

# 【深度学习】基于 Pytorch 从 0 实现 UNet

惯例：这是我在科研过程中所用到的工具的学习笔记，并不是专业教程，仅作保存，以及自用查错。想学相关知识的话 B 站有很优秀的 UP 可以学习，我就不发表拙见了。若有错漏之处，还请多多包涵，欢迎评论区进行讨论，其实我发出来也有向大家请教的意思，毕竟一个人闷头做不可能所有方面都顾及到。

另：由于 B 站专栏对代码格式没有支持，我使用的都是截图，具体细节可以查询文件，代码文件也有详细的注释，建议结合代码文件观看本文，这样理解起来会简单很多。

GitHub 地址：

声明：本文只负责搭建网络并使用，不涉及网络优化、调整参数、结果评价等进阶操作。如果是老手，直接下载代码跑就完事了，（老手甚至都不用看我代码）。

## 背景：UNet

一句话：UNet 广泛应用于医学图像分割，效果很不错。

## 准备工作：

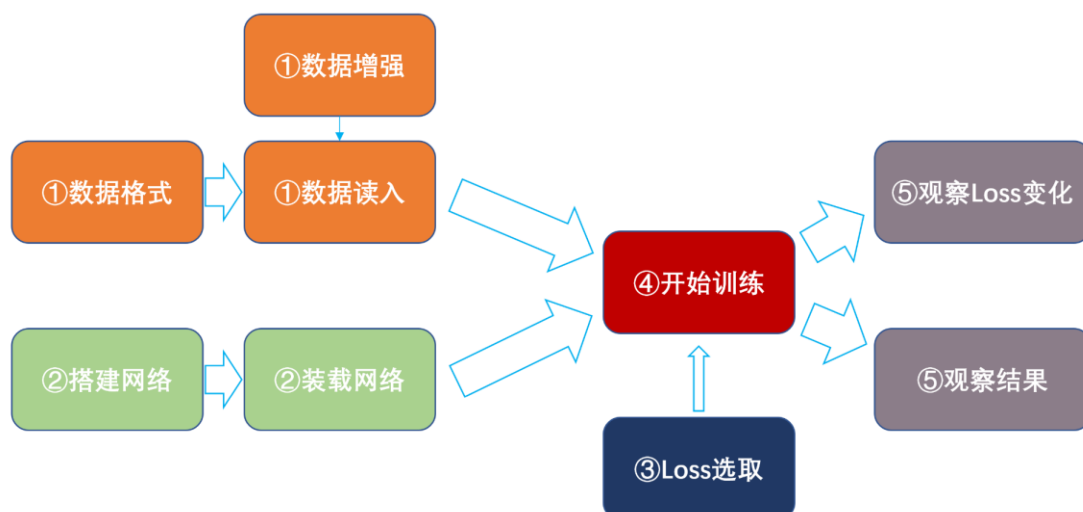
要求对机器学习以及深度学习有一定的了解，至少知道神经网络是什么，Loss 是什么。安装好了的 Anaconda，以及相关的包。

图省事的话可以在安装好 Anaconda 以后直接使用 Github 上的 requirement.txt 进行环境的配置。

## 目标：使用 Pytorch 构建 UNet，并应用于具体图像数据

本文主要目的为使用深度学习框架 Pytorch 来搭建一个最基本的 UNet 神经网络，从数据读取到网络搭建，再到训练和预测，全过程使用 Pytorch 封装好的类或者自定义函数从 0 实现 UNet 的应用。力求一针见血，只做必要的步骤，只要能跑通就行。2333

一图流：



## 一、数据

一切方法的基础就是数据, 没有好的数据, 就算算法能做到单张图像就能完成任务都是白搭。要让程序能跑起来, 必须先将数据用合适的方法读进内存, 并以合适的方法保存。

### Image:

读图像我主要用的是 PIL 的 Image 类, 这个类有读取图像的 open 方法, 也可以在读取图像时知道图像类型, 比如灰度\RGB 图像, 还能决定图像尺寸。

代码:

```
Image.open(self.imgs + '/' + self.img_path[index]).resize((self.img_H, self.img_W)).convert('RGB')
```

Ps: index 属于 torch.utils.data.Dataset 部分, 会在后面进行解释,

其实 pytorch 对 Image 格式支持还是挺不错的, 但是需要注意的点就是 Image 保存图像一般是[H, W, C]/ [B, H, W, C], 但是在 torch.tensor 中图像保存格式则是[B, H, W, C], 这也是后面为什么要变换 tensor 维度顺序。

```
img.permute(0, 3, 1, 2)
```

B: Batch, H: 图像的高, W: 图像的宽, C: 图像的通道数, RGB 一般每个分量一个通道, 即三通道(24 位), 灰度图则是一个通道(8 位)。

附赠一个 RGB 转灰度公式:

$$\text{gray} = R * 0.299 + G * 0.587 + B * 0.114$$

为了节省资源开销以及方便训练时数据的读入,

Pytorch 有自己专门的数据读取方法, Dataset 和 DataLoader。

流程可总结为:

图像文件->Dataset(数据的索引)-> DataLoader(按照用户定义的规格打包数据, 等待训练)

## Dataset:

可以使用官方默认预定义的 Dataset，不过一般来说不同人的数据不一样，基本都是要自己重新定义的。

官方文档

```
class Dataset(object):
    """An abstract class representing a Dataset.

    All other datasets should subclass it. All subclasses should override
    ``__len__``, that provides the size of the dataset, and ``__getitem__``,
    supporting integer indexing in range from 0 to len(self) exclusive.
    """

    def __getitem__(self, index):
        raise NotImplementedError

    def __len__(self):
        raise NotImplementedError

    def __add__(self, other):
        return ConcatDataset([self, other])
```

`__init__()`: 构造函数，定义类内变量，为读取数据做准备。

`__getitem__()`: `_DataLoaderIter()`类中有调用，`DataLoader` 会根据 `index` 循环调用函数，分步将数据读进内存，这里 `index` 的范围与 `__len__()` 相关。

`__len__()`: 返回数据长度的函数，一般与数据大小相等，也有例外，下面会讲到。

Ex: 数据增强:

通俗点来讲即使增加训练集。

众所周知，深度学习其实是一种需要大量样本来进行训练的方法。但是很多时候我们找不到很多数据，比如罕见疾病的图像、医生标注的金标准。

这里就要用到数据增强，又称数据增广、图像扩增。

指利用旋转、缩放、灰度变换等手段处理数据，以达到获取“新”样本的目的，`pytorch` 中也有自带的变换函数 `torchvision.transforms`，不过本文没有用到这种方法，原因则是 UNet 输入的训练集以及标签均为图像，数据增强时需要对样本和标签进行同样的处理。

