

基于 3D-U-Net 模型的小样本肺部肿瘤分割研究

何瑞杰 25110801

摘 要

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

目录

1. 研究背景与目的	2
2. 数据集及初步分析	2
2.1. 数据集介绍	2
2.2. 数据初步分析、增强和预处理	4
2.2.1. 灰度分布	4
2.2.2. 病灶大小	4
2.2.3. CT 数据的预处理	5
3. 模型	6
3.1. 2D 和 3D U-Net	6
3.1.1. 2D U-Net	6
3.1.2. 3D U-Net	6
3.2. 3D U-Net 的变体	7
3.2.1. 3D 残差 U-Net	7
3.2.2. 3D 残差 SE U-Net	7
4. 训练优化	8
4.1. 混合精度训练	8
4.2. 数据切分和分块读取	8
4.3. LRU 缓存	8
5. 实验与结论	10
5.1. 机器参数和训练超参数	10
5.1.1. 机器参数	10
5.1.2. 训练超参数	10
6. 运行项目代码	11
参考文献	12

1. 研究背景与目的

非小细胞肺癌 (NSCLC) 作为肺癌中最常见的类型, 占据了大约 85% 的肺癌病例, 主要包含鳞状细胞癌、腺癌和大细胞癌等亚型。与小细胞肺癌相比, NSCLC 的生长速度通常较慢, 治疗手段也更为多样化, 通常取决于肿瘤的具体类型、发展阶段以及患者整体的健康状况。在 NSCLC 的诊断和治疗过程中, CT 扫描扮演着至关重要的角色。它能提供关于肿瘤大小、形状和位置的详细信息, 帮助医生确定病变的精确阶段, 并指导手术和放疗计划。此外, CT 图像对于监测肿瘤对治疗的响应和检测复发或转移至关重要。

本研究尝试使用深度学习方法, 基于病人 CT 影像中由医护人员手动标注的 NSCLC 数据学习一个 3D 分割模型, 输入为处理过的 CT 影像数据, 输出为分割后的影像数据, 其中包含可能的病灶位置。凭借已在数据集上训练好的鲁棒模型, 可以快速帮助病人和医生定位可能的病灶位置, 节省时间以便后续的处理治疗。

2. 数据集及初步分析

2.1. 数据集介绍

本研究采用的数据为数据集是医学图像分割十项全能挑战赛 (Medical Segmentation Decathlon, MSD)[1] 中的第 6 个子任务, 即 MSD Lung Tumours 数据集[2]。其目标是从 CT 图像中分割出肺部肿瘤, MSD 选择该数据集的原因是“在大的背景中分割出小目标”。该数据集包含 96 例 (实为 95 例) NSCLC 患者的薄层 CT 扫描, 官方划分为 64 例训练集 (实为 63 例) 和 32 例测试集, 其中测试集可以通过官网提交分割结果进行测试。该数据集旨在系统评估机器学习算法在非小细胞肺癌 CT 影像分割任务中的泛化能力与鲁棒性挑战, 作为迄今规模最大且临床异质性最强的公开医学图像分割基准之一, 其数据全部来源于 The Cancer Imaging Archive 获取的经病理证实的 NSCLC 患者术前薄层 CT 扫描, 经严格脱敏处理后以 NIFTI 标准格式呈现。整个数据集共收录 96 例三维 CT 影像, 其中 64 例作为训练集并配有像素级标注, 32 例作为独立测试集用于算法性能评估, 目标区域明确界定为原发肿瘤实体范围, 标注由单一专家完成, 涵盖肿瘤坏死与实性成分的完整勾画。

