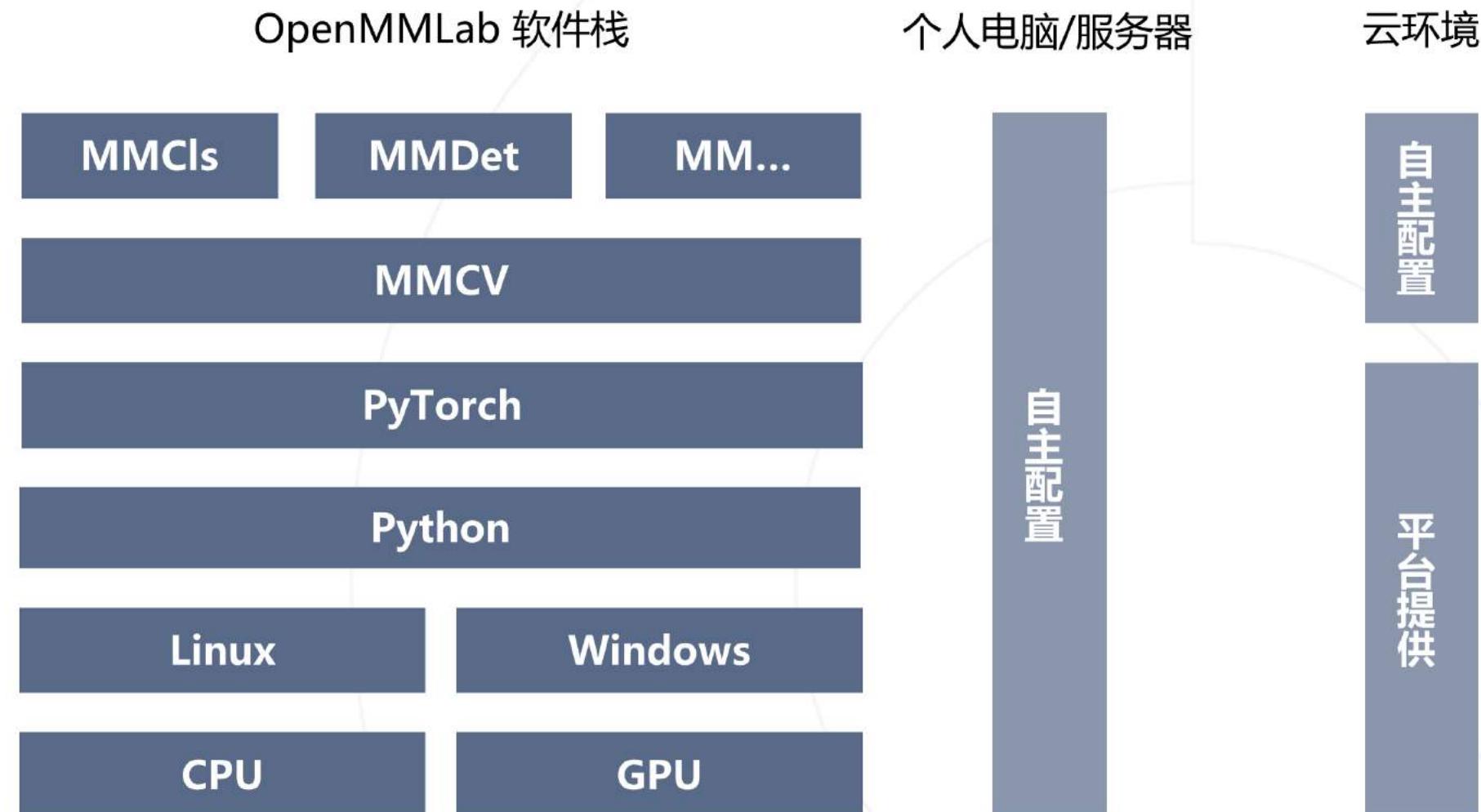


通用视觉框架OpenMMLab
实践1 搭建 OpenMMLab 运行环境



Colaboratory 介绍

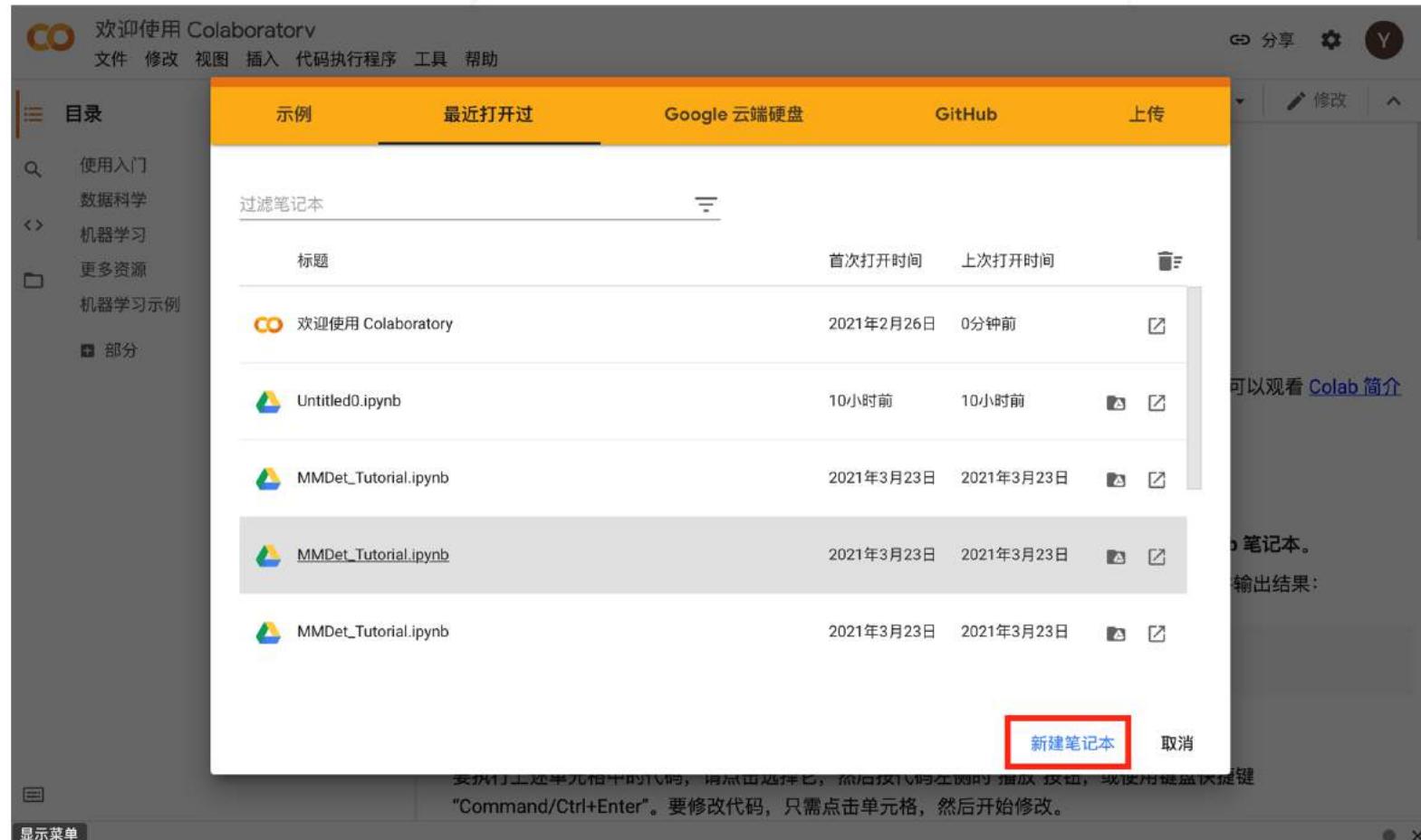
Colaboratory 是一个云端运行的 Python 交互式编程环境。借助 Colaboratory，我们可以编写和执行 Python 代码、保存和共享分析结果，以及利用其强大的计算资源。
所有这些都不需要进行任何复杂的配置，仅仅通过浏览器就可以使用，而且完全免费。



Colab 的网址为 colab.research.google.com，这里我们需要连接国际互联网，从而畅通运行。



点击右上角登录，登录 Google 账户。



登陆后我们会进入到当前界面，我们可以通过多种方式打开 Jupyter Notebook：从 Google Drive 中直接打开文件、从 GitHub 中打开文件、从本地上传文件。这里我们从新建笔记本开始，点击右下角 “**新建笔记本**” 开始使用 Colab。



至此我们已经打开了一个新的 Jupyter Notebook。Jupyter Notebook 是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。

