**人工智能综合设计课程设计报告**

|  |  |
| --- | --- |
| **作业名称：** | RVSC数据集的心脏医学图像分割 |
| **学 院：** | 计算机学院 |
| **班 级：** | 10042101 |
| **学 号：** | 2021302623 |
| **姓 名：** | 赵雨航 |
| **团队组成：** | 2021301166-刘子炀 2021301931-解晗轲 |

**西北工业大学**

**2023年12月30日**

**1、任务说明（描述所要完成的任务和问题）**

|  |
| --- |
| 心脏图像分割是医学影像处理中的重要任务，旨在识别心脏的各个结构（如心肌、心腔等）并准确地对其进行分割。该模型能够帮助医生诊断心脏病变、进行手术规划以及进行相关医疗研究。这项任务有几个关键方面和潜在问题：  **任务描述：**   1. **图像获取与预处理：** 获取心脏影像，可能是通过MRI、CT等医学成像技术获得的图像数据。预处理包括去噪、标准化、图像配准等步骤，确保数据质量和一致性。 2. **图像分割：** 对心脏图像进行分割，识别和标记出心脏各个结构的边界或区域，比如心室、心房、心肌等。这可能需要借助深度学习模型（如U-Net、Mask R-CNN等）或传统的图像处理技术。 3. **性能评估：** 使用多种指标（如Dice系数、IOU、精确率、召回率等）评估分割模型的性能，确定模型对不同结构的分割效果。   **潜在问题：**   1. **数据质量与标注：** 数据质量对模型训练至关重要，而且精准的标注也是图像分割任务的基础。不准确或者不充分的标注可能会影响模型性能。 2. **模型选择与训练：** 选择适合的模型架构、优化器以及损失函数进行训练。同时，需要充分考虑数据集的大小、类别平衡性等因素。 3. **过拟合与泛化：** 防止模型在训练集上过度拟合，同时确保模型在新数据上的泛化能力，特别是对于来自不同设备或环境的数据。 4. **不规则形状与噪声：** 心脏结构可能存在不规则形状、大小差异、噪声等问题，这可能会增加分割的难度。 5. **实时性与效率：** 在临床应用中，实时性和效率也是重要考虑因素，需要确保模型能够在合理的时间内处理大量图像数据。   小组分工：解晗轲：数据集构建、模型设计：Unet++ U2net CBAM(Unet+C+S) 评价指标设计 代码优化  刘子炀：数据集构建、模型设计：Unet SEAttention（Unet+C）SPA-Unet（Unet+S）评价指标设计 代码优化  赵雨航：数据集构建、评价指标设计 代码优化修改 模型优化 报告撰写 |

**2、开发工具（列出所使用的机器学习平台和相关库）**

|  |
| --- |
| 平台：百度飞桨paddle平台  框架：paddlepaddle  相关库：  pandas: 用于数据处理和分析的Python库，广泛应用于数据清洗、转换和分析。  tqdm: 一个用于在Python中添加进度条的库，用于在循环中显示进度。  logging: Python的标准库，用于记录日志信息，方便程序的调试和追踪。  numpy: 用于科学计算的Python库，提供多维数组和矩阵运算，常用于处理图像、数组数据等。  PIL (Python Imaging Library) / Pillow: 用于图像处理的Python库，能够打开、操作和保存多种格式的图像。  cv2 (OpenCV): OpenCV是一个开源的计算机视觉库，提供了大量图像处理和计算机视觉相关的功能，包括图像处理、特征提取、对象识别等。  pydicom: 用于处理DICOM（医学影像存储和传输标准）格式的Python库，常用于医学影像数据的读取和处理。  matplotlib: 用于绘制图表和数据可视化的Python库，常用于显示图像、绘制图表等。  scipy: 科学计算库，提供了许多数学、科学和工程计算方面的功能，包括图像处理、数值优化、统计等。  paddle.nn：PaddlePaddle中的神经网络模块，提供了构建神经网络的API，包括各种层（如卷积层、池化层等）和激活函数，以及构建模型所需的各种神经网络组件。  paddle.nn.functional:PaddlePaddle中的函数库 包含了一系列的函数，比如激活函数、损失函数等，用于神经网络的构建和优化。  PaddleSeg: 是基于PaddlePaddle的语义分割领域的开发工具库，提供了丰富的语义分割模型、数据集、数据增强方法等。  paddleseg.models: 包含了一系列用于语义分割的模型，比如UNet、UNetPlusPlus，UNet3Plus等，可以直接用于图像分割任务。  paddleseg.transforms: Paddleseg中提供了数据增强的方法，用于增加训练数据的多样性和模型的鲁棒性。  paddleseg.datasets: Paddleseg中的数据管理提供了用于加载和处理不同语义分割数据集的API。  paddleseg.utils, paddleseg.core: Paddle中的工具和核心模块提供了一系列的工具函数、评估指标、日志记录等功能，用于模型训练、评估和推理过程中的辅助操作。  paddleseg.cvlibs: 计算机视觉库,借用 manager 模块，用于管理配置、模型、数据集等方面的功能。 |

**3、设计内容（整体软硬件设计框架）**

|  |
| --- |
| 1、软件设计框架：  （1）数据收集和预处理：  ①数据采集：获取心脏图像数据集，使用RVSC数据集。  ②数据集划分：创建训练集、验证集、测试集所对应的文件夹，创建两个文本文件train\_list.txt和val\_list.txt，分别用于存储训练集和验证集的数据路径，然后遍历所有文件夹，列出图像和标签路径，打乱文件顺序，按照9:1记录训练和验证集的图像路径和对应的标签路径。  ③数据预处理：根据之前标注的路径信息读取图像数据转换为png格式，根据标签信息创建相应的掩膜图像，再次保存路径。  然后使用cv2库对图像进行读取、保存和处理操作，包括灰度转换、阈值化、图像蒙版的创建等。  模型选择与训练：  ④创建用于训练和测试的数据列表文件，方法同②  （2）模型训练  ①选择模型：根据任务需求，选择合适的深度学习模型，如UNet、U2Net，UNet++等，用于图像分割。  ②模型训练：使用训练集对选择的模型进行训练，并进行验证以优化超参数，以达到较好的分割效果，需要指定训练集和验证集的数据集、优化器、保存模型的路径、迭代次数、批量大小、保存模型的间隔、日志间隔、工作进程数量等训练相关的参数。定义损失函数BCELoss()（二元交叉熵损失函数），然后绘制损失值loss下降曲线和重叠指数dice上升曲线。  ③调优与评估：通过评估指标（如IoU、Dice系数）对模型进行评估和调优，确保模型的准确性和泛化能力。此处选择IoU计算预测结果和标签数据之间的交集和并集，并计算交并比。  *IoU*=Union(*A*,*B*)/Intersection(*A*,*B*)​  （3）部署与应用：  ①模型部署：将训练好的模型部署到实际应用环境paddle云端服务器中。  ②应用开发：开发针对心脏图像的应用程序，用于进行图像分割并展示结果。  2、硬件设计框架：  （1）训练和开发环境：  ①GPU服务器：用于训练深度学习模型的高性能GPU服务器（默认GPU），加速模型训练过程。  ②开发设备：用于软件开发和数据处理的计算机，可能包括个人电脑或者笔记本电脑。  （2）模型部署环境（非必要）：  云服务器：如果需要在云端部署模型，可以使用云服务提供商的计算资源。  边缘设备：如果需要在本地部署模型，可能会使用一些嵌入式设备或者边缘计算设备来部署模型并进行图像分割。 |

