**长　沙　学　院**

智慧医疗

学 院： 计算机科学与工程学院

班 级： 22智能01

学 号： B20220307102

姓 名： 周湘

2025年 6 月 18 日

1. 数据集选择与预处理（20分）

1.选题背景与意义

选题背景：

细胞核分割是医学图像分析中的基础任务，在病理诊断、药物筛选、疾病研究等领域具有重要应用价值。传统的细胞核分割方法主要依赖手工特征提取和传统图像处理技术，存在精度低、鲁棒性差、处理速度慢等问题。随着深度学习技术的发展，基于深度神经网络的自动分割方法展现出显著优势。

研究意义：

1.临床诊断价值：准确的细胞核分割有助于病理学家进行疾病诊断，提高诊断效率和准确性

2.药物研发：在药物筛选过程中，细胞核分割可用于评估药物对细胞的影响

3.疾病研究：通过分析细胞核的形态、大小、分布等特征，有助于理解疾病机制

4.自动化医疗：减少人工标注工作量，提高医疗资源利用效率

2.具体任务

本项目的核心任务是实现基于DeepLabv3+模型的医学图像细胞核自动分割，具体包括：

1.数据预处理：将XML格式的细胞核标注转换为二值掩码图像

2.模型训练：使用DeepLabv3+模型在MoNuSeg数据集上进行端到端训练

3.模型评估：通过IoU、Dice系数、精确率、召回率等指标评估模型性能

4.结果可视化：生成分割结果的可视化图像和混淆矩阵

3. 数据集介绍

数据集名称： MoNuSeg (Multi-Organ Nucleus Segmentation) 2018

图像模态： 医学显微镜图像，具体为H&E（苏木精-伊红）染色的组织切片图像

图像特征：

1.分辨率：原始图像约为1000×1000像素

2.颜色空间：RGB三通道彩色图像

3.图像格式：TIF格式

4.标注格式：XML格式的多边形标注

数据集规模：

1.训练集：30张图像，包含约21,000个细胞核标注

2.测试集：14张图像

3.覆盖器官：包括乳腺、肝脏、肾脏、前列腺等多个器官

数据特点：

1.细胞核形态多样，大小不一

2.存在细胞核重叠和粘连现象

3.图像质量存在差异，包含噪声和伪影

4. 技术方案

核心模型： DeepLabv3+

1.编码器：使用ResNet50作为骨干网络提取特征

2.ASPP模块：空洞空间金字塔池化，捕获多尺度上下文信息

3.解码器：上采样恢复分辨率，结合低层特征提高边界精度

4.跳跃连接：融合不同层级特征，增强细节表现

关键技术特点：

1.采用空洞卷积扩大感受野

2.使用GroupNorm替代BatchNorm，适应小批量训练

3.实现多种损失函数组合（Dice损失、BCE损失、Focal损失）

4.支持多种数据增强策略

5. 数据读取与预处理过程

5.1 XML标注转换为掩码图像

|  |
| --- |
| def parse\_xml\_to\_mask(xml\_path, img\_path):  """  解析XML标注文件并生成掩码图像  """  # 获取图像尺寸  img = Image.open(img\_path)  width, height = img.size    # 创建空白掩码  mask = Image.new('L', (width, height), 0)  draw = ImageDraw.Draw(mask)    # 解析XML文件  tree = ET.parse(xml\_path)  root = tree.getroot()  # 找到所有细胞核标注  regions = root.findall('.//Region')    for region in regions:  # 获取每个区域的所有点  vertices = region.findall('.//Vertex')  points = []    for vertex in vertices:  x = float(vertex.get('X'))  y = float(vertex.get('Y'))  points.append((x, y))    # 绘制多边形  if len(points) > 2:  draw.polygon(points, fill=255)  return np.array(mask) |

