目录

1 项目概览

1.1 项目说明

本项目结合目前全球发病率最高的肺癌早期诊断方法开展研究，运用先进的人工智能方法进行基于肺部 CT 图像的自动识别技术及诊断技术，确定早期肺癌的病灶，为患者提供早期治愈的机会。

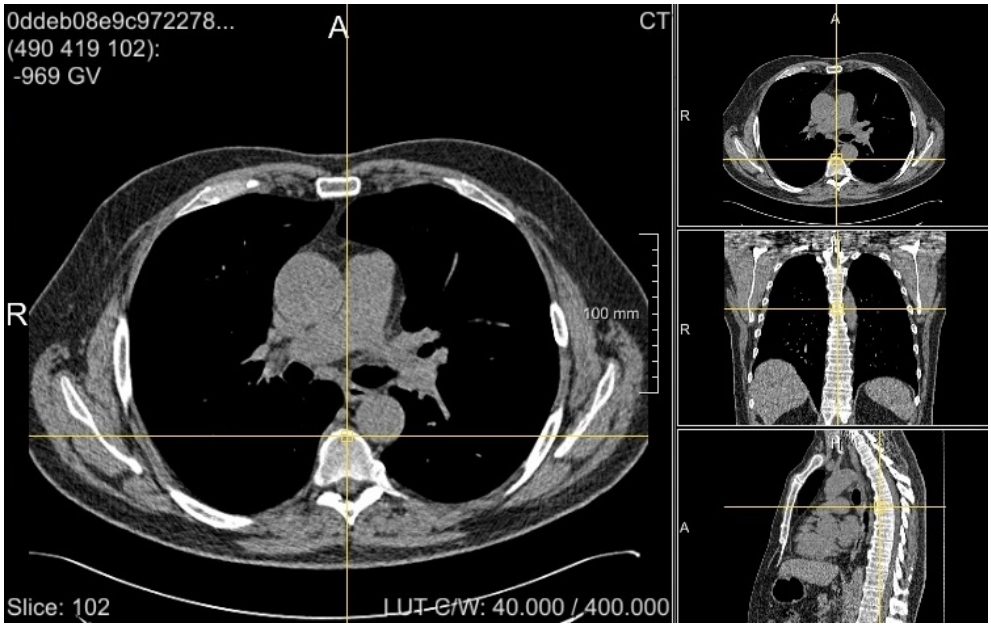

图 1: 肺结节检测

肺结节检测，假阳性降低和肺结节分割是计算机辅助分析肺部 CT 图像的三个代表性任务，在传统医学图像领域已经提出了针对每个任务的具体方法，而基于深度学习的方法近来受到广泛青睐。但是，训练深度学习模型来分别解决每个子任务很可能得不到最优解（资源密集型任务，且无法共享每个环节的特征图谱）。本项目提出了一种新的端到端的 3D 卷积神经网络，以多任务方式联合解决检测、降低假阳性和分割三个任务。

1.2 项目依赖

- 训练环境：Linux version 4.4.0-176-generic, gcc version 5.4.0, Ubuntu 16.04
- 显卡使用：TITAN Xp × 6, 72 GB
- 训练框架：pytorch 1.5.0
- 数据来源：LIDC-IDRI

表 1: LIDC-IDRI 数据集

Collection Statictics	Concrete infos
数据大小	124 GB
图像类型	CT (computed tomography)
图片数	244527
患者数	1010

2 问题分析

2.1 前人工作整理

人工检测结节需要花费大量时间分析每张切片（slice）来估计结节的形状、尺寸和生长率，这无疑是项耗时且难以做到精确的工作。近年来，利用深度学习研究自动化检测和分割肺结节的方法不断出现，在各个公开数据集诸如 Luna16，LIDC-IDRI 均取得了相对不错的结果。现在领域内最优秀的模型均采用相似的分析流程 [2]。

针对肺结节检测，首先使用 3D RPN（region proposal network）筛选可能的候选（candidate screening），然后通过一个 3D 的分类器（classifier）来降低假阳性。虽然亦有人提出了 1-stage 的检测方案，但实际来看效果不佳，更多的解决方案是 2-stage 的，即使用额外的分类器修正肺结节检测的误差。

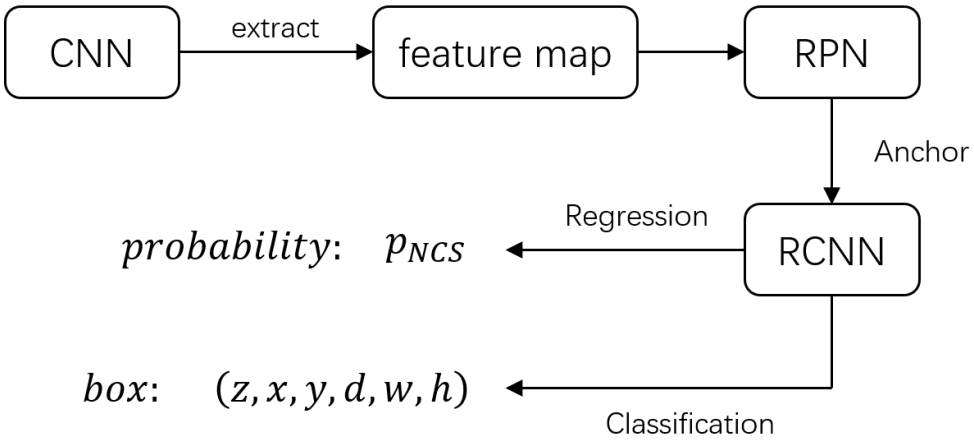

图 2: 训练流程

针对肺结节分割，U-net [3] 和 V-net 等相似结构的网络被证明是目前的最优方案，在上采样的过程中融合网络早期提取的特征图谱，再通过 FCN 进行像素级别的分类，这在小样本医疗影像分割领域取得了很不错成绩。