保存笔记本

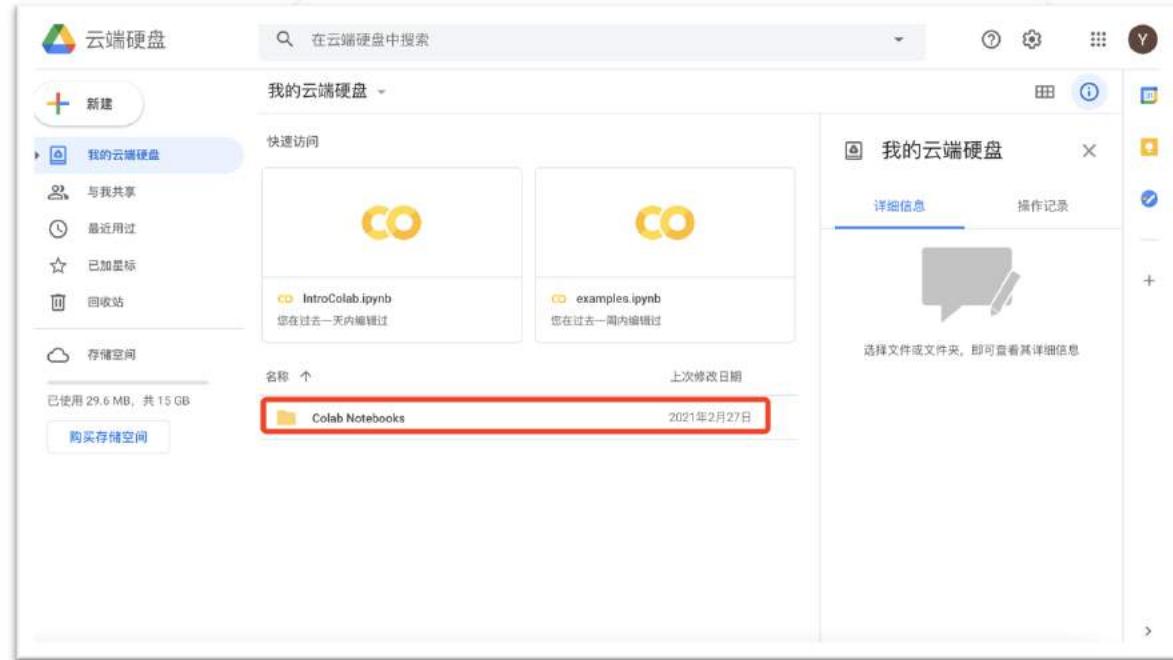
This screenshot shows a Jupyter Notebook interface with a file named 'examples.ipynb'. The notebook contains Python code for printing 'Hello, world!' and calculating the first five Fibonacci numbers. A status bar at the bottom indicates '正在保存...' (Saving...). A red box highlights the status bar.

```
print('Hello, world!')  
Hello, world!  
  
def fibonacci(n):  
    assert n >= 0  
    a, b = 0, 1  
    for _ in range(n):  
        a, b = b, a + b  
    return a  
  
for i in range(5):  
    print(fibonacci(i))  
  
0  
1  
1  
2  
3  
  
import torch  
torch.__version__  
'1.8.1+cu101'
```

This screenshot shows the same Jupyter Notebook after saving. The status bar now displays '已保存所有更改' (All changes saved). A red box highlights the status bar. A message box at the bottom says '已成功保存!' (Saved successfully!).

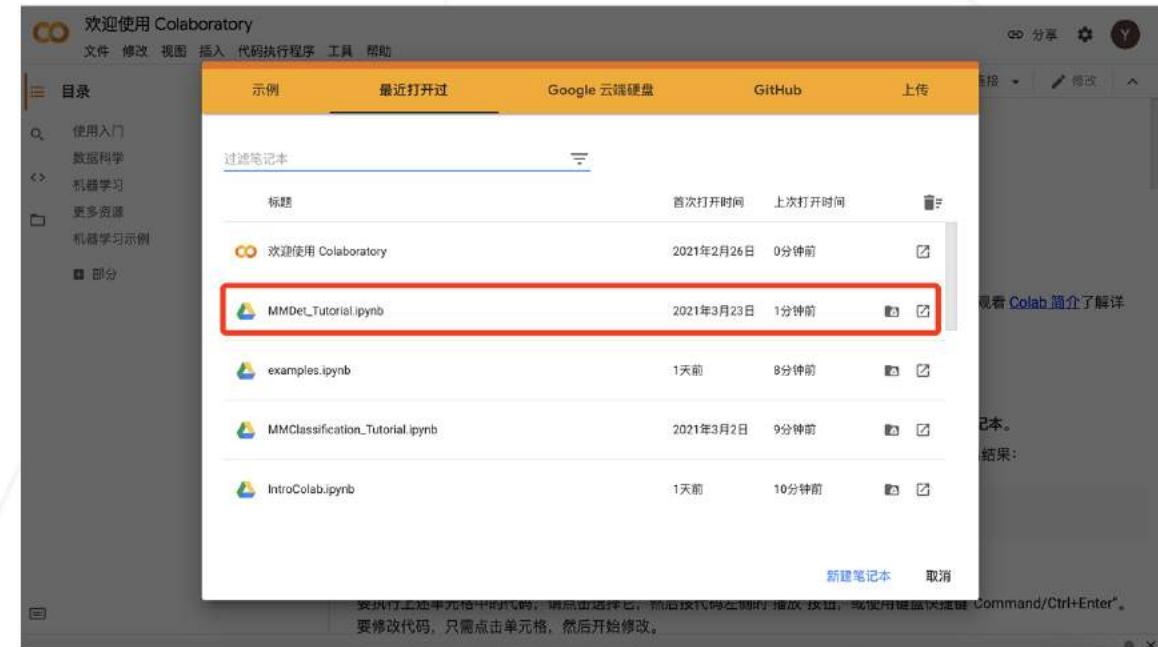
```
print('Hello, world!')  
Hello, world!  
  
def fibonacci(n):  
    assert n >= 0  
    a, b = 0, 1  
    for _ in range(n):  
        a, b = b, a + b  
    return a  
  
for i in range(5):  
    print(fibonacci(i))  
  
0  
1  
1  
2  
3  
  
import torch  
torch.__version__  
'1.8.1+cu101'
```

