基于 3D U-Net 的肋骨骨折检测

王资 519030910345
薛峥嵘 519030910349

1. 问题描述

1.1 研究背景

医学影像的识别（recognition）、分割（segmentation）和解析（parsing）是医学影像分析的核心任务。医学影像的一大挑战是临床应用对精度、稳定性和速度的严格要求。读片和诊断通常不允许出错。尽管要求很高的精度和稳定性，速度仍不能慢，一个快速的工作流能够确保医院的高吞吐量。放射科和外科医生不会愿意花几个小时甚至几分钟去等待一个分析结果。

不同于自然图像，医学影像具有很强的上下文信息，例如有限数量的解剖目标，约束和结构化的背景，不同解剖结构之间的关系，强先验姿态参数信息等。能够通过大量数据获取这样的上下文信息的统计机器学习方法非常适用于医学影像处理。

然而，出于各方面的原因，医学图像数据集的整理较自然图像要困难许多，医学图片的样本数量不够多，这就给训练带来了困难，所以我们要求网络能够充分利用数据。相较于自然图像，医学图像（如 CT、MRI 图像）的内存占用往往十分庞大的，这给我们的模型训练带来了不小的压力，因此我们需要一个尽可能简单的网络。这两大限制使得近些年来几乎所有的基于深度学习的医学影像研究[1]都采用了 U-Net 或类 U-Net 的网络结构。在有限的时间内，我们不认为我们能做出结构上的重大创新，因此我们也采用了类 U-Net 的网络结构。

1.2 数据集

本次研究采用的数据集来自于 MICCAI2020 的一个公开比赛[2]，这个数据集是华东医院用了近十年的时间整理的。公开的数据集有 660 张 CT 照片，这些照片都是胸片，总共包含了约 5000 处骨折。其中，420 张作为训练集，80 张作为验证集，160 张作为测试集。CT 照片是用.nii.gz 格式的文件保存的，每张照片有相对应的标签，标签的格式是 voxel-wise 的 0/1 标签。

我们可以用一些专业软件打开 CT 图片来更直观地查看骨折区域的形态学特征。通过请教医学院的同学，我们了解到，肋骨骨折区域最主要的形态学特征是：在骨折区域，肋骨外侧的白色层是不封闭的；同时肋骨内部的骨质颜色相较于正常区域较浅。医生在检查肋骨骨折是往往是按照肋骨在空间上的分布顺序，逐肋骨地进行检查。