2.2 问题难点

肺部 CT 图像智能分析的任务有三，分别是：

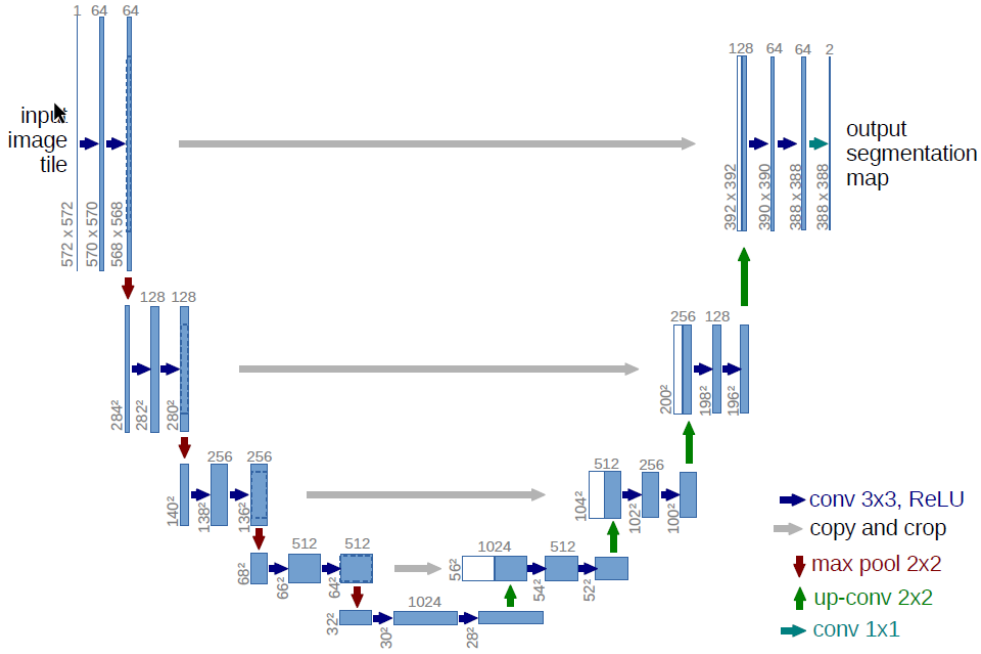

图 3: U net

- 肺结节检测（pulmonary nodule detection）
- 假阳性降低（false positive reduction）
- 肺结节分割（segmentation）

在医学实践中，用于肺结节检测和分割的计算机辅助诊断（CAD）系统通常由几个独立的子系统组成，并分别进行优化

事实上，通过计算机完全独立解决这三个问题存在了很大的限制。首先，训练多个深层次的神经网络既耗时又耗资源，虽然每个子任务的目标需求不同，但它们共享提取表征肺结节特征的通用方法，分开训练会造成极大的资源冗余；其次，独立子系统的表现未必是最佳的，因为分开训练的神经网络阻止了子系统间的通信，它们无法互相学习潜在的特征信息。直觉上看，肺结节分割的掩码（mask）应该为神经网络学习判别特征提供有力的指导，进而可以提高结节检测的性能。

2018 年的论文《On Awakening the Classification Power of Faster RCNN》中已指出，Faster RCNN 的错误情形 [4] 有很多属于假阳性，即误将背景识别为前景物体，或是 nms 无法处理的小框多次识别的情形，原因皆与分类器较弱的分辨能力有关，而感受野的大小一定程度上也影响着分类器的能力。

深层来看，造成这种假阳性的根本原因如下：

- 定位（localization）和分类（classication）共享了所有特征。事实上分类和定位工作是有矛盾的，分类需要学习平移不变量（translation invariant），而定位需要学习平移协变量（translation covariant）