我们通过“ctrl+s”来保存当前正在编辑的 Jupyter Notebook，按下时会显示“正在保存”，并于结束时显示“保存成功”。

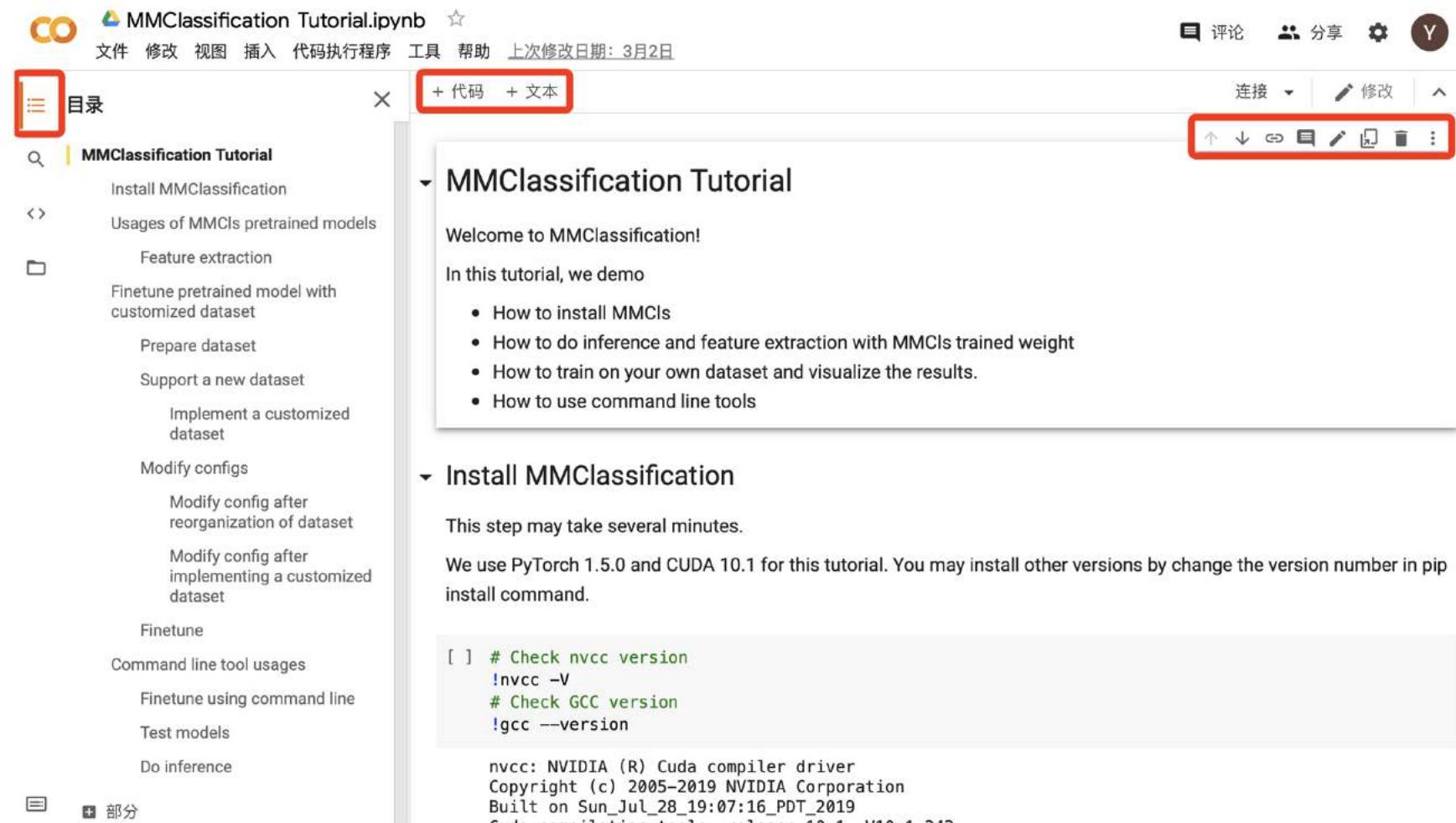


Colab 会自动将我们的 Jupyter Notebook 保存在相应账户 Google Drive 中的 Colab Notebooks 文件夹下。

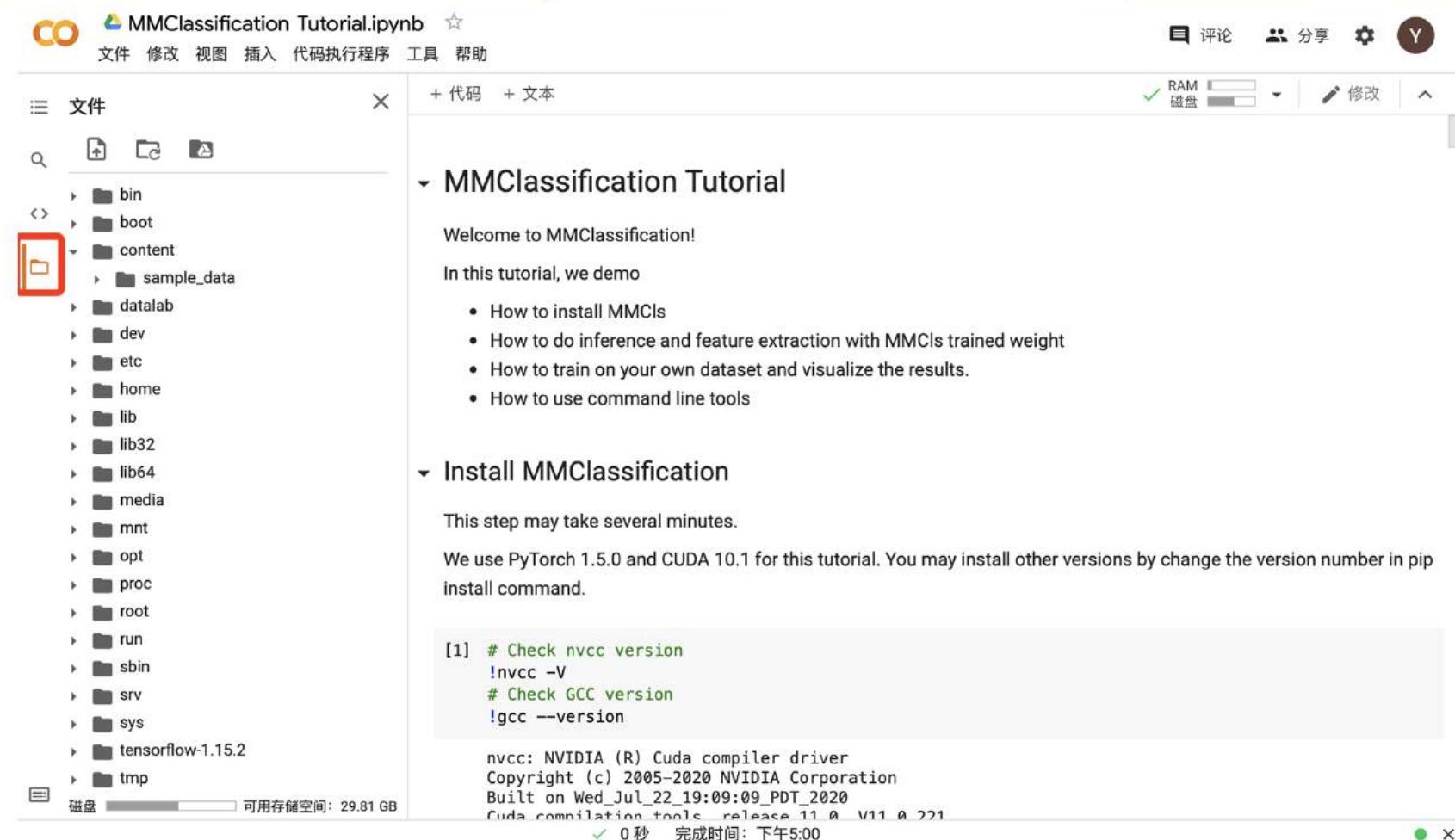
打开已有笔记本



再次打开 Jupyter Notebook 时, 我们可以从 Google Drive 中直接双击相应文件, 也可以在 Colab 中通过“文件” - “打开笔记本” - “最近打开过” 中选择相应文件。



点击左侧“目录”可以查看当前 Jupyter Notebook 行文结构；点击上方的“代码”、“文本”可以添加代码块、文本块，这里的文本块采用 Markdown 语法；右上方是当前代码块的选项栏。



点击左侧“文件夹”可以查看文件夹结构。这里我们可以看到 Linux 的目录结构，一般情况下，我们默认的工作目录是在 `/content/` 文件夹下。接下来我们将展示如何配置 OpenMMLab 的应用环境，并以 MMClassification 为例。

```
[3] # Check pytorch  
!pip list | grep torch  
  
torch                  1.8.1+cu101  
torchsummary            1.5.1  
torchtext               0.9.1  
torchvision             0.9.1+cu101
```

Colab 在默认的环境中已经集成了一系列 Python 包，例如 PyTorch、Tensorflow 等深度学习框架。我们可以使用 pip list 指令来查看已有的 python 包。OpenMMLab 基于 PyTorch。可以看到 Colab 自带的 PyTorch 版本是最新的 1.8.1。

不同于 Python 代码，调用 pip 需要在开头加上英文叹号 “!”，这是因为 pip 不是 Python 代码，而是 Python 以外的 Linux 环境中的命令。

需要注意的是，在 Linux 中，我们会使用 cd 命令切换目录，但在 Colab 的代码中，如果希望切换目录，不能使用 !cd，而应该使用 IPython 的魔术命令 (英文magics) %cd。



```
# Check nvcc version  
!nvcc -V
```

```
↳ nvcc: NVIDIA (R) Cuda compiler driver  
Copyright (c) 2005-2020 NVIDIA Corporation  
Built on Wed_Jul_22_19:09:09_PDT_2020  
Cuda compilation tools, release 11.0, V11.0.221  
Build cuda_11.0_bu.TC445_37.28845127_0
```

```
[2] # Check GCC version  
!gcc --version
```

```
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0  
Copyright (C) 2017 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

一些情况下，我们可以通过`nvcc`和`gcc`命令来分别查看 CUDA 及 GCC 的相关信息。Nvcc 是 CUDA 代码的编译器，GCC 是 C++ 代码的编译器，在个别情况下，我们需要自己从源代码编译 OpenMMLab 中的 MMCV 工具包。

Colab 上会安装较新版本的 CUDA 和 GCC，在本教程制作时，CUDA 的版本是 11.0，GCC 的版本是 7.5.0。



PyTorch 支持 GPU 加速计算，而 Colab 提供免费的 GPU、TPU 硬件加速，默认情况下是关闭的，我们需要手动开启。我们可以在：“修改” - “笔记本设置” 中查看硬件加速选项。

这里我们使用“GPU”硬件加速，并点击保存。

```
# Test gpus
import torch

num_gpu = torch.cuda.device_count()
device = torch.device("cuda:0" if (torch.cuda.is_available() and num_gpu > 0) else "cpu")
print("Number of gpu: {}\nDevice: {}".format(num_gpu, device))
```

```
Number of gpu: 1
Device: cuda:0
```

```
▶ torch.cuda.get_device_name(0)
```

```
⇨ 'Tesla T4'
```

```
[8] print(torch.rand(3,3).cuda())
```

```
tensor([[0.3195, 0.1171, 0.4933],
       [0.0991, 0.0286, 0.4879],
       [0.6125, 0.4662, 0.6659]], device='cuda:0')
```

当前可以使用1个 GPU 进行加速。我们可以通过最下方的例子看到，PyTorch 可以正常使用 GPU 加速。

配置 MMCV 与 OpenMMLab

```
# Install mmcv
!pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.0/index.html

[1] Looking in links: https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.0/index.html
Collecting mmcv-full
  Downloading https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.0/mmcv\_full-1.3.0-cp37-cp37m-manylinux1\_x86\_64.whl (25.6M)
    |██████████| 25.6MB 114kB/s
Collecting yapf
  Downloading https://files.pythonhosted.org/packages/5f
    |██████████| 194kB 26.0MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/site-packages (1.19.2)
Requirement already satisfied: opencv-python>=3 in /usr/local/lib/python3.7/site-packages (3.4.19.1)
Collecting addict
  Downloading https://files.pythonhosted.org/packages/6a
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/site-packages (6.2.2)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/site-packages (5.4.1)
Installing collected packages: yapf, addict, mmcv-full
Successfully installed addict-2.4.0 mmcv-full-1.3.0 yapf-0.5.1.0
```

[3] # Check pytorch
!pip list | grep torch

torch	1.8.1+cu101
torchsummary	1.5.1
torchtext	0.9.1
torchvision	0.9.1+cu101

MMCV 是 OpenMMLab 中的核心模块，我们需要安装 MMCV 来保证其他库的正常使用。

这里我们通过 [-f https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.0/index.html](https://download.openmmlab.com/mmcv/dist/cu101/torch1.8.0/index.html) 来下载对应版本 MMCV。

MMCV 的版本号分为三部分，一部分是 MMCV 自身的版本，目前最新版本为 1.3.0，不指定则安装最新版本。第二部分是 PyTorch 的版本 1.8.0，第三部分是 CUDA 版本 cu101，这里我们需要选择与 Colab 对应的版本。如果实在不能匹配，也可以使用 pip 命令安装旧版本的 PyTorch。更多关于 MMCV 的详细信息可以参考 <https://github.com/open-mmlab/mmcv> 中安装说明的部分。

```
▶ # Install mmcls
!git clone https://github.com/open-mmlab/mmclassification.git
%cd mmclassification

!pip install -e .

↳ Cloning into 'mmclassification'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (57/57), done.
remote: Total 2013 (delta 30), reused 68 (delta 27), pack-reused 1928
Receiving objects: 100% (2013/2013), 1.97 MiB | 14.94 MiB/s, done.
Resolving deltas: 100% (1213/1213), done.
/content/mmclassification
Obtaining file:///content/mmclassification
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from mmcls==0.10.0) (3.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from mmcls==0.10.0) (1.19.5)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->mmcls==0.10.0) (1.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->mmcls==0.10.0) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplot
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->mmcls==0.10.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib->mmcls==0.10.0) (1.
Installing collected packages: mmcls
  Running setup.py develop for mmcls
Successfully installed mmcls
```

MMClassification 是用于图像分类任务的工具包，是 OpenMMLab 的重要部分。我们可以使用 git 克隆代码库，并通过 pip 配合 -e 参数从源代码安装。OpenMMLab 中其他工具包的安装也遵循同样的流程。更多信息可以参考 <https://github.com/open-mmlab/mmclassification>。

```
▶ # Check Pytorch installation
    import torch, torchvision
    print(torch.__version__, torch.cuda.is_available())

    # Check MMClassification installation
    import mmcls
    print(mmcls.__version__)

↳ 1.8.1+cu101 True
    0.10.0
```

开始使用 MMClassification 之前，检查其是否正确安装。在这里我们可以看到 mmcls 可以正常 import，并输出正确的版本号。

```
▶ # Download pretrained model
!mkdir checkpoints
!wget https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_batch256_imagenet_20200708-cfb998bf.pth -P checkpoints