**4、算法描述（详细描述解决问题的算法原理等）**

|  |
| --- |
| 1、Unet：unet\_model = UNet(num\_classes=2)  U-Net 是一种流行的卷积神经网络架构，用于解决图像分割问题。它的设计结构使其适用于医学图像分割等任务。  U-Net 包含编码器和解码器两部分，它们之间通过跳跃连接（skip connections）进行连接（将编码器中的特征图与解码器中对应的层次连接起来，使得底层和高层特征可以直接传输，有助于更好地保留细节和上下文信息）整体构成一个 U 形的网络结构。  （1）编码器（收缩路径）：  编码器通过卷积和池化操作逐步缩减图像尺寸，并提取图像特征。每次缩减都会增加通道数，以捕获更高级别的特征。  编码器可以理解为特征提取器，它将输入图像转换为低分辨率但高级别的特征表示。  （2）解码器（扩展路径）：  解码器通过上采样和卷积操作逐步恢复图像尺寸，并将特征映射与编码器相对应的层次连接起来，以重建细节和边界信息。  跳跃连接允许网络在解码器阶段使用编码器阶段的特征信息，这有助于更好地保留空间信息和细节。  训练过程：  ①设置可用GPU卡  ②损失函数：通常使用与任务相关的损失函数，如BCE交叉熵损失（对于分类问题）或 Dice 系数（对于图像分割问题）。  ③优化器：选择适当的优化器（如 Adam、SGD 等）以最小化损失函数，本次选择unt\_optimizer，设置了权重衰减  ④训练参数：  使用 train() 函数对 U-Net 模型进行训练，传入了相关参数：  ·model：指定要训练的 U-Net 模型。  ·filename指定保存模型的文件名。  ·dicepath：指定保存 Dice 值（一种评估指标）的文件名。  ·train\_dataset 和 val\_dataset：训练数据集和验证数据集。  ·optimizer：使用之前初始化的优化器进行模型优化。  ·save\_dir：指定保存训练结果的目录。  ·iters、batch\_size、save\_interval 等参数：用于控制训练的迭代次数、批次大小和模型保存间隔等。  ·num\_workers：用于数据加载的并行工作数，一般取0.  ·losses 和 use\_vdl=True：用于设置损失函数和是否使用可视化工具。  ⑤算法原理：  数据预处理：输入图像与对应的标签进行预处理和对齐，以便在网络中进行训练。  特征提取：编码器部分对图像进行特征提取和抽象，逐步减少分辨率并捕获图像中的语义信息。  特征融合：通过跳跃连接，将编码器中提取的特征与解码器中的特征结合，以保留更多细节和空间信息。  特征重建：解码器部分逐步恢复图像尺寸，并将合并的特征映射转换为分割结果。  优化训练：使用损失函数和优化器对模型进行训练，优化网络参数，使其能够准确地预测图像的分割结果。  2、Unet++：unetpp\_model = UNetPlusPlus(num\_classes=2)  U-Net++ 是基于 U-Net 模型的改进版本，旨在进一步改善图像分割的性能。它通过引入多层级的连接来增强 U-Net 中的信息传递和特征重用，以提高分割准确性。  （1）多层级连接：U-Net++ 结构引入了密集连接和多层级连接，以促进特征信息的传播和重用。它在原始的 U-Net 结构中加入了更多的跨层连接，使得不同层级之间的特征可以相互传递，提升了特征的表达和语义信息的传播。  （2）嵌套结构：UNet++ 使用了类似于“嵌套”或“金字塔”结构的设计，即每个下采样和上采样阶段都包含多个分支，每个分支都与前后相邻的多个层级相连接。这种结构可以有效地捕获不同层级的特征和语义信息，提升模型对不同尺度和层次的特征表达能力。  （3）信息传播和融合：UNet++ 在不同层级之间进行信息传播和融合，采用了密集的跳跃连接机制，允许更多层级之间的特征交互和融合。这有助于提升模型对上下文信息的感知和利用，增强了特征的表达能力和语义分割的精度。  （4）更好的特征表示：UNet++ 结构的改进使得网络能够更好地利用不同层级的特征信息，提升了对特征的表示和利用效率。它能够更全面地捕获图像中的语义信息，从而在语义分割任务中取得更好的性能。 训练原理同1，下同  3、U2net：u2net\_model = U2Net(num\_classes=2)  U2Net 是一个专注于高精度图像分割任务的神经网络模型。它基于 U-Net 架构，并使用了较大的网络深度和有效的注意力机制来提高分割性能。U2Net 最初设计用于自然图像和物体边缘的分割，但也适用于其他领域的图像分割任务。  U2Net 的特点：  （1）网络深度：U2Net 模型非常深，并且具有多个级别的特征提取器，这有助于捕获图像中各种尺度和抽象级别的特征。  （2）注意力机制：引入了 U2Net 中的U2NetP模块（也称为U2Net+），利用了U2NetP中的自适应特征选择模块，帮助网络集中注意力于图像中关键区域和特征。  （3）多级特征融合：使用了级联和跳跃连接机制，以便于高效地融合不同层级和不同尺度的特征。  （4）高分辨率分割：U2Net 被设计用于处理高分辨率图像，并在边缘细节、物体轮廓等方面表现出色。  4、SEAttention（Unet+C）：seunet\_model = SEAttention\_UNet()  Unet+C 是指在 U-Net 模型中加入了注意力机制的改进版本。在 U-Net 的基础上，引入了注意力机制，提高了网络在图像分割任务中对重要特征的关注程度，从而提升了模型的性能和准确性。  特点和改进：  （1）引入注意力机制：在 U-Net 结构中，加入了注意力模块，使模型能够在学习过程中自适应地关注和强调重要的特征区域。  （2）增强特征学习：通过注意力机制，模型可以更好地学习到图像中各个位置的重要特征，有助于提高对图像不同区域的感知能力。  （3）提高分割准确性：通过增强对重要特征的关注，Unet+C 在分割任务中能够更准确地识别目标边界、保留细节信息等。  SEAttention原理：  （1）压缩（S-Squeeze）：  在这个步骤中，SEAttention 首先通过全局池化操作，对每个特征通道进行压缩，将每个通道的特征图转换为单个数值。通常使用全局平均池化（Global Average Pooling）或全局最大池化（Global Max Pooling）来实现这一步骤，将每个通道的特征值汇总成一个数。  （2）激励（E-Excitation）：  在激励阶段，SEAttention学习了每个通道重要性的分布。通过引入一个小型的神经网络（通常是全连接层），这个网络被用来学习每个通道的权重。这些权重被应用到压缩阶段得到的特征数值上，以增强重要通道的响应，并抑制不重要通道的响应。通常包括一个或多个全连接层和激活函数（比如 ReLU），最终产生一个通道注意力权重。  5、SPA-Unet（Unet+S）: spaunet\_model = SPA\_UNet()  SPA-Unet模型中的SPA是指Spatial Attention Module（空间注意力模块）。这个模块被设计用于增强神经网络对于空间特征的关注和利用，以提升特征的表达能力。  SPA原理：  （1）初始化：模块接受输入通道数和减少率（reduction）作为参数。  （2）空间注意力处理：模块首先使用了三个不同大小的自适应平均池化（1x1、2x2、4x4）来捕获输入特征图的全局信息。  （3）全连接网络：池化后的特征经过展平处理后，连接成一个较长的特征向量，并通过全连接神经网络进行处理。这个全连接网络包含了两个线性层和ReLU激活函数，最后通过Sigmoid激活函数输出权重。  （4）特征融合：计算得到的权重通过reshape操作和广播乘法应用到输入特征图上，以增强特征图中每个位置的表示能力。  SPA特点：  （1）空间特征加强：SPA模型通过自适应平均池化和全连接网络，对输入的特征图进行处理，以增强特征图中不同位置的表示能力。这有助于网络更加关注重要的空间特征，提升了模型对图像局部和全局信息的感知能力。  （2）位置敏感：SPA模块在处理特征图时，考虑到了不同尺度和大小的感受野。通过对不同尺度的池化操作和特征融合，有利于模型更细致地捕获不同位置的特征信息。  （3）轻量级设计：SPA模块通过少量参数和简单的结构实现了空间注意力的增强效果。使用全局池化和小型的全连接网络，既提升了网络性能，又保持了较低的计算复杂度。  （4）可嵌入性：SPA模块作为一个模块化的部分，可以轻松地集成到其他网络结构中，例如UNet、ResNet等，以增强这些网络对空间特征的关注能力。  6、CBAM(Unet+C+S)：cbamunet\_model = CBAM\_UNet()  CBAM（Convolutional Block Attention Module）是一种综合利用通道注意力（Channel Attention）和空间注意力（Spatial Attention）的注意力机制，用于增强卷积神经网络（CNN）在特征提取和特征重要性学习方面的能力。CBAM 最初提出于论文 "CBAM: Convolutional Block Attention Module" 中。  CBAM 的原理和特点：  （1）通道注意力（Channel Attention）：在 CBAM 中，通道注意力用于建模特征通道之间的相关性。首先，通过全局平均池化（Global Average Pooling，GAP）获取每个通道的全局特征，然后经过两个全连接层，通过学习得到每个通道的权重系数，从而对每个通道进行加权，突出重要的特征通道。  （2）空间注意力（Spatial Attention）：空间注意力用于建模特征图中不同空间位置的相关性。首先，在通道注意力之后，对特征图进行最大池化和平均池化操作，获取每个通道的最大值和平均值特征图。然后，通过堆叠这两种特征图并经过卷积操作，获得空间注意力图。这个注意力图对不同空间位置的重要性进行建模，强调重要的空间区域。  （3）结合通道和空间注意力：CBAM将通道和空间注意力结合在一个模块中，使得网络可以同时考虑通道和空间维度上的特征重要性，从而提升网络对特征的表达能力。  （4）轻量级设计：CBAM的设计相对轻量，可方便地嵌入到不同的神经网络架构中，并且不引入过多的计算开销。它可以作为一个模块，简单地添加到现有的网络结构中，而不会增加太多的复杂度。  （5）通用性：CBAM是通用的注意力机制，适用于各种视觉任务，如图像分类、目标检测和语义分割等。这种通用性使得CBAM可以灵活应用于不同领域的视觉问题中，并取得显著的性能提升。 |

