OpenMMLab 学习笔记

huang_rui4@dahuatech.com

导航

1 前言

以下是在学习、使用 OpenMMLab2.0 的过程中，总结的学习报告、博客等等，以及针对部分问题提出的个人看法，特此记录！

2 新架构

特点： 通用、统一、灵活

- 通用：新的训练器以统一的方式实现数据、模型、评测等组件的构造流程供各算法库调用
- 统一：将不同的算法的训练流程拆解成数据、数据变换、模型、评测、可视化等抽象。统一接口的同时，通过 MMEngine[2.1] 注册器管理
- 细粒度的模块化设计，提供“乐高”式训练

2.1 MMEngine

用户说明文档：https://mmengine.readthedocs.io/zh_CN/latest/

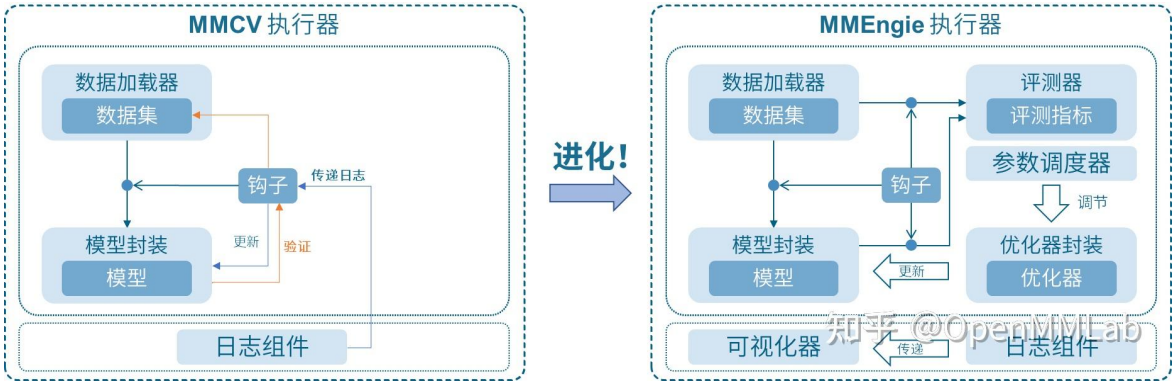


图 1: 更加强大的 Runner

训练引擎的核心模块是执行器（Runner），如图1。为了允许用户拓展、插入和执行自定义逻辑，执行器设置了丰富的钩子（Hook），如图2。

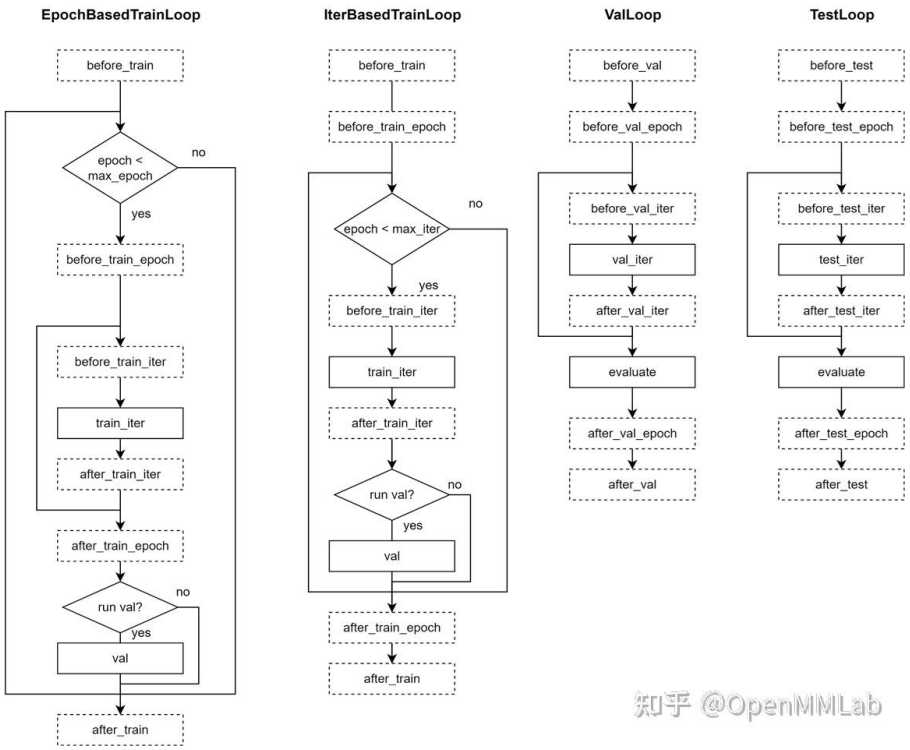


图 2: 丰富的 Hook 点位

2.1.1 执行器

执行器主要调用如下组件来完成训练和推理过程中的循环：

- 数据集（Dataset）
- 模型（Model）
- 优化器（Optimizer）：执行反向传播优化模型。在 MMEngine 中，官方对优化器做了一层封装：OptimWrapper，其优点如下：