但是 `transforms` 每次只支持输入一个样本，并且每次处理都涉及到概率问题，即有可能无法对样本和标签进行同样的处理，所以这里我使用了其他方法：

```
label = Image.open(
    self.labels + '/' + self.label_path[int(index/self.times)])
angle = transforms.RandomRotation.get_params(
    [-180, 180]) # 旋转随机角度，范围为[-180, 180]
img = torch.from_numpy(np.array(img.rotate(angle)))
label = torch.from_numpy(
    np.array(label.rotate(angle))).unsqueeze(0)
```

times 决定数据扩增几倍，angle 则是每次扩增时都随机取角度，并对 img 和 label 进行相同的操作。

其实这一步可以进行更多种类的变换，这里只是为了示意就实现了一种。

有条件的还可以使用 Albumentations 这个库，功能强大。

## DataLoader:

Dataloader 是个很重要的函数，一定要学会怎么用，结合 dataset 为佳。

简单直接上代码：

```
train_dataloader = DataLoader(training_data, batch_size=batch_size)
```

在上一步定义好\_\_getitem\_\_()后，在这里就可以直接将数据读进来了，通知给它指定你想要的 batch\_size，这个参数应该是深度学习里面的经典参数了，也是你秀显卡性能的最直观舞台。

DataLoader 会按照你定义的\_\_getitem\_\_函数将样本按照 batch\_size 打包成 batch，将来在训练的时候就可以以此将它喂给你的网络。

Ps: epoch 就不展开说明了，可以直观地将其理解为整个训练流程重复的次数（网络权重会一直保留）。

这样一来我们就完成了训练数据的准备。其实一般来说还要对训练集划分验证集（val），这个需要你对机器学习的训练机理有一定的了解（推荐西瓜书），如果不想考究太多的话见到 val 字眼直接无视掉就好。

接下来就是重头戏，定义网络了。

## 二、搭建网络

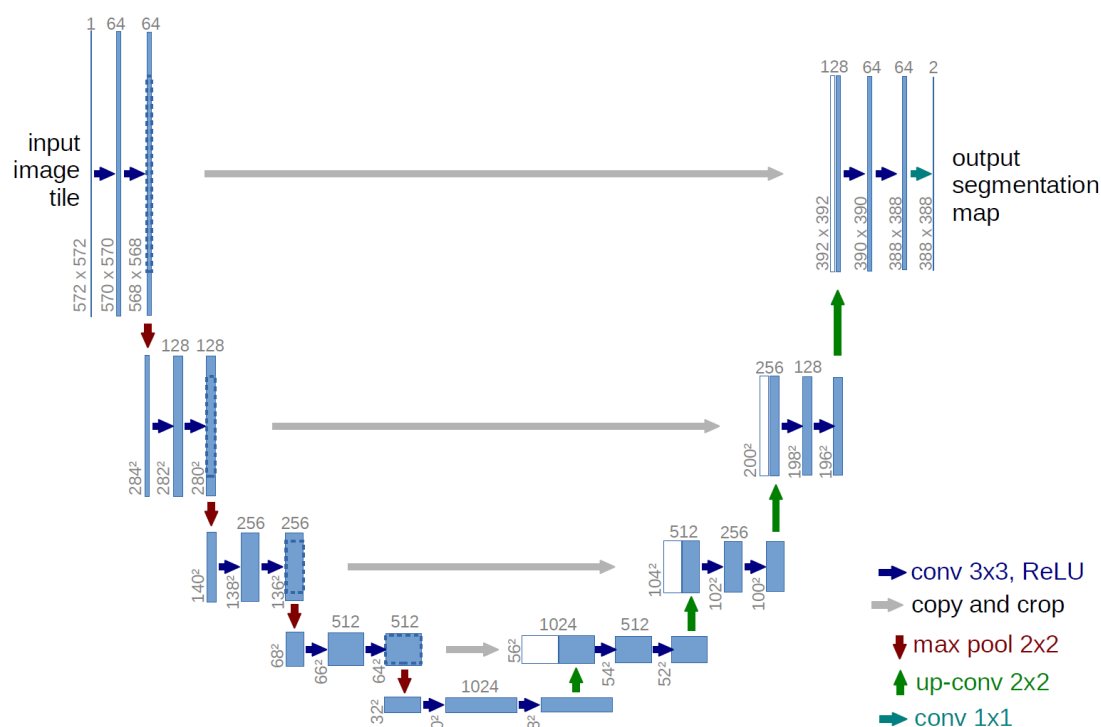
深度学习说白了学的就是网络的参数，至于网络的参数具体指的是什么，有兴趣了解深度学习的童鞋可以看一下花书，或者不理解也可以，就把它当作一个黑箱模型就好，我把我的东西输进去，得到的就是我要的东西（端到端）。

不过就算是黑箱，也可以了解下黑箱的表面构造，即如何搭建网络。

我将其称为搭积木，如果再有示意图的前提下，甚至比搭积木还简单。

接下来我给大家展示一下什么叫做“照葫芦画瓢”（不感兴趣的同学可以直接跳到 UNet 末尾）。

# UNet:



这个就是大名鼎鼎的 UNet 结构图了，事实上现在很多网络都用到这种结构 (或部分结构)，我们一般将其称为编码器(Encoder)-解码器(Decoder)模式。

从图例可以看出，蓝色箭头为卷积(conv)，灰色箭头为拼接(copy and crop)，红色箭头为最大池化(max pool)，绿色箭头为上采样 (up-conv 反卷积)，天蓝色箭头本文自定义为用户指定通道数的卷积。

上面提到的词有：

卷积：nn.Conv2d

拼接：torch.cat

最大池化：nn.MaxPool2d

上采样：nn.ConvTranspose2d

可以看到卷积操作一般都是进行两次的，

在这里定义为双卷积：DoubleConv

具体函数的作用以及函数对应的参数可以查阅参考资料以及代码，也可以通过对比图里的数字 (通道数) 以及代码中的数字去尝试理解这幅图。

现在就开始搭积木了：

左边下降的部分：

输入-双卷积-最大池化-双卷积-最大池化-双卷积-最大池化-双卷积-最大池化-双卷积

对应代码：

```

self.conv1 = DoubleConv(in_ch, 32)
self.pool1 = nn.MaxPool2d(2)
self.conv2 = DoubleConv(32, 64)
self.pool2 = nn.MaxPool2d(2)
self.conv3 = DoubleConv(64, 128)
self.pool3 = nn.MaxPool2d(2)
self.conv4 = DoubleConv(128, 256)
self.pool4 = nn.MaxPool2d(2)
self.conv5 = DoubleConv(256, 512)