**5、程序实现（核心程序片段）**

|  |
| --- |
| （1）导入基本库： **import** pandas **as** pd  **import** os  **from** tqdm **import** tqdm  **import** logging  **import** numpy **as** np  **from** PIL **import** Image  **import** cv2  **import** pydicom  **import** matplotlib.pyplot **as** plt  **import** scipy.misc  **import** matplotlib.image **as** mpimg  **import** os  **import** numpy **as** np  **import** cv2  **import** pydicom  **import** matplotlib.pyplot **as** plt  **import** scipy.misc  **import** random  （2）设置图片路径  **import** os  **import** numpy **as** np  **import** cv2  **import** pydicom  **import** matplotlib.pyplot **as** plt  **import** scipy.misc  **import** random  **def** **mkdir**(path):    folder = os.path.exists(path)    **if** **not** folder: *#判断是否存在文件夹如果不存在则创建为文件夹*  os.makedirs(path) *#makedirs 创建文件时如果路径不存在会创建这个路径*  print("--- OK ---")    **else**:  print ("--- There is this folder! ---")  **def** **create\_datalist**(data\_path,image\_path,label\_path):  data\_names = os.listdir(image\_path)  random.shuffle(data\_names) *# 打乱数据*  k=0  **with** open(os.path.join(data\_path, 'train\_list.txt'), 'w') **as** tf:  **with** open(os.path.join(data\_path, 'val\_list.txt'), 'w') **as** vf:  **for** i **in** data\_names:  patient\_img=image\_path+'/'+i  label\_img=label\_path+'/'+i  **for** j **in** os.listdir(patient\_img):  *# print(patient\_img+'/'+j)*  *# print(label\_img+'/'+j)*  **if** k%9==0:  vf.write(patient\_img+'/'+j + ' ' + label\_img+'/'+j + '\n')  **else**:  tf.write(patient\_img+'/'+j + ' ' + label\_img+'/'+j + '\n')  k=k+1  **def**  **MakeDataset**(train\_path,label\_path,data\_path,label\_path\_in,label\_path\_out,model='train'):  mkdir(label\_path\_out)  mkdir(label\_path\_in)  mkdir(data\_path)  **for** i **in** os.listdir(train\_path):  *# print(i)*  patient=i[7:11]  file=train\_path+'/patient'+str(patient)+'/P'+str(patient)+'list.txt'  print(file)  names=[]  **with** open(file) **as** f:  line = f.readline()  **while** line:  *# print(line.split('\\')[3][0:8])*  names.append(line.split('\\')[3][0:8])  *# line\_float=[float(line.split()[0]),float(line.split()[1])]*  *# array\_in.append(line\_float)*  line = f.readline()  line = f.readline()  **for** name **in** names:  patient=name[1:3]  idx=name[4:10]  *# print(patient,idx)*    path=train\_path+'/patient'+str(patient)+'/P'+str(patient)+'dicom/P'+str(patient)+'-'+str(idx)+'.dcm'  **if**(model=='train'):  in\_path=label\_path+'/patient'+str(patient)+'/P'+str(patient)+'contours-manual/P'+str(patient)+'-'+str(idx)+'-icontour-manual.txt'  out\_path=label\_path+'/patient'+str(patient)+'/P'+str(patient)+'contours-manual/P'+str(patient)+'-'+str(idx)+'-ocontour-manual.txt'  **else**:  in\_path=label\_path+'/P'+str(patient)+'contours-manual/P'+str(patient)+'-'+str(idx)+'-icontour-manual.txt'  out\_path=label\_path+'/P'+str(patient)+'contours-manual/P'+str(patient)+'-'+str(idx)+'-ocontour-manual.txt'  ds = pydicom.read\_file(path) *#读取.dcm文件*  img = ds.pixel\_array *# 提取图像信息*    *# plt.axis('off')*  *# plt.imshow(img)*    *# plt.savefig(data\_path+'/P'+str(patient)+'-'+str(idx)+'.png',bbox\_inches='tight')*  cv2.imwrite(data\_path+'/P'+str(patient)+'-'+str(idx)+'.png',img\*255)  *# plt.close()*  array\_in=[]  array\_out=[]  **with** open(in\_path) **as** f:  line = f.readline()  **while** line:  line\_float=[float(line.split()[0]),float(line.split()[1])]  array\_in.append(line\_float)  line = f.readline()    **with** open(in\_path) **as** f:  line = f.readline()  **while** line:  line\_float=[float(line.split()[0]),float(line.split()[1])]  array\_out.append(line\_float)  line = f.readline()  array\_in=np.array(array\_in)  array\_out=np.array(array\_out)  *# print(type(array\_in))*  imgName=data\_path+'/P'+str(patient)+'-'+str(idx)+'.png'  *# 展示原图*  img\_mask = cv2.imread(imgName)  *# plt.imshow(img\_mask)*  *# plt.axis('off')*  *# 创建掩膜*  mask = np.zeros(img.shape[:2], dtype=np.uint8)    polygon\_in = np.array(array\_in, np.int32) *# 坐标为顺时针方向*  cv2.fillConvexPoly(mask, polygon\_in, (255, 255, 255))  *# # 展示掩膜图片*  *# plt.imshow(mask)*  cv2.imwrite(label\_path\_in+'/P'+str(patient)+'-'+str(idx)+'.png',mask)  img=cv2.imread(label\_path\_in+'/P'+str(patient)+'-'+str(idx)+'.png')  img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  difference = (img\_gray.max() - img\_gray.min()) // 2  \_, img\_binary = cv2.threshold(img\_gray, difference, 1, cv2.THRESH\_BINARY)  cv2.imwrite(label\_path\_in+'/P'+str(patient)+'-'+str(idx)+'.png',img\_binary)  *# plt.savefig(label\_path\_in+'/P'+str(patient)+'-'+str(idx)+'.png',bbox\_inches='tight')*  *# plt.close()*  mask = np.zeros(img.shape[:2], dtype=np.uint8)  polygon\_out = np.array(array\_out, np.int32) *# 坐标为顺时针方向*  cv2.fillConvexPoly(mask, polygon\_in, (255, 255, 255))  *# plt.axis('off')*  *# plt.imshow(mask)*  *# plt.savefig(label\_path\_out+'/P'+str(patient)+'-'+str(idx)+'.png',bbox\_inches='tight')*  cv2.imwrite(label\_path\_out+'/P'+str(patient)+'-'+str(idx)+'.png',mask)  img=cv2.imread(label\_path\_out+'/P'+str(patient)+'-'+str(idx)+'.png')  img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  difference = (img\_gray.max() - img\_gray.min()) // 2  \_, img\_binary = cv2.threshold(img\_gray, difference, 1, cv2.THRESH\_BINARY)  cv2.imwrite(label\_path\_out+'/P'+str(patient)+'-'+str(idx)+'.png',img\_binary)  *# plt.close()*  （3）定义损失函数BCE loss  class BCELoss(nn.Layer):  def \_\_init\_\_(self,  weight=None,  pos\_weight=None,  ignore\_index=255,  edge\_label=False):  super().\_\_init\_\_()  self.weight = weight # 权重，用于赋予不同类别或样本不同的重要性，可以是Tensor或者'dynamic'  self.pos\_weight = pos\_weight # 正类样本的权重  self.ignore\_index = ignore\_index # 忽略的标签索引  self.edge\_label = edge\_label # 是否为边缘标签  self.EPS = 1e-10 # 防止除零的小值  # 参数检查和初始化  def forward(self, logit, label):  # 损失函数的前向传播  # 调整标签形状  if len(label.shape) != len(logit.shape):  label = paddle.unsqueeze(label, 1)  # 创建mask，标记要忽略的区域  mask = (label != self.ignore\_index)  mask = paddle.cast(mask, 'float32')  # 如果标签的类别数量不等于logit的类别数量，转换标签为one-hot编码  if label.shape[1] != logit.shape[1]:  label = label.squeeze(1)  label = F.one\_hot(label, logit.shape[1])  label = label.transpose((0, 3, 1, 2))  # 计算权重  if isinstance(self.weight, str):  # 根据字符串类型的权重计算权重值  # 根据类别不平衡情况动态调整正负类别权重  else:  weight = self.weight  # 计算正类样本的权重  if isinstance(self.pos\_weight, str):  # 根据字符串类型的正类样本权重计算权重值  else:  pos\_weight = self.pos\_weight  # 将标签转为float类型  label = label.astype('float32')  # 计算二元交叉熵损失  loss = paddle.nn.functional.binary\_cross\_entropy\_with\_logits(  logit,  label,  weight=weight,  reduction='none',  pos\_weight=pos\_weight)  # 应用mask，计算平均损失  loss = loss \* mask  loss = paddle.mean(loss) / (paddle.mean(mask) + self.EPS)  # 设置梯度停止  label.stop\_gradient = True  mask.stop\_gradient = True  return loss # 返回计算得到的损失  （4）构建训练集、测试集、验证集  %cd /home/aistudio/PaddleSeg/  *# 参数调整*  **import** paddle  **from** paddleseg.models **import** UNet, UNetPlusPlus  **import** paddleseg.transforms **as** T  **from** paddleseg.datasets **import** Dataset  *# from paddleseg.models.losses import DiceLoss*  *# from paddleseg.models.losses import CrossEntropyLoss,FocalLoss,LovaszSoftmaxLoss*  *# 构建训练集*  train\_transforms = [  T.RandomHorizontalFlip(), *# 水平翻转*  T.RandomVerticalFlip(), *# 垂直翻转*  T.RandomRotation(), *# 随机旋转*  T.RandomScaleAspect(), *# 随机缩放*  T.RandomDistort(), *# 随机扭曲*  T.Resize(target\_size=(256, 256)), *# 这里为了加快速度，改为256x256*  T.Normalize() *# 归一化*  ]  train\_dataset = Dataset(  transforms=train\_transforms,  dataset\_root='dataset',  num\_classes=2,  mode='train',  train\_path='dataset/train\_list.txt',  separator=' ',  )  *# 构建验证集*  val\_transforms = [  T.Resize(target\_size=(256, 256)),  T.Normalize()  ]  val\_dataset = Dataset(  transforms=val\_transforms,  dataset\_root='dataset',  num\_classes=2,  mode='val',  val\_path='dataset/val\_list.txt',  separator=' ',  )  *# 优化器及损失*  epochs = 5  batch\_size = 16  *# iters = epochs \* 7278 // batch\_size //2*  iters=2000  base\_lr = 2e-3  losses = {}  *# losses['types'] = [LovaszSoftmaxLoss()]*  *# losses['coef'] = [1]*  losses['types'] = [BCELoss()]  losses['coef'] = [1]  （5）重写evaluate函数  **import** os  **import** numpy **as** np  **import** time  **import** paddle  **import** paddle.nn.functional **as** F  **from** paddleseg.utils **import** metrics, TimeAverager, calculate\_eta, logger, progbar  **from** paddleseg.core **import** infer  np.set\_printoptions(suppress=True)  **def** **evaluate**(model,  eval\_dataset,  aug\_eval=False,  scales=1.0,  flip\_horizontal=False,  flip\_vertical=False,  is\_slide=False,  stride=None,  crop\_size=None,  precision='fp32',  amp\_level='O1',  num\_workers=0,  print\_detail=True,  auc\_roc=False,  use\_multilabel=False):    model.eval()  nranks = paddle.distributed.ParallelEnv().nranks  local\_rank = paddle.distributed.ParallelEnv().local\_rank  **if** nranks > 1:  *# Initialize parallel environment if not done.*  **if** **not** paddle.distributed.parallel.parallel\_helper.\_is\_parallel\_ctx\_initialized(  ):  paddle.distributed.init\_parallel\_env()  batch\_sampler = paddle.io.DistributedBatchSampler(  eval\_dataset, batch\_size=1, shuffle=False, drop\_last=False)  loader = paddle.io.DataLoader(  eval\_dataset,  batch\_sampler=batch\_sampler,  num\_workers=num\_workers,  return\_list=True, )  total\_iters = len(loader)  intersect\_area\_all = paddle.zeros([1], dtype='int64')  pred\_area\_all = paddle.zeros([1], dtype='int64')  label\_area\_all = paddle.zeros([1], dtype='int64')  logits\_all = None  label\_all = None  **if** print\_detail:  logger.info("Start evaluating (total\_samples: {}, total\_iters: {})...".  format(len(eval\_dataset), total\_iters))  *#TODO(chenguowei): fix log print error with multi-gpus*  progbar\_val = progbar.Progbar(  target=total\_iters, verbose=1 **if** nranks < 2 **else** 2)  reader\_cost\_averager = TimeAverager()  batch\_cost\_averager = TimeAverager()  batch\_start = time.time()  **with** paddle.no\_grad():  **for** iter, data **in** enumerate(loader):  reader\_cost\_averager.record(time.time() - batch\_start)  label = data['label'].astype('int64')  **if** aug\_eval:  **if** precision == 'fp16':  **with** paddle.amp.auto\_cast(  level=amp\_level,  enable=True,  custom\_white\_list={  "elementwise\_add", "batch\_norm",  "sync\_batch\_norm"  },  custom\_black\_list={'bilinear\_interp\_v2'}):  pred, logits = infer.aug\_inference(  model,  data['img'],  trans\_info=data['trans\_info'],  scales=scales,  flip\_horizontal=flip\_horizontal,  flip\_vertical=flip\_vertical,  is\_slide=is\_slide,  stride=stride,  crop\_size=crop\_size)  *# use\_multilabel=False)*  **else**:  pred, logits = infer.aug\_inference(  model,  data['img'],  trans\_info=data['trans\_info'],  scales=scales,  flip\_horizontal=flip\_horizontal,  flip\_vertical=flip\_vertical,  is\_slide=is\_slide,  stride=stride,  crop\_size=crop\_size)  *# use\_multilabel=False)*  **else**:  **if** precision == 'fp16':  **with** paddle.amp.auto\_cast(  level=amp\_level,  enable=True,  custom\_white\_list={  "elementwise\_add", "batch\_norm",  "sync\_batch\_norm"  },  custom\_black\_list={'bilinear\_interp\_v2'}):  pred, logits = infer.inference(  model,  data['img'],  trans\_info=data['trans\_info'],  is\_slide=is\_slide,  stride=stride,  crop\_size=crop\_size)  *# use\_multilabel=False)*  **else**:  pred, logits = infer.inference(  model,  data['img'],  trans\_info=data['trans\_info'],  is\_slide=is\_slide,  stride=stride,  crop\_size=crop\_size)  *# use\_multilabel=False)*  intersect\_area, pred\_area, label\_area = metrics.calculate\_area(  pred,  label,  eval\_dataset.num\_classes,  ignore\_index=eval\_dataset.ignore\_index)  *# use\_multilabel=False)*  *# Gather from all ranks*  **if** nranks > 1:  intersect\_area\_list = []  pred\_area\_list = []  label\_area\_list = []  paddle.distributed.all\_gather(intersect\_area\_list,  intersect\_area)  paddle.distributed.all\_gather(pred\_area\_list, pred\_area)  paddle.distributed.all\_gather(label\_area\_list, label\_area)  *# Some image has been evaluated and should be eliminated in last iter*  **if** (iter + 1) \* nranks > len(eval\_dataset):  valid = len(eval\_dataset) - iter \* nranks  intersect\_area\_list = intersect\_area\_list[:valid]  pred\_area\_list = pred\_area\_list[:valid]  label\_area\_list = label\_area\_list[:valid]  **for** i **in** range(len(intersect\_area\_list)):  intersect\_area\_all = intersect\_area\_all + intersect\_area\_list[  i]  pred\_area\_all = pred\_area\_all + pred\_area\_list[i]  label\_area\_all = label\_area\_all + label\_area\_list[i]  **else**:  intersect\_area\_all = intersect\_area\_all + intersect\_area  pred\_area\_all = pred\_area\_all + pred\_area  label\_area\_all = label\_area\_all + label\_area  **if** auc\_roc:  logits = F.softmax(logits, axis=1)  **if** logits\_all **is** None:  logits\_all = logits.numpy()  label\_all = label.numpy()  **else**:  logits\_all = np.concatenate(  [logits\_all, logits.numpy()]) *# (KN, C, H, W)*  label\_all = np.concatenate([label\_all, label.numpy()])  batch\_cost\_averager.record(  time.time() - batch\_start, num\_samples=len(label))  batch\_cost = batch\_cost\_averager.get\_average()  reader\_cost = reader\_cost\_averager.get\_average()  **if** local\_rank == 0 **and** print\_detail:  progbar\_val.update(iter + 1, [('batch\_cost', batch\_cost),  ('reader cost', reader\_cost)])  reader\_cost\_averager.reset()  batch\_cost\_averager.reset()  batch\_start = time.time()  metrics\_input = (intersect\_area\_all, pred\_area\_all, label\_area\_all)  class\_iou, miou = metrics.mean\_iou(\*metrics\_input)  acc, class\_precision, class\_recall = metrics.class\_measurement(  \*metrics\_input)  kappa = metrics.kappa(\*metrics\_input)  