⇒ --2021-04-08 02:43:49-- https://download.openmmlab.com/mmclassification/v0/resnet/resnet50\_batch256\_imagenet\_20200708-cfb998bf.pth
Resolving download.openmmlab.com (download.openmmlab.com)... 47.252.96.35
Connecting to download.openmmlab.com (download.openmmlab.com)|47.252.96.35|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102491894 (98M) [application/octet-stream]
Saving to: 'checkpoints/resnet50_batch256_imagenet_20200708-cfb998bf.pth'

resnet50_batch256_i 100%[=====] 97.74M 9.22MB/s    in 11s

2021-04-08 02:44:02 (8.80 MB/s) - 'checkpoints/resnet50_batch256_imagenet_20200708-cfb998bf.pth' saved [102491894/102491894]

[14] # Do inference
!python demo/image_demo.py demo/demo.JPEG configs/resnet/resnet50_b32x8_imagenet.py checkpoints/resnet50_batch256_imagenet_20200708-cfb998bf.pth
```

MMClassification 可以完成多项图像分类任务，并包含多种预训练模型，这里我们以 ResNet50 来对图片进行简单的预测。更多关于 MMClassification 以及其他工具包的使用会在后续课程中讲述，同学们也可以参考 GitHub 上的代码文档。

同学们可以参考这份材料或 GitHub 上的文档以及 Tutorial，在 Colab 或者自己的 Linux 环境下配置 OpenMMLab 的运行环境。

如果有问题可以加入我们的QQ群提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。

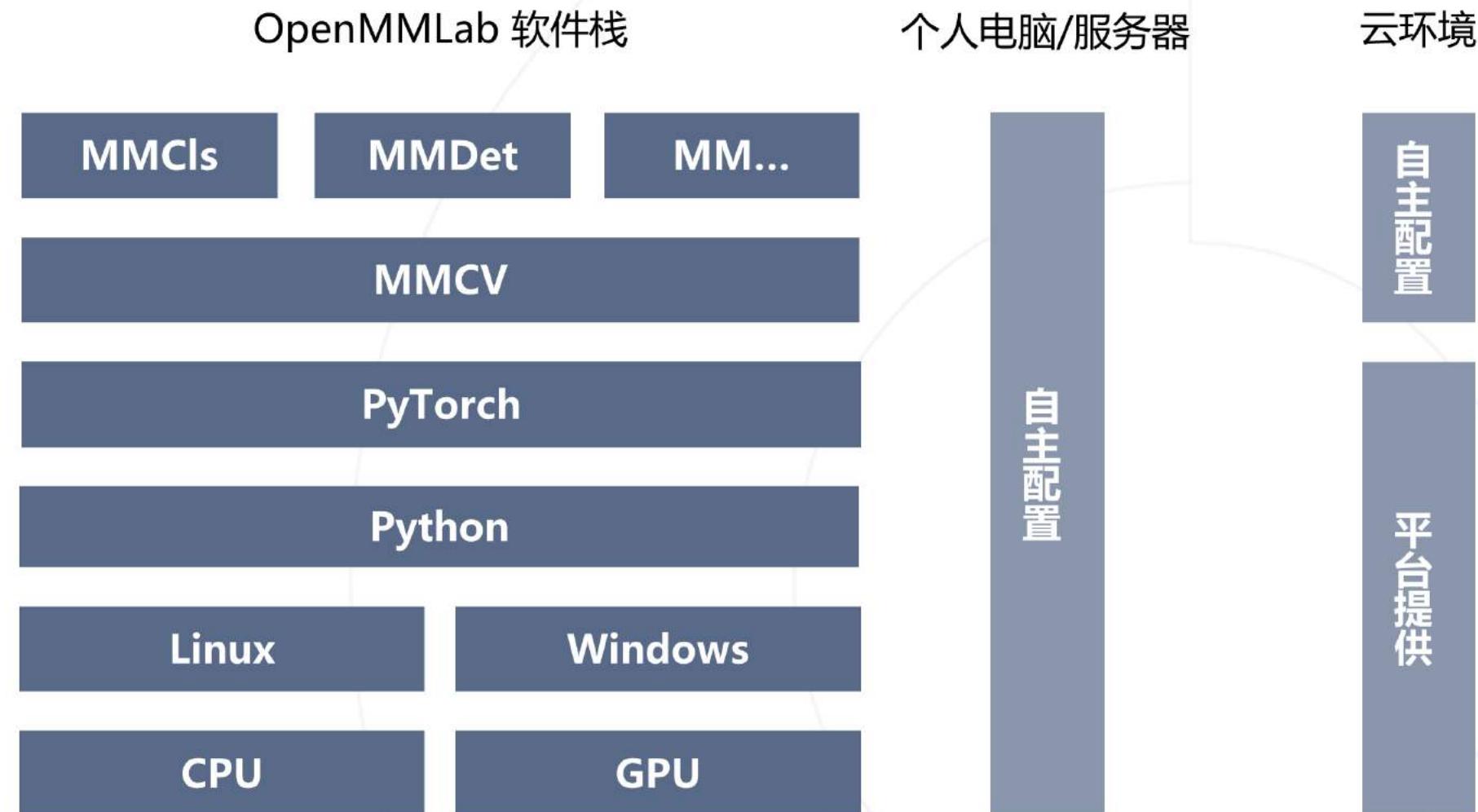


QQ

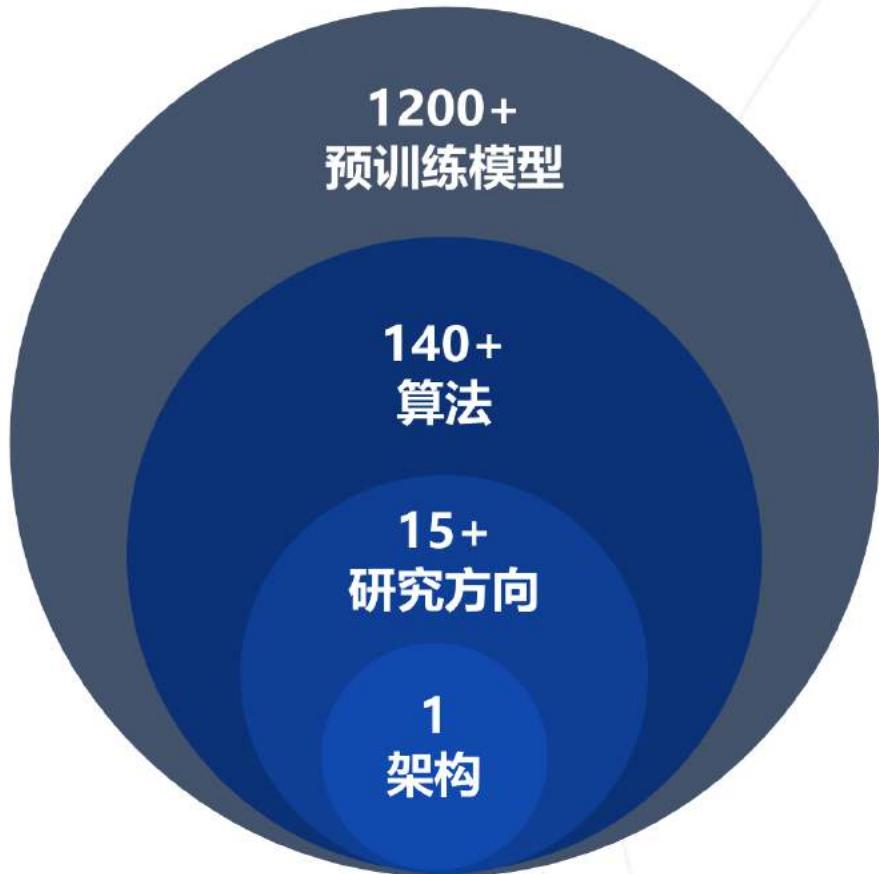
通用视觉框架OpenMMLab 实践2 MMClassification

王若晖
2021年4月

- MMClassification 运行环境搭建
- 使用预训练模型完成图像分类
- 训练自己的图像分类模型



- MMClassification 运行环境搭建