```

右边上升部分：

上采样-拼接-双卷积-上采样-拼接-双卷积-上采样-拼接-双卷积-上采样-拼接-双卷积-输出

对应代码：

```

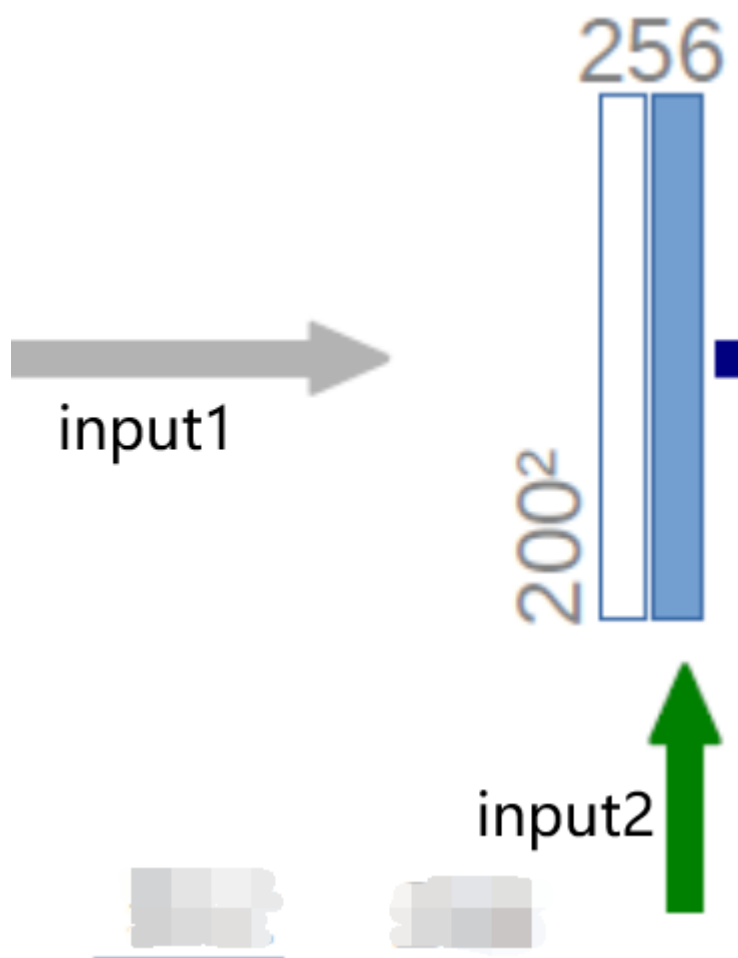
up_6 = self.up6(c5)
merge6 = torch.cat([up_6, c4], dim=1)
c6 = self.conv6(merge6)
up_7 = self.up7(c6)
merge7 = torch.cat([up_7, c3], dim=1)
c7 = self.conv7(merge7)
up_8 = self.up8(c7)
merge8 = torch.cat([up_8, c2], dim=1)
c8 = self.conv8(merge8)
up_9 = self.up9(c8)
merge9 = torch.cat([up_9, c1], dim=1)
c9 = self.conv9(merge9)
c10 = self.conv10(c9)

```

Ps：在网络卷积后加入 Batchnormalization(nn.BatchNorm2d)，会很大程度上提升网络的性能。

为了直观，上下代码不是同一个函数（定义）内的。

Tips：拼接(copy and crop)不要看作是一个对称的过程，而是看做一个卷积层的输入（如下图），这样可能会比较好理解一点



最终得到一个模型，也是本文的核心：

```
model = Unet(3, 1).
```

由于继承了 nn.Module 这个类，上面的那个实例化方法其实就装载好模型了。

### 三、Loss 选取

简单说一下 Loss 的作用：让模型收敛，往我们希望它学习到的方向去学习。

就好像空有一身肌肉的壮士，没有师父的训练，他终究只是一介莽夫。

本文中比较了三中 Loss：CrossEntropyLoss（交叉熵）、FocalLoss、DiceLoss

这里也是用一张图表达，具体的可以查阅参考资料或者代码。

#### CrossEntropyLoss（交叉熵）：

$$L = -y \log y' - (1 - y) \log(1 - y') = \begin{cases} -\log y', & y = 1 \\ -\log(1 - y'), & y = 0 \end{cases}$$

y 为标签(label)，y' 为网络预测出的图像。

交叉熵对于正样本而言，输出概率越大损失越小。

缺点：损失函数在大量简单样本的迭代过程中比较缓慢且可能无法优化至最优。

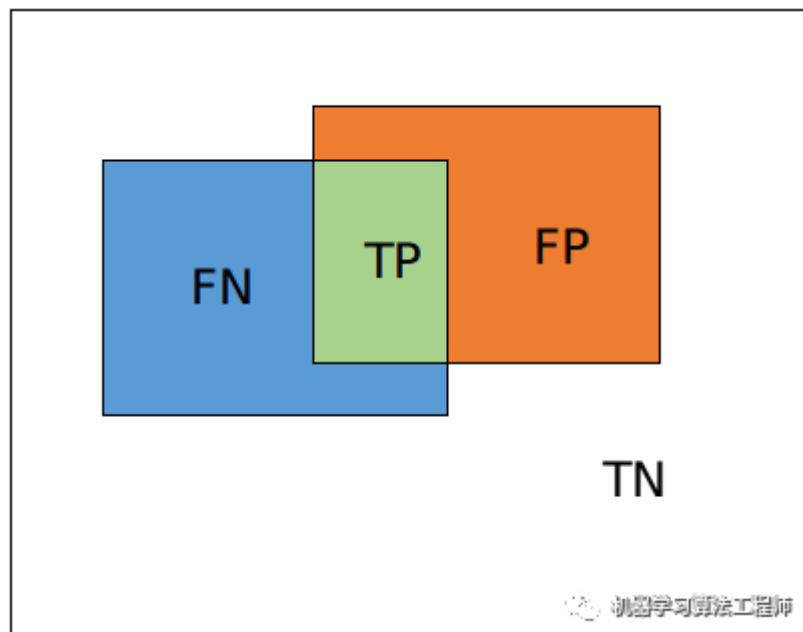
## FocalLoss: (改进交叉熵)

$$L_{fl} = \begin{cases} -(1-y')^\gamma \log y' & , \quad y = 1 \\ -y'^\gamma \log(1-y'), & y = 0 \end{cases}$$

特点：使得更关注于困难的、错分的样本。

## DiceLoss:

这个就更直接了，也是图像分割经常使用的 Loss



$$dice = \frac{2TP}{2TP + FP + FN} \quad L_{dice} = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$$

但是为了防止出现等于 0 的情况，一般都会分子分母同时加上一个数。

## 四、开始训练

```
model = Unet(3, 1).to(device)

loss_fn = DiceLoss()
```

现在我们得到了一个“肌肉猛汉”，以及一位“师父”，那要怎么才能让他们两个“开始训练”呢？这里就用到一个优化器，也就是“师父”使用的工具，即“教鞭”。



```
optimizer = torch.optim.Adam(model.parameters())
```

“猛汉”使用数据进行预测，并交给“师父”评价。

```
model.train()  
pred = model(img)  
loss = loss_fn(pred, label)
```

而“师父”是通过一个梯度回传来“使用”这个“教鞭”。

```
loss.backward()  
optimizer.step()
```

比喻终究只是比喻，想了解具体的作用机理的话还是建议看一下网络训练的资料以及相关知识，比如梯度下降法。还是那句话，结合代码来阅读本文的话会直观很多。

训练过程：

神经网络的训练一般可以按 batch 来划分，

```
for batch, (img, label) in enumerate(train_dataloader):
```