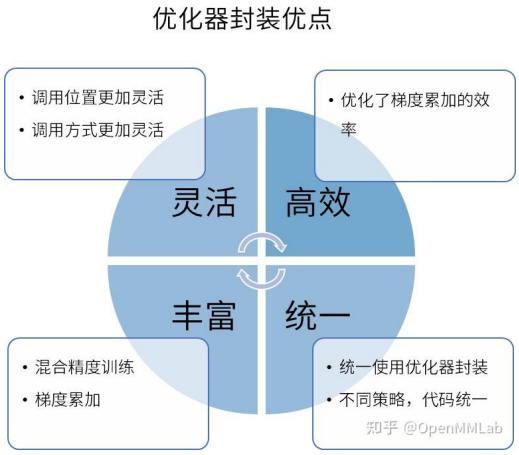
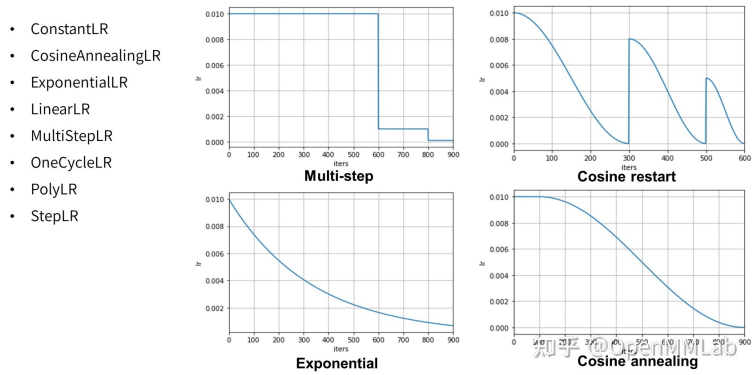


图 3: 优化器封装优点

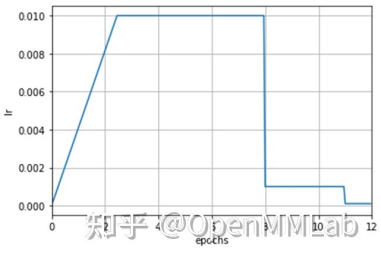
- 参数调度器（Parameter Scheduler）：训练过程中对学习率、动量等优化器超参数动态调整，还支持调度器之间的自由组合



(a)

用户可以设置生效区间（begin，end）组合多个 scheduler，
例如学习率预热可以由两个 scheduler 组成

```
param_scheduler = [
    # 线性学习率预热调度器
    dict(type='LinearLR',
         start_factor=0.001,
         by_epoch=False,  # 按迭代更新学习率
         begin=0,
         end=50),  # 预热前 50 次迭代
    # 主学习率调度器
    dict(type='MultiStepLR',
         by_epoch=True,  # 按轮次更新学习率
         milestones=[8, 11],
         gamma=0.1)]
```



(b)

图 4: 灵活的参数调度器

模型和各模块之间具体的数据流如下：

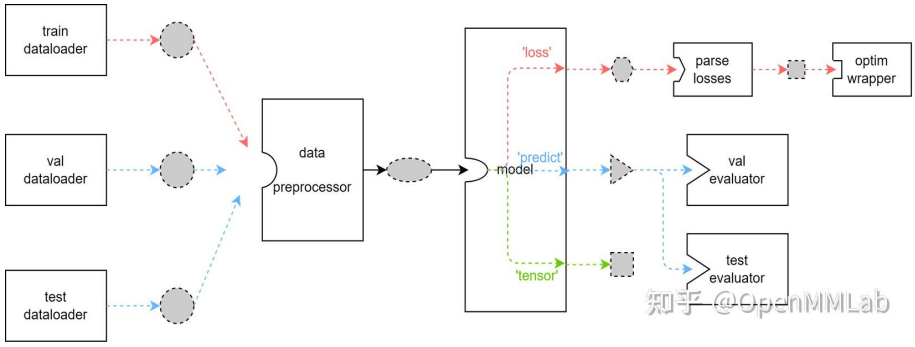


图 5: 模型和各模块之间具体的数据流

在训练间隙或者测试阶段，评测指标与评测器（Metrics & Evaluator）负责对模型性能进行评测。其中，

- Metric 负责实现根据测试数据和模型预测结果，完成模型特定精度指标的计算
- Evaluator 类则位于 Metric 上层，为 Metric 提供格式转换以及分布式通信的支持