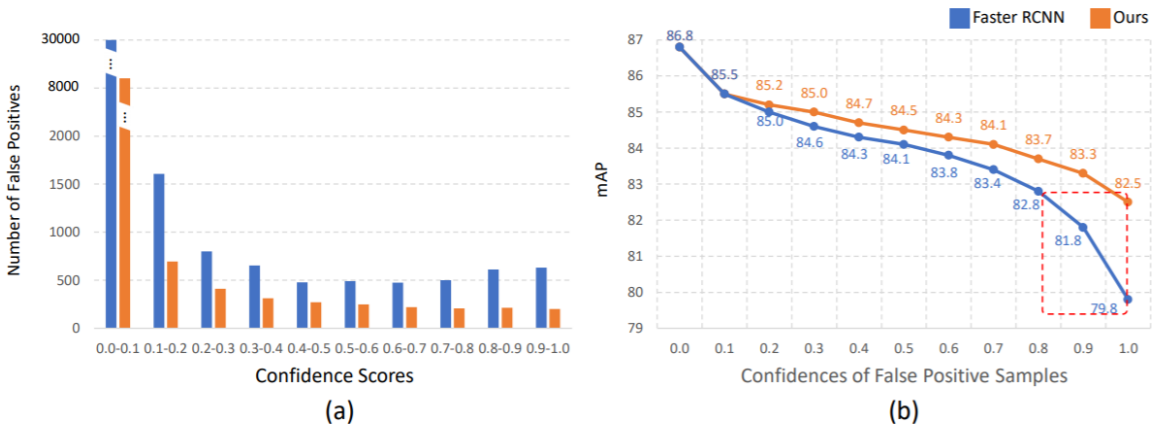

图 4: Faster RCNN 中的假阳性情形

- Faster RCNN 中的分类和回归是联合优化的，这很可能最终优化到两者折中的次优，而对每个子任务来说都不是最优
- Faster RCNN 是直接从特征图谱上获取感受野，这使得特征图上每个点的视野域都很大，每个 ROI 几乎覆盖了整张图片，这引入了大量的背景（负样本）信息

原文的作者认为多任务学习（MTL）无法充分挖掘新的信息，因此提出将分类与定位完全解耦 [5]，并更改 RCNN 感受野的提取位置，让其尽量不包含多余的背景信息。实验证明这种做法的确提高了一定的精度，但一定程度上牺牲了速度，且针对负样本的舍去问题也很值得商榷。

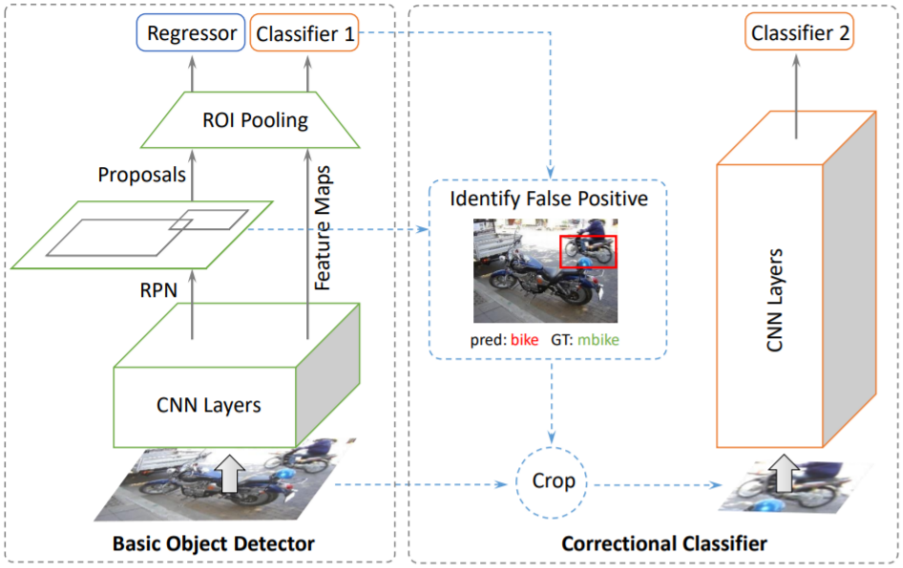

图 5: 完全解耦方案

尽管关于多任务学习（MTL）和特征共享的探讨提供了一种新的解决思路，但是单纯的

实现可能会引起其他问题。首先，由于定位和分类的目标不匹配，如果使用相同的特征图谱执行这两个任务可能不是最佳选择，但完全解耦又会牺牲速度；其次，过大的感受野（ROI）会整合图像中过多的背景信息，这在 3D 图像中尤甚，会对根瘤尤其是小根瘤的分类造成严重的干扰。因此，解耦定位和分类显得至关重要，如何平衡特征共享与解耦成为了很关键的问题。