 - Python 和 PyTorch 环境配置（或使用云平台提供的环境）
 - MMCV 安装
 - MMClassification 安装
- 使用预训练模型完成图像分类
 - 配置文件
 - 预训练模型
 - Python API
- 训练自己的图像分类模型
 - 修改配置文件
 - 使用预训练模型
 - 使用训练和测试工具



1 架构

- 所有项目基于一致架构开发

15+ 研究方向

- 涵盖多个研究热点方向，算法覆盖完善

140+ 算法

- 包括 140+ 先进算法，性能领先

1200+ 预训练模型

- 拥有超过 1200 个预训练模型，真正实现开箱即用

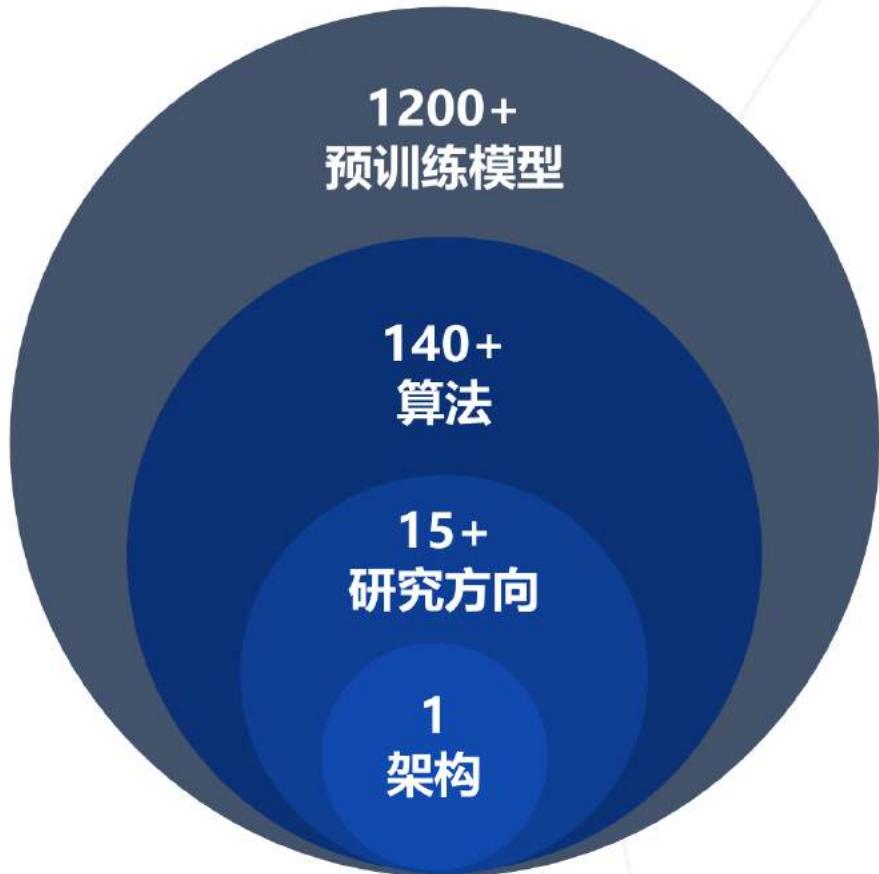
更多内容可参见 GitHub 上的代码文档以及教程。
如有问题欢迎加入我们的QQ群进行提问。



通用视觉框架OpenMMLab 实践3 MMDetection 上

王若晖
2021年4月

- MMDetection 项目一览
- MMDetection 运行环境搭建
- 使用 Faster RCNN 模型对单张图像进行推理
 - 推理结果解析
 - 模型结构解析
- 使用 RPN 模型对单张图像进行推理
 - 理解提议框的概念



1 架构

- 所有项目基于一致架构开发

15+ 研究方向

- 涵盖多个研究热点方向，算法覆盖完善

140+ 算法

- 包括 140+ 先进算法，性能领先

1200+ 预训练模型

- 拥有超过 1200 个预训练模型，真正实现开箱即用

更多内容可参见 GitHub 上的代码文档以及教程。
如有问题欢迎加入我们的QQ群进行提问。



通用视觉框架OpenMMLab 实践4 MMDetection 下

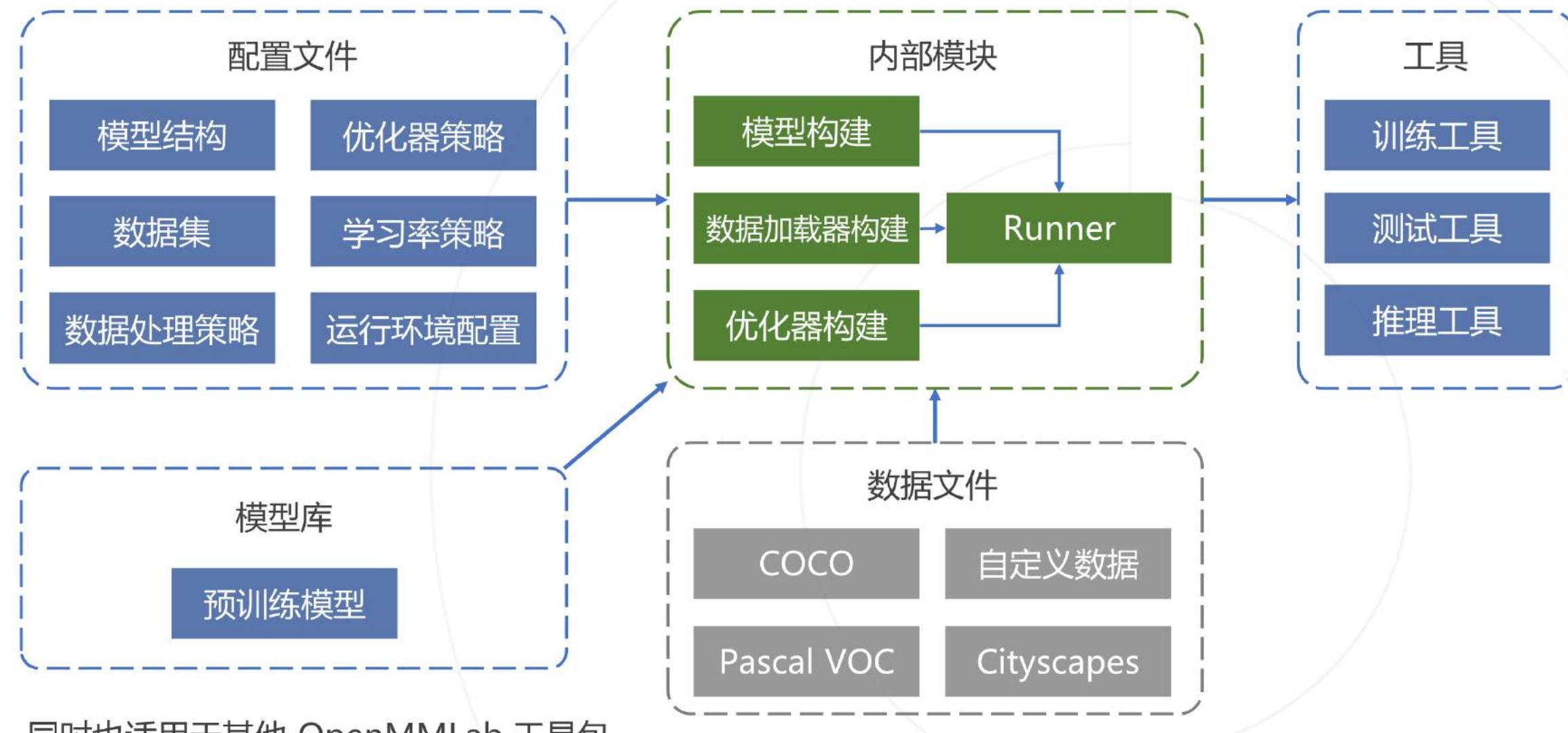
王若晖
2021年5月

④ 本节内容：

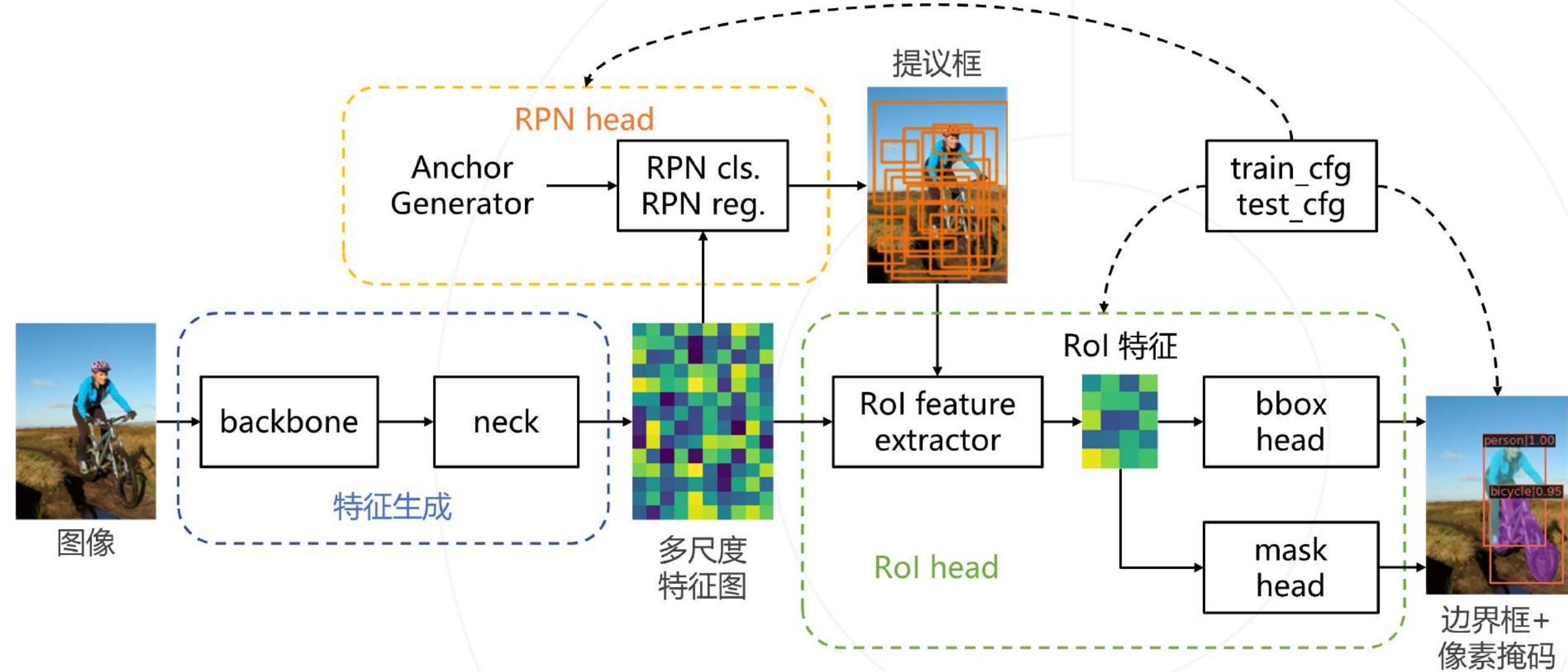
- RetinaNet 模型配置文件解读
- Mask RCNN 模型配置文件解读
- COCO 数据集介绍
- 数据配置文件解读
- 优化器配置文件解读
- 使用自定义数据集训练目标检测模型
 - 支持自定义数据集的三种方法

④ 上节回顾：

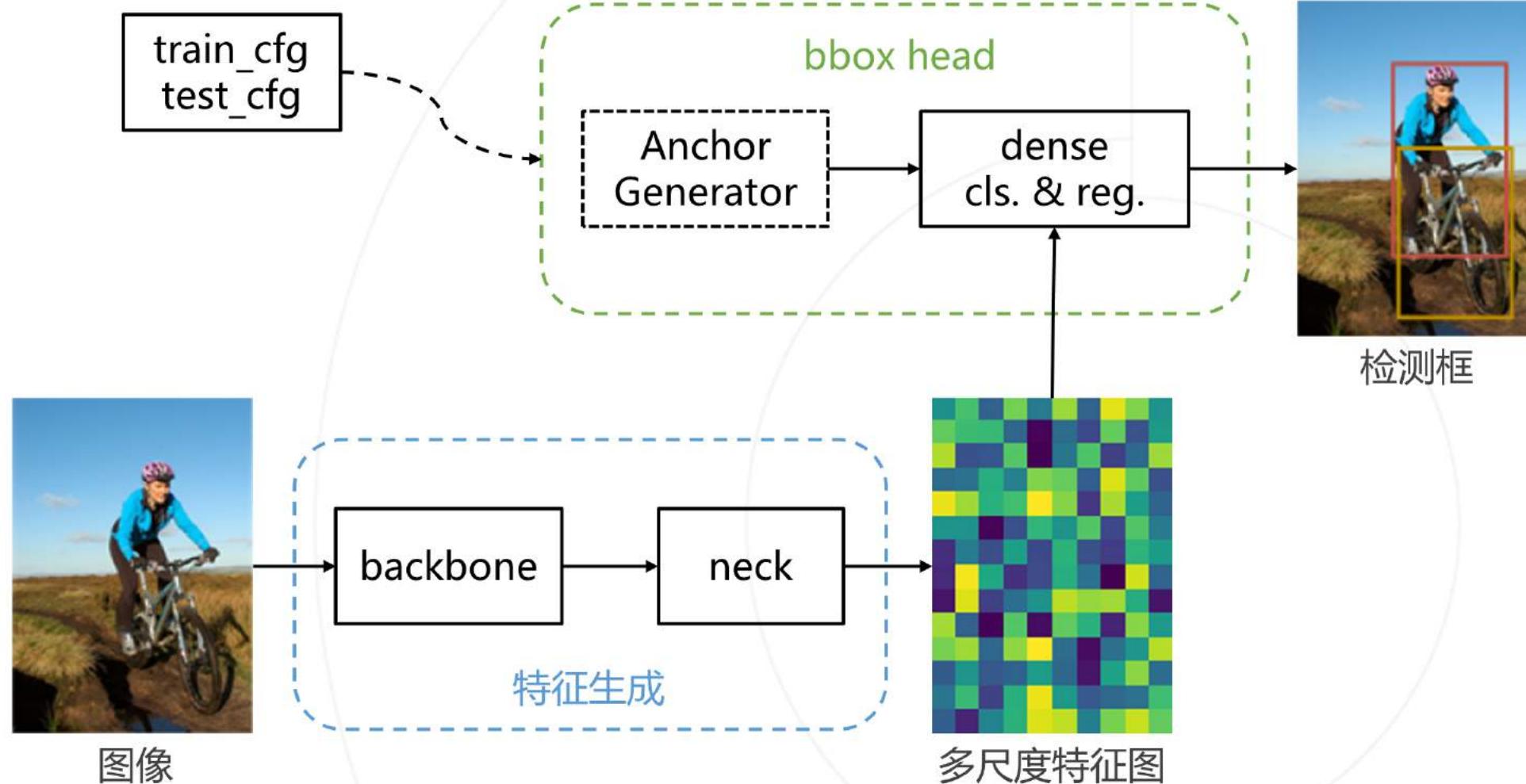
- MMDetection 项目一览
- MMDetection 运行环境搭建
- 使用 Faster RCNN 模型和 RPN 模型对单张图像进行推理
 - 推理结果的解析
- Faster RCNN 模型配置文件解读



两阶段检测器的构成



单阶段检测器的构成



ResNet 50 模型

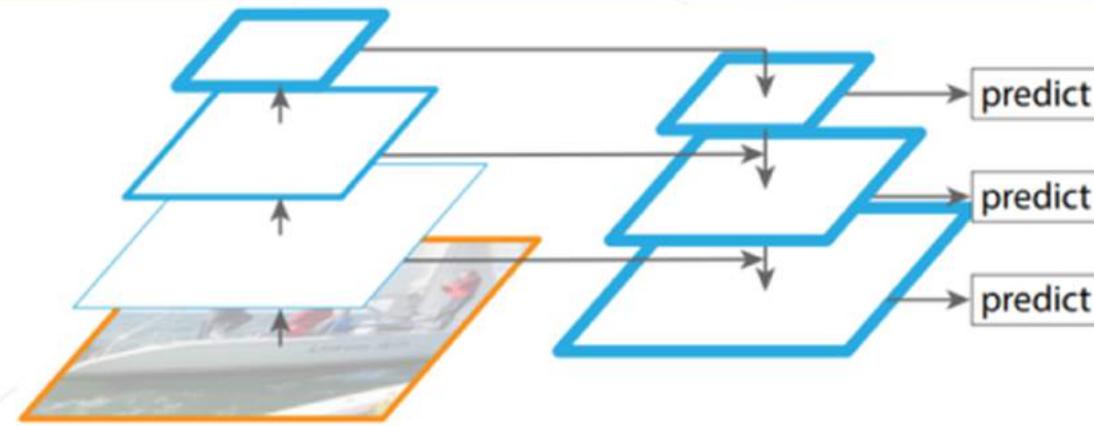
使用全部4组卷积层
并输出4组卷积层的特征

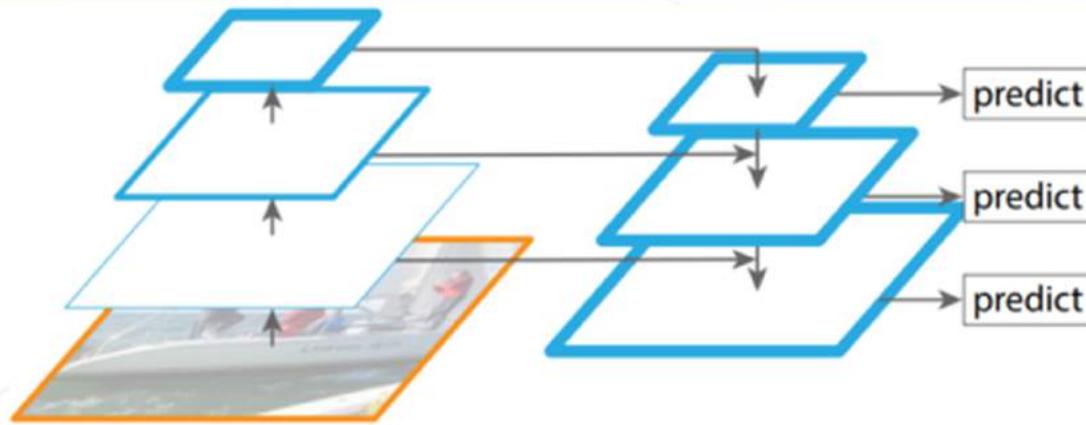
训练时冻结第1组卷积层参数

使用BN层

冻结BN层的统计量