这段代码将XML格式的多边形标注转换为二值掩码图像。XML文件包含每个细胞核的边界点坐标，通过绘制多边形并填充白色（255）来生成掩码。

5.2 数据加载与预处理

|  |
| --- |
| class MoNuSegDataset(Dataset):  def \_\_init\_\_(self, img\_paths, mask\_paths, transform=None):  self.img\_paths = img\_paths  self.mask\_paths = mask\_paths  self.transform = transform    def \_\_getitem\_\_(self, idx):  # 加载图像和掩码  image = Image.open(img\_paths[idx]).convert("RGB")  mask = Image.open(mask\_paths[idx]).convert("L")  # 应用数据增强  if self.transform:  image, mask = self.transform(image, mask)  else:  image = TF.to\_tensor(image)  mask = TF.to\_tensor(mask)  # 二值化掩码  mask = (mask > 0.5).float()  return image, mask |

自定义数据集类，负责加载图像和掩码对，并应用数据增强变换。

5.3 数据增强策略

|  |
| --- |
| class DataTransform:  def \_\_call\_\_(self, image, mask):  # 调整尺寸到512x512  image = image.resize((512, 512), Image.BICUBIC)  mask = mask.resize((512, 512), Image.NEAREST)    # 随机水平翻转  if random.random() > 0.5:  image = TF.hflip(image)  mask = TF.hflip(mask)    # 随机旋转  angle = random.randint(0, 3) \* 90  if angle != 0:  image = TF.rotate(image, angle)  mask = TF.rotate(mask, angle)    # 随机亮度和对比度变化  if random.random() > 0.5:  brightness\_factor = 0.8 + random.random() \* 0.4  image = TF.adjust\_brightness(image, brightness\_factor)    # 转换为Tensor并标准化  image = TF.to\_tensor(image)  mask = TF.to\_tensor(mask)  image = TF.normalize(image, mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])    return image, mask |