根据以上分析，我们提出了一种新的部分解耦的假阳性降低方案来解决这个问题，在共享特征提取流程的同时改变不同子任务特征图谱的选择。

3 模型架构与分析

3.1 Backbone

我们基于 pytorch 完成了整个模型的搭建，训练流程图如 [6] 所示。

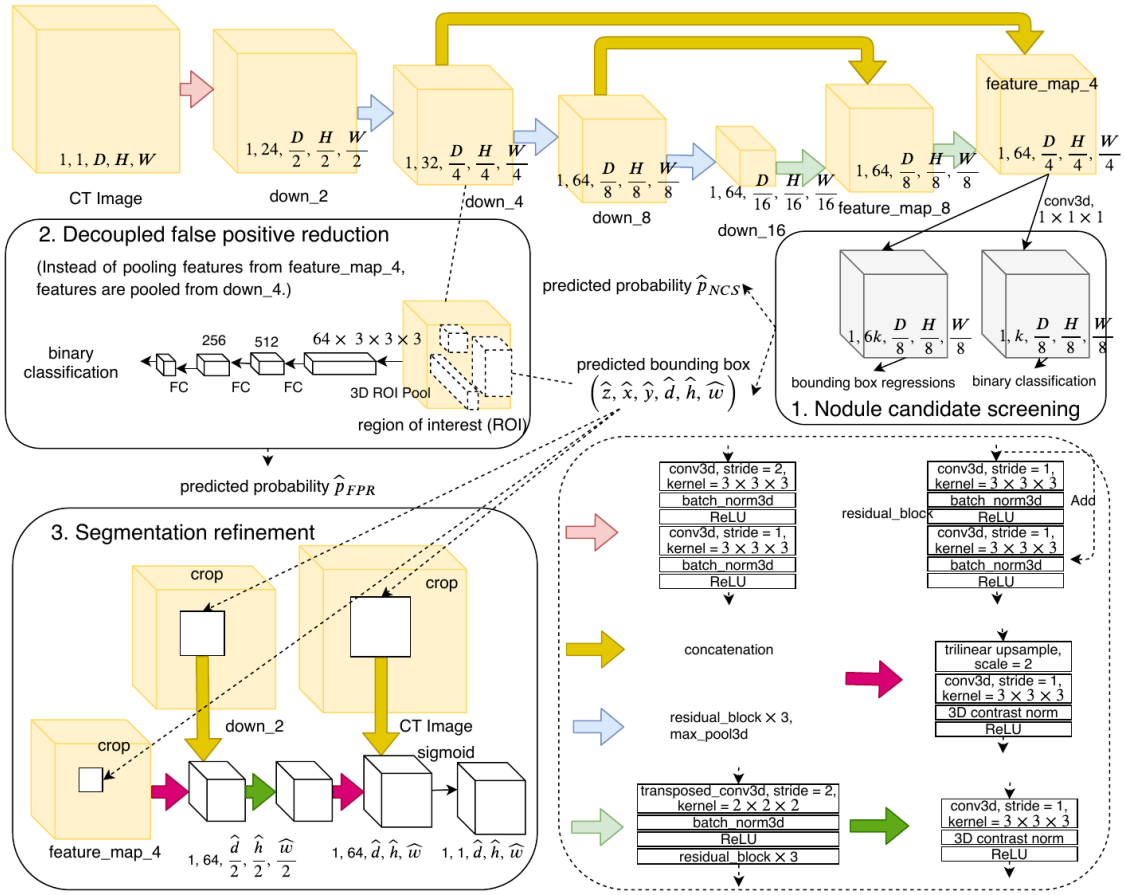

图 6: 模型概览

```
class NoduleNet(nn.Module):
    def __init__(self, cfg, mode='train'):
        super(NoduleNet, self).__init__()
        self.cfg = cfg
        self.mode = mode
```

```
        self.feature_net = FeatureNet(config, 1, 128)
        self.rpn = RpnHead(config, in_channels=128)
        self.rcnn_head = RcnnHead(config, in_channels=64)
        self.rcnn_crop = CropRoi(self.cfg, cfg['rcnn_crop_size'])
        self.mask_head = MaskHead(config, in_channels=128)
```

我们选用 U-net 作为 backbone 实现特征提取，经多层卷积和特征融合后得到特征图 feature-map-4 用以接下来的候选筛选工作。

```
class FeatureNet(nn.Module):
    def __init__(self, config, in_channels, out_channels):
        super(FeatureNet, self).__init__()
        self.preBlock = nn.Sequential(
            nn.Conv3d(in_channels, 24, kernel_size = 3, padding = 1, stride=2),
            nn.BatchNorm3d(24, momentum=bn_momentum),
            nn.ReLU(inplace = True),
            nn.Conv3d(24, 24, kernel_size = 3, padding = 1),
            nn.BatchNorm3d(24, momentum=bn_momentum),
            nn.ReLU(inplace = True))

        self.forw1 = nn.Sequential(
            ResBlock3d(24, 32),
            ResBlock3d(32, 32))

        self.forw2 = nn.Sequential(
            ResBlock3d(32, 64),
            ResBlock3d(64, 64))

        self.forw3 = nn.Sequential(
            ResBlock3d(64, 64),
            ResBlock3d(64, 64),
            ResBlock3d(64, 64))

        self.forw4 = nn.Sequential(
            ResBlock3d(64, 64),
            ResBlock3d(64, 64),
            ResBlock3d(64, 64))

        # skip connection in U-net
        self.back2 = nn.Sequential(
            # 64 + 64 + 3, where 3 is the channeld dimension of coord
            ResBlock3d(128, 128),
            ResBlock3d(128, 128),
            ResBlock3d(128, 128))

        # skip connection in U-net
        self.back3 = nn.Sequential(
            ResBlock3d(128, 64),
            ResBlock3d(64, 64),
            ResBlock3d(64, 64))

        self.maxpool1 = nn.MaxPool3d(kernel_size=2, stride=2, return_indices=True)
        self.maxpool2 = nn.MaxPool3d(kernel_size=2, stride=2, return_indices=True)
        self.maxpool3 = nn.MaxPool3d(kernel_size=2, stride=2, return_indices=True)
        self.maxpool4 = nn.MaxPool3d(kernel_size=2, stride=2, return_indices=True)

        # upsampling in U-net
        self.path1 = nn.Sequential(
            nn.ConvTranspose3d(64, 64, kernel_size=2, stride=2),
            nn.BatchNorm3d(64),
            nn.ReLU(inplace=True))
```

```
        # upsampling in U-net
        self.path2 = nn.Sequential(
            nn.ConvTranspose3d(64, 64, kernel_size=2, stride=2),
            nn.BatchNorm3d(64),
            nn.ReLU(inplace=True))
```

3.2 肺结节候选筛选 (Nodule candidate screening)