然而，在实践的过程中，我们发现这些医学先验很难发挥作用。对于医学影像处理领域的初学者而言，使用深度学习的端对端方法依然是最便捷有效的分割方法。

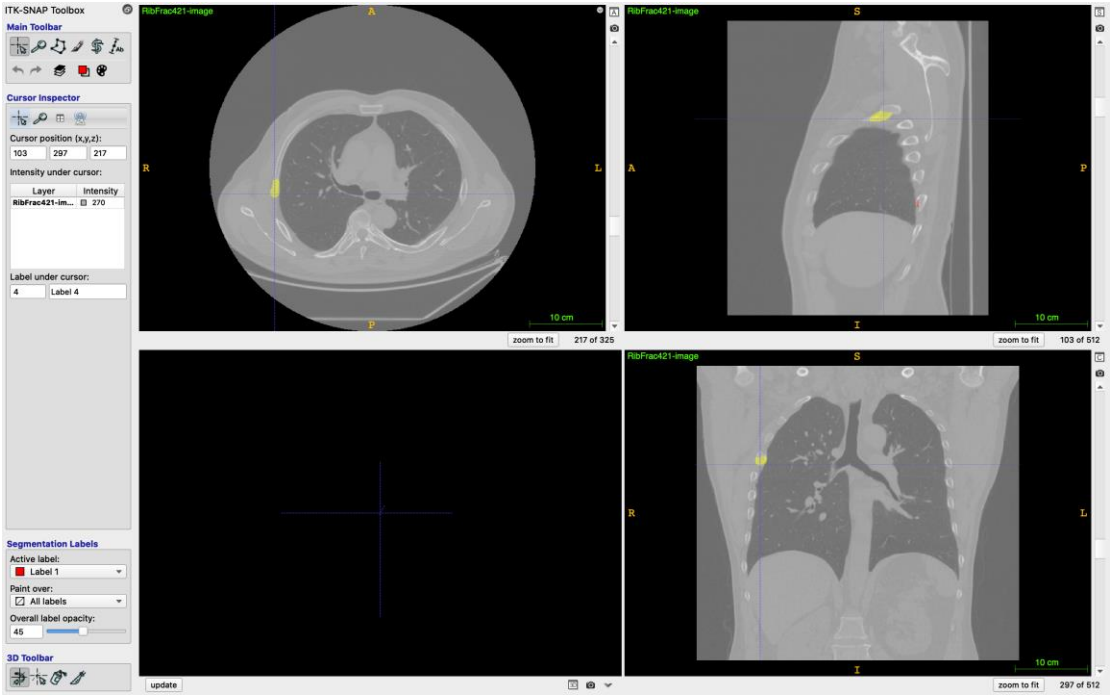

图 1 使用 ITK-SNAP 查看胸片形态和骨折区域的局部特征

2. 方法论

2.1 网络结构

在本实验中，我们分别实现了原版的 3D U-Net[3]网络结构和 FracNet[4]的网络结构。我们发现相较于原版的 3D U-Net，FracNet 的网络参数更少，在基本不损失表征能力的前提下，FracNet 的运行速度更快。因此，在之后的实验中，我们均采用 FracNet 的网络结构。我们在这里对 FracNet 的网络结构做一个简单的介绍。

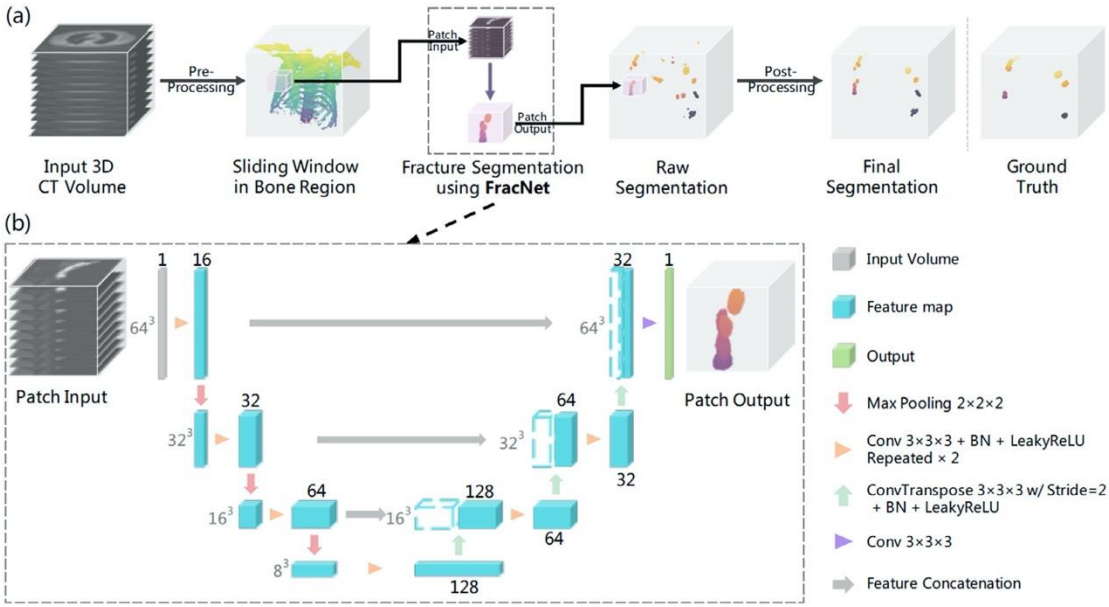

图 2 FracNet 的网络结构

尺寸为 64×64×64 的输入图像块先通过编码路径进行信息压缩，再通过解码路径进行信息还原。每一条路径都有 4 个分辨率级别，在解码的过程中，编码路径的信息将作为新通道直接拼接到解码路径信息中。这样做的直观是，编码-解码的过程要求网络用更少的信息完成任务，从而迫使学到更有用的信息，丢弃冗余的信息；不同分辨率的设置使网络可以关注到不同尺度的信息，直接拼接的操作使网络在同一层既能关注到 low-level 的表征信息，又能关注到 high-level 的表征信息。值得一提的是，3D 点云领域的开山之作 PointNet[5]也采用这样的网络结构，从发表时间上看，3D U-Net 很有可能是受到了 PointNet 的启发。