该数据集在 MSD 挑战赛框架中被明确归类为**小目标-小样本-大视野**任务的典型代表, 其设计初衷在于真实再现临床实践中算法面临的三大核心瓶颈。首要挑战在于目标尺度的极端不平衡性, 肿瘤病灶体积相较于全肺扫描视野占比不足百分之一, 这种悬殊比例要求算法必须具备对微小结构的精准识别能力, 同时导致传统 Dice 重叠度指标对单像素偏差呈现高度敏感, 显著增加了模型优化的难度。其次, 数据集刻意设置了数据稀缺性条件, 64 例训练样本的规模远低于深度学习模型的常规数据需求, 迫使算法必须在有限样本条件下学习高维特征空间, 从而直接检验模型的数据效率与过拟合抑制能力。此外, 数据集充分呈现了解剖与病理层面的异质性特征, 病例覆盖不同肿瘤位置、大小、形态学表现及周围组织浸润模式, 且 CT 扫描协议存在多中心差异, 包括层厚、重建算法和造影剂使用等方面的变异, 全面考验算法跨域泛化的稳健性。

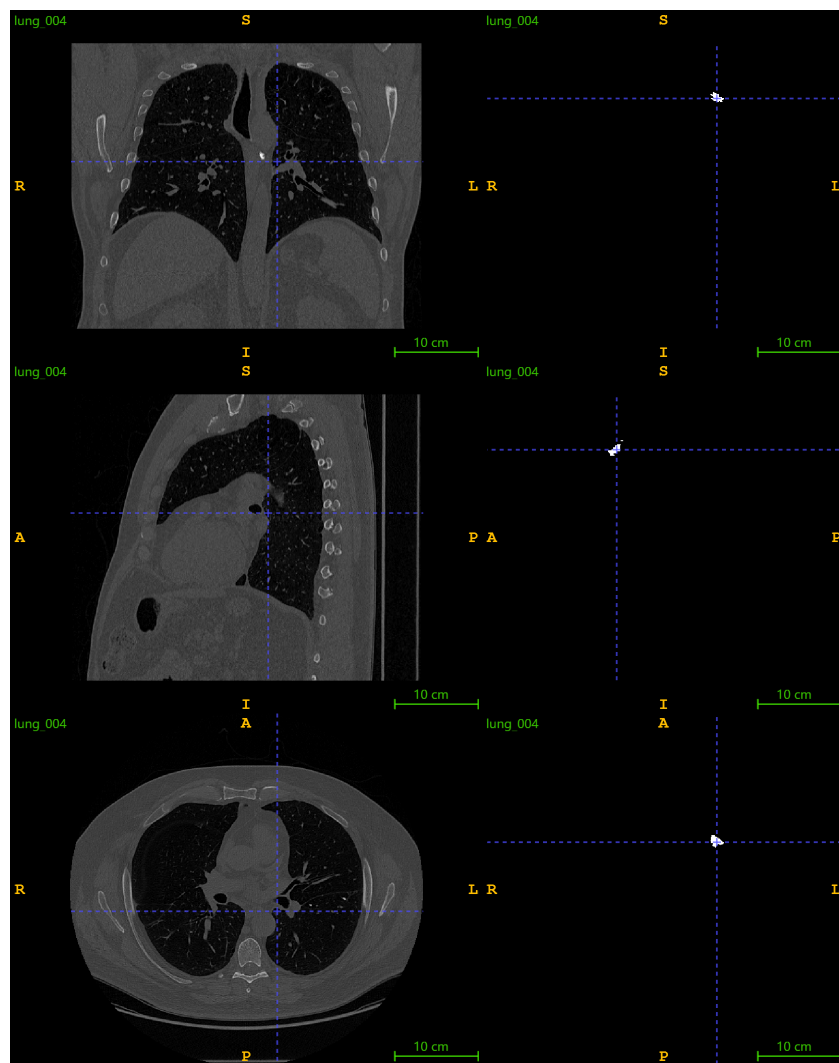


图 1 MSD 肺部肿瘤数据集中的一个样本 lung_004 的 CT 数据（左）和分割标签（右）在开源预览软件 ITK-SNAP 中的预览图像，从上至下分别是冠状面、矢状面和横断面视图。

2.2. 数据初步分析、增强和预处理

相比通常的数据，医学影像数据稀少、受个体影响大甚至有时缺乏完全正确的标注，这使得我们在对待医学影像数据时需要格外注意。对于本项目中涉及的 CT 影像数据有必要分析其数值范围和分布，以确定合适的预处理方式。