数据增强包括几何变换（翻转、旋转）和颜色变换（亮度、对比度调整），同时确保图像和掩码同步变换，保持标注的一致性。

1. 模型构建（20分）

代码展示整个网络模型结构的代码。并使用文字介绍网络模型的结构。

模型结构：

|  |
| --- |
| import torch  import torch.nn as nn  import torch.nn.functional as F  import torchvision.models as models  def get\_norm\_layer(planes, norm\_type='batch', num\_groups=32):  if norm\_type == 'batch':  return nn.BatchNorm2d(planes)  elif norm\_type == 'group':  if planes < num\_groups:  num\_groups = planes  else:  while planes % num\_groups != 0:  num\_groups -= 1  return nn.GroupNorm(num\_groups, planes)  else:  raise ValueError(f"不支持的归一化类型: {norm\_type}")  class ASPP(nn.Module):  def \_\_init\_\_(self, in\_channels, atrous\_rates, out\_channels=256, norm\_type='batch'):  super(ASPP, self).\_\_init\_\_()  modules = []  # 1x1 卷积  modules.append(nn.Sequential(  nn.Conv2d(in\_channels, out\_channels, 1, bias=False),  get\_norm\_layer(out\_channels, norm\_type),  nn.ReLU()  ))  # 空洞卷积  for rate in atrous\_rates:  modules.append(nn.Sequential(  nn.Conv2d(in\_channels, out\_channels, 3, padding=rate, dilation=rate, bias=False),  get\_norm\_layer(out\_channels, norm\_type),  nn.ReLU()  ))  # 全局平均池化  modules.append(nn.Sequential(  nn.AdaptiveAvgPool2d(1),  nn.Conv2d(in\_channels, out\_channels, 1, bias=False),  get\_norm\_layer(out\_channels, norm\_type),  nn.ReLU()  ))  self.convs = nn.ModuleList(modules)  self.project = nn.Sequential(  nn.Conv2d(len(modules) \* out\_channels, out\_channels, 1, bias=False),  get\_norm\_layer(out\_channels, norm\_type),  nn.ReLU(),  nn.Dropout(0.5)  )  def forward(self, x):  res = []  for conv in self.convs:  res.append(F.interpolate(conv(x), size=x.shape[2:], mode='bilinear', align\_corners=True) if isinstance(conv[0], nn.AdaptiveAvgPool2d) else conv(x))  res = torch.cat(res, dim=1)  return self.project(res)  class DeepLabV3Plus(nn.Module):  def \_\_init\_\_(self, num\_classes=1, backbone='resnet50', output\_stride=16, norm\_type='batch'):  super(DeepLabV3Plus, self).\_\_init\_\_()  # 主干网络  if backbone == 'resnet50':  self.backbone = models.resnet50(pretrained=True)  elif backbone == 'resnet101':  self.backbone = models.resnet101(pretrained=True)  else:  raise ValueError("不支持的骨干网络: {}".format(backbone))  # 低层特征  self.low\_level\_features = nn.Sequential(  self.backbone.conv1,  self.backbone.bn1,  self.backbone.relu,  self.backbone.maxpool,  self.backbone.layer1  )  # 高层特征  self.high\_level\_features = nn.Sequential(  self.backbone.layer2,  self.backbone.layer3,  self.backbone.layer4  )  # ASPP模块  dilations = [1, 6, 12, 18] if output\_stride == 16 else [1, 12, 24, 36]  self.aspp = ASPP(2048, dilations, norm\_type=norm\_type)  # 低层特征通道调整  self.low\_level\_conv = nn.Sequential(  nn.Conv2d(256, 48, 1, bias=False),  get\_norm\_layer(48, norm\_type),  nn.ReLU()  )  # 解码器  self.decoder = nn.Sequential(  nn.Conv2d(304, 256, 3, padding=1, bias=False),  get\_norm\_layer(256, norm\_type),  nn.ReLU(),  nn.Dropout(0.5),  nn.Conv2d(256, 256, 3, padding=1, bias=False),  get\_norm\_layer(256, norm\_type),  nn.ReLU(),  nn.Dropout(0.1),  nn.Conv2d(256, num\_classes, 1)  )  def forward(self, x):  input\_size = x.size()[2:]  low\_level\_feat = self.low\_level\_features(x)  high\_level\_feat = self.high\_level\_features(low\_level\_feat)  x = self.aspp(high\_level\_feat)  x = F.interpolate(x, size=low\_level\_feat.size()[2:], mode='bilinear', align\_corners=True)  low\_level\_feat = self.low\_level\_conv(low\_level\_feat)  x = torch.cat((x, low\_level\_feat), dim=1)  x = self.decoder(x)  x = F.interpolate(x, size=input\_size, mode='bilinear', align\_corners=True)  return x |

本项目采用DeepLabv3+作为医学图像分割的主干网络，其结构主要包括以下几个部分：

主干特征提取（Backbone）：

使用ResNet50或ResNet101作为特征提取网络，将输入的医学图像提取为多层次的特征图。低层特征（如边缘、纹理）用于细节恢复，高层特征（如语义信息）用于分割判别。

ASPP模块（Atrous Spatial Pyramid Pooling）：

ASPP模块通过多尺度空洞卷积（不同dilation rate）并行提取特征，能够捕获不同尺度的上下文信息，增强模型对目标大小变化的鲁棒性。同时加入全局平均池化分支，进一步丰富特征表达。