问题 2.1

Q：既然 Runner、Hook 这些核心组件都在 MMEngine 里实现了，那 MMCV 还有啥嘞？架构升级后，还能用 MMCV 做哪些事情?

A：详见2.2节

问题 2.2

Q：数据预处理模型 data_preprocessor 和数据集 trasform[2.2] 有何不同?

A：这一模块在从 dataloader 获得数据后，对数据进行搬运，并加速预处理。和 trasform 的不同在于：

- transform 每次调用只处理一张图片，而 data_preprocessor 每次调用处理一个 batch 的图片
- transform 可以包含复杂逻辑，对每张图像执行不同操作，由 CPU 执行，而 data_preprocessor 可以获取模型所在的设备，能够利用 GPU 加速对一个 batch 的图像进行批量处理，进而实现加速

用法上，原先数据集 pipeline 中的 Normalize 变换操作可以移除，代之以单独的 data_preprocessor 配置字段。

为了统一接口，OpenMMLab 2.0 中各个算法库的评测器，模型和数据之间交流的接口都使用了数据元素（Data Element）来进行封装。

在训练、推理执行过程中，上述各个组件都可以调用日志管理模块和可视化器进行结构化和非结构化日志的存储与展示。日志管理（Logging Modules）：负责管理执行器运行过程中产生的各种日志信息。其中消息枢纽（MessageHub）负责实现组件与组件、执行器与执行器之间的数据共享，日志处理器（Log Processor）负责对日志信息进行处理，处理后的日志会分别发送给执行器的日志器（Logger）和可视化器（Visualizer）进行日志的管理与展示。可视化器（Visualizer）：可视化器负责对模型的特征图、预测结果和训练过程中产生

的结构化日志进行可视化，支持 Tensorboard 和 MLflow 等多种可视化后端。

2.1.2 可视化后端

TensorBoard

- 介绍
  TensorBoard 最初是随 TensorFlow 提出的一款可视化工具包，其便捷性和完善的记录功能使它得到了广泛应用，并扩展到 PyTorch 等多种深度学习框架。TensorBoard 支持记录多种数据类型：
  - 指标和损失
  - 超参数和模型 config
  - 图片数据
  - 模型图
  - Embedding Projector（在低维空间可视化高维数据）
- 使用
  - 安装 tensorboard：pip install tensorboard
  - 修改 vis_backends 字段

```
vis_backends = [
    dict(
        type='TensorboardVisBackend',
        save_dir = work_dir)
]
```

  - 运行：`tensorboard –logdir work_dir`
  - 后台展示：浏览器输入http://localhost:6006/

MLflow

- 介绍
  MLflow 是一个用于记录机器学习生命周期的开源工具，实验记录和可视化只是其中一个基础功能，因此它的可视化功能不如 Neptune 和 WandB 那样丰富灵活。MLflow 支持记录的数据类型有：
  - 指标和损失
  - 超参数和模型 config
  - Git 信息
  - Artifacts（图片、模型、数据等）

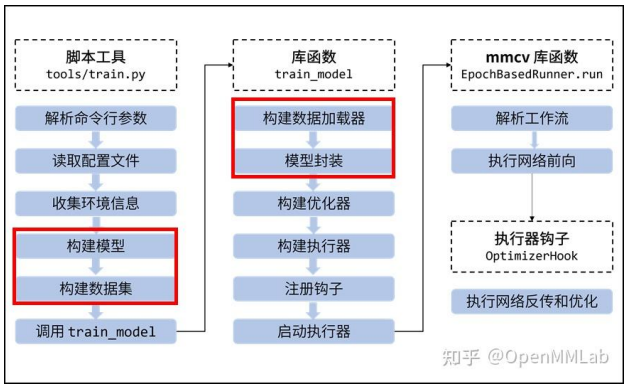
  注：MLflow 只能以 artifacts 的形式记录图片，没有交互式功能，因此很难从图片中直接获取实验数据。MLflow 也不适用于大型实验，过多的实验可能导致 UI 滞后。然而，MLflow 的主要优势在于机器学习生命周期的完整记录，包括实验可复现性的实现、模型注册、模型和数据的版本管理等。