更加具体的实现细节是：在编码路径上，每一层的操作是做两遍 3×3×3 卷积 + Batch Normalization + Leaky ReLU，后接一个 Max Pooling 下采样。在解码路径上，每一层的操作是先做一个步长为 2 的 3×3×3 的反卷积层 + Batch Normalization + Leaky ReLU，再将来自编码路径的相同尺寸的特征图拼接过来，最后再做两遍 3×3×3 卷积 + Batch Normalization + Leaky ReLU。最后的输出层和输入层的尺寸相同，每个体素的值表征了该位置出现骨折的概率。

```
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size = 3, stride = 1, padding = 1):
        super(ConvBlock, self).__init__()
        self.conv3d = nn.Conv3d(in_channels = in_channels, out_channels = out_channels,
                                kernel_size = kernel_size, stride = stride, padding = padding)
        self.batch_norm = nn.BatchNorm3d(num_features = out_channels)
        self.relu = nn.LeakyReLU(inplace = True)

    def forward(self, x):
        x = self.conv3d(x)
        x = self.batch_norm(x)
        x = self.relu(x)

        return x

class ConvTranspose(nn.Module):
    def __init__(self,in_channels, out_channels, kernel_size = 2, stride = 2, padding = 0, output_padding = 0):
        super(ConvTranspose, self).__init__()
        self.conv3d_transpose = nn.ConvTranspose3d(in_channels = in_channels, out_channels = out_channels,
                                                   kernel_size = kernel_size, stride = stride,
                                                   padding = padding, output_padding = output_padding)
        self.batch_norm = nn.BatchNorm3d(num_features = out_channels)
        self.relu = nn.LeakyReLU(inplace = True)
    def forward(self, x):
        x = self.conv3d_transpose(x)
        x = self.batch_norm(x)
        x = self.relu(x)

        return x
```

图 3 编码和解码的代码块

```
class U_Net(nn.Module):
    def __init__(self):
        super(U_Net, self).__init__()
        self.max_pool = nn.MaxPool3d(kernel_size = 2)

        self.encoder_1_1 = ConvBlock(1, 16)
        self.encoder_1_2 = ConvBlock(16, 16)
        self.encoder_2_1 = ConvBlock(16, 32)
        self.encoder_2_2 = ConvBlock(32, 32)
        self.encoder_3_1 = ConvBlock(32, 64)
        self.encoder_3_2 = ConvBlock(64, 64)
        self.encoder_4_1 = ConvBlock(64, 128)
        self.encoder_4_2 = ConvBlock(128, 128)

        self.deconv_4 = ConvTranspose(128, 128)
        self.deconv_3 = ConvTranspose(64, 64)
        self.deconv_2 = ConvTranspose(32, 32)

        self.decoder_3_1 = ConvBlock(192, 64)
        self.decoder_3_2 = ConvBlock(64, 64)
        self.decoder_2_1 = ConvBlock(96, 32)
        self.decoder_2_2 = ConvBlock(32, 32)

        self.final = nn.Conv3d(in_channels = 48, out_channels = 1,kernel_size = 1)

        for m in self.modules():
            if isinstance(m, nn.Conv3d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='leaky_relu')
            elif isinstance(m, nn.BatchNorm3d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
```

图 4 编码和解码的路径

2.2 数据预处理

输入的 patch 尺寸是 64×64×64，将全部训练集分割成小块喂进所带来的内存开销是我们不能接受的。在这个问题中，数据预处理最核心的问题就是找到一种合适的采样方法，有选择地从原图片中提出对训练有帮助的 patch。

一个十分符合直观的想法是，提出 label 中含有正样本的区域就可以了。我们只需简单地调用 skimage.measure 中的 regionprops 函数就可以把正样本区域提取出来，以区域的 centriod 为中心，提出一个 64×64×64 的 patch，就是我们感兴趣的正样本。由于在测试时我们并不能保证骨折区域在中心位置出现，我们需要对选择的中心做一些抖动来提升模型的泛化能力，在实验中，我们在各个方向上加入了(-10, 10)的随机抖动。

接下来需要提取负样本。由于 CT 图片在 y 方向上基本是沿着脊柱对称的，我们只需要提出一个和正样本在 y 方向上对称的区域就可以了。由于脊柱的位置往往并不是完全居中，因此这样选取也保证了泛化的效果。

接下来需要思考，是否选取正样本区域和与其对称的负样本区域就足够了呢？对于人体外围或者肺实质这些明显与肋骨有区分的区域，我们相信网络具有足够的表征能力去区分这些区域。然而，对于脊柱这一和肋骨在形态学上有些相似的区域，如果网络没有见过一定量的样本的话，我们不能保证它能做出可靠的区分。因此，我们需要在每张