```
backbone=dict(  
    type='ResNet',  
    depth=50,  
    num_stages=4,  
    out_indices=(0, 1, 2, 3),  
    frozen_stages=1,  
    norm_cfg=dict(type='BN', requires_grad=True),  
    norm_eval=True,  
    style='pytorch'),
```





neck=dict (

 FPN模型 type='FPN',

每级特征图的输入通道数 in_channels=[256, 512, 1024, 2048],

FPN的输出通道数 out_channels=256,

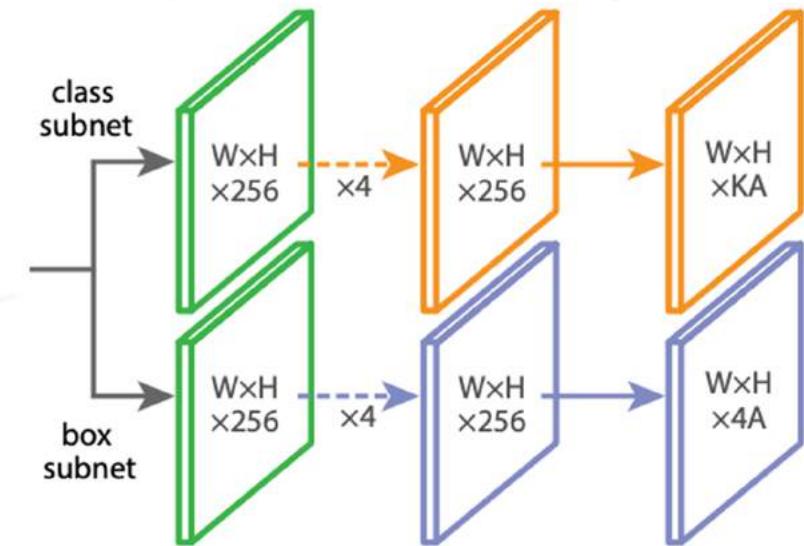
使用backbone的第2层到最后
一层特征图构建特征金字塔

在backbone的最高层特征图上
使用卷积层生成额外的特征图

输出5级特征金字塔 num_outs=5),

类别数
输入通道数
中间卷积层的个数
中间特征通道数
锚框生成
在基础锚框尺寸的基础上产生 $2^{\frac{1}{3}}, 2^{\frac{2}{3}}$ 比例放大的锚框
锚框的长宽比
不同尺度特征图对应的锚框在原图上的步长

```
bbox_head=dict(
    type='RetinaHead',
    num_classes=80,
    in_channels=256,
    stacked_convs=4,
    feat_channels=256,
    anchor_generator=dict(
        type='AnchorGenerator',
        octave_base_scale=4,
        scales_per_octave=3,
        ratios=[0.5, 1.0, 2.0],
        strides=[8, 16, 32, 64, 128]),
```



- 边界框编码器**
- RCNN的编码方式
- 偏移量的归一化均值
- 偏移量的归一化标准差

- 分类损失函数**
- 使用 Focal Loss
- 使用 Sigmoid 激活函数
K个二分类问题
- Focal loss 的超参设置

- 边界框损失函数**
- L1 损失函数