与传统 RCNN 相仿，首先让 feature-map-4 经过一个大小为 1 x 1 x 1 的卷积核降低通道数，再由 RPN 生成候选锚框和坐标信息，经训练同时执行分类和回归两个子任务，得到预测的类别概率和锚框坐标及尺寸信息。

```
class RpnHead(nn.Module):
    def __init__(self, config, in_channels=128):
        super(RpnHead, self).__init__()
        self.drop = nn.Dropout3d(p=0.5, inplace=False)
        self.conv = nn.Sequential(nn.Conv3d(in_channels, 64, kernel_size=1),
                                  nn.ReLU())
        self.logits = nn.Conv3d(64, 1 * len(config['anchors']), kernel_size=1)
        self.deltas = nn.Conv3d(64, 6 * len(config['anchors']), kernel_size=1)
```

```
class RcnnHead(nn.Module):
    def __init__(self, cfg, in_channels=128):
        super(RcnnHead, self).__init__()
        self.num_class = cfg['num_class']
        self.crop_size = cfg['rcnn_crop_size']

        self.fc1 = nn.Linear(in_channels * self.crop_size[0] * self.crop_size[1] * self.
                                                    crop_size[2], 512)
        self.fc2 = nn.Linear(512, 256)
        self.logit = nn.Linear(256, self.num_class)
        self.delta = nn.Linear(256, self.num_class * 6)
```

3.3 假阳性降低 (Decoupled false positive reduction)

我们采取部分解耦的思路来解决假阳性过高的问题，即在保证共享特征提取流程的前提下改变分类和定位两个子任务选择的特征图。我们选用网络早期的特征图 down-4（U-net 结构中它和 feature-map-4 保存了相同的语义信息），使用 3D 的 ROI 池化层从 down-4 上提取特征。

显然，由于 down-4 保留了更多的浅层特征，相对于深层特征图来说，ROI 会生成相对更小的感受野（receptive field），其中包含的背景信息也会更少，降低了后续分类器的学习难度。同时，我们也整合了从 feature-map-4 上学习的特征，如此既避免了分开训练造成的信息隔离，又实现了部分解耦，让定位和分类这两个子任务学到了不同特征。

```
class CropRoi(nn.Module):
    def __init__(self, cfg, rcnn_crop_size):
        super(CropRoi, self).__init__()
        self.cfg = cfg
        self.rcnn_crop_size  = rcnn_crop_size
        self.scale = cfg['stride']
```

```
        self.DEPTH, self.HEIGHT, self.WIDTH = cfg['crop_size']

    def forward(self, f, inputs, proposals):
        self.DEPTH, self.HEIGHT, self.WIDTH = inputs.shape[2:]
        crops = []
        for p in proposals:
            b = int(p[0])
            center = p[2:5]
            side_length = p[5:8]
            c0 = center - side_length / 2 # left bottom corner
            c1 = c0 + side_length # right upper corner
            c0 = (c0 / self.scale).floor().long()
            c1 = (c1 / self.scale).ceil().long()
            minimum = torch.LongTensor([[0, 0, 0]]).cuda()
            maximum = torch.LongTensor(
                np.array([[self.DEPTH, self.HEIGHT, self.WIDTH]]) / self.scale).cuda()

            c0 = torch.cat((c0.unsqueeze(0), minimum), 0)
            c1 = torch.cat((c1.unsqueeze(0), maximum), 0)
            c0, _ = torch.max(c0, 0)
            c1, _ = torch.min(c1, 0)

            # Slice 0 dim, should never happen
            if np.any((c1 - c0).cpu().data.numpy() < 1):
                print(p)
                print('c0:', c0, ', c1:', c1)
            crop = f[b, :, c0[0]:c1[0], c0[1]:c1[1], c0[2]:c1[2]]
            crop = F.adaptive_max_pool3d(crop, self.rcnn_crop_size)
            crops.append(crop)

        crops = torch.stack(crops)
        return crops
```

3.4 肺结节分割（Segmentation）

与传统 Mask-RCNN 相仿，结合前两个任务的结果，针对 feature-map-4，将含有结节的候选框裁剪出来上采样至 down-2 和原始 CT 图像两个区域进行特征融合，经过 FCN 进行像素级别的分类，最终实现分割效果。由于融合了浅层特征，分割的精度也有所提高；同时由于我们仅对含有结节的区域进行上采样，大大提高了训练速度。

```
class MaskHead(nn.Module):
    def __init__(self, cfg, in_channels=128):
        super(MaskHead, self).__init__()
        self.num_class = cfg['num_class']

        self.up1 = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='trilinear'),
            nn.Conv3d(in_channels, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))

        self.up2 = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='trilinear'),
            nn.Conv3d(in_channels, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))
```

```
        self.up3 = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='trilinear'),
            nn.Conv3d(64, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))

        self.back1 = nn.Sequential(
            nn.Conv3d(128, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))

        self.back2 = nn.Sequential(
            nn.Conv3d(96, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))

        self.back3 = nn.Sequential(
            nn.Conv3d(65, 64, kernel_size=3, padding=1),
            nn.InstanceNorm3d(64, momentum=bn_momentum, affine=affine),
            nn.ReLU(inplace = True))
```

3.5 优化与再创新

3.5.1 样本均衡与假阳性问题

我们知道假阳性的出现与正负样本不均衡有关，无论是单纯的数据集中存在过多的负样本，还是 ROI 时过大的感受野，都会使得网络学习过多不必要的特征，进而降低分类器的准确率。虽然减小感受野一定程度上能减轻假阳性的情况，但终究未能充分利用负样本的潜在信息。