图片上提出一些脊柱的负样本，告诉网络脊柱和肋骨的区别。

为了保证正负样本数量的均衡，我们在大体上保证肋骨正样本、肋骨负样本、脊柱负样本三者的数量是均衡的。

2.3 数据增强

TorchIO 库[6]提供了数量丰富的针对医学影像的数据增强工具。如空间上的翻转、放射变换，体素强度上的模糊、MRI 的各种伪影等。我们对于这个库的功能做了一定的了解，但出于一些技术原因和时间的限制，我们没有来得及实现全部的功能。

2.4 损失函数

我们选用了 Dice Loss 和 Focal Loss 的混合作为损失函数。

Dice Loss 源于二分类，本质上是衡量两个样本的重叠部分。该指标范围从 0 到 1，其中 1 表示完整的重叠。其计算公式为：

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

其中，$|A \cap B|$表示集合 A、B 之间的共同元素，$|A|$表示 A 中的元素的个数，$|B|$也用相似的表示方法。

假设我们用p来表示预测值，用t来表示真实标签值，那么 Dice Loss 的值是$1 - \frac{2pt}{p+t}$或$1 - \frac{2pt}{p^2+t^2}$，其关于p的梯度形式为$\frac{2t^2}{(p+t)^2}$或$\frac{2t(t^2-p^2)}{(p^2+t^2)^2}$，可以看到在极端情况下即$p$和$t$都很小的时候，计算得到的梯度值可能会非常大，即会导致训练十分不稳定。

为了在一定程度上解决这一问题，我们需要引入交叉熵 Loss。在这里，我们使用了交叉熵 Loss 的变种，即何凯明团队在 RetinaNet 论文中提出的 Focal Loss [7]。在直观上，Focal Loss 主要解决了两个问题，一个是正负样本的数量不匹配的问题，这个问题已经被我们阐述的采样方法很好地解决了。另一个问题是，论文认为易分样本（即，置信度高的样本）对模型的提升效果非常小，模型应该主要关注与那些难分样本。这个问题在我们的任务中确实是存在的，所以我们有针对性地调整了该 Loss 的参数来主要解决第二个问题。

在实验中，我们尝试使用过一些结构更加复杂的损失函数，比如 GHM Loss [8]，但并没有获得比较明显的效果提升。

2.5 数据后处理

在医学图像的处理中，由于先验知识的存在，数据后处理特别重要。

2.5.1 基于阈值的后处理

我们对于人类标注的骨折区域大小和预测结果的骨折区域大小和置信做了一个统计分析，我们发现，人类标注的区域一般体积较大，因此，我们抛弃了体积过小的预测区域。对于置信这一指标，我们发现即便某一区域的置信较小，它依然可能成为真实的骨折区域，因此我们设定了一个比较低的阈值。

public_id	label_id	confidence	label_code	area	centroid
RibFrac421	0	0.0	0	0	0.0
RibFrac421	1	0.24323445068453900	1	1455	(54.69278350515464, 315.87560137457046, 184.46666666666667)
RibFrac421	2	0.9340748437231690	1	7174	(87.21522163367717, 317.6576526345135, 141.0383328686925)
RibFrac421	3	0.6122143431444520	1	7304	(108.72713581599123, 364.4769989047098, 129.07817634173057)
RibFrac421	4	0.19121883968322000	1	2257	(102.57111209570226, 300.65352237483387, 225.49712007089056)
RibFrac421	5	0.38329520029105800	1	728	(125.17582417582418, 380.4546703296703, 182.5)
RibFrac421	6	0.30276486262239200	1	153	(125.37908496732027, 382.45098039215685, 158.88235294117646)
RibFrac421	7	0.18629047251631400	1	100	(158.04, 313.57, 286.49)
RibFrac421	8	0.21170803759442500	1	256	(166.015625, 369.03515625, 61.421875)
RibFrac421	9	0.2122497730424150	1	234	(177.52991452991452, 236.77777777777777, 279.9316239316239)
RibFrac421	10	0.3558011088195210	1	5294	(193.45258783528521, 258.98451076690594, 307.398375519456)
RibFrac421	11	0.12606777223441400	1	1797	(211.92376182526434, 249.6989426822482, 283.985531441291)
RibFrac421	12	0.3784441769080210	1	363	(249.41322314049586, 187.39944903581267, 256.79338842975204)
RibFrac421	13	0.41895697311787000	1	1188	(314.4587542087542, 254.82996632996634, 287.40488215488216)
RibFrac421	14	0.8952749801106380	1	5498	(322.75154601673336, 459.58312113495816, 314.7906511458712)
RibFrac421	15	0.12747202368396700	1	2265	(327.03796909492274, 281.0944812362031, 313.878587196468)
RibFrac421	16	0.1229781714338360	1	258	(341.468992248062, 291.4418604651163, 286.1085271317829)