每个 batch 进行一次 Loss 的记录，Loss 的变化是训练效果最直观的表现，也有不少博客专门讲怎么通过 Loss 变化评价网络性能的，比如过拟合。

## 五、观察结果(Loss)

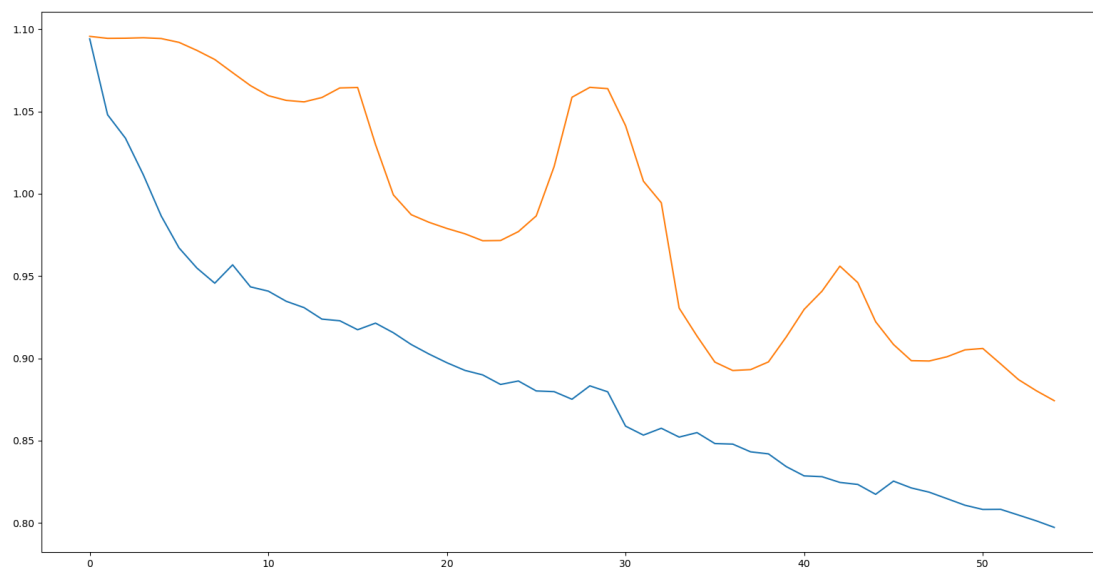
蓝色为训练 Loss，橙色为验证 Loss。

### CrossEntropyLoss（交叉熵）：

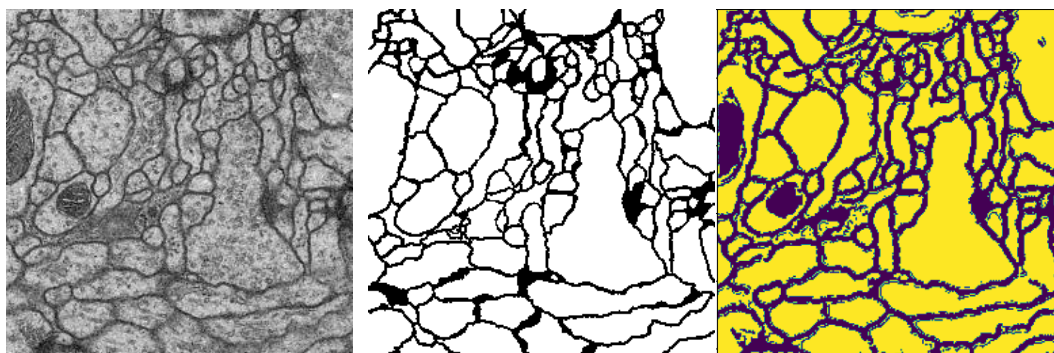
注意：在没有 Batchnormalization 的时候该损失函数没有结果。

数据扩增：否

Loss 变化：

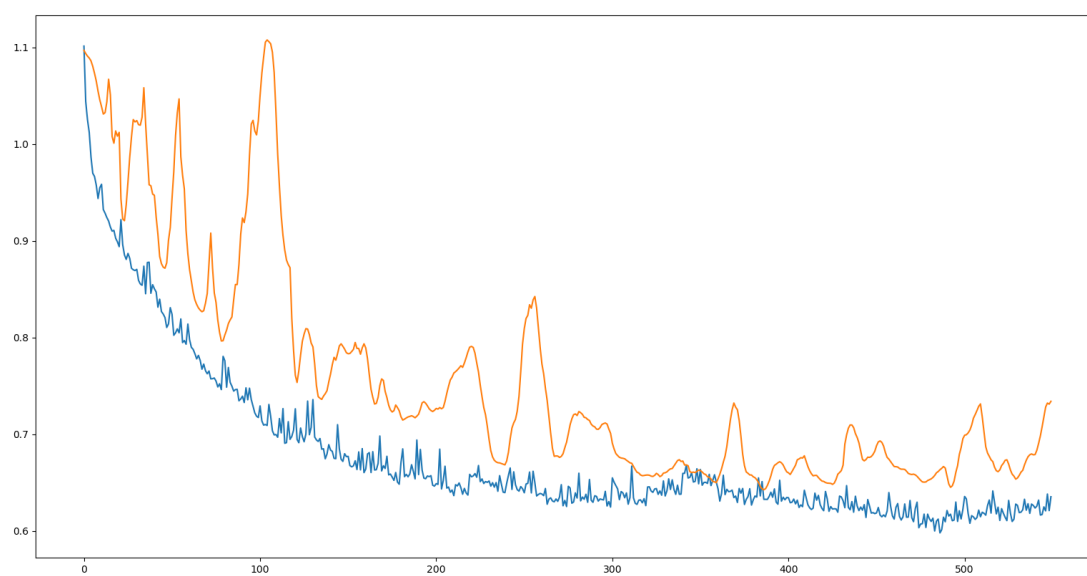