class\_dice, mdice = metrics.dice(\*metrics\_input)  **if** auc\_roc:  auc\_roc = metrics.auc\_roc(  logits\_all, label\_all, num\_classes=eval\_dataset.num\_classes)  auc\_infor = ' Auc\_roc: {:.4f}'.format(auc\_roc)  **if** print\_detail:  infor = "[EVAL] #Images: {} mIoU: {:.4f} Acc: {:.4f} Kappa: {:.4f} Dice: {:.4f}".format(  len(eval\_dataset), miou, acc, kappa, mdice)  infor = infor + auc\_infor **if** auc\_roc **else** infor  logger.info(infor)  logger.info("[EVAL] Class IoU: \n" + str(np.round(class\_iou, 4)))  logger.info("[EVAL] Class Precision: \n" + str(  np.round(class\_precision, 4)))  logger.info("[EVAL] Class Recall: \n" + str(np.round(class\_recall, 4)))  **return** miou, acc, class\_iou, class\_precision, kappa , mdice  （6）重写train函数  **import** os  **import** time  **from** collections **import** deque  **import** shutil  **import** paddle  **import** paddle.nn.functional **as** F  **from** paddleseg.utils **import** (TimeAverager, calculate\_eta, resume, logger,  worker\_init\_fn, train\_profiler, op\_flops\_funs)  *# 因为重写了evaluate函数，所以没有用paddleseg自身的*  *# from paddleseg.core.val import evaluate*  **def** **check\_logits\_losses**(logits\_list, losses):  len\_logits = len(logits\_list)  len\_losses = len(losses['types'])  **if** len\_logits != len\_losses:  **raise** RuntimeError(  'The length of logits\_list should equal to the types of loss config: {} != {}.'  .format(len\_logits, len\_losses))  **def** **loss\_computation**(logits\_list, labels, losses, edges=None):  check\_logits\_losses(logits\_list, losses)  loss\_list = []  **for** i **in** range(len(logits\_list)):  logits = logits\_list[i]  loss\_i = losses['types'][i]  coef\_i = losses['coef'][i]  **if** loss\_i.\_\_class\_\_.\_\_name\_\_ == 'MixedLoss':  mixed\_loss\_list = loss\_i(logits, labels)  **for** mixed\_loss **in** mixed\_loss\_list:  loss\_list.append(coef\_i \* mixed\_loss)  **elif** loss\_i.\_\_class\_\_.\_\_name\_\_ **in** ("KLLoss", ):  loss\_list.append(coef\_i \*  loss\_i(logits\_list[0], logits\_list[1].detach()))  **else**:  loss\_list.append(coef\_i \* loss\_i(logits, labels))  **return** loss\_list  *# 新增了一个参数 filename 就是保存的npy文件位置 位置就是save\_dir+filename，这个文件中保存的是loss的信息*  *# 还要增加一个参数dicepath，这个npy文件中保存的是dice的信息*  **def** **train**(model,  train\_dataset,  filename,  dicepath,  val\_dataset=None,  optimizer=None,  save\_dir='output',  iters=10000,  batch\_size=2,  resume\_model=None,  save\_interval=1000,  log\_iters=10,  num\_workers=0,  use\_vdl=False,  losses=None,  keep\_checkpoint\_max=5,  test\_config=None,  precision='fp32',  amp\_level='O1',  profiler\_options=None,  to\_static\_training=False):  filename=save\_dir+'/'+filename  print(filename)  dicepath=save\_dir+'/'+dicepath  print(dicepath)  model.train()  nranks = paddle.distributed.ParallelEnv().nranks  local\_rank = paddle.distributed.ParallelEnv().local\_rank  start\_iter = 0  **if** resume\_model **is** **not** None:  start\_iter = resume(model, optimizer, resume\_model)  **if** **not** os.path.isdir(save\_dir):  **if** os.path.exists(save\_dir):  os.remove(save\_dir)  os.makedirs(save\_dir)  *# use amp*  **if** precision == 'fp16':  logger.info('use AMP to train. AMP level = {}'.format(amp\_level))  scaler = paddle.amp.GradScaler(init\_loss\_scaling=1024)  **if** amp\_level == 'O2':  model, optimizer = paddle.amp.decorate(  models=model,  optimizers=optimizer,  level='O2',  save\_dtype='float32')  **if** nranks > 1:  paddle.distributed.fleet.init(is\_collective=True)  optimizer = paddle.distributed.fleet.distributed\_optimizer(  optimizer) *# The return is Fleet object*  ddp\_model = paddle.distributed.fleet.distributed\_model(model)  batch\_sampler = paddle.io.DistributedBatchSampler(  train\_dataset, batch\_size=batch\_size, shuffle=True, drop\_last=True)  loader = paddle.io.DataLoader(  train\_dataset,  batch\_sampler=batch\_sampler,  num\_workers=num\_workers,  return\_list=True,  worker\_init\_fn=worker\_init\_fn, )  **if** use\_vdl:  **from** visualdl **import** LogWriter  log\_writer = LogWriter(save\_dir)  **if** to\_static\_training:  model = paddle.jit.to\_static(model)  logger.info("Successfully to apply @to\_static")  avg\_loss = 0.0  avg\_loss\_list = []  iters\_per\_epoch = len(batch\_sampler)  best\_mean\_iou = -1.0  best\_model\_iter = -1  reader\_cost\_averager = TimeAverager()  batch\_cost\_averager = TimeAverager()  save\_models = deque()  batch\_start = time.time()    *# 为了画出dice图像*  *# 定义一个用于记录Dice的列表*  dice\_save\_list = []    iter = start\_iter  loss\_save\_list=[]  **while** iter < iters:  **for** data **in** loader:  iter += 1  **if** iter > iters:  version = paddle.\_\_version\_\_  **if** version == '2.1.2':  **continue**  **else**:  **break**  reader\_cost\_averager.record(time.time() - batch\_start)  images = data['img']  labels = data['label'].astype('int64')  *# print(labels.shape)*  edges = None  **if** len(data) == 3:  edges = data[2].astype('int64')  **if** hasattr(model, 'data\_format') **and** model.data\_format == 'NHWC':  images = images.transpose((0, 2, 3, 1))    logits\_list = ddp\_model(images) **if** nranks > 1 **else** model(images)  loss\_list = loss\_computation(  logits\_list=logits\_list,  labels=labels,  losses=losses,  edges=edges)  loss = sum(loss\_list)  *# 这一步保存loss*  loss\_save\_list.append(loss)  loss.backward()  *# if the optimizer is ReduceOnPlateau, the loss is the one which has been pass into step.*  **if** isinstance(optimizer, paddle.optimizer.lr.ReduceOnPlateau):  optimizer.step(loss)  **else**:  optimizer.step()  lr = optimizer.get\_lr()  *# update lr*  **if** isinstance(optimizer, paddle.distributed.fleet.Fleet):  lr\_sche = optimizer.user\_defined\_optimizer.\_learning\_rate  **else**:  lr\_sche = optimizer.\_learning\_rate  **if** isinstance(lr\_sche, paddle.optimizer.lr.LRScheduler):  lr\_sche.step()  train\_profiler.add\_profiler\_step(profiler\_options)  model.clear\_gradients()  avg\_loss += loss.numpy()[0]    **if** **not** avg\_loss\_list:  avg\_loss\_list = [l.numpy() **for** l **in** loss\_list]  **else**:  **for** i **in** range(len(loss\_list)):  avg\_loss\_list[i] += loss\_list[i].numpy()  batch\_cost\_averager.record(  time.time() - batch\_start, num\_samples=batch\_size)  **if** (iter) % log\_iters == 0 **and** local\_rank == 0:  avg\_loss /= log\_iters  avg\_loss\_list = [l[0] / log\_iters **for** l **in** avg\_loss\_list]  remain\_iters = iters - iter  avg\_train\_batch\_cost = batch\_cost\_averager.get\_average()  avg\_train\_reader\_cost = reader\_cost\_averager.get\_average()  eta = calculate\_eta(remain\_iters, avg\_train\_batch\_cost)  logger.info(  "[TRAIN] epoch: {}, iter: {}/{}, loss: {:.4f}, lr: {:.6f}, batch\_cost: {:.4f}, reader\_cost: {:.5f}, ips: {:.4f} samples/sec | ETA {}"  .format((iter - 1  ) // iters\_per\_epoch + 1, iter, iters, avg\_loss,  lr, avg\_train\_batch\_cost, avg\_train\_reader\_cost,  batch\_cost\_averager.get\_ips\_average(), eta))  **if** use\_vdl:  log\_writer.add\_scalar('Train/loss', avg\_loss, iter)  *# Record all losses if there are more than 2 losses.