图 5 对于 RibFrac421 预测结果的指标统计

2.5.2 基于提取肺实质区域的后处理

通过检查预测结果，我们注意到，尽管我们已经尽可能在采样阶段增加网络所能见到的不同样本，但网络依然会提出一些明显不在肋骨区域的骨折区域，有些甚至位于人体之外。我们检查了若干张照片的预测区域，发现不在肋骨上的预测区域为数不少，如果能给出肋骨的 RoI，那么想必能显著地降低假阳性预测的数量。

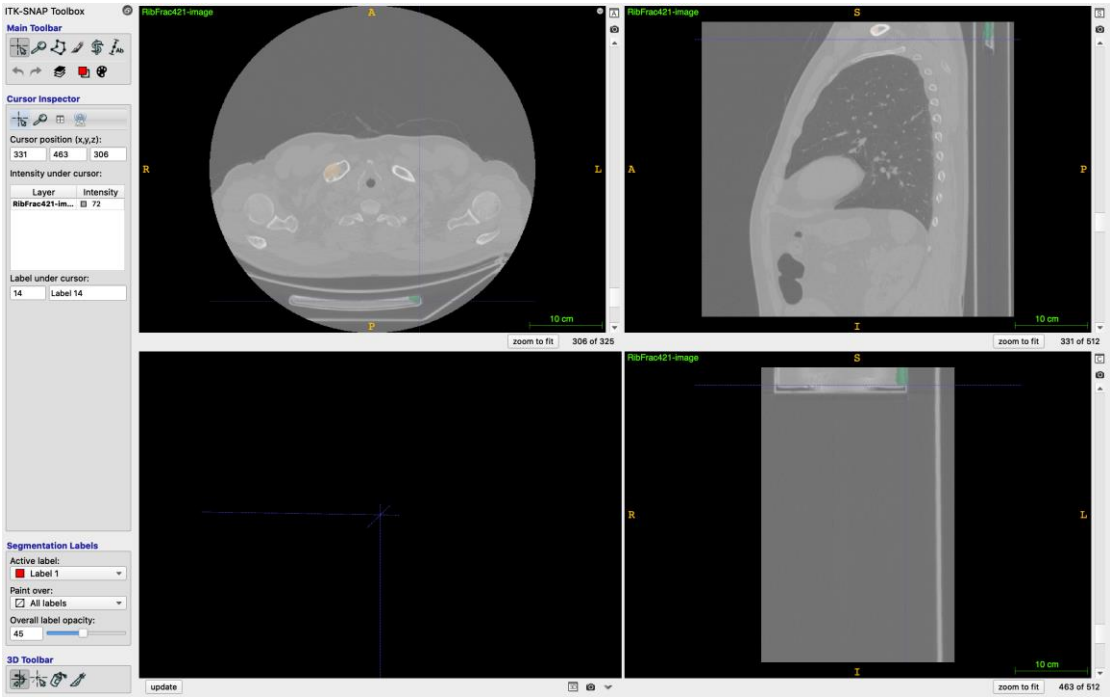

图 6 两个假阳性预测，一个位于体外，一个位于锁骨

然而，我们并没有找到任何标注出肋骨 RoI 的数据集，这使得提取肋骨 RoI 的工作难以进行。于是，我们选择退而求其次，先提取出肺实质，然后将肺实质向外膨胀若干体素，以达到包裹住肋骨的目的。

在实现的过程中，我们参考了一篇 MICCAI2017 的文章[9]中的预处理方法，这篇文章利用形态学的方法提出了肺实质。基本思路是，我们可以用体素强度和空间位置作为阈值大致划定出一些可能属于肺实质的体素，从这些体素出发，建一个图，建图的依据是按照体素强度的接近程度，然后，寻找这个图的最大连通分量，得到的结果就是肺实质。又注意到肺和气管是联通的，算法会将肺实质和气管一起提出，文章又提出了一些更加复杂的形态学方法去进一步分割肺和气管。