2.2.1. 灰度分布

由于人体解剖结构大致相同，可从数据中取一条进行灰度分布的可视化分析。计算整一条数据中体素的灰度分布、矢状面、冠状面和横截面的平均灰度值，其结果如下图所示。首先可以看到数据中的体素值分布在约 -1000 至 $+3000$ 之间，绝大部分灰度值分布位于 -1000 到 2000 之间，对应空气、水和身体组织。小部分在 3000 左右，对应骨骼等致密物。

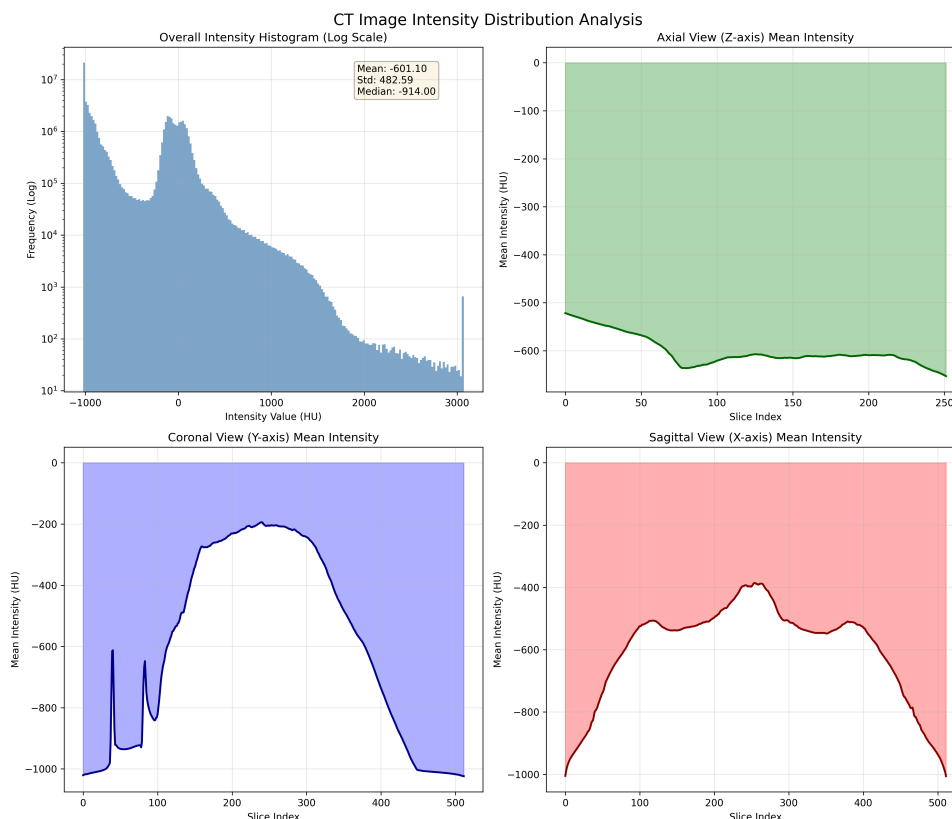


图 2 对 lung_053 CT 数据的灰度值分布分析。左上角为整个数据数组的灰度分布直方图，

2.2.2. 病灶大小

另外我们考虑所有有标签样本（共 63 1‰

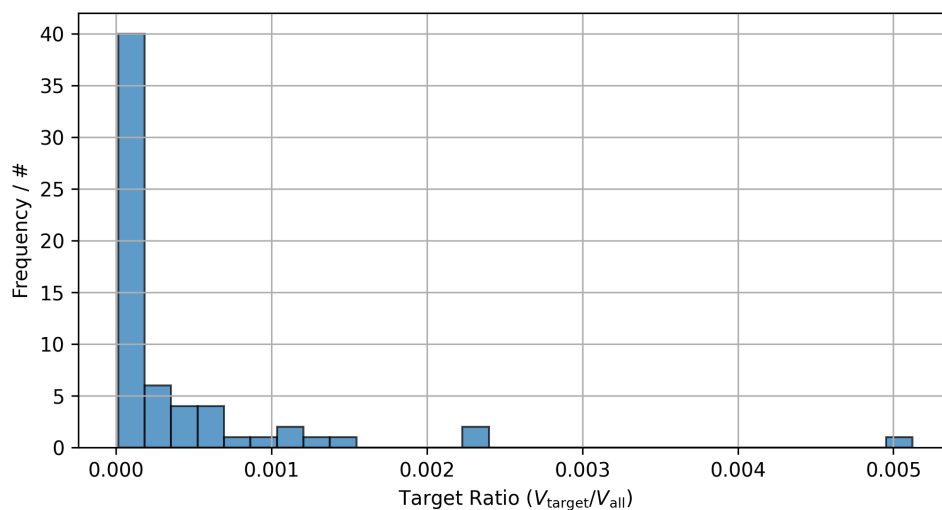


图 3

2.2.3. CT 数据的预处理

CT 扫描生成的 3D 数据中每一体素的强度单位为 Hounsfield (HU)，其中临床参考值为空气 -1000 HU、水对应 0 ± 10 HU，组织对应 $-600 \sim 1500$ HU。

$$x \leftarrow \frac{\text{clamp}_{[-600, 1500]}(x) + 600}{2100}$$

3. 模型

3.1. 2D 和 3D U-Net

3.1.1. 2D U-Net