*  **if** len(avg\_loss\_list) > 1:  avg\_loss\_dict = {}  **for** i, value **in** enumerate(avg\_loss\_list):  avg\_loss\_dict['loss\_' + str(i)] = value  **for** key, value **in** avg\_loss\_dict.items():  log\_tag = 'Train/' + key  log\_writer.add\_scalar(log\_tag, value, iter)  log\_writer.add\_scalar('Train/lr', lr, iter)  log\_writer.add\_scalar('Train/batch\_cost',  avg\_train\_batch\_cost, iter)  log\_writer.add\_scalar('Train/reader\_cost',  avg\_train\_reader\_cost, iter)  avg\_loss = 0.0  avg\_loss\_list = []  reader\_cost\_averager.reset()  batch\_cost\_averager.reset()  **if** (iter % save\_interval == 0 **or**  iter == iters) **and** (val\_dataset **is** **not** None):  num\_workers = 1 **if** num\_workers > 0 **else** 0  **if** test\_config **is** None:  test\_config = {}  mean\_iou, acc, \_, \_, \_ , mdice = evaluate(  model,  val\_dataset,  num\_workers=num\_workers,  precision=precision,  amp\_level=amp\_level,  \*\*test\_config,  use\_multilabel=False)    *# 保存dice*  dice\_save\_list.append(mdice)  model.train()  **if** (iter % save\_interval == 0 **or** iter == iters) **and** local\_rank == 0:  current\_save\_dir = os.path.join(save\_dir,  "iter\_{}".format(iter))  **if** **not** os.path.isdir(current\_save\_dir):  os.makedirs(current\_save\_dir)  paddle.save(model.state\_dict(),  os.path.join(current\_save\_dir, 'model.pdparams'))  paddle.save(optimizer.state\_dict(),  os.path.join(current\_save\_dir, 'model.pdopt'))  save\_models.append(current\_save\_dir)  **if** len(save\_models) > keep\_checkpoint\_max > 0:  model\_to\_remove = save\_models.popleft()  shutil.rmtree(model\_to\_remove)  **if** val\_dataset **is** **not** None:  **if** mean\_iou > best\_mean\_iou:  best\_mean\_iou = mean\_iou  best\_model\_iter = iter  best\_model\_dir = os.path.join(save\_dir, "best\_model")  paddle.save(  model.state\_dict(),  os.path.join(best\_model\_dir, 'model.pdparams'))  logger.info(  '[EVAL] The model with the best validation mIoU ({:.4f}) was saved at iter {}.'  .format(best\_mean\_iou, best\_model\_iter))  **if** use\_vdl:  log\_writer.add\_scalar('Evaluate/mIoU', mean\_iou, iter)  log\_writer.add\_scalar('Evaluate/Acc', acc, iter)  batch\_start = time.time()  *# Calculate flops.*  **if** local\_rank == 0 **and** **not** (precision == 'fp16' **and** amp\_level == 'O2'):  \_, c, h, w = images.shape  \_ = paddle.flops(  model, [1, c, h, w],  custom\_ops={paddle.nn.SyncBatchNorm: op\_flops\_funs.count\_syncbn})  *# Sleep for half a second to let dataloader release resources.*  time.sleep(0.5)  **if** use\_vdl:  log\_writer.close()    np.save(filename, loss\_save\_list)  np.save(dicepath, dice\_save\_list)  （6）unet训练过程 !export CUDA\_VISIBLE\_DEVICES=0 *# 设置1张可用的卡*  print(type(unet\_model))  *# paddle.summary(unet\_model, (1, 3, 128, 128)) # 查看网络结构*  lr = paddle.optimizer.lr.CosineAnnealingDecay(base\_lr, T\_max=(iters // 3), last\_epoch=0.5)  unt\_optimizer = paddle.optimizer.Adam(lr, parameters=unet\_model.parameters(),weight\_decay=4.0e-5)  *# 训练*  os.environ['FLAGS\_eager\_delete\_tensor\_gb'] = "0.0"  *# 更改对应文件名即可*  train(  model=unet\_model,  filename='unet.npy',#其余训练过程道理完全相同，仅更改训练模型对应的filename，因此不做重复展示  dicepath = 'dice\_values.npy',  train\_dataset=train\_dataset,  val\_dataset=val\_dataset,  optimizer=unt\_optimizer,  save\_dir='output\_unet',  iters=iters,  batch\_size=batch\_size,  save\_interval=int(iters/5),  log\_iters=10,  num\_workers=0,  losses=losses,  use\_vdl=True)  （8）loss绘图  loss=np.load('output\_unet/unet.npy')  x=range(1,len(loss)+1)  plt.plot(x,loss)  （9）dice绘图  dice=np.load('output\_unet/dice\_values.npy')  x=range(1,len(dice)+1)  plt.plot(x,dice)  （10）编码器  *# 加入注意力机制之后，重写unet，形成senet-unet和cbam-unet*  *# 可以参考SE与CBAM注意力机制改进的U-Net\_副本，之前上课的一个练习*  *# 编码器*  *# 其中nn.Layer表示该类继承自PaddlePaddle的深度学习网络层（layer），*  *# 表明Encoder本质上是一个神经网络模型。*  **class** **Encoder**(nn.Layer):  **def** **\_\_init\_\_**(self, num\_channels, num\_filters,attention=False):  *# super(Encoder, self)调用Encoder类的父类，并且将当前实例对象*  *# self作为参数传递给它，以便在父类中进行相应的初始化工作。然后通过\_\_init\_\_()*  *# 方法完成Encoder这个子类自己的初始化工作。*  super(Encoder,self).\_\_init\_\_()  **if** attention **is** False:  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters, act="relu")  )  **elif** attention=='CBAM':  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  CBAM(num\_filters),  nn.BatchNorm(num\_filters, act="relu")  )  **elif** attention=='SEAttention':  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  SEAttention(num\_filters),  nn.BatchNorm(num\_filters, act="relu")  )  **elif** attention == 'SPA':  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  SPA(num\_filters),  nn.BatchNorm(num\_filters, act="relu")  )  *# 池化层，图片尺寸减半[H/2 W/2]*  self.pool = nn.MaxPool2D(kernel\_size=2, stride=2, padding="SAME")  **def** **forward**(self,inputs):  x = self.features(inputs)  x\_conv = x  x\_pool = self.pool(x)  *# 返回池化之前的与 池化之后的*  **return** x\_conv, x\_pool  *# paddle.summary(Encoder(3,64,'SEAttention'), (1, 3, 128, 128))*  （11）解码器  **class** **Decoder**(nn.Layer):  **def** **\_\_init\_\_**(self, num\_channels,num\_filters,attention=False):  super(Decoder,self).\_\_init\_\_()  **if** attention **is** False:  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters, act="relu")  )  **elif** attention=='CBAM':  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  CBAM(num\_filters),  nn.BatchNorm(num\_filters, act="relu")  )  **elif** attention=='SEAttention':  self.features = nn.Sequential(  *# 3\*3卷积核，步长为1，填充1，不变图片尺寸*  nn.Conv2D(in\_channels=num\_channels,  out\_channels=num\_filters,  kernel\_size = 3,  stride=1,  padding=1),  nn.BatchNorm(num\_filters,act="relu"),  nn.Conv2D(in\_channels=num\_filters,  out\_channels=num\_filters,  kernel\_size=3,  stride=1,  padding=1),  SEAttention(num\_filters),  nn.BatchNorm(num\_filters, act="relu")  )  *# 图片尺寸变大一倍[2\*H 2\*W]*  self.up = nn.Conv2DTranspose(num\_channels,num\_filters,2,2,padding=0)  **def** **forward**(self,input\_conv,input\_pool):    x = self.up(input\_pool)  h\_diff = (input\_conv.shape[2] - x.shape[2])  w\_diff = (input\_conv.shape[3] - x.shape[3])  *# #以下采样保存的feature map为基准，填充上采样的feature map尺寸*  *# padding=[上，下，左，右]填充空白像素*  pad = nn.Pad2D(padding=[h\_diff//2, h\_diff - h\_diff//2, w\_diff//2, w\_diff-w\_diff//2])  x = pad(x)  *# in\_channels扩大两倍*  x = paddle.concat([input\_conv, x], axis=1)  x = self.features(x)  **return** x  （12）Unet模型 **class** **UNet**(nn.Layer):  **def** **\_\_init\_\_**(self,num\_classes=2):  super(UNet,self).