```
bbox_coder=dict(  
    type='DeltaXYWHBBoxCoder',  
    target_means=[.0, .0, .0, .0],  
    target_stds=[1.0, 1.0, 1.0, 1.0]),  
  
loss_cls=dict(  
    type='FocalLoss',  
    use_sigmoid=True,  
    gamma=2.0,  
    alpha=0.25,  
    loss_weight=1.0),  
  
loss_bbox=dict(  
    type='L1Loss',  
    loss_weight=1.0))
```

Mask ROI 提取器

```
mask_roi_extractor=dict(  
    type='SingleRoIExtractor',  
    roi_layer=dict(type='RoIAlign', output_size=14, sampling_ratio=0),  
    out_channels=256,
```

ROI 层为 RoIAxis

输出通道数 256

特征图步长

Mask 头部

```
mask_head=dict(  
    type='FCNMaskHead',  
    num_convs=4,  
    in_channels=256,  
    conv_out_channels=256,  
    num_classes=80,  
    loss_mask=dict(  
        type='CrossEntropyLoss', use_mask=True, loss_weight=1.0))),
```

卷积层个数

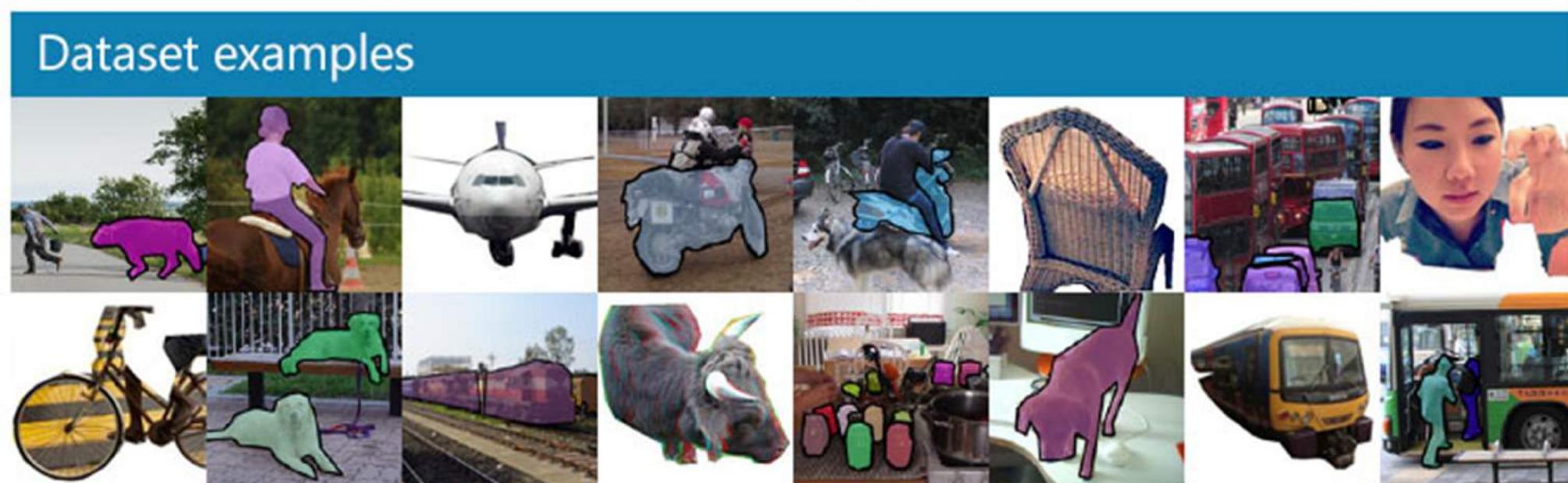
输入通道数 256

卷积输出通道数 256

Mask 损失函数为交叉熵

微软于2014年提出，最常用的是2017年版本

- 全集 33W 张图像
- 针对多种任务进行了标注
- 80类、150W 物体标注用于**目标检测与实例分割**



数据集整体信息
全局一个字典

图像信息
每个图像一个字典

```
{  
    "info" : info,  
    "images" : [image],  
    "annotations" : [annotation],  
    "licenses" : [license],  
    "categories" : [categories],  
}
```

```
image {  
    "id" : int,  
    "width" : int,  
    "height" : int,  
    "file_name" : str,  
    "license" : int,  
    "flickr_url" : str,  
    "coco_url" : str,  
    "date_captured" : datetime,  
}
```

全局信息

```
{  
    "info" : info,  
    "images" : [image],  
    "annotations" : [annotation],  
    "licenses" : [license],  
    "categories" : [categories],  
}
```

所有类别的信息
全局一个字典

```
categories[  
    {"id" : int,  
     "name" : str,  
     "supercategory" : str,  
}]
```

所有物体标注的信息
每个物体一个字典

```
annotation {  
    "id" : int,  
    "image_id" : int,  
    "category_id" : int,  
    "segmentation" : RLE or [polygon],  
    "area" : float,  
    "bbox" : [x,y,width,height],  
    "iscrowd" : 0 or 1,  
}
```

COCO 数据集的根目录

配置 batch_size

配置 Dataloader 进程数

针对训练、验证、测试子集，

分别配置：

- 数据类型 = COCO 数据集
- 标注文件的路径
- 图像目录的路径
- 数据处理流水线

```
dataset_type = 'CocoDataset'  
data_root = 'data/coco/'  
data = dict(  
    samples_per_gpu=2,  
    workers_per_gpu=2,  
    train=dict(  
        type=dataset_type,  
        ann_file=data_root + 'annotations/instances_train2017.json',  
        img_prefix=data_root + 'train2017/',  
        pipeline=train_pipeline),  
    val=dict(  
        type=dataset_type,  
        ann_file=data_root + 'annotations/instances_val2017.json',  
        img_prefix=data_root + 'val2017/',  
        pipeline=test_pipeline),  
    test=dict(  
        type=dataset_type,  
        ann_file=data_root + 'annotations/instances_val2017.json',  
        img_prefix=data_root + 'val2017/',  
        pipeline=test_pipeline))
```

一个列表包含所有图像，
每个图像包含所有物体标注

```
[  
    {  
        'filename': 'a.jpg',  
        'width': 1280,  
        'height': 720,  
        'ann': {  
            'bboxes': <np.ndarray> (n, 4),  
            'labels': <np.ndarray> (n, ),  
            'bboxes_ignore': <np.ndarray> (k, 4), (optional field)  
            'labels_ignore': <np.ndarray> (k, ) (optional field),  
            'masks': [poly]  
        }  
    },  
    ...  
]
```

自定义数据集的根目录

自定义数据集的类别名

配置 batch_size

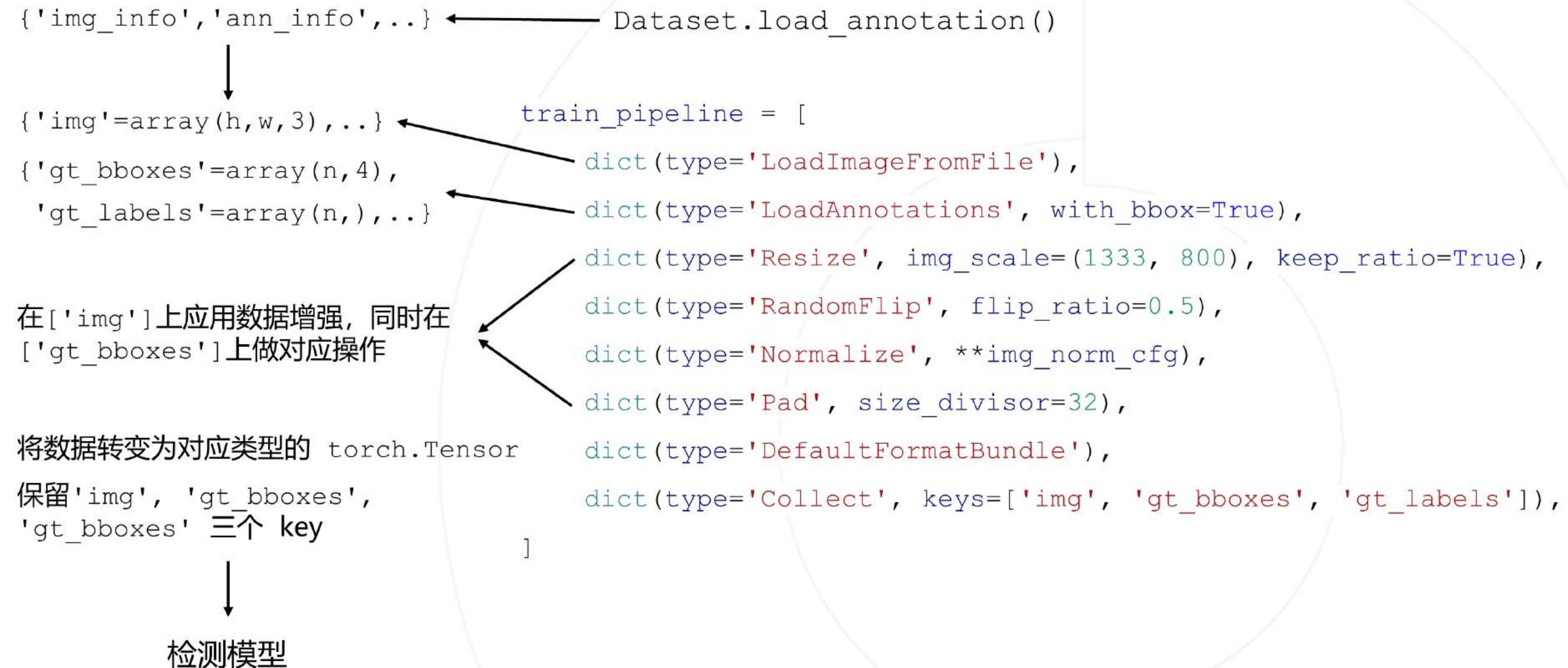
配置 Dataloader 进程数

针对训练、验证、测试子集，

分别配置：

- 数据类型 = 自定义数据集
- 标注文件的路径（自定义格式中包含ndarray，需要用pkl格式）
- 图像目录的路径
- 数据处理流水线