U-Net 最初由 Olaf Ronneberger 等人【】于 2015 年提出，为的是解决 2D 医学影像分割任务，其得名于论文中的 U 形网络结构，如图 4 所示。由图可见，U-Net 中有一些基本结构： 3×3 卷积和 ReLU 组成的基本卷积模块、 1×1 卷积模块、复制剪贴模块 (copy and crop)、最大值池化下采样模块、以转置卷积为例的上采样模块。数据进入模型后，通过不断通过 3×3 卷积模块和下采样模块，得到多种大小尺度的特征图。在经过 U-Net 最下一层的瓶颈层之后再不断经过卷积模块和上采样模块使数据还原为和输入相同的形状。每经过一层上采样，都会与先前下采样时留下的特征图合并，再经过后续的模块。这样的结构使得网络可以同时捕捉多尺度的信息，在后来的 DDPM 等一众扩散生成模型【】中，被用作噪声预测器。

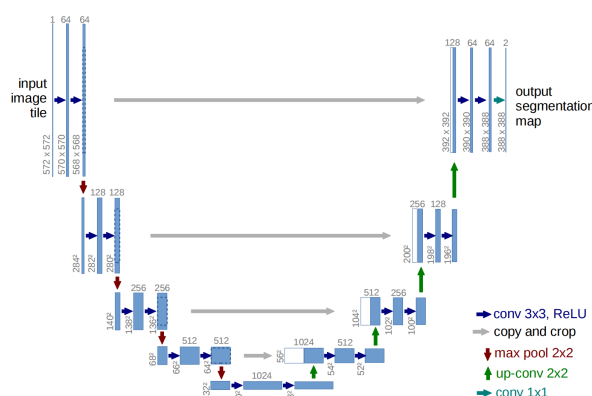


图 4 U-Net 的原始架构。主要由卷积模块、 1×1 卷积模块、上下采样模块和特征图拼接操作构成，赋予了其优秀的多尺度信息提取能力。

3.1.2. 3D U-Net

在面临 3D 医学影像分割任务时，虽然可以将其切分成若干层 2D 医学影像分别经过 U-Net 等模型做分割任务，但这样的方法破坏了影像在第三维度上的连续性，也无法利用这一特点进行高效准确的预测。为充分利用三维影像的这一特点，Özgün Çiçek 等人【】提出了 **3D U-Net 模型**，如图 5 所示。