分割结果：

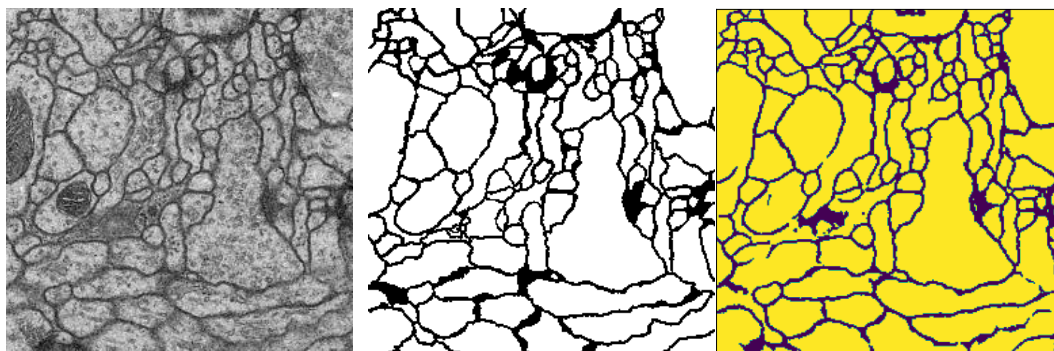


数据扩增：是，10 倍

Loss 变化：



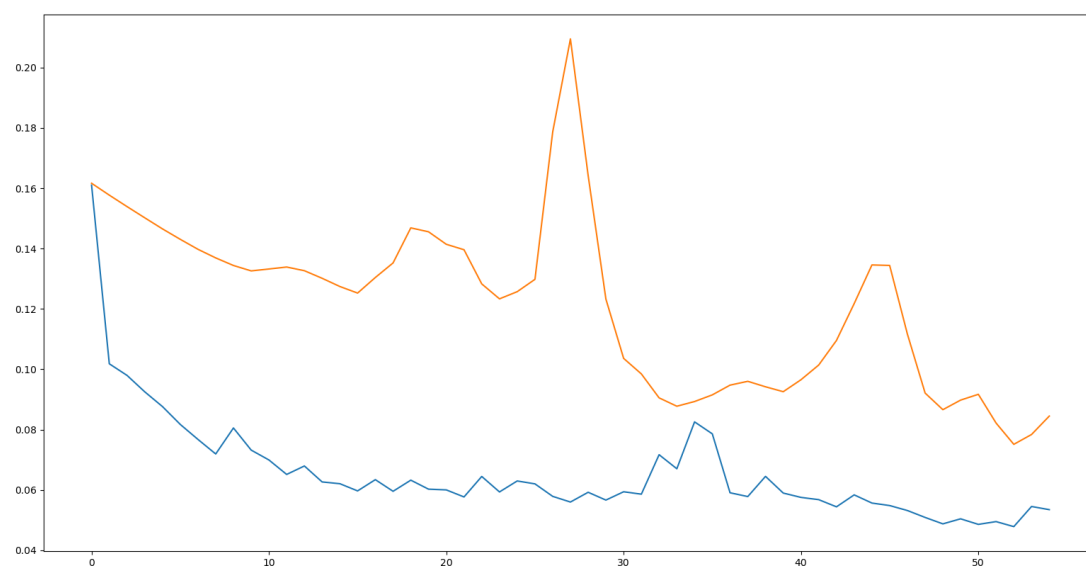
分割结果：



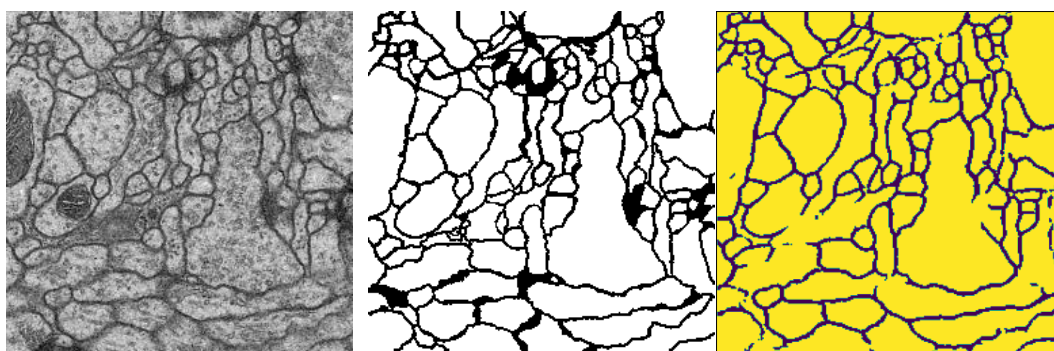
## FocalLoss: (改进交叉熵):

数据扩增: 否

Loss 变化:

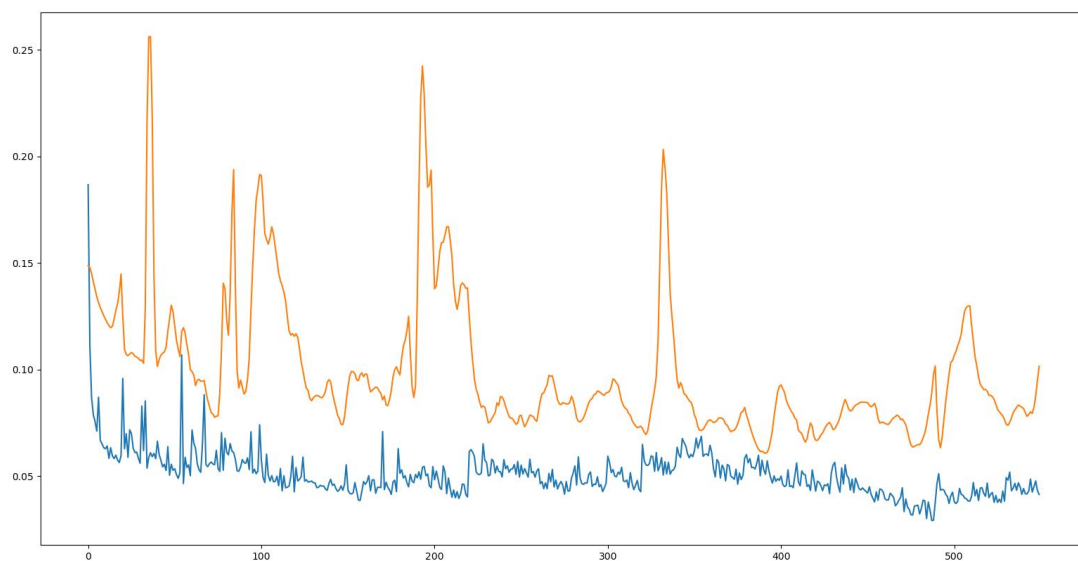


分割结果:

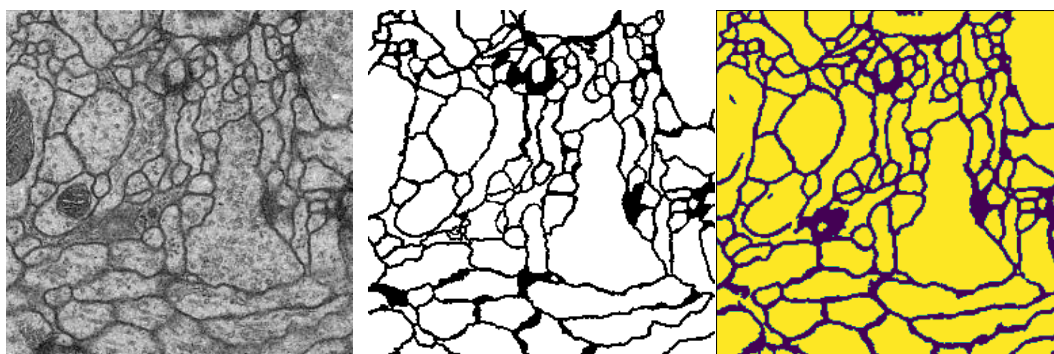


数据扩增: 是, 10 倍

Loss 变化:



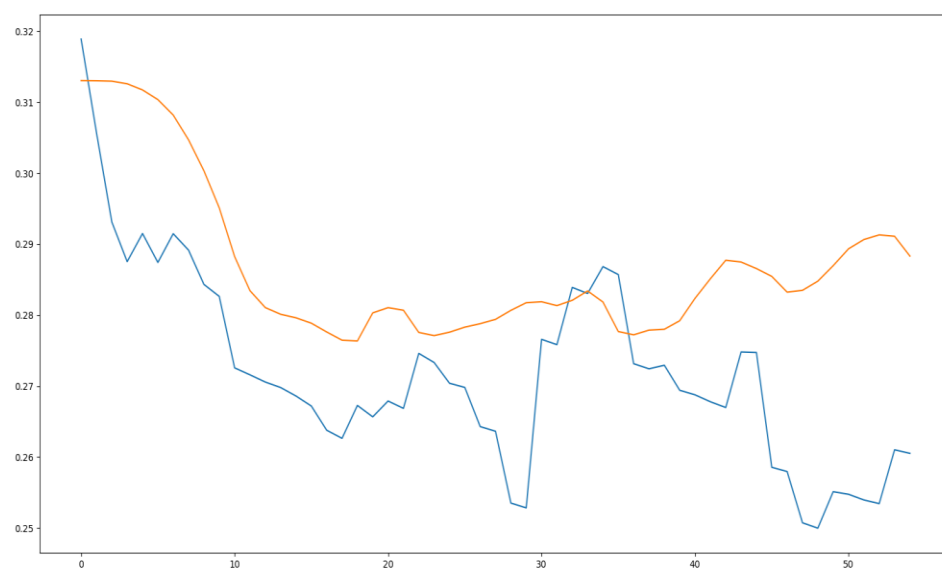
分割结果:



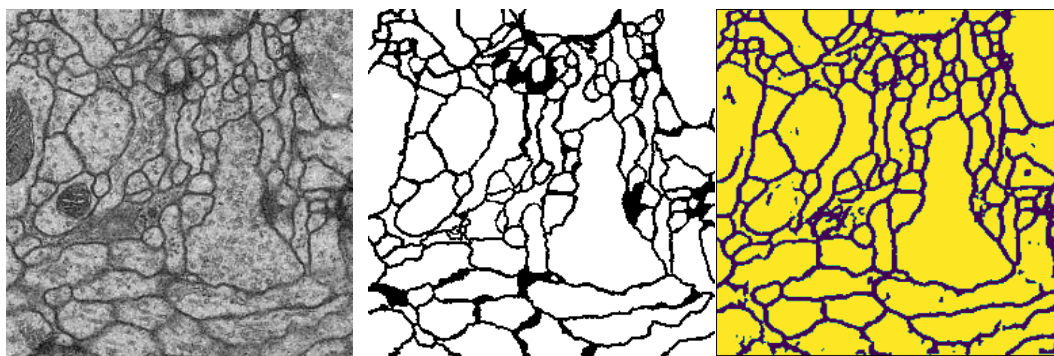
**DiceLoss:**

数据扩增: 否

Loss 变化:

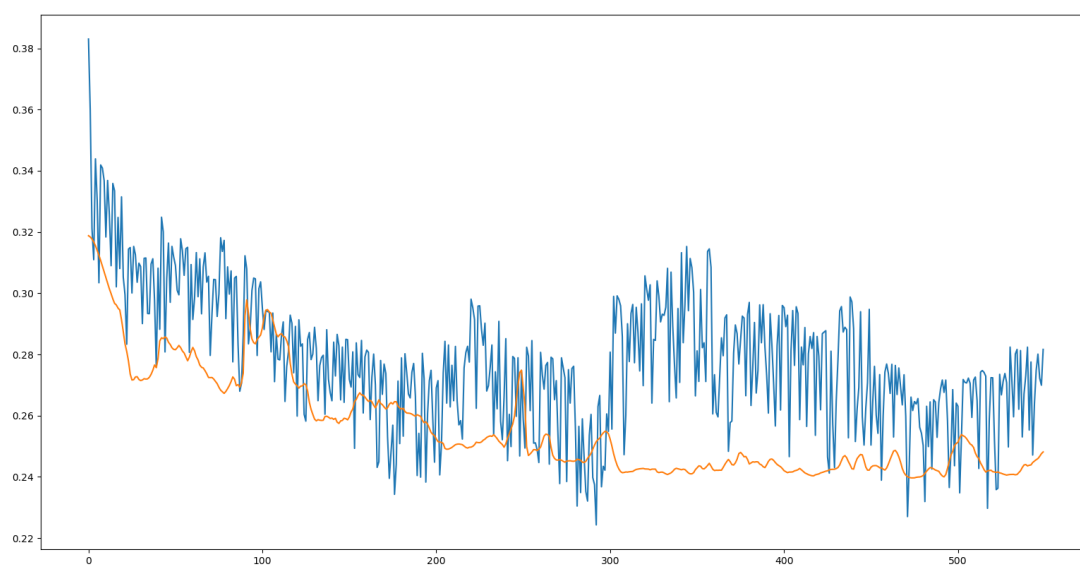


分割结果：

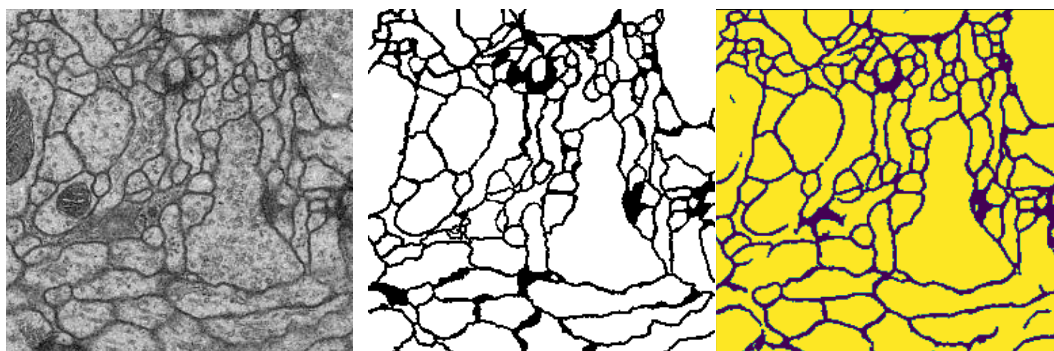


数据扩增：是，10 倍

Loss 变化：

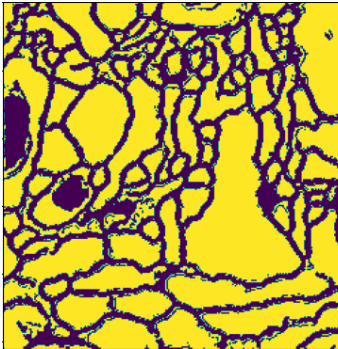
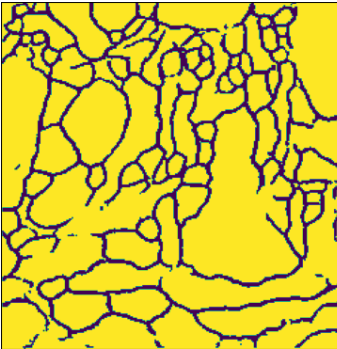
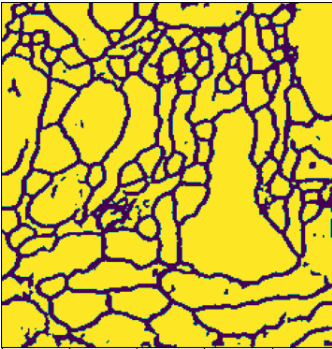
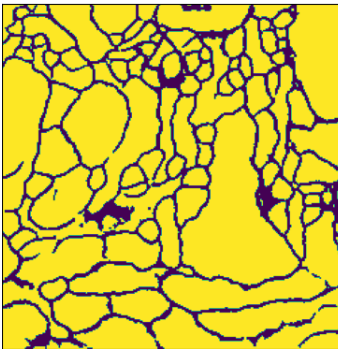
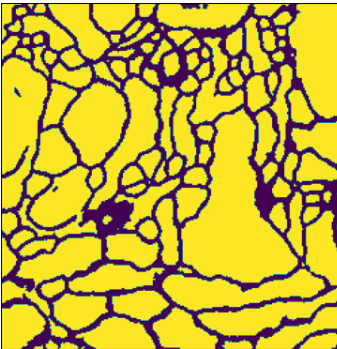
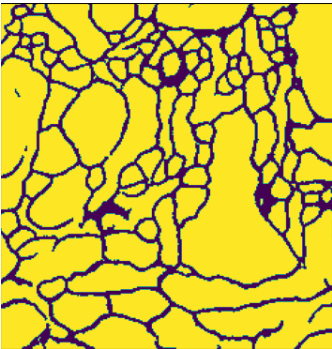


分割结果：





## 对比表：

	CrossEntropyLoss	FocalLoss	DiceLoss
数据不扩增			
数据扩增			

## 六、补充说明

有些地方比如怎么保存模型还有如何测试数据等，本文就不再赘述，但是代码中会给出较为详细的注释

（因为有的坑踩了，没人告诉你还真跳不出去 T\_T）

以前我用 Keras 实现过 UNet，不过是很久之前的事了，现在重新捡起来 UNet，还是花了点时间去熟悉，但不得不说，Pytorch 用起来太流畅了。

## 参考资料：（关键词+网址）

### 背景：

Pytorch 入门：<https://pytorch.org/tutorials/beginner/basics/intro.html#>

UNet 背景：<https://www.zhihu.com/question/269914775?sort=created>

UNet 框架 1：<https://github.com/milesial/Pytorch-UNet>

UNet 框架 2：<https://github.com/Andy-zhujunwen/UNET-ZOO/blob/master/UNet.py>

UNet 框架 3：<https://zhuanlan.zhihu.com/p/97488817>

医学图像数据集汇总：<https://zhuanlan.zhihu.com/p/102855802>

## 数据读取

数据读取之 Dataset: <https://blog.csdn.net/tfcy694/article/details/85251036>

Dataset: <https://blog.csdn.net/tfcy694/article/details/85251036>

pytorch 数据读取\_\_getitem\_\_1: <https://blog.csdn.net/zzzpy/article/details/88749592>

pytorch 数据读取\_\_getitem\_\_2:

[https://blog.csdn.net/qg\\_40178291/article/details/102326077](https://blog.csdn.net/qg_40178291/article/details/102326077)

Dataset and DataLoader1: <https://zhuanlan.zhihu.com/p/128679151>

Dataset and DataLoader2: [https://www.cnblogs.com/cxq1126/p/13949136.html#\\_label1](https://www.cnblogs.com/cxq1126/p/13949136.html#_label1)

Pytorch 读取数据 1 (图像图片):

[https://blog.csdn.net/weixin\\_36815313/article/details/108441164](https://blog.csdn.net/weixin_36815313/article/details/108441164)

Pytorch 读取数据 2: <https://zhuanlan.zhihu.com/p/27434001>

Pytorch 读取数据 3: <https://blog.csdn.net/shunshune/article/details/89316572>

Pytorch 读取数据 4: [https://zhuanlan.zhihu.com/p/103339003?utm\\_source=qg](https://zhuanlan.zhihu.com/p/103339003?utm_source=qg)

使用 torch.utils.data.random\_split()划分数据集:

[https://blog.csdn.net/qg\\_42951560/article/details/115445317](https://blog.csdn.net/qg_42951560/article/details/115445317)

Tensor 与 Image 转换之 torchvision.transforms.ToTensor 与 ToPILImage 1:

[https://blog.csdn.net/qg\\_37385726/article/details/81771980](https://blog.csdn.net/qg_37385726/article/details/81771980)

Tensor 与 Image 转换之 torchvision.transforms.ToTensor 与 ToPILImage 2:

<https://blog.csdn.net/nekoli/article/details/89918058>

Tensor 与 Image 转换之 torchvision.transforms.ToTensor 与 ToPILImage 3:

[https://blog.csdn.net/qg\\_40264206/article/details/109131376](https://blog.csdn.net/qg_40264206/article/details/109131376)

Img 灰度与"RGB"转换: <https://blog.csdn.net/u012977885/article/details/105733554>

PIL.Image.Resize (图像尺寸):

[https://blog.csdn.net/weixin\\_40446557/article/details/87896752](https://blog.csdn.net/weixin_40446557/article/details/87896752)

PIL.Image.Rotate (图像旋转):

[https://blog.csdn.net/Tornado\\_Liao/article/details/109508848](https://blog.csdn.net/Tornado_Liao/article/details/109508848)

Pytorch 使用 Albumentations:

[https://zhuanlan.zhihu.com/p/147594227?from\\_voters\\_page=true](https://zhuanlan.zhihu.com/p/147594227?from_voters_page=true)

Pytorch 数据扩增 1: <https://www.cnblogs.com/cxq1126/p/13325478.html>

Pytorch 数据扩增 2: <https://cloud.tencent.com/developer/article/1795473>

Pytorch 数据扩增 3: <https://zhuanlan.zhihu.com/p/54527197>

Pytorch 数据扩增 4: <https://zhuanlan.zhihu.com/p/54566524>

同时对两张图片进行数据扩增:

<https://blog.csdn.net/happyeveryday62/article/details/104350332>

Pytorch 传统数据扩增: [https://blog.csdn.net/weixin\\_42437114/article/details/107220420](https://blog.csdn.net/weixin_42437114/article/details/107220420)

二值化图像技巧:

<https://www.pythonheidong.com/blog/article/614138/37673e2e51e691aa3e7f/>

Tensor 升降维度: [https://blog.csdn.net/flysky\\_jay/article/details/81607289](https://blog.csdn.net/flysky_jay/article/details/81607289)

Tensor 维度变换: <https://blog.csdn.net/zjc910997316/article/details/111566753>

对比 numpy 维度变换: [https://blog.csdn.net/qg\\_34806812/article/details/89385831](https://blog.csdn.net/qg_34806812/article/details/89385831)