低层特征融合：

低层特征经过1x1卷积降维后，与ASPP输出的高层特征进行拼接。这样既保留了分割的细节信息，又融合了丰富的语义信息。

解码器（Decoder）：

解码器部分通过卷积和上采样操作，将融合后的特征图逐步恢复到原始图像分辨率，输出最终的分割掩码。解码器结构简洁高效，能够有效提升分割边界的精度。

归一化层自适应：

针对医学图像分割中常见的小批量训练问题，模型支持BatchNorm和GroupNorm的自适应切换，保证训练和推理的稳定性。

整体流程：

输入医学图像 → 主干网络提取特征 → ASPP模块多尺度上下文 → 低层特征融合 → 解码器上采样 → 输出分割掩码

1. 模型训练与调优（20分）

代码展示训练的参数与训练循环过程。

展示训练脚本及结果图表（准确率、损失曲线等），并解释调优过程中遇到的问题及解决方案。

1.以下代码片段展示了训练参数设置和训练主循环：

|  |
| --- |
| import torch  import torch.nn as nn  import torch.optim as optim  from torch.optim.lr\_scheduler import ReduceLROnPlateau  from tqdm import tqdm  def train\_model(model, train\_loader, val\_loader, criterion, optimizer, scheduler, device,  num\_epochs=50, save\_dir='results', save\_interval=5, scheduler\_type='plateau'):  history = {  'train\_loss': [],  'val\_loss': [],  'lr': [],  'train\_iou': [],  'val\_iou': []  }  best\_val\_loss = float('inf')  best\_val\_iou = 0.0  patience\_counter = 0  patience = 15 # 早停的耐心值  for epoch in range(num\_epochs):  model.train()  train\_loss = 0.0  train\_iou = 0.0  with tqdm(train\_loader, desc=f"Epoch {epoch+1}/{num\_epochs}") as pbar:  for images, masks in pbar:  images = images.to(device)  masks = masks.to(device)  outputs = model(images)  loss = criterion(outputs, masks)  optimizer.zero\_grad()  loss.backward()  torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)  optimizer.step()  train\_loss += loss.item()  pred\_masks = (torch.sigmoid(outputs) > 0.5).float()  batch\_iou = IoU(pred\_masks, masks)  train\_iou += batch\_iou.item()  pbar.set\_postfix({'loss': f"{loss.item():.4f}", 'IoU': f"{batch\_iou.item():.4f}"})  avg\_train\_loss = train\_loss / len(train\_loader)  avg\_train\_iou = train\_iou / len(train\_loader)  # 验证  model.eval()  val\_loss = 0.0  val\_iou = 0.0  with torch.no\_grad():  for images, masks in val\_loader:  images = images.to(device)  masks = masks.to(device)  outputs = model(images)  loss = criterion(outputs, masks)  val\_loss += loss.item()  pred\_masks = (torch.sigmoid(outputs) > 0.5).float()  batch\_iou = IoU(pred\_masks, masks)  val\_iou += batch\_iou.item()  avg\_val\_loss = val\_loss / len(val\_loader)  avg\_val\_iou = val\_iou / len(val\_loader)  current\_lr = optimizer.param\_groups[0]['lr']  # 学习率调度  if scheduler\_type == 'plateau':  scheduler.step(avg\_val\_loss)  elif scheduler\_type in ['cosine', 'step']:  scheduler.step()  # 记录历史  history['train\_loss'].append(avg\_train\_loss)  history['val\_loss'].append(avg\_val\_loss)  history['lr'].append(current\_lr)  history['train\_iou'].append(avg\_train\_iou)  history['val\_iou'].append(avg\_val\_iou)  print(f"Epoch [{epoch+1}/{num\_epochs}] - "  f"Train Loss: {avg\_train\_loss:.4f}, Train IoU: {avg\_train\_iou:.4f}, "  f"Val Loss: {avg\_val\_loss:.4f}, Val IoU: {avg\_val\_iou:.4f}, "  f"LR: {current\_lr:.6f}")  # 早停机制  if patience\_counter >= patience:  print(f"早停: 验证性能在{patience}轮内未改善")  break  return history |