- 使用

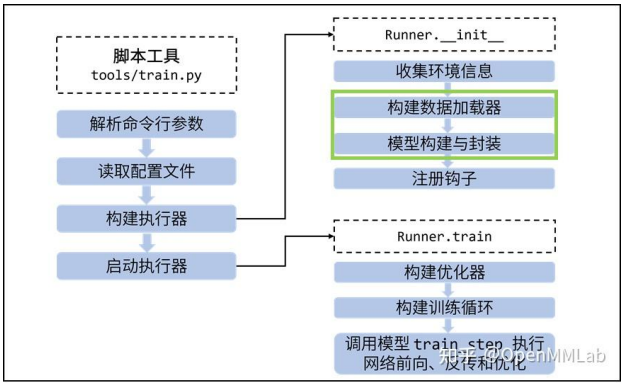
  - 安装 mlflow：pip install mlflow
  - 修改 vis_backends 字段

```
vis_backends = [
    dict(
        type='MLflowVisBackend',
        save_dir = work_dir,
        exp_name = "356542",
        run_name = "20220724",
        artifact_suffix = ('.pth'))
]
```

  - 运行：`tensorboard –logdir work_dir`
  - 后台展示
    - windows：浏览器输入http://localhost:5000/
    - linux：使用管道或者`mlflow server –host 0.0.0.0`



(a)



(b)

图 6: 全新的架构设计。图6(a)为原来的架构训练逻辑；图6(b)为新的架构训练逻辑

总而言之，在新的架构中，MMEngine 的执行器集中了所有的模块构建功能，训练脚本只用于最基本的配置解析，如图6(b)，这样新的训练流程不仅逻辑更加清晰，大大减少了代码量，还能为用户带来更方便的模型调试体验，让用户灵活地定义模型的前向和方向过程。

2.2 MMCV

特点： MMCV2.x 两大变化：模块增删 & 包名变更

2.2.1 模块增删

MMCV1.x 中主要包含 Runner、Hook、Parallel、Registry、Config、FileIO、Image/Video、CNN 和 OPS 组件。

在 MMCV2.x 中，和训练流程相关的组件被删除了，由 MMEngine 提供，只保留图像视频处理、网络基础模块和算子。除此之外，还新增了数据预处理模块（Transform），如图7。