一般来说，传统解决正负样本不均匀的方式是重采样，使用数据清洗、数据生成的方法均衡正负样本及前背景的相对比例，但这样做同样没有充分利用整个数据集而不能达到更高的准确率。因此，不基于重采样的样本均衡解决方案被逐一提出。本项目采用了 Retina Net 提出的 focal loss [7] 解决样本均衡问题以进一步降低假阳性。

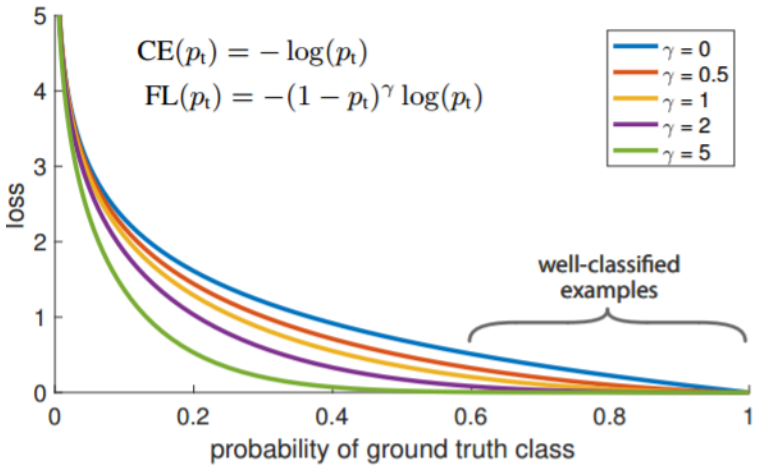

图 7: focal loss

focal loss 的核心在于修正了标准交叉熵对于正负样本的权重问题。标准交叉熵的权值均为 1，即训练过程中不同样本对于损失函数的影响是相同的。而 focal loss 修正了权重分配。

$$L_f = \begin{cases} -\alpha(1-\hat{y})^\gamma \log \hat{y} & y = 1 \\ -(1-\alpha)\hat{y}^\gamma \log(1-\hat{y}) & y = 0 \end{cases}$$

γ 降低了强正、负类样本的分类难度，减少了简单样本的影响，使得模型更关注于困难的、错分的样本（预测概率在 0.5 左右）。同时，α 平衡了正负样本的数量均衡问题，最终提高了前背景的分类精度。我们的模型参考了 Retina Net，选取 $\alpha = 0.25$ 和 $\gamma = 2$ 来训练模型。

```
# focal loss
def weighted_focal_loss_for_cross_entropy(logits, labels, weights, gamma=2.):
    log_probs = F.log_softmax(logits, dim=1).gather(1, labels)
    probs     = F.softmax(logits, dim=1).gather(1, labels)
    probs     = F.softmax(logits, dim=1).gather(1, labels)

    loss = - log_probs * (1 - probs) ** gamma
    loss = (weights * loss).sum()/(weights.sum()+1e-12)

    return loss.sum()
```

3.5.2 多尺度检测

为充分挖掘基于 U-net 的多尺度融合的能力，我们更换了 RPN 接受的特征图来源，选择续接一个 FPN（feature pyramid net）[8] 进一步融合浅层和深层特征生成特征金字塔，针对不同大小的锚框选用不同尺度的特征图进行后续训练，最终使得无论是大结节还是小结节均有较高的识别和分割精度。

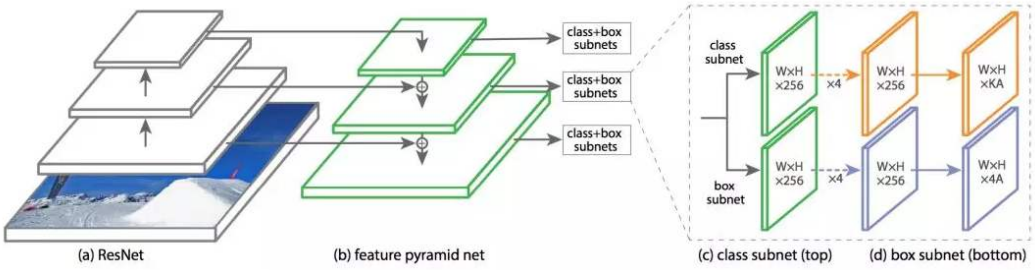

图 8: FPN

```
class PyramidFeatures(nn.Module):
    def __init__(self, C3_size, C4_size, C5_size, feature_size=256):
        super(PyramidFeatures, self).__init__()

        # upsample C5 to get P5 from the FPN paper
        self.P5_1 = nn.Conv2d(C5_size, feature_size, kernel_size=1, stride=1, padding=0)
        self.P5_upsampled = nn.Upsample(scale_factor=2, mode='nearest')
        self.P5_2 = nn.Conv2d(feature_size, feature_size, kernel_size=3, stride=1, padding=1)

        # add P5 elementwise to C4
```

```
        self.P4_1 = nn.Conv2d(C4_size, feature_size, kernel_size=1, stride=1, padding=0)
        self.P4_upsampled = nn.Upsample(scale_factor=2, mode='nearest')
        self.P4_2 = nn.Conv2d(feature_size, feature_size, kernel_size=3, stride=1, padding=1)

        # add P4 elementwise to C3
        self.P3_1 = nn.Conv2d(C3_size, feature_size, kernel_size=1, stride=1, padding=0)
        self.P3_2 = nn.Conv2d(feature_size, feature_size, kernel_size=3, stride=1, padding=1)

        # "P6 is obtained via a 3x3 stride-2 conv on C5"
        self.P6 = nn.Conv2d(C5_size, feature_size, kernel_size=3, stride=2, padding=1)

        # "P7 is computed by applying ReLU followed by a 3x3 stride-2 conv on P6"
        self.P7_1 = nn.ReLU()
        self.P7_2 = nn.Conv2d(feature_size, feature_size, kernel_size=3, stride=2, padding=1)
```