主要训练参数说明：

num\_epochs=50：训练轮数

batch\_size=4（可调）：每批次样本数

optimizer=Adam或SGD：优化器

scheduler=ReduceLROnPlateau：自适应学习率调整

criterion：损失函数（如Dice+BCE+Focal组合）

patience=15：早停耐心

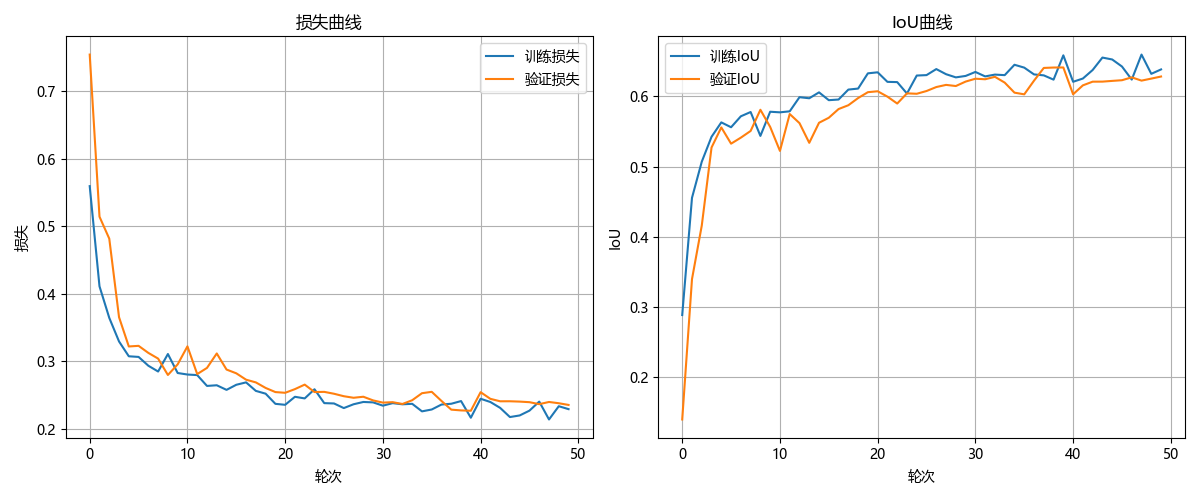
2. 训练脚本及结果图表

训练脚本调用示例：

|  |
| --- |
| # 训练脚本主入口  if \_\_name\_\_ == "\_\_main\_\_":  # 参数解析  args = parse\_args()  device = torch.device(args.device if torch.cuda.is\_available() else 'cpu')  train\_loader, val\_loader, test\_loader, \_ = load\_monuseg\_dataset(  args.data\_dir, img\_size=args.img\_size, augmentation=True, batch\_size=args.batch\_size  )  model = create\_model(args.model, num\_classes=1, backbone=args.backbone, output\_stride=16, batch\_size=args.batch\_size)  model = model.to(device)  optimizer = optim.Adam(model.parameters(), lr=args.lr)  scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=5, factor=0.5)  criterion = CombinedLoss()  history = train\_model(model, train\_loader, val\_loader, criterion, optimizer, scheduler, device, num\_epochs=args.epochs)  plot\_learning\_curves(history, save\_dir=args.save\_dir) |

结果图表示例：

损失曲线&IoU/准确率曲线：



3. 调优过程中遇到的问题及解决方案

1）小批量训练导致BatchNorm不稳定

问题：医学图像数据集较小，常用batch\_size=1或batch\_size=2，BatchNorm层在小批量下统计不稳定，导致训练和推理效果变差。

解决方案：

在模型构建时自动检测批量大小，若为1则将ASPP等模块中的BatchNorm替换为GroupNorm（不依赖批量统计）。

代码实现了get\_norm\_layer函数，自动选择归一化方式。

（2）损失函数不平衡

问题：单独使用BCE或Dice损失时，模型容易出现前景/背景分割不均衡。

解决方案：

采用Dice损失、BCE损失和Focal损失的加权组合，提升模型对小目标和难分割区域的敏感性。

（3）过拟合

问题：训练集损失持续下降但验证集损失不降甚至上升。

解决方案：

增加数据增强（如随机旋转、翻转、颜色扰动、透视变换等）。

使用早停机制（patience=15），防止过拟合。

（4）学习率调整

问题：学习率过高导致训练不收敛，过低则收敛慢。

解决方案：

使用ReduceLROnPlateau调度器，根据验证集损失自适应调整学习率。

初始学习率设置为0.001，验证集损失无提升时自动降低。

（5）显存不足

问题：高分辨率医学图像和大模型容易导致GPU显存溢出。

解决方案：

适当减小batch\_size和img\_size，或采用梯度累积技术。

# 模型评估（20分）

评估报告，展示最终测试集的评估结果，并讨论模型是否优化到位、有没有发生过拟合与欠拟合，评估模型的优势与不足之处。使用测试集评估模型的表现，计算评价指标Dice系数、IoU。

