3 4

1	2
3	4

🎈公式一 —— 目标矩阵到源矩阵的坐标映射：

(注：这里公式可能跟你平常看到的高和宽是反过来的，因为这里的 x 指代的是行，y 指代的是列，所以 x 对应高，y 对应宽；平常看到的 x 和 y 是那种平常接触的坐标系，x 是列，y 是行，所以才会看到 x 对应宽，y 对应高，两种做法都是一样的)

$$X_{src} = (X_{dst}0.5) * \left(\frac{\text{Height}_{src}}{\text{Height}_{dst}}\right) - 0.5$$
$$Y_{src} = (Y_{dst}0.5) * \left(\frac{\text{Width}_{src}}{\text{Width}_{dst}}\right) - 0.5$$

A 的坐标是（0，1）【第 0 行第 1 列】那么由公式得源矩阵坐标为（-0.25，0.25），是小数没事。

为了找到负数坐标点，我们将源矩阵扩展为下面的形式，中间红色的部分为源矩阵。

1 1 2 2
1 1 2 2
3 3 4 4
3 3 4 4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

那么 (0.25,1.25) 应该在这里面

1 2
1 2

🎈公式二 —— 具体点的值：

$$f(iu, jv) = (1-u)(1-v)f(i,j)(1-u)vf(i,j1)u(1-v)f(i1,j)uvf(i1,j1)$$

可得 i = -1, u = 0.75, j = 0, v = 0.25；再由这个公式可得 A 的值为 1.25

其他值以此类推

对应代码

nn.Upsample(scale_factor=2, mode='bilinear')

🌳跳级结构

FCN 中的跳级结构解释全卷积网络 FCN 详解_tt 丫的博客-CSDN 博客

FCN 是采用逐点相加的方法，而 U-Net 采用将特征在 channel 维度拼接在一起，形成更“厚”的特征，对应 caffe 的 ConcatLayer 层， 对应 tensorflow 的 tf.concat()。

对应代码

torch.cat([low_layer_features, deep_layer_features], dim=1)
这两种方法都是为了特征融合。

三、U-Net 的 pytorch 版实现
首先先导入所需要的库

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

对 Decorder 先进行定义

```
class Decoder(nn.Module):
  def __init__(self, in_channels,out_channels):
    super(Decoder, self).__init__()
    self.up = nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=2)
    #up-conv 2*2
    self.conv_relu = 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, high, low):
    x1 = self.up(high)
    offset = x1.size()[2]-low.size()[2]
    padding = 2*[offset//2,offset//2]
    #计算应该填充多少（这里可以是负数）
    x2 = F.pad(low,padding)#这里相当于对低级特征做一个crop操作
    x1 = torch.cat((x1, x2), dim=1)#拼起来
    x1 = self.conv_relu(x1)#卷积走起
    return x1
```

U-Net 整体网络框架

```
class UNet(nn.Module):
    def __init__(self, n_class):
        super().__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1,64,3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64,64,3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(64,128,3),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128,128,3),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )
```

```
        self.layer3 = nn.Sequential(
            nn.Conv2d(128,256,3),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256,256,3),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )
        self.layer4 = nn.Sequential(
            nn.Conv2d(256,512,3),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512,512,3),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
        )
        self.layer5 = nn.Sequential(
            nn.Conv2d(512,1024,3),
            nn.BatchNorm2d(1024),
            nn.ReLU(inplace=True),
            nn.Conv2d(1024,1024,3),
```

```
            nn.BatchNorm2d(1024),
            nn.ReLU(inplace=True)
        )

        self.maxpool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))

        self.decorder4 = Decoder(1024,512)
        self.decorder3 = Decoder(512,256)
        self.decorder2 = Decoder(256,128)
        self.decorder1 = Decoder(128,64)
        self.last = nn.Conv2d(64, n_class, 1)


    def forward(self, input):
        #Encorder
        layer1 = self.layer1(input)
        layer2 = self.layer2(self.maxpool(layer1))
        layer3 = self.layer3(self.maxpool(layer2))
        layer4 = self.layer4(self.maxpool(layer3))
        layer5 = self.layer5(self.maxpool(layer4))
```

```
61
62          #Decorder
63          layer6 = self.decorder4(layer5,layer4)
64          layer7 = self.decorder3(layer6,layer3)
65          layer8 = self.decorder2(layer7,layer2)
66          layer9 = self.decorder1(layer8,layer1)
67          out = self.last(layer9)#n_class预测种类数
68
69          return out
```

原文链接：https://blog.csdn.net/weixin_55073640/article/details/123060574