3.6 训练成果

我们在实验室机器上进行了多轮训练和验证，验证示例如 [9] ，与其他方法的对比情况如 [2]

表 2: 结果对比

Approach	# Train	# Test	Consensus	IoU(%)	DSC(%)
Deeplung, 2017	1404	1404	3	null	73.89 ± 3.87
iw-net, 2018	1593	1593	3	55.00 ± 14.00	null
Central focused Net, 2017	350	493	4	71.16 ± 12.22	82.15 ± 10.76
NoduleNet	1131	1131	3	69.98 ± 10.80	81.80 ± 8.65
NoduleNet	1131	712	4	71.85 ± 10.48	83.10 ± 8.85

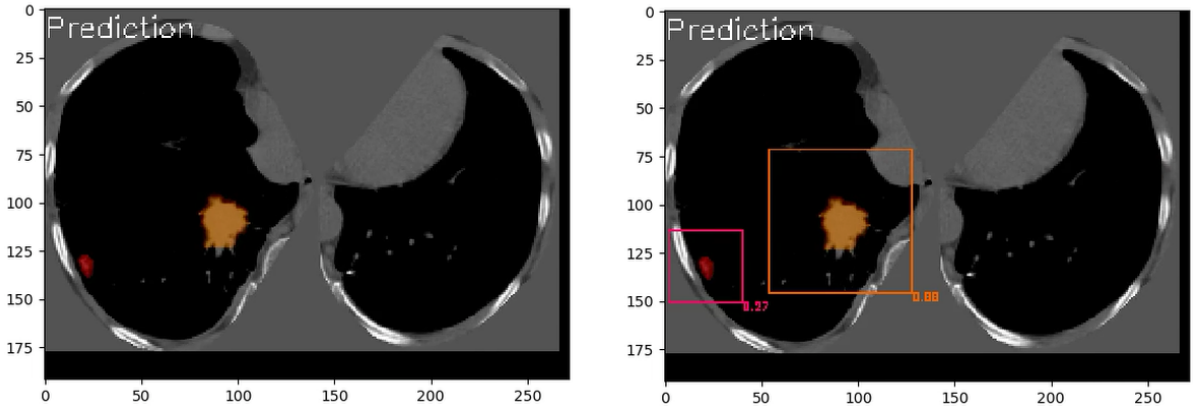

图 9: 测试结果示例

4 项目部署

在完成前期主要工作的基础上，主要采用 pyqt5 框架，搭建可视化应用系统平台。该平

台功能及实现方法如下：

1) 使用vtk的vtkImageViewer2和vtkImagePlaneWidget类分别呈现呈现原始CT文件的二维、三维图像

2) 使用paramiko连接对应服务器，上传目标CT文件并使用网络执行检测，并将结果文件下载到本地

3) 使用matplotlib、cv2以及vtk的vtkImageViewer2可视化肺结节检测结果，将前期网络生成的mask呈现在二维图像上，并附有检测框与目标结节的置信度

4) 将所有的功能汇集，并嵌入设计好的pyqt5框架

检测系统工作流程如 [10] 所示；检测系统界面截图如下，包括医学影像图片的二维与三维呈现 [11]、肺结节检测结果呈现 [12]。

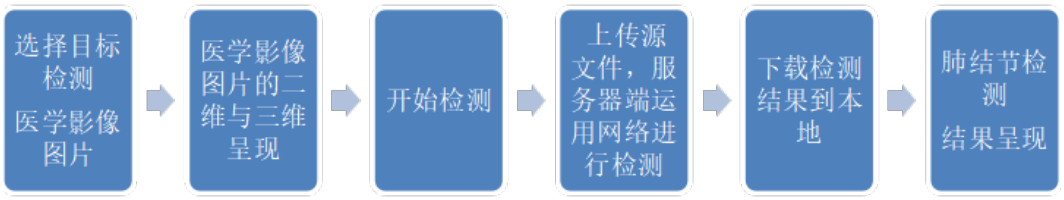

图 10: 检测系统工作流程

5 学习成果与未来展望

在本次实践中我们阅读了大量的论文，丰富了自己关于 cv 和医学图像处理的相关知识，积累了在 Linux 环境下进行深度学习的项目经历，对 2-stage 的 RCNN项目有了更深入的理解。

在具体开发的过程中，学习了vtk、paramiko等开源工具包的使用方法，了解了基本的医学图像格式（如：DICOM、mhd和raw等）的相关知识，掌握了医学图像可视化与图像处理的相关技术，掌握了基于python语言的服务器远程控制技术，实际运用pyqt5与QtDesigner掌握了界面与功能分离的图形界面开发。

在具体实践的过程中，我们也遇到了一些困难，问题主要在于代码具体实现的细节问题。通过这次大创项目实践，我们提升了信息检索能力、对于开源函数库的学习能力以及编程实践能力。