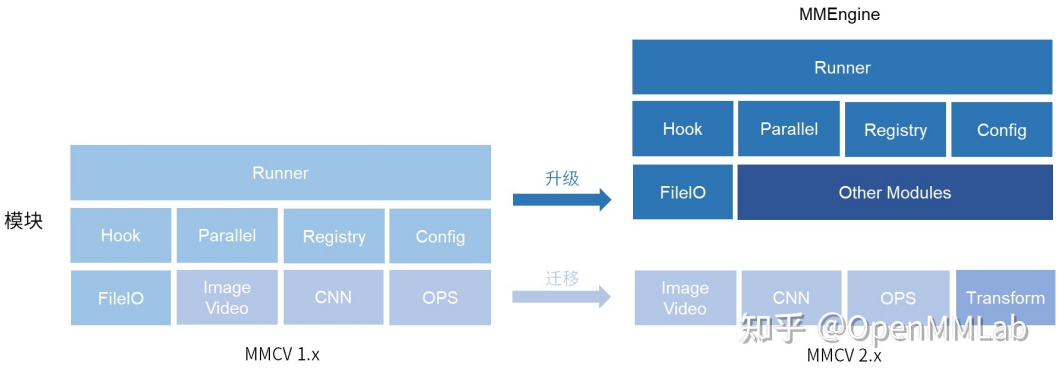


图 7: MMCV2.x 模块的变化

以分类任务为例，图8展示的是一个典型的数据流水线。

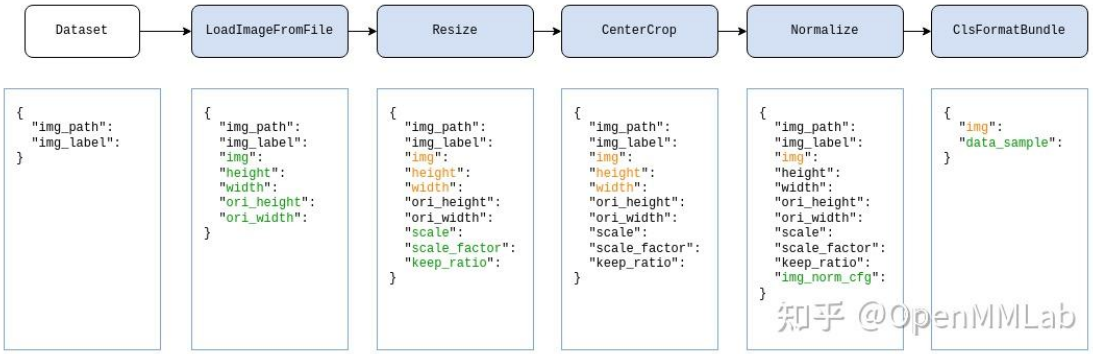


图 8: 一个典型的数据流水线。对每个样本，数据集中保存的基本信息是最左侧所示的字典，之后每经过一个由蓝色块代表的数据变换操作，数据字典中都会加入新的字段（标记为绿色）或更新现有的字段（标记为橙色）

数据加载

为了支持大规模数据集的加载，通常在 Dataset 初始化时不加载数据，只加载相应的路径。因此需要在数据流水线中进行具体数据的加载。

数据变换类	功能
LoadImageFromFile	根据路径加载图像
LoadAnnotations	加载和组织标注信息，如 bbox、语义分割图等

表 1: 数据加载

数据变换

按照功能，常用的数据变换类有数据预处理与增强、数据格式化。

数据预处理和增强通常是对图像本身进行变换，如裁剪、填充、缩放等。

数据变换类	功能
Pad	填充图像边缘
CenterCrop	居中裁剪
Normalize	对图像进行归一化
Resize	按照指定尺寸或比例缩放图像
RandomResize	缩放图像至指定范围的随机尺寸
RandomMultiscaleResize	缩放图像至多个尺寸中的随机一个尺寸
RandomGrayscale	随机灰度化
RandomFlip	图像随机翻转
MultiScaleFlipAug	支持缩放和翻转的测试时数据增强

表 2: 数据预处理及增强

数据格式化操作通常是对数据进行的类型转换。

数据变换类	功能
ToTensor	将指定的数据转换为 torch.Tensor
ImageToTensor	将图像转换为 torch.Tensor

表 3: 数据格式化

此外，数据变换还提供了四个特殊的数据变换类，主要作用是对其中定义的数据变化行为进行增强。分别是：

- **字段映射（KeyMapper）**：用于对数据字典中的字段进行映射。以 RandomFlip 为例

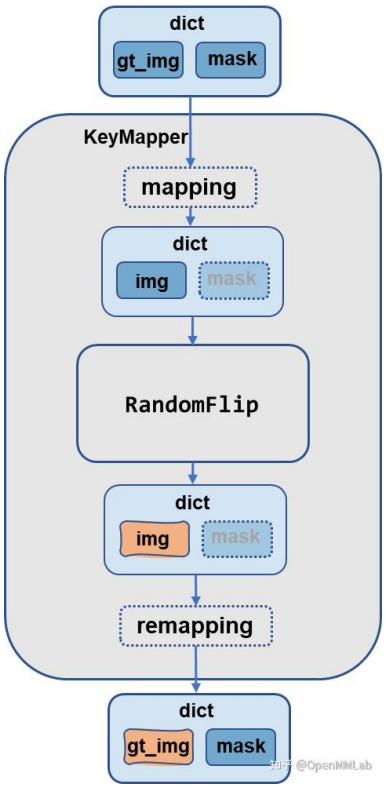


图 9: RandomFlip 要求输入的字典中包含 img 字段，但是用户的输入只包含了 gt_img 字段，这种情况下可以用 KeyMapper 将 gt_img 字段映射为 img 字段，在 RandomFlip 完成处理后再将 img 映射回 gt_img

- **随机选择（RandomChoice）**：用于从一系列数据变换组合中随机应用一个数据变换组合。以 AutoAugment 为例

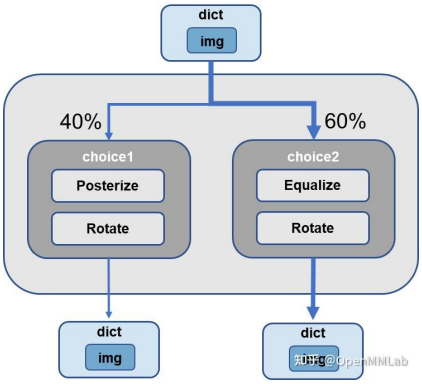


图 10: 两组 transform，其中有 40% 的概率选择第一组，60% 的概率选择第二组

- **随机执行（RandomApply）**
- **多目标扩展（TransformBroadcaster）**

2.2.2 包名变更

包名	<2.0	≥ 2.0
mmcv-full	包含 CUDA 算子	无
mmcv	不包含 CUDA 算子	包含 CUDA 算子
mmcv-lite	无	不包含 CUDA 算子