\_\_init\_\_()  self.down1 = Encoder(num\_channels= 3, num\_filters=64,attention=False) *#下采样*  self.down2 = Encoder(num\_channels= 64, num\_filters=128,attention=False)  self.down3 = Encoder(num\_channels=128, num\_filters=256,attention=False)  self.down4 = Encoder(num\_channels=256, num\_filters=512,attention=False)    self.mid\_conv1 = nn.Conv2D(512,1024,1) *#中间层*  self.mid\_bn1 = nn.BatchNorm(1024,act="relu")  self.mid\_conv2 = nn.Conv2D(1024,1024,1)  self.mid\_bn2 = nn.BatchNorm(1024,act="relu")  self.up4 = Decoder(1024,512,attention=False) *#上采样*  self.up3 = Decoder(512,256,attention=False)  self.up2 = Decoder(256,128,attention=False)  self.up1 = Decoder(128,64,attention=False)    self.last\_conv = nn.Conv2D(64,num\_classes,1) *#1x1卷积，softmax做分类*    **def** **forward**(self,inputs):  x1, x = self.down1(inputs)  x2, x = self.down2(x)  x3, x = self.down3(x)  x4, x = self.down4(x)    x = self.mid\_conv1(x)  x = self.mid\_bn1(x)  x = self.mid\_conv2(x)  x = self.mid\_bn2(x)  x = self.up4(x4, x)  x = self.up3(x3, x)  x = self.up2(x2, x)  x = self.up1(x1, x)    x = self.last\_conv(x)    **return** x  （13）SEAttention模型 **class** **SEAttention\_UNet**(nn.Layer):  **def** **\_\_init\_\_**(self,num\_classes=2):  super(SEAttention\_UNet,self).\_\_init\_\_()  self.down1 = Encoder(num\_channels= 3, num\_filters=64, attention='SEAttention') *#下采样*  self.down2 = Encoder(num\_channels= 64, num\_filters=128,attention='SEAttention')  self.down3 = Encoder(num\_channels=128, num\_filters=256,attention='SEAttention')  self.down4 = Encoder(num\_channels=256, num\_filters=512,attention='SEAttention')  self.mid\_conv1 = nn.Conv2D(512,1024,1) *#中间层*  self.mid\_bn1 = nn.BatchNorm(1024,act="relu")  self.mid\_conv2 = nn.Conv2D(1024,1024,1)  self.mid\_bn2 = nn.BatchNorm(1024,act="relu")  self.up4 = Decoder(1024,512,attention=False) *#上采样*  self.up3 = Decoder(512,256,attention=False)  self.up2 = Decoder(256,128,attention=False)  self.up1 = Decoder(128,64,attention=False)    self.last\_conv = nn.Conv2D(64,num\_classes,1) *#1x1卷积，softmax做分类*    **def** **forward**(self,inputs):  x1, x = self.down1(inputs)  x2, x = self.down2(x)  x3, x = self.down3(x)  x4, x = self.down4(x)  x = self.mid\_conv1(x)  x = self.mid\_bn1(x)  x = self.mid\_conv2(x)  x = self.mid\_bn2(x)  x = self.up4(x4, x)  x = self.up3(x3, x)  x = self.up2(x2, x)  x = self.up1(x1, x)  x = self.last\_conv(x)  logit\_list = [x]  *# print("Output shape:", x.shape)*  **return** logit\_list  （14）SPA-Unet模型  **class** **SPA\_UNet**(nn.Layer):  **def** **\_\_init\_\_**(self,num\_classes=2):  super(SPA\_UNet,self).\_\_init\_\_()  self.down1 = Encoder(num\_channels= 3, num\_filters=64, attention='SPA') *#下采样*  self.down2 = Encoder(num\_channels= 64, num\_filters=128,attention='SPA')  self.down3 = Encoder(num\_channels=128, num\_filters=256,attention='SPA')  self.down4 = Encoder(num\_channels=256, num\_filters=512,attention='SPA')      self.mid\_conv1 = nn.Conv2D(512,1024,1) *#中间层*  self.mid\_bn1 = nn.BatchNorm(1024,act="relu")  self.mid\_conv2 = nn.Conv2D(1024,1024,1)  self.mid\_bn2 = nn.BatchNorm(1024,act="relu")  self.up4 = Decoder(1024,512,attention=False) *#上采样*  self.up3 = Decoder(512,256,attention=False)  self.up2 = Decoder(256,128,attention=False)  self.up1 = Decoder(128,64,attention=False)    self.last\_conv = nn.Conv2D(64,num\_classes,1) *#1x1卷积，softmax做分类*    **def** **forward**(self,inputs):  x1, x = self.down1(inputs)  x2, x = self.down2(x)  x3, x = self.down3(x)  x4, x = self.down4(x)    x = self.mid\_conv1(x)  x = self.mid\_bn1(x)  x = self.mid\_conv2(x)  x = self.mid\_bn2(x)  x = self.up4(x4, x)  x = self.up3(x3, x)  x = self.up2(x2, x)  x = self.up1(x1, x)    x = self.last\_conv(x)  logit\_list = [x]    **return** logit\_list  *# paddle.summary(CBAM\_UNet(), (1, 3, 128, 128))*  （15）CBAM模型 **class** **CBAM\_UNet**(nn.Layer):  **def** **\_\_init\_\_**(self,num\_classes=2):  super(CBAM\_UNet,self).\_\_init\_\_()  self.down1 = Encoder(num\_channels= 3, num\_filters=64, attention='CBAM') *#下采样*  self.down2 = Encoder(num\_channels= 64, num\_filters=128,attention='CBAM')  self.down3 = Encoder(num\_channels=128, num\_filters=256,attention='CBAM')  self.down4 = Encoder(num\_channels=256, num\_filters=512,attention='CBAM')      self.mid\_conv1 = nn.Conv2D(512,1024,1) *#中间层*  self.mid\_bn1 = nn.BatchNorm(1024,act="relu")  self.mid\_conv2 = nn.Conv2D(1024,1024,1)  self.mid\_bn2 = nn.BatchNorm(1024,act="relu")  self.up4 = Decoder(1024,512,attention=False) *#上采样*  self.up3 = Decoder(512,256,attention=False)  self.up2 = Decoder(256,128,attention=False)  self.up1 = Decoder(128,64,attention=False)    self.last\_conv = nn.Conv2D(64,num\_classes,1) *#1x1卷积，softmax做分类*    **def** **forward**(self,inputs):  x1, x = self.down1(inputs)  x2, x = self.down2(x)  x3, x = self.down3(x)  x4, x = self.down4(x)    x = self.mid\_conv1(x)  x = self.mid\_bn1(x)  x = self.mid\_conv2(x)  x = self.mid\_bn2(x)  x = self.up4(x4, x)  x = self.up3(x3, x)  x = self.up2(x2, x)  x = self.up1(x1, x)    x = self.last\_conv(x)  logit\_list = [x]    **return** logit\_list  *# paddle.summary(CBAM\_UNet(), (1, 3, 128, 128))*  （16）预测模型，保存模型路径（以unet为例，其余原理相同，仅更改对应模型名以得到不同的保存路径，不做重复展示）  !pwd  **from** paddleseg.utils **import** get\_sys\_env, logger, get\_image\_list  *# image\_path='dataset/test2\_img/'*  *# image\_list, image\_dir = get\_image\_list(image\_path)*  **import** paddleseg.transforms **as** T  test\_transforms = T.Compose([  T.Resize(target\_size=(256, 256)),  T.Normalize()  ])  **from** paddleseg.core **import** predict  predict(  model=unet\_model,  model\_path='output\_unet/best\_model/model.pdparams',  transforms=test\_transforms,  image\_list=image\_list,  image\_dir='dataset/test1\_img',  save\_dir='output\_unet/results1'  )  （17）评价准确率（IOU）分数列表（以unet为例，其余相同，仅更改模型路径名，不做重复展示）  def evaulate\_miou(img\_path, label\_path):  iou = [] # 初始化IoU列表  # 遍历图像路径中的每张图像  for img in os.listdir(img\_path):  imgpath = img\_path + img  img\_1 = cv2.imread(imgpath)    img\_test = np.zeros(img\_1.shape[0:2])  for n, i in enumerate(img\_1):  for m, j in enumerate(i):  if j[1] == 128:  img\_test[n][m] = 1  data\_path = label\_path + img  data = cv2.imread(data\_path, -1)  target = data  prediction = img\_test  # 计算Intersection over Union (IoU)  intersection = np.logical\_and(target, prediction) # 计算交集  union = np.logical\_or(target, prediction) # 计算并集  iou\_score = np.sum(intersection) / np.sum(union) # 计算IoU分数  iou.append(iou\_score) # 将IoU分数添加到列表中  return iou  （18）打印准确率，绘制IoU分布图（以unet为例，其余原理相似，仅更改对应的路径名和标签名，不做重复展示）  img\_path='/home/aistudio/PaddleSeg/output\_unet/results1/pseudo\_color\_prediction/'  label\_path='/home/aistudio/PaddleSeg/dataset/test1\_label\_img\_in/'  iou\_unet=evaulate\_miou(img\_path,label\_path)  print(iou\_unet)  print(len(iou\_unet))  print(sum(iou\_unet)/len(iou\_unet))  x=range(0,len(iou\_unet))  y=iou\_unet  plt.plot(x,y)  plt.show()  （19）绘制箱型图，观察离群值，说明大部分是正常预测的，除了极端部分（以unet为例，其余原理相同，不做重复展示）  fig, ax = plt.subplots() *# 子图*  data=iou\_unet  ax.boxplot(data) |