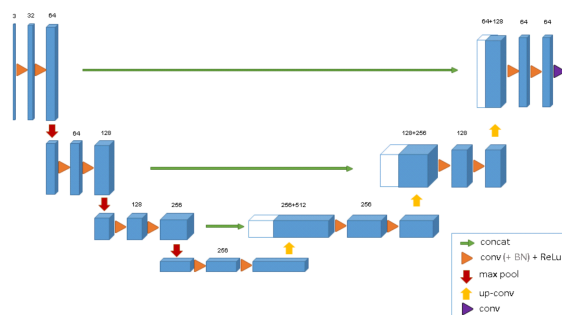


图 5 3D U-Net 的架构。相比于 2D U-Net，其所有卷积核形状改为 $3 \times 3 \times 3$ ，最大池化与上采样核的形状相应变为 $2 \times 2 \times 2$ 。

相比于 2D U-Net，3D U-Net 的所有卷积核形状改为 $3 \times 3 \times 3$ ，最大池化与上采样核的形状相应变为 $2 \times 2 \times 2$ ，使感受野在三个正交方向均衡扩展。

3.2. 3D U-Net 的变体

3.2.1. 3D 残差 U-Net

在此之后，Kisuk Lee 等人【】提出了**残差 3D U-Net 的架构**，并在 SNEMI3D Connectomics 竞赛中超越人类专家水平。残差 3D-UNet 架构如图 6 所示，它将 3D-UNet 中每个尺度上的变换模块改为了由一个 $3 \times 3 \times 1$ 卷积、两个 $3 \times 3 \times 3$ 卷积和一个残差链接组成的残差卷积模块，使得反向传播的梯度流的衰减或扩大效应更弱。

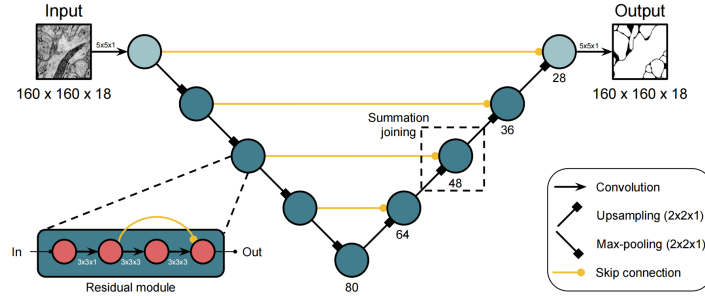


图 6 3D 残差 U-Net 的架构。将每个尺度下的 $3 \times 3 \times 3$ 卷积模块更换为了一个残差卷积模块。

3.2.2. 3D 残差 SE U-Net

以 3D 残差为基础，在每一个残差卷积块后添加一个**挤压刺激 (squeeze and excitation, SE) 模块**【】，就得到了**3D 残差 SE U-Net**。SE 模块可被理解为某种“注意力”模块，它根据输入通过一个卷积层特征图计算空间网格和 (或) 通道的权重，最后和输入特征图做逐元素乘法，达到调整特征图各部分强度的目的。SE 一般有三种，通道 SE、空间 SE 和空间通道 SE，三者的具体结构如图 7 所示。