表 4: 为避免用户混淆，将包名重命名

3 MMYOLO

问题 3.1

Q：为啥要推出 MMYOLO？为何要单独开一个仓库而不是直接放到 MMDetection 中？
A：答案可以归纳为以下三点：

- **统一运行和推理平台**：目前目标检测领域出现了非常多 YOLO 的改进算法，并且非常受大家欢迎，但是这类算法基于不同框架不同后端实现，存在较大差异，缺少统一便捷的从训练到部署的公平评测流程
- **协议限制**：众所周知，YOLOv5 以及其衍生的 YOLOv6 和 YOLOv7 等算法都是 GPL3.0 协议，不同于 MMDetection 的 Apache 协议。由于协议问题，无法将 MMYOLO 直接并入 MMDetection 中
- **多任务支持**：MMYOLO 任务不局限于 MMDetection，还会支持基于 MMPose 实现关键点相关的应用，以及基于 MMTracking 实现追踪相关的应用，因此不太适合直接并入 MMDetection

3.1 训练技巧

3.1.1 提升检测性能

- **多尺度训练**：在 YOLO 中大部分模型的训练输入都是单尺度的 640x640，原因有两个方面：1. 单尺度训练速度快。当训练 epoch 在 300 或者 500 的时候训练效率是用户非常关注的，多尺度训练会比较慢。2. 训练 pipeline 中隐含了多尺度增强，等价于应用了多尺度训练，典型的如 Mosaic、RandomAffine 和 Resize 等，故没有必要再次引入模型输入的多尺度训练。因此，如果直接在 YOLOv5 的 DataLoader 输出后再次引入多尺度训练增强实际性能提升非常小，但是这不代表用户自定义数据集微调模式下没有明显增益。如果想在 MMYOLO 中对 YOLO 系列算法开启多尺度训练，可以参考多尺度训练文档
- **使用 Mask 标注优化目标检测性能**
- **训练后期关闭强增强提升检测性能**：该策略是在 YOLOX 算法中第一次被提出，可以极大地提升检测性能。论文中指出虽然 Mosaic+MixUp 可以极大地提升目标检测性能，但是它生成的训练图片远远脱离自然图片的真实分布，并且 Mosaic 大量的裁剪操作会带来很多不准确的标注框，所以 YOLOX 提出在最

后 15 个 epoch 关掉强增强，转而使用较弱的增强，从而让检测器避开不准确标注框的影响，在自然图片的数据分布下完成最终的收敛。

4 MMSegmentation

4.1 数据结构

MMSegmentation1.0 引入了 SegDataSample 数据结构，将语义分割中的数据封装起来，用于各个功能模块之间的数据传递，SegDataSample 里面的字段有三个：gt_sem_seg，pred_sem_seg 和 seg_logits。前两个分别是标签和模型预测对应的分割掩膜（segmeation mask），seg logits 是模型最后一层没有经过归一化的输出。

```
if C > 1:
    i_seg_pred = i_seg_logits.argmax(dim=0, keepdim=True)
else:
    i_seg_logits = i_seg_logits.sigmoid()
    i_seg_pred = (i_seg_logits > self.decode_head.threshold).to(i_seg_logits)
data_samples[i].set_data({
    'seg_logits': PixelData(**{'data': i_seg_logits}),
    'pred_sem_seg': PixelData(**{'data': i_seg_pred})
```

4.2 数据集和数据变化

MMSegmentation1.0 新定义了 BaseSegDataset，规范了语义分割数据集功能和接口，是 MMEngine 中 BaseDataset 的子类。数据集主要的功能是加载数据信息，数据信息有两种，一种是数据集的元信息，包括类别信息和调色板信息，就是渲染时类别对应的颜色；另一种是数据信息，保存了具体数据集中图片路径和对应的标签路径。