**6、调试过程（算法训练测试过程描述，以及中间结果图表）**

|  |
| --- |
| 1、安装包  C:\Users\zyh09\Desktop\QQ截图20231230161437.png  2、导入unet模型  C:\Users\zyh09\Desktop\QQ截图20231230163712.png  3、unet模型训练过程（epoch选择2000轮）  C:\Users\zyh09\Desktop\QQ截图20231230165053.png  4、unet模型下的loss下降曲线图和dice曲线图  C:\Users\zyh09\Desktop\QQ截图20231230165657.png  C:\Users\zyh09\Desktop\QQ截图20231230165752.png  5、Unet++模型训练过程  C:\Users\zyh09\Desktop\QQ截图20231230171428.png  6、unet++模型下的loss下降曲线图和dice曲线图  C:\Users\zyh09\Desktop\QQ截图20231230171525.png  C:\Users\zyh09\Desktop\QQ截图20231230171534.png  7、U2net模型训练过程  C:\Users\zyh09\Desktop\QQ截图20231230172409.png  8、U2net模型下的loss下降曲线图和dice曲线图  C:\Users\zyh09\Desktop\QQ截图20231230172450.png  C:\Users\zyh09\Desktop\QQ截图20231230172551.png  9、SEAttention模型训练过程  C:\Users\zyh09\Desktop\QQ截图20231230174129.png  10、SEAttention模型loss下降曲线和dice曲线  C:\Users\zyh09\Desktop\QQ截图20231230174258.png  C:\Users\zyh09\Desktop\QQ截图20231230174308.png  11、SPA-Unet训练模型  C:\Users\zyh09\Desktop\QQ截图20231230175959.png  12、SPA-Unet模型loss下降曲线和dice曲线  C:\Users\zyh09\Desktop\QQ截图20231230180114.png  C:\Users\zyh09\Desktop\QQ截图20231230180125.png  13、CBAM训练模型  C:\Users\zyh09\Desktop\QQ截图20231230181918.png  14、CBAM模型loss下降曲线和dice曲线  C:\Users\zyh09\Desktop\QQ截图20231230182027.png  C:\Users\zyh09\Desktop\QQ截图20231230182233.png |

**7、结果分析（运行测试结果图表和分析，包括失败例子的分析）**

|  |
| --- |
| 1、unet  测试结果：0.7229642194962903  IoU分布图  C:\Users\zyh09\Desktop\下载.png  箱型图：  C:\Users\zyh09\Desktop\下载 (1).png  2、Unet++  测试结果：0.654437705707149  IoU分布图：  C:\Users\zyh09\Desktop\下载 (2).png  箱型图：  C:\Users\zyh09\Desktop\下载 (3).png  3、U2net：  测试结果：0.4579907689429558  IoU分布图：  C:\Users\zyh09\Desktop\下载 (4).png  箱型图：  C:\Users\zyh09\Desktop\下载 (5).png  4、SEAttention  测试结果：0.6336178377381347  IoU图像：  C:\Users\zyh09\Desktop\下载 (6).png  箱型图：  C:\Users\zyh09\Desktop\下载 (7).png  5、SPA\_unet  测试结果：0.5778058013015369  IoU图像：  C:\Users\zyh09\Desktop\下载 (8).png  箱型图：  C:\Users\zyh09\Desktop\下载 (9).png  6、CBAMunet  测试结果：0.5778058013015369  IoU图像：  C:\Users\zyh09\Desktop\下载 (10).png  箱型图：  C:\Users\zyh09\Desktop\下载 (11).png  分析：本实验中预测准确率最高的是Unet模型，准确率达到70%以上，loss和dice曲线最为稳定，而Unet2+则准确率最低，准确率不足50%。先期的时候预测准确率总体都非常低，后来调试参数发现先期选择的epoch值设为200太小了，没有足够收敛，改成300后训练效果有所提升，最后我们定在了2000，达到了一个相对收敛的结果。目前添加了注意力机制后的模型却仍然不如Unet模型的准确率，可能得原因有**：**   1. 模型复杂度增加： 注意力机制引入了额外的参数和计算复杂度，有时可能会增加模型的复杂度，导致过拟合或者训练困难。 2. 数据适配性不足： 注意力机制的性能通常依赖于数据集和任务。在某些情况下，数据集可能不适合或不充分支持注意力机制的有效性，这可能导致性能下降。 3. 超参数选择不当： 注意力机制中的超参数选择可能会影响模型性能。例如，注意力机制中的权重或比例可能需要调整以适应特定的任务。 4. 模型训练不充分： 引入新的机制可能需要更多的训练时间和调优，如果模型没有充分训练可能无法发挥注意力机制的优势。   这些问题也是该问题需要继续处理解决的问题，不同的模型相关参数可能还是需要继续优化改进，也是未来医学图像分割领域发展的一个重点研究方向所在。 |

**8、程序开发总结（简要叙述编写本作业的收获与思考）**

|  |
| --- |
| 通过这次的综合设计作业，我们的收获主要是：   1. 对unet,unet++以及注意力机制更加熟悉了，并且完成了一个图形分割任务，也加强了对这个工作流程的熟悉 2. 增强了代码能力，并且积攒了一些debug的经验。小组成员使用markdown来日常记录一些出现的问题和解决方法，这也有利于我们小组成员之间的交流，比如：         3、通过大作业，增强了小组成员之间的沟通和协作能力 |