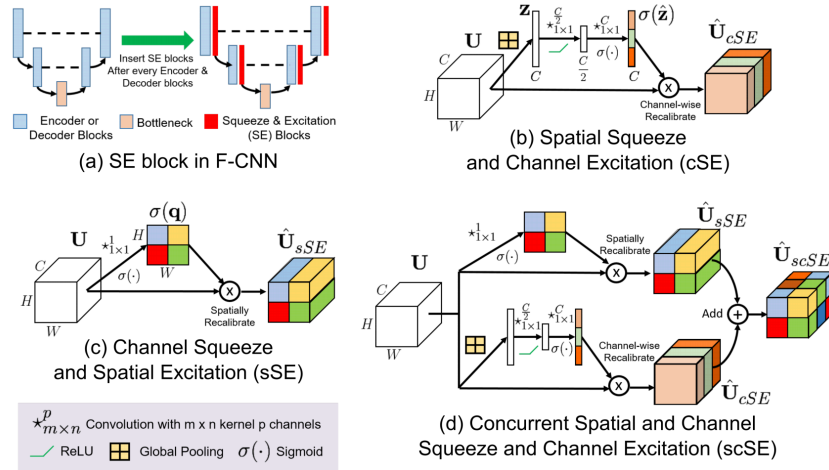


图 7 SE 模块。(a) SE 模块一般位于卷积模块之后，有 (b) 空间 SE、(c) 通道 SE 和 (d) 空间通道 SE 三种。

本项目将使用 3D U-Net、3D 残差 U-Net 和 3D 残差 SE U-Net 进行分割任务。

4. 训练优化

本项目中涉及的 3D 数据和 3D 模型为笔者首次尝试，它涉及到的数据和模型参数的体量相比于同类型的 2D 数据和 2D 模型更大，但数据量相比于 2D 数据小得多。对于前者的问题，一般采取半精度训练等方法加快模型的训练和推理速度，降低 I/O 延迟；对于后者，一般采取数据增强和数据切分的方法。本节中详述在本项目中用到的这些方法。

4.1. 混合精度训练

通常的神经网络工作在 32 位浮点数空间，为加快模型的正向传播速度，可以在模型调用 `forward()` 方法时让其在更小的浮点数空间进行，例如 16 位浮点数空间。正向传播时调整至混合精度相对容易

4.2. 数据切分和分块读取

本项目的 3D 分割任务中数据相比通常的 2D 分割任务稀少，一条数据的形状大致为几百 \times 几百 \times 几百，相比 2D 图像任务中一条数据大小高出许多，遑论设置较大的 `batch_size`。本项目沿用分割中的经典做法，即仿照 `pytorch` 中卷积网络的做法，将输入数据切分成若干块 (patch)。具体而言，设置一个小块的尺寸为 `[px, py, pz]`，和步长 `[tx, ty, tz]`，然后按照该尺寸和步长从一条数据中采样处多个块，每个小块作为一条新的数据输入网络中，这样可以增加网络的训练推理速度，也可以使用较大的 `batch_size` 加速训练过程。在提取过程中，对于有标签的数据，程序将同时在输入数据和标签的相同位置切取，切取得到的数组形状相同。

此外，本项目中 3D 分割任务另一特点为目标标签稀少，肿瘤大小较小，这使得一份病人的 CT 影像中有绝大部分体素的分割标签都是 0。为了减弱训练时标签的不均衡，可以在切分过程中测量对应标签样本的正标签体素占比，并拒绝占比低于某个阈值的块，以达到降低训练样本量和降低标签不平衡的目的。

4.3. LRU 缓存

LRU 缓存是工程上常用的技巧，其全称为 `least recently used cache`。所谓缓存，为处于两种访问速度不同介质之间提高访问速度和效率的高速存储器，例如计算机中 CPU 和内存、内存和硬盘、GPU 的全局内存和线程块之间都存在不同级别的缓存。当处理器通过缓存向存储器中读取数据时，缓存会根据访问的空间局部性原则从存储器中读出一块数据，并将其存入缓存。如果下一次处理器想要读取的数据恰好在缓存中，就免去了再访问慢速存储器的时间开销，这称为 **缓存命中 (cache hit)**；如果缓存中没有处理器想访问的数据，这称为 **缓存脱靶 (cache miss)**，缓存需要重新在慢速存储器中读取新的一块数据，并按照 **缓存替换算法** 整理缓存中存放的数据。

LRU 是一种缓存替换算法，**当缓存未滿时，每次缓存脱靶后缓存将读取新的一块数据；当缓存已滿时，脱靶后读取的新的数据将替换最近最少使用的数据块**。在 Python 中常常使用 LRU 缓存优化经常被调用的函数，例如需要递归计算的斐波那契数列函数。在 Python 中，可以轻易通过添加装饰器或是以显式函数的方式将 LRU 缓存作用在某个函数上：

```
1 # apply LRU by decorating the function
2 @lru_cache(maxsize=self.cache_max_size)
3 def fib(n):
4     if n in [0, 1]:
5         return 1
6     else:
7         return f(n-1) + f(n-2)
8
9 # or we can apply lru_cache() directly on function object
```