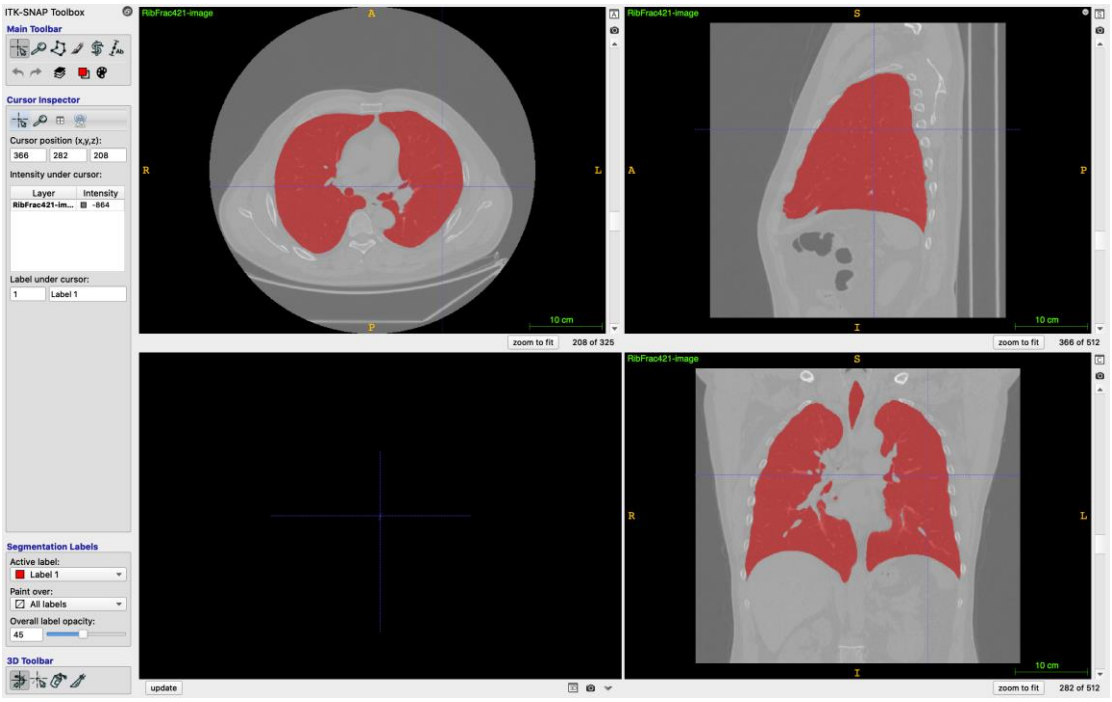

图 7 用最大连通分量提取出的肺

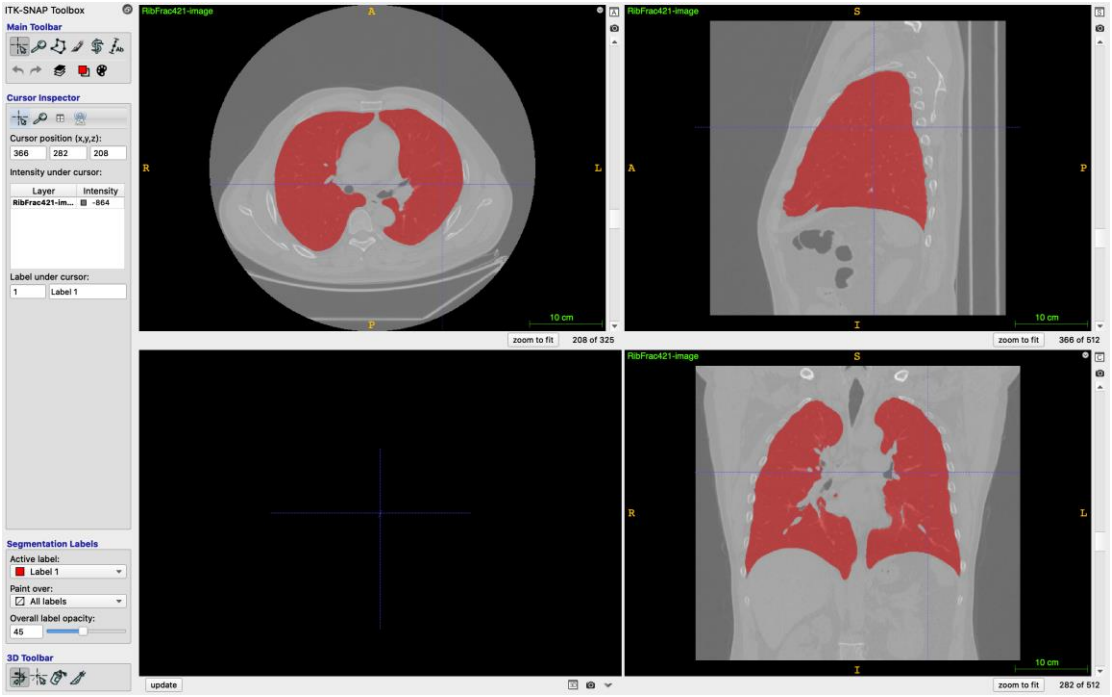

图 8 切除气管后得到的更加干净的肺

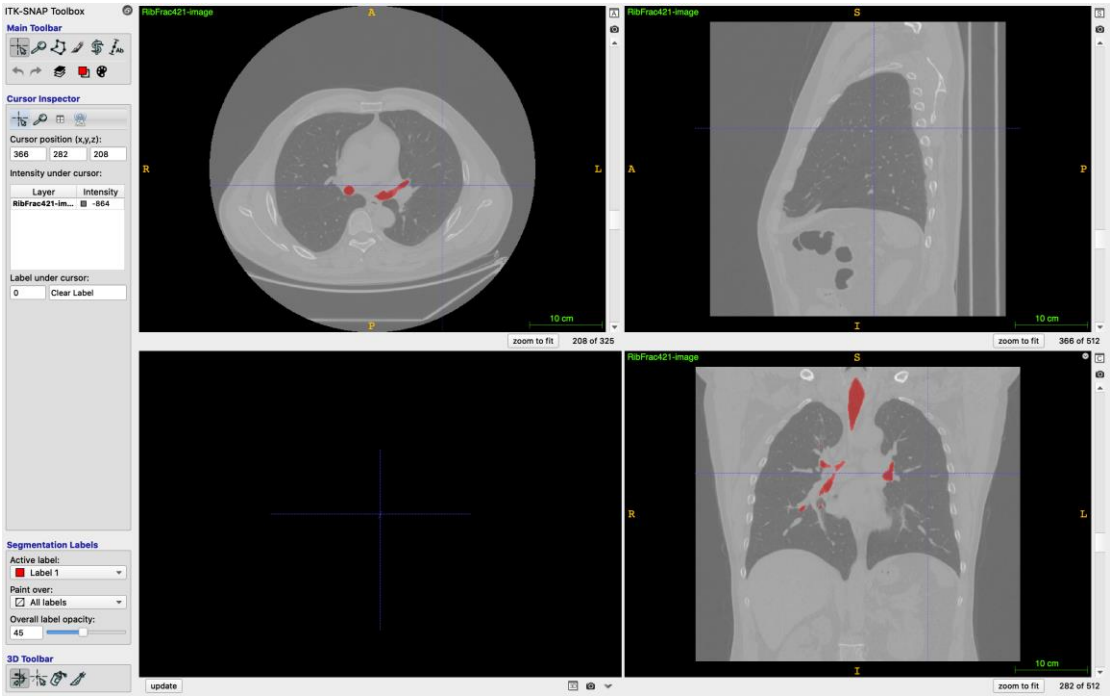

图 9 用更复杂的形态学方法得到的气管

可以直观地看到，算法的效果是十分不错的。但是我们必须要考虑一些工程上的问题。我们拥有现成的高效的算法去寻找图的最大连通分量，因此粗分割肺的时间几乎可以忽略不计。但是，进一步分割肺和气管是基于一个迭代算法完成的，这个算法的收敛时间很大程度上取决于某张 CT 照片的具体形态，平均需要约 5 分钟的时间。这个时间虽然比较长，但仍然可以接受。

一个更加严重的问题是，分割肺和气管的算法的鲁棒性并不是特别好。我们通过检查分割区域体积的方式粗略地检查了算法了鲁棒性，粗分割肺的鲁棒性很好，只需通过调参的方法去简单地处理极个别异常值就可以了，但由于这个数据集的数据并不是特别规整，分割肺和气管的错误率比较高，我们尝试了若干方法试图提升鲁棒性，但均以失败告终。

图 10 数据集内的照片在亮度、对比度、形态上存在差异

因此，我们放弃了更细的分割，直接采用粗分割。这样做带来的坏处是在锁骨等距离气管较近的区域可能会引入更多的假阳性，但通过控制肺向外膨胀的体素的大小，我们获得了可以让人接受的效果。唯一的问题是由于人体结构的原因，最下方的一根肋骨已经深入腹腔，距离肺实质较远，因此不能被膨胀后的肺实质包裹住，这可能会导致最大召回率有小幅的下降。我们暂时不能从根本上解决这一问题，但无论如何，这已经是我们能够获得的最好的折衷的方案。