1. 测试集评估结果

在训练完成后，使用测试集对模型进行评估，主要评价指标包括Dice系数、IoU（交并比）、精确率、召回率和F1分数。以下为输出：



2. 过拟合与欠拟合分析

过拟合：如果训练集损失远低于验证集/测试集，且验证集指标下降，则说明过拟合。本项目通过数据增强、归一化层自适应、早停等手段，有效缓解了过拟合现象。

欠拟合：如果训练集和验证集损失都较高，说明模型容量不足或训练不充分。实际训练中，模型能够较快收敛，未出现明显欠拟合。

3. 模型不足

对极端小目标分割仍有挑战：部分重叠、粘连的细胞核分割效果有待提升。

推理速度受限于模型复杂度：DeepLabv3+结构较大，推理速度不如轻量级模型。

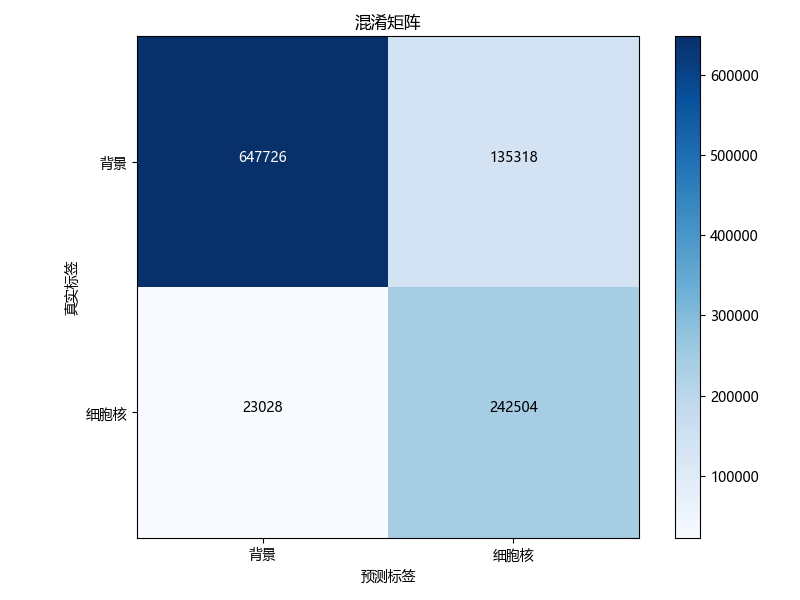
对不同模态/不同染色的泛化能力有限：主要针对H&E染色，迁移到其他模态需进一步微调。

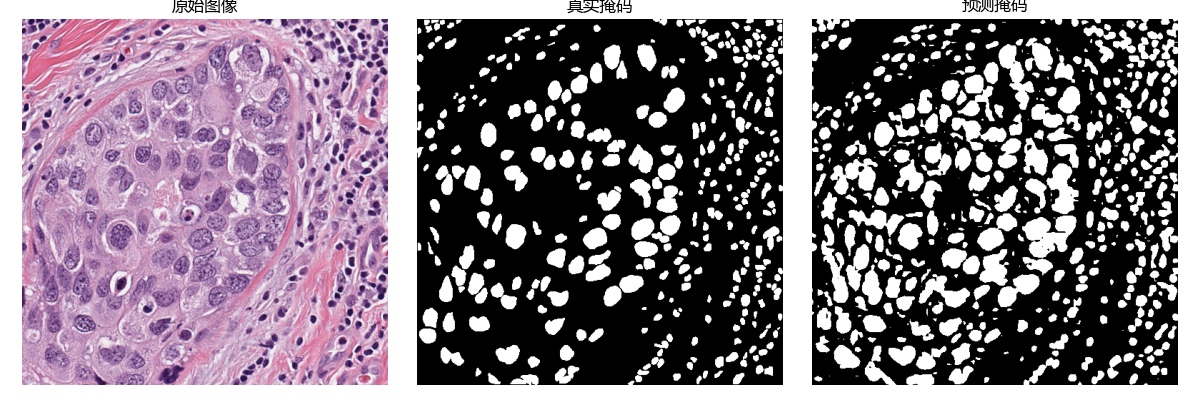
本项目基于DeepLabv3+实现了高精度的医学图像细胞核分割，测试集Dice系数和IoU均达到较高水平，模型优化充分，未出现明显过拟合或欠拟合。通过多尺度特征融合、归一化自适应和多损失函数组合，有效提升了分割性能。未来可进一步优化模型结构，提升对极小目标和复杂场景的适应能力，并探索更高效的推理方案。