划分验证集: <https://zhuanlan.zhihu.com/p/199238910>

验证集的作用: [https://blog.csdn.net/qg\\_34107425/article/details/104097370](https://blog.csdn.net/qg_34107425/article/details/104097370)

## Loss 选取:

CrossEntropyLoss (交叉熵) 1: [https://blog.csdn.net/geter\\_CS/article/details/79849386](https://blog.csdn.net/geter_CS/article/details/79849386)  
CrossEntropyLoss (交叉熵) 2: [https://blog.csdn.net/pain\\_gain0/article/details/109407742](https://blog.csdn.net/pain_gain0/article/details/109407742)  
CrossEntropyLoss (交叉熵) 3: <https://discuss.pytorch.org/t/multiclass-segmentation/54065>  
CrossEntropyLoss (交叉熵) 4: [https://blog.csdn.net/ft\\_sunshine/article/details/92074842](https://blog.csdn.net/ft_sunshine/article/details/92074842)  
FocalLoss1: [https://blog.csdn.net/qg\\_35054151/article/details/114272442](https://blog.csdn.net/qg_35054151/article/details/114272442)  
FocalLoss2: <https://www.kaggle.com/c/tgs-salt-identification-challenge/discussion/65938>  
FocalLoss3: <https://www.cnblogs.com/king-lps/p/9497836.html>  
DiceLoss1: <https://jishuin.proginn.com/p/763bfd2aeb3>  
DiceLoss2: <https://blog.csdn.net/liangjiu2009/article/details/107352164>  
DiceLoss3: [https://zhuanlan.zhihu.com/p/144582930?from\\_voters\\_page=true](https://zhuanlan.zhihu.com/p/144582930?from_voters_page=true)  
DiceLoss4: <https://jishuin.proginn.com/p/763bfd2aeb3>

## UNet 网络搭建:

UNet 理解: [https://blog.csdn.net/weixin\\_43839245/article/details/108101643](https://blog.csdn.net/weixin_43839245/article/details/108101643)  
Torch.nn.Conv2d() 用法 1: [https://blog.csdn.net/qg\\_38863413/article/details/104108808](https://blog.csdn.net/qg_38863413/article/details/104108808)  
Torch.nn.Conv2d() 用法 2: [https://blog.csdn.net/qg\\_26369907/article/details/88366147](https://blog.csdn.net/qg_26369907/article/details/88366147)  
Torch.nn.MaxPool2d 1: [https://blog.csdn.net/qg\\_36387683/article/details/107638184](https://blog.csdn.net/qg_36387683/article/details/107638184)  
Torch.nn.MaxPool2d 2: <https://blog.csdn.net/iblctw/article/details/107088462>  
Feature map: <https://www.cnblogs.com/spore/p/13282959.html>  
ConvTranspose2d 原理 (上采样):  
[https://blog.csdn.net/qg\\_27261889/article/details/86304061/](https://blog.csdn.net/qg_27261889/article/details/86304061/)  
Pytorch 的 torch.cat: [https://blog.csdn.net/qg\\_39709535/article/details/80803003](https://blog.csdn.net/qg_39709535/article/details/80803003)  
Batch Normalization: <https://zhuanlan.zhihu.com/p/52749286>  
BatchNorm2d 理解: <https://blog.csdn.net/heiheihei000000/article/details/107497273>

## 开始训练:

数据不平衡: <https://www.cnblogs.com/hotsnow/p/10954624.html>  
优化器、梯度下降法: <https://blog.csdn.net/wydbxyr/article/details/84822806>  
Epoch: [https://blog.csdn.net/qg\\_43232545/article/details/100181595](https://blog.csdn.net/qg_43232545/article/details/100181595)  
model.train()和 model.eval()用法和区别 1:  
[https://blog.csdn.net/qg\\_38410428/article/details/101102075](https://blog.csdn.net/qg_38410428/article/details/101102075)  
model.train()和 model.eval()用法和区别 2:  
<https://www.cnblogs.com/luckypli/p/13424561.html>  
Albumentations: <https://github.com/albumentations-team/albumentations>  
多 GPU 训练、保存与装载 1:  
[https://blog.csdn.net/weixin\\_42105432/article/details/99582322](https://blog.csdn.net/weixin_42105432/article/details/99582322)  
多 GPU 训练、保存与装载 2: [https://blog.csdn.net/gaishi\\_hero/article/details/81139045](https://blog.csdn.net/gaishi_hero/article/details/81139045)



多 GPU 训练、保存与装载 3: <https://www.cnblogs.com/blog4lly/p/11711173.html>

## 观察结果:

绘制 Loss 曲线: <https://blog.csdn.net/tequilaro/article/details/81841748>

保存每次的 Loss: [https://blog.csdn.net/weixin\\_38324105/article/details/90202840](https://blog.csdn.net/weixin_38324105/article/details/90202840)

Pytorch loss.item()大坑!!! (爆显存):

<https://blog.csdn.net/StarfishCu/article/details/112473856>

Pytorch 各种坑: <https://www.zhihu.com/question/67209417/answer/344752405>

## 代码报错参考:

- RuntimeError: Expected object of scalar type Long but got scalar type Float for argument #2 'target' in call to \_thnn\_nll\_loss2d\_forward:  
<http://www.weainfo.net/news/detail/472613>
- expected input to have 3 channels, but got 1 channels instead:  
<https://www.cnblogs.com/pprp/p/11705791.html>
- Pytorch 自定义 Dataset 和 DataLoader 去除不存在和空的数据 collate\_fn (callable, optional): <https://blog.csdn.net/guyuealian/article/details/91129367>
- 'builtin\_function\_or\_method' object is not subscriptable:  
<https://blog.csdn.net/bjttwendy/article/details/80288034>
- SyntaxError: encoding problem:gbk:  
<https://blog.csdn.net/yuyilahanbao/article/details/104477167/>
- warning: LF will be replaced by CRLF:  
<https://www.zhihu.com/question/50862500/answer/123197258>
- SyntaxError: encoding problem:gbk(UTF-8):  
[https://blog.csdn.net/qq\\_27492735/article/details/108850160](https://blog.csdn.net/qq_27492735/article/details/108850160)
- RuntimeError: 1only batches of spatial targets supported (3D tensors) but got:  
[https://blog.csdn.net/weixin\\_39190382/article/details/114433884](https://blog.csdn.net/weixin_39190382/article/details/114433884)
- RuntimeError: 1only batches of spatial targets supported (non-empty 3D tensors) but got targets of size: <https://www.cnblogs.com/dyc99/p/12665778.html>
- RuntimeError: 1only batches of spatial targets supported (non-empty 3D tensors) but got targets of size: <https://blog.csdn.net/qazwsrx/article/details/103974636>
- RuntimeError: Expected object of scalar type Long but got scalar type Float for argument #2 'target': <https://blog.csdn.net/shangxiaqiusuo1/article/details/95749684>