最后，我们的项目将投入智能医学进行肺结节检测和治疗的具体实践中，为计算机辅助治疗提供了新的思路和方法。

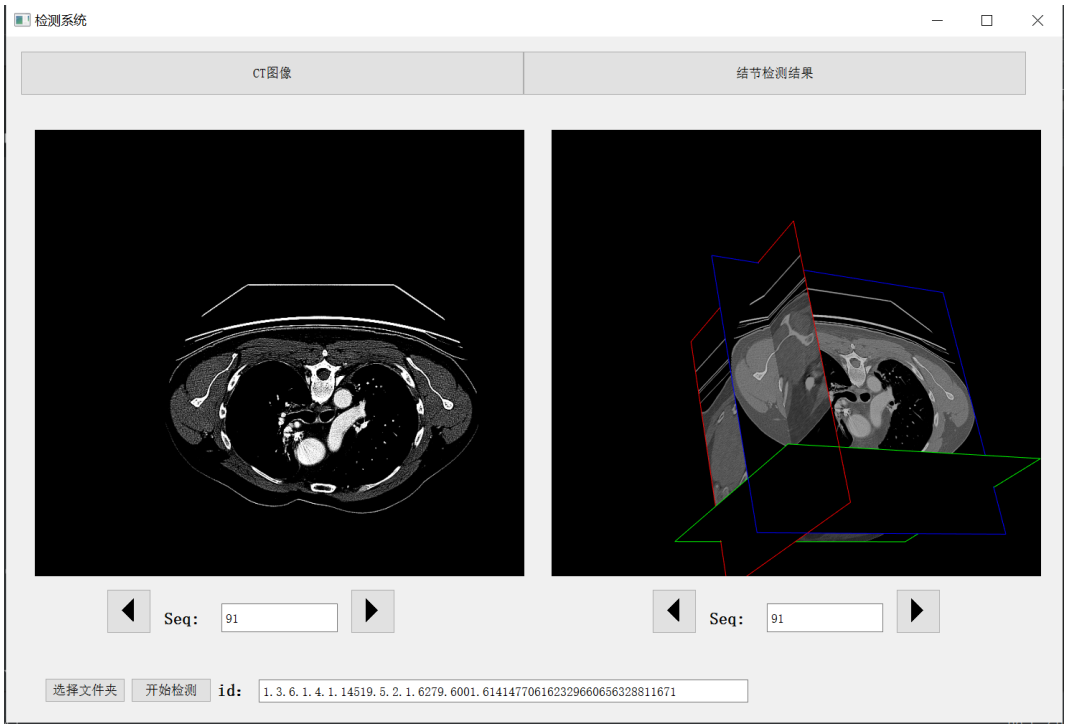

图 11: 肺部 CT 可视化

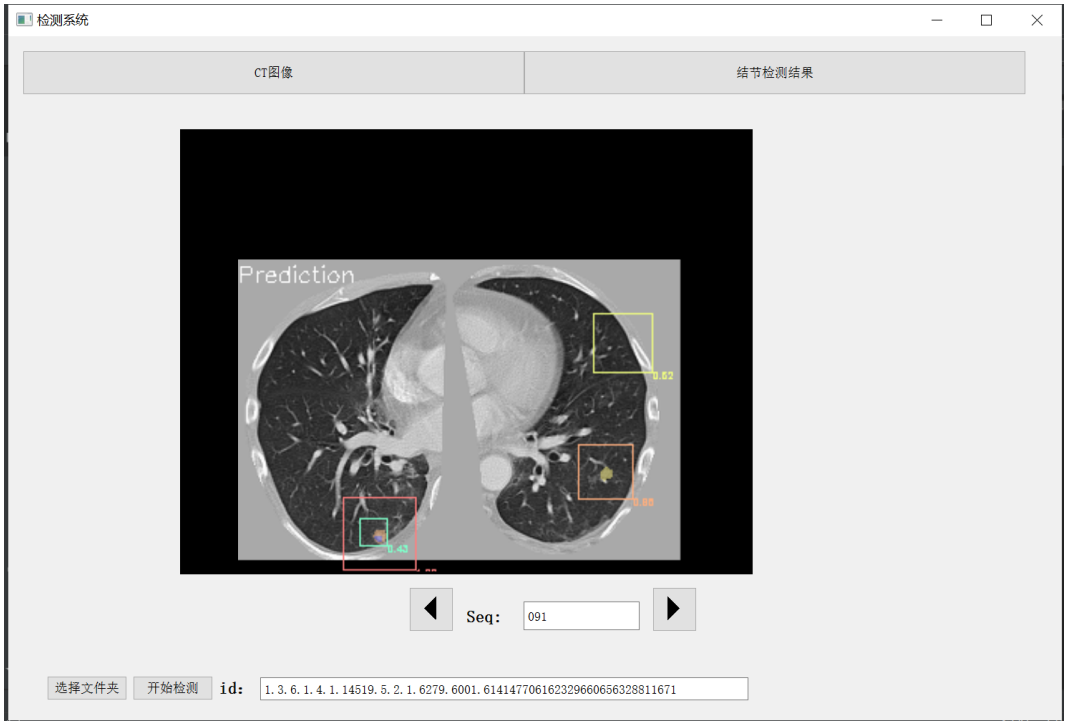

图 12: 肺结节检测