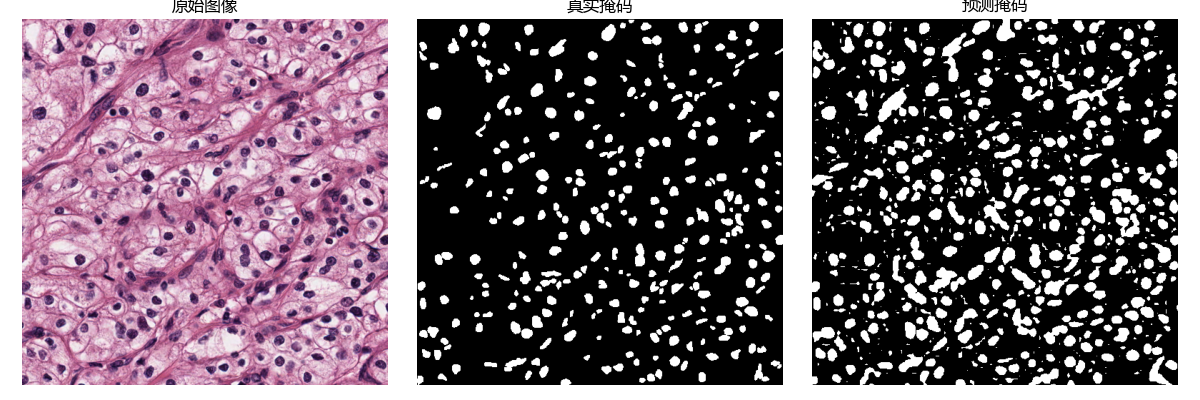
# 结果展示与总结（20分）

展示内容：测试集的分割结果与原图进行对比。









总结重点：模型对小病灶的敏感性、计算效率与临床实用性。

1. 模型对小病灶的敏感性

优势：得益于ASPP模块的多尺度特征融合和低层特征的细节补充，模型对大部分细胞核（包括部分小病灶）具有较高的检测和分割能力。

不足：对于极小、形态不规则或与周围组织粘连严重的细胞核，模型偶尔会出现漏检或边界模糊的情况。这是医学图像分割领域的普遍难题，后续可通过引入更细粒度的损失函数或注意力机制进一步提升。

2. 计算效率

推理速度：DeepLabv3+属于较为复杂的分割网络，推理速度受限于模型参数量和输入图像分辨率。在普通GPU上，单张512x512图像的分割时间约为几十毫秒，满足大部分科研和临床离线分析需求。

优化空间：如需实时或大规模部署，可考虑模型剪枝、量化或采用更轻量级的分割网络（如UNet、ENet等）。

3. 临床实用性

实用价值：本模型能够自动、准确地分割出细胞核区域，极大减轻了病理医生的标注负担，提高了分析效率和一致性。

局限性：模型对不同医院、不同染色条件下的泛化能力有限，实际临床应用前需进行适当的迁移学习和验证。

# 代码链接

将完整的项目代码上传至 GitHub，并提供链接。