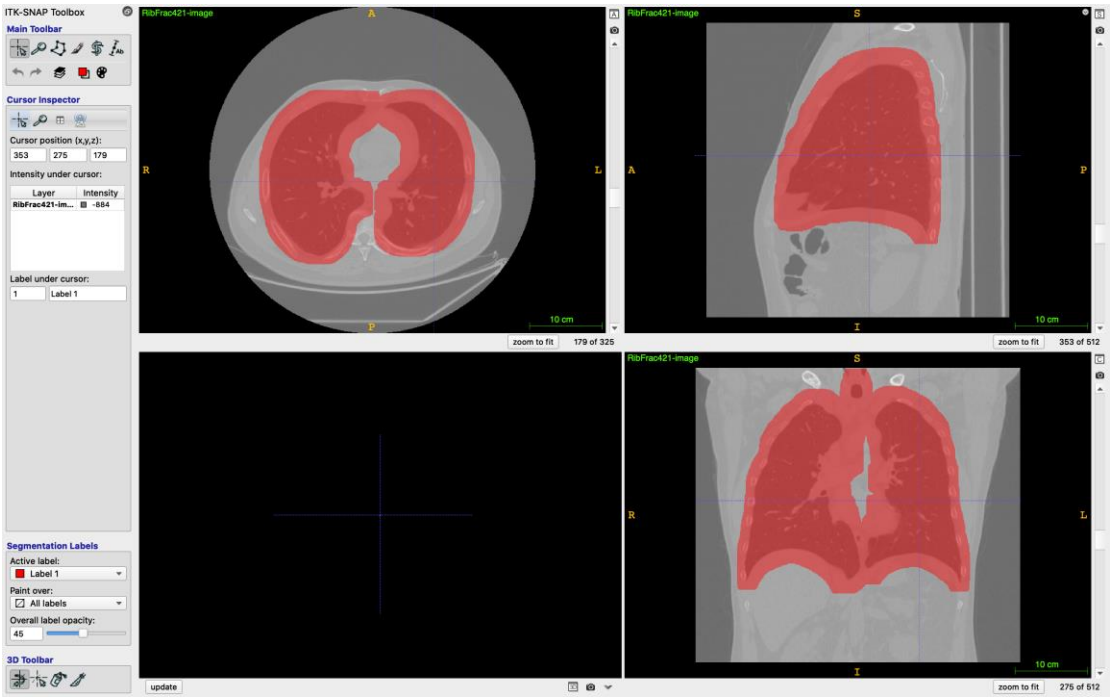

图 11 肺实质在膨胀后得到的结果

3. 实验

3.1 调参细节

几个重要的参数：学习率、batch size、每张图片提取出的样本数、后处理时去掉小区域部分的阈值、肺实质向周围膨胀的像素值。

学习率经过从 1e-1 到 1e-4 进行调整，发现效果最好的是 1e-3。

本实验在 4 张 Titan Xp 上运行，所以最终设置的 batch size 是 8，每张图片提取出的样本数是 16 .

我们观察了预测出来的骨折区域的大小，发现大部分假阳性的骨折区域大小均小于 700.所以我们最终把所有体积小于 700voxels 的骨折区域全部滤除。

我们尝试了 15、20、25、30 四种膨胀的像素值，发现 30 效果更好，并且经过可视化观察，没有触碰到锁骨区域。所以最终采用了 30。

3.2 实验指标

在 validation 上进行预测，然后调用 ribfrac challenge 中的 evaluation 函数进行评判。得到的最好的结果为 froc 为 57.45

如图：

```
============================================================
Recall at key FP
          FP=0.5      FP=1      FP=2      FP=4      FP=8
Recall  0.382184  0.489655  0.593487  0.692209  0.714943
Average recall: 0.5745
Maximum recall: 0.7149
Average FP per scan at maximum recall: 6.5125
```

参考文献

[1] 3D Deep Learning on Medical Images: A Review. https://arxiv.org/abs/2004.00218.

[2] The RibFrac challenge. https://ribfrac.grand-challenge.org.

[3] 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. https://arxiv.org/abs/1606.06650.

[4] Deep-learning-assisted detection and segmentation of rib fractures from CT scans: Development and validation of FracNet. https://www.thelancet.com/journals/ebiom/article/PIIS2352-3964(20)30482-5/fulltext.

[5] PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. https://arxiv.org/abs/1612.00593.

[6] TorchIO Library. https://torchio.readthedocs.io/transforms/augmentation.html#.

[7] Focal Loss for Dense Object Detection. https://arxiv.org/abs/1708.02002.

[8] Gradient Harmonized Single-stage Detector. https://arxiv.org/abs/1811.05181.

[9] Discriminative Localization in CNNs for Weakly-Supervised Segmentation of Pulmonary Nodules. https://arxiv.org/abs/1707.01086.