一个典型的语义分割模型训练时的数据变换流水线，如图11所示。

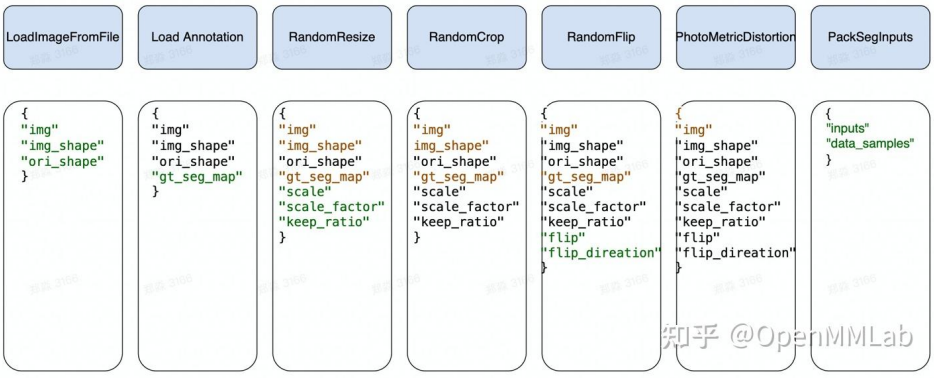


图 11: 一个典型的语义分割模型训练时的数据变换流水线。对每个样本，数据集中保存的基本信息是最左侧所示的字典，之后每经过一个由蓝色块代表的数据变换操作，数据字典中都会加入新的字段（标记为绿色）或更新现有的字段（标记为橙色）

4.3 模型

MMSegmentation 中将语义分割算法模型称为 segmentor，共 6 个模块，分别是：

- data_preprocessor，详见2.1.1
- Backbone，常见的模型有 ResNet，Swin transformer 等
- Neck，常见的网络有 Feature Pyramid Network（FPN）
- decode_head
- auxiliary_head（可选）
- loss

segmentor 的模型结构根据是否由多个 decode_head 集联，分为 encoder_decoder 和 cascade_encoder_decoder 两种。

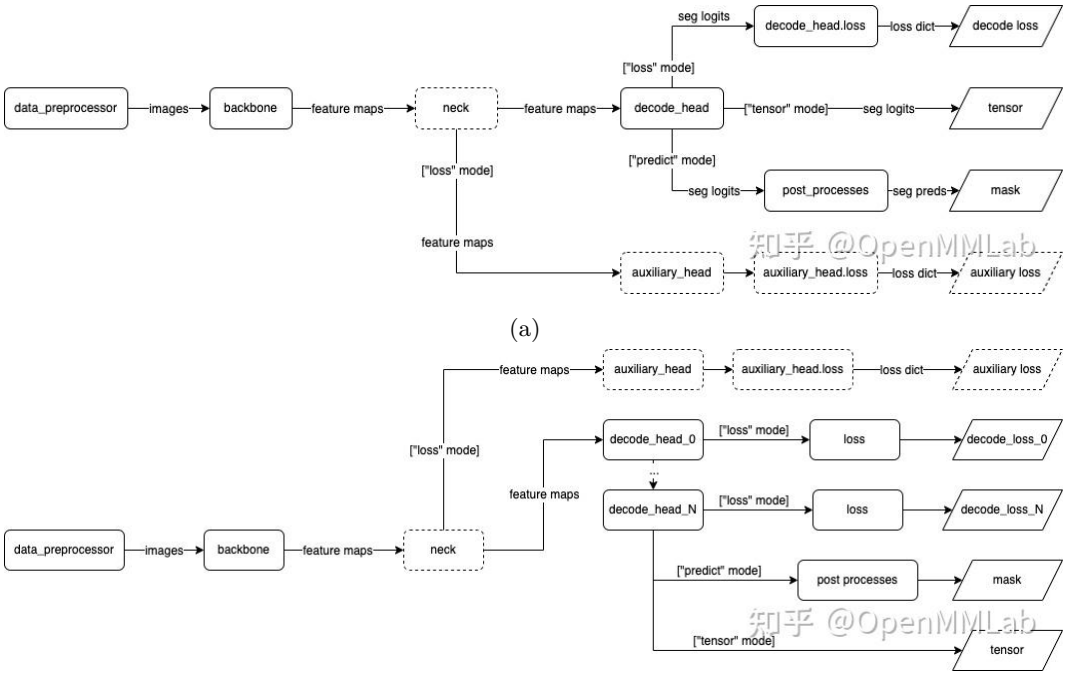


图 12: 语义分割数据流。图12(a)为 encoder_decoder 数据流；图12(b)为 cascade_encoder_decoder 数据流

数据流协议分为训练和测试两种。

如图13(a)，训练的时候，dataloader 搬运经过 data transforms 处理的数据，传给模型 train_step 方法，模型里会先调数据预处理模块，再传给模型的 forward 函数，前传并计算损失。这个 loss dict 会经过 parse_losses 模块解析，得到一个 loss scalar，然后在 opitimizer warpper 里的 update_params 对模型反传，计算梯度，并更新参数。