Python

```
10 fib = lru_cache(maxsize=self.cache_max_size)(fib)
```

但在本项目的情境下,为了读取一个小数据块而频繁地读取一个较大的数据文件(一般为几十到一两百 M)使得 LRU 缓存是一个提升效率的可选项。需要对其进行调整,笔者使用的 Windows 机器在设置 Dataloader 中的 num_workers 非零的情况下系统会创建子进程,相比于 Linux 或 Unix 系统,Windows 系统会使用 spawn() 模式启动新的 Python 解释器,并通过 pickle 序列化传递对象,这意味着主进程中的所有对象必须是可序列化的,否则解释器将会抛出 PicklingError,而如果使用 lru_cache() 装饰读取数据块的函数,得到的将会是不可序列化的对象。解决方法为不再直接使用 lru_cache() 装饰器,而是直接使用有序字典对象 (OrderedDict) 重新实现一个自定义带有 LRU 缓存的函数,同时注意键可哈希的要求。

5. 实验与结论

5.1. 机器参数和训练超参数

5.1.1. 机器参数

本项目的训练过程中使用两台不同的机器。

1 号机器配备 AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz CPU、32.0 GB 内存和 NVIDIA RTX 3050Ti GPU (4 GB);

2 号机器配备 Intel(R) Xeon(R) Gold 6430 CPU、120 GB 内存和 NVIDIA GeForce RTX 4090 GPU (24 GB)。两个机器的环境和训练超参数一致。

5.1.2. 训练超参数

模型训练的超参数主要存储在 *.yaml 文件中，部分超参数以默认参数的形式显式地存储于 *.py 文件中。较为重要的超参数值如下表所示。

6. 运行项目代码

本项目使用 Python 编写，主要参照了 Github 上的 [pytorch-3dunet](#) 仓库，并基于本项目的实际需求做了诸多简化和优化调整。本项目的环境基于 uv 搭建，可使用命令行直接下载 uv：

```
1 # if you have curl installed on your machine
2 curl -LsSf https://astral.sh/uv/install.sh | sh
3 # or you can use wget
4 wget -qO- https://astral.sh/uv/install.sh | sh
```

Shell

然后根据下载后的指示激活 uv 即可创建环境：

```
1 cd /path/to/project
2 uv init --python 3.10
```

Shell

由于本项目在配置环境时出现问题，您可能在终端直接运行 uv sync 时会得到不兼容报错，此时推荐使用

```
1 uv pip install xxx
```

Shell

绕过 uv 的兼容性检查直接在环境中安装。注意环境中的 pytorch 和 torchvision 包需要选择符合您机器的 CUDA toolkit 的版本。您可以在终端中运行 nvidia-smi 来查询您的 CUDA 版本，根据其选择合适的 pytorch 和 torchvision 版本。假如您的机器的 CUDA 版本为 12.4，则您可以通过下面的代码向环境中添加 pytorch 和 torchvision。

```
1 uv pip install torch torchvision --torch-backend=cu126
```

Shell

在运行主程序 trainers.py 之前，需要在终端中运行下面的代码。

```
1 cd /path/to/project
2
3 # linux or macos
4 source ./venv/bin/activate
5 # windows
6 .\.venv\Scripts\activate
```

Shell

最后，主程序 trainers.py 中读取的超参数文件位于 *.yaml 文件中。

参考文献

- [1] M. Antonelli, A. Reinke, S. Bakas, and others, “The Medical Segmentation Decathlon,” *Nature Communications*, 2022, doi: [10.1038/s41467-022-30695-9](https://doi.org/10.1038/s41467-022-30695-9).
- [2] A. L. Simpson *et al.*, “A large annotated medical image dataset for the development and evaluation of segmentation algorithms.” 2019.
- [3] P. A. Yushkevich *et al.*, “User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability,” *NeuroImage*, vol. 31, no. 3, pp. 1116–1128, 2006, doi: <https://doi.org/10.1016/j.neuroimage.2006.01.015>.
- [4] A. Wolny *et al.*, “Accurate and versatile 3D segmentation of plant tissues at cellular resolution,” *eLife*, vol. 9, p. e57613, July 2020, doi: [10.7554/eLife.57613](https://doi.org/10.7554/eLife.57613).