如图13(b)，测试时，数据会传给模型的 test_step 方法，同样是先经过预处理，predict 输出 datasample。这里的 datasample 就是输入网络的 datasample，只不过新增了 pred_sem_seg 和 seg_logits 两个字段用以保存网络的预测结果。将这个修改后的 data sample 和 inputs 送到评测器计算评测指标，或者送到可视化器中进行处理。

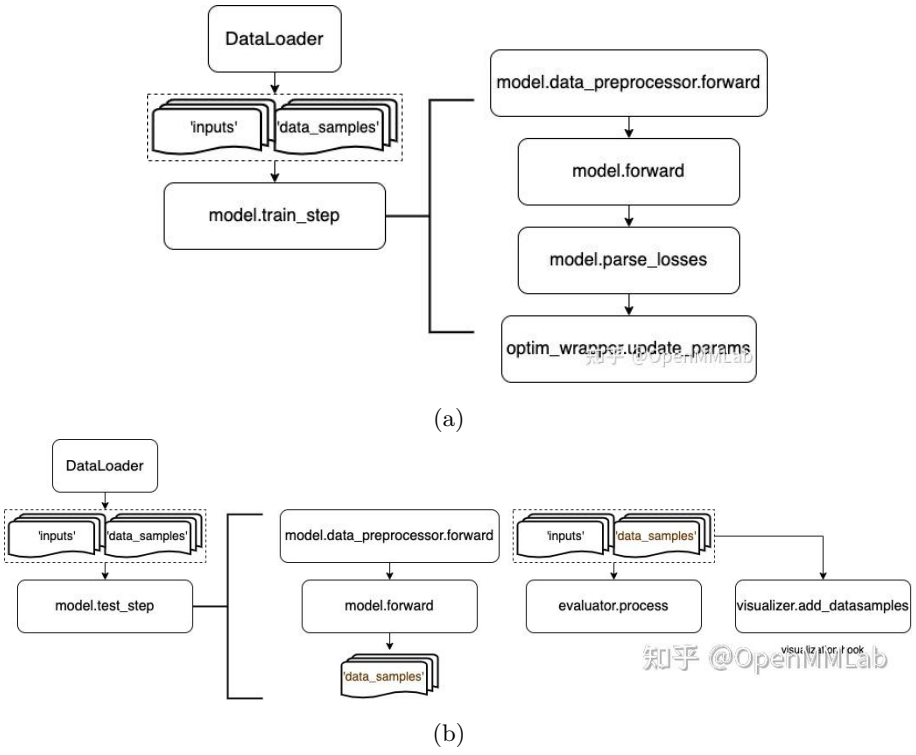


图 13: 数据流协议。图13(a)为训练时的数据流；图13(b)为测试时的数据流

5 OpenMMLab VS PaddlePaddle

OpenMMLab & PaddlePaddle 均可分为定制和通用两个部分。

流程	详情	
	定制	**通用**
	设计网络结构	网络模块 & 原子函数
步骤 1	指定 Loss 函数	Loss 函数实现
	指定优化算法	优化算法实现
步骤 2	提供数据格式 & 接入数据方式	为模型批量送入数据 (Feed、Py_reader)
步骤 3	单机和多机配置	单机到多机转换 (transpile) & 训练程序
步骤 4	确定保存模型和加载模型的环节点	保存模型
步骤 5	指定评估指标	指标实现 & 图形化工具
步骤 6	主程序	-
附	个性化评估定制	性能分析

表 5: OpenMMLab & Paddle 算法框架系列 Star 对比

系列	OpenMMLab	Paddle
OS	4.3k	**5.3k**
OC	1.5k	**4.3k**
OD	**21k**	8.3k

OpenMMLab 基于 PyTorch，是一个适用于学术研究和工业应用的开源算法体系【迄今最为完备，偏向科研】

PaddlePaddle 的 API 代码风格基本模仿 PyTorch【学术上几乎没有份额，工业上依靠百度自己开源的项目】，上手快，使用下来最大的问题是 Paddle 性能有点拉跨，但是由于是百度自己开源的，相对稳定。