```
dataset_type = 'CustomDataset'  
data_root = 'dataset_path/'  
classes=('classname1', 'classname2', ...)  
data = dict(  
    samples_per_gpu=2,  
    workers_per_gpu=2,  
    train=dict(  
        type=dataset_type,  
        ann_file='train/custom_anno.pkl',  
        img_prefix='train',  
        pipeline=train_pipeline),  
    val=dict(  
        type=dataset_type,  
        ann_file='val/custom_anno.pkl',  
        img_prefix='val',  
        pipeline=test_pipeline),  
    test=dict(  
        type=dataset_type,  
        ann_file='val/custom_anno.pkl',  
        img_prefix='val',  
        pipeline=test_pipeline))
```



通常使用SGD算法配合不同的学习率策略：

学习率策略	第一次降低	第二次降低	总轮数
1x	8 轮	11 轮	12 轮
2x	16 轮	22 轮	24 轮
20e	16 轮	19 轮	20 轮

所有策略均包含一个学习率升温过程。

```
optimizer = dict(type='SGD', lr=0.02, momentum=0.9,
weight_decay=0.0001)

optimizer_config = dict(grad_clip=None)
# learning policy
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=0.001,
    step=[8, 11])

runner = dict(type='EpochBasedRunner', max_epochs=12)
```

代码实操演示

更多内容可参见 GitHub 上的代码文档以及教程；
如有问题欢迎加入我们的QQ群进行提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。



QQ

通用视觉框架OpenMMLab 实践5 语义分割工具包 MM Segmentation

王若晖
2021年5月

④ 本节内容：

- MMSegmentation 项目概述
- MMSegmentation 的模块化设计
 - PSPNet 模型配置文件解读
- 数据集与数据流水线配置解读
- 常用优化器配置
- 代码实践
 - 使用预训练模型对单张图像进行推理
 - 使用自定义数据集训练语义分割模型

MM Segmentation



算法丰富

378 个
预训练模型

27 篇
论文复现

模块化设计

配置简便

容易拓展

统一超参

大量消融实验

支持公平对比

使用方便

训练工具

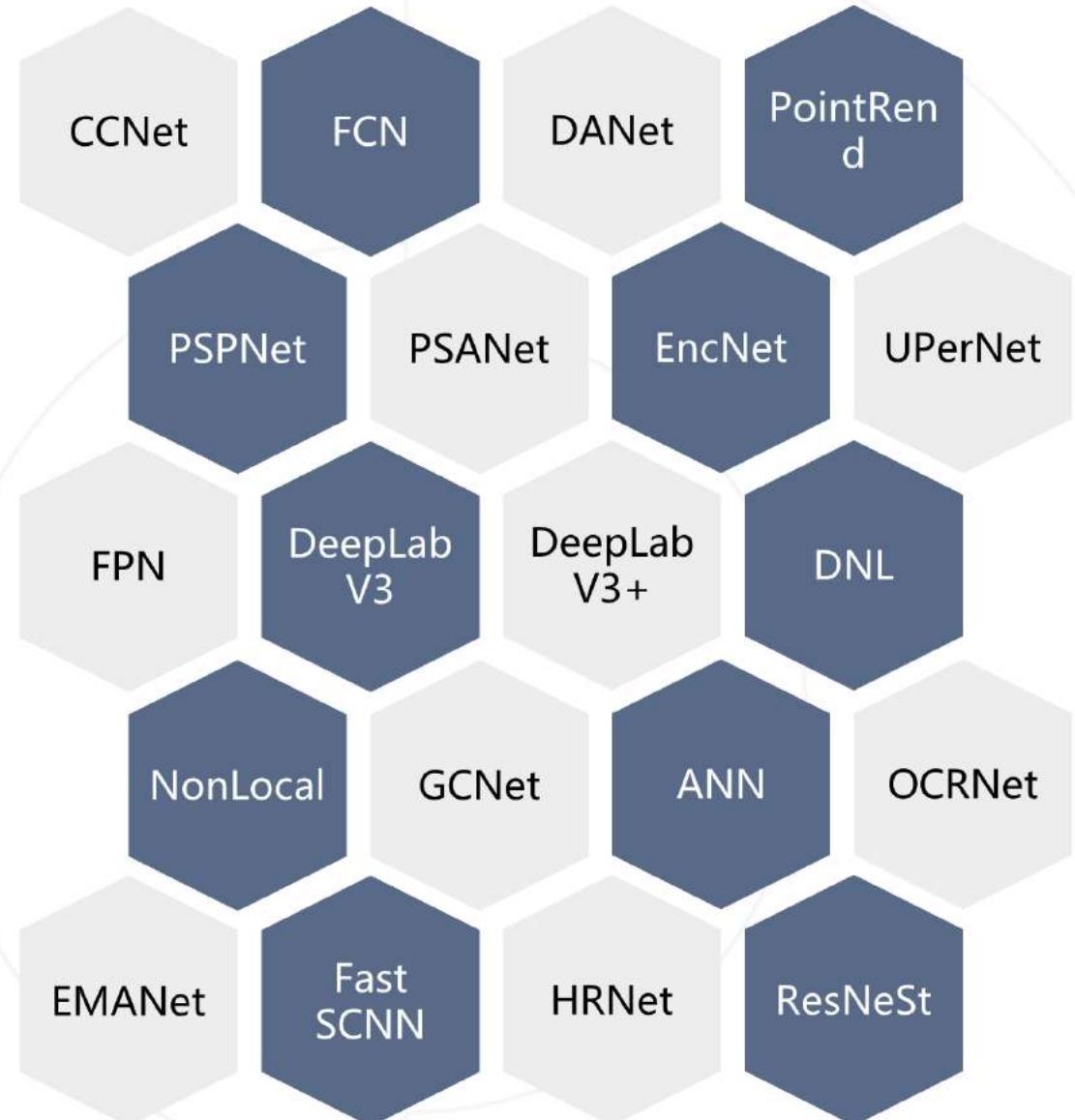
测试工具

推理 API

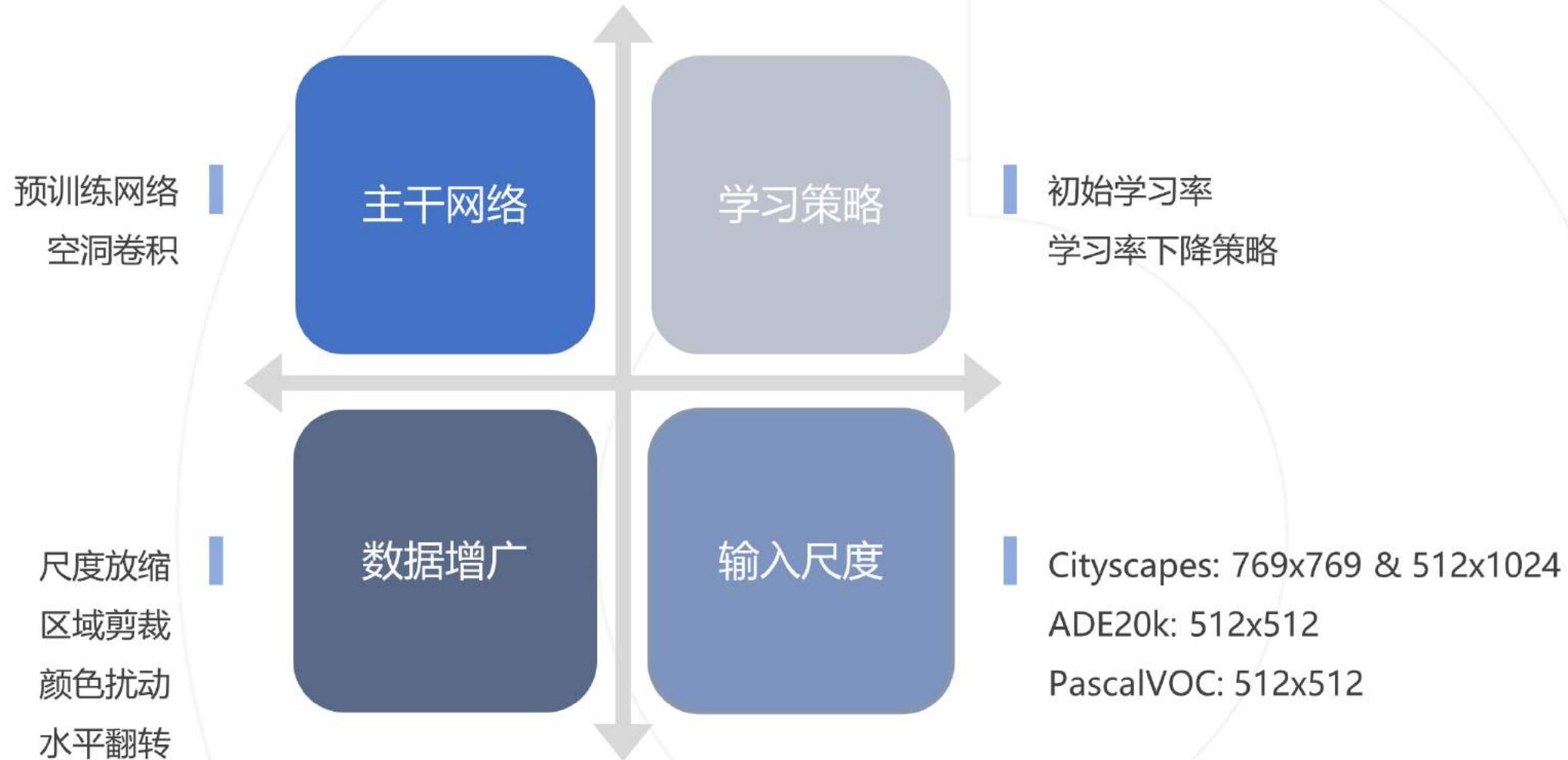
代码库: <https://github.com/open-mmlab/mmsegmentation>
文档: <https://mmsegmentation.readthedocs.io/en/latest/>

20+ 算法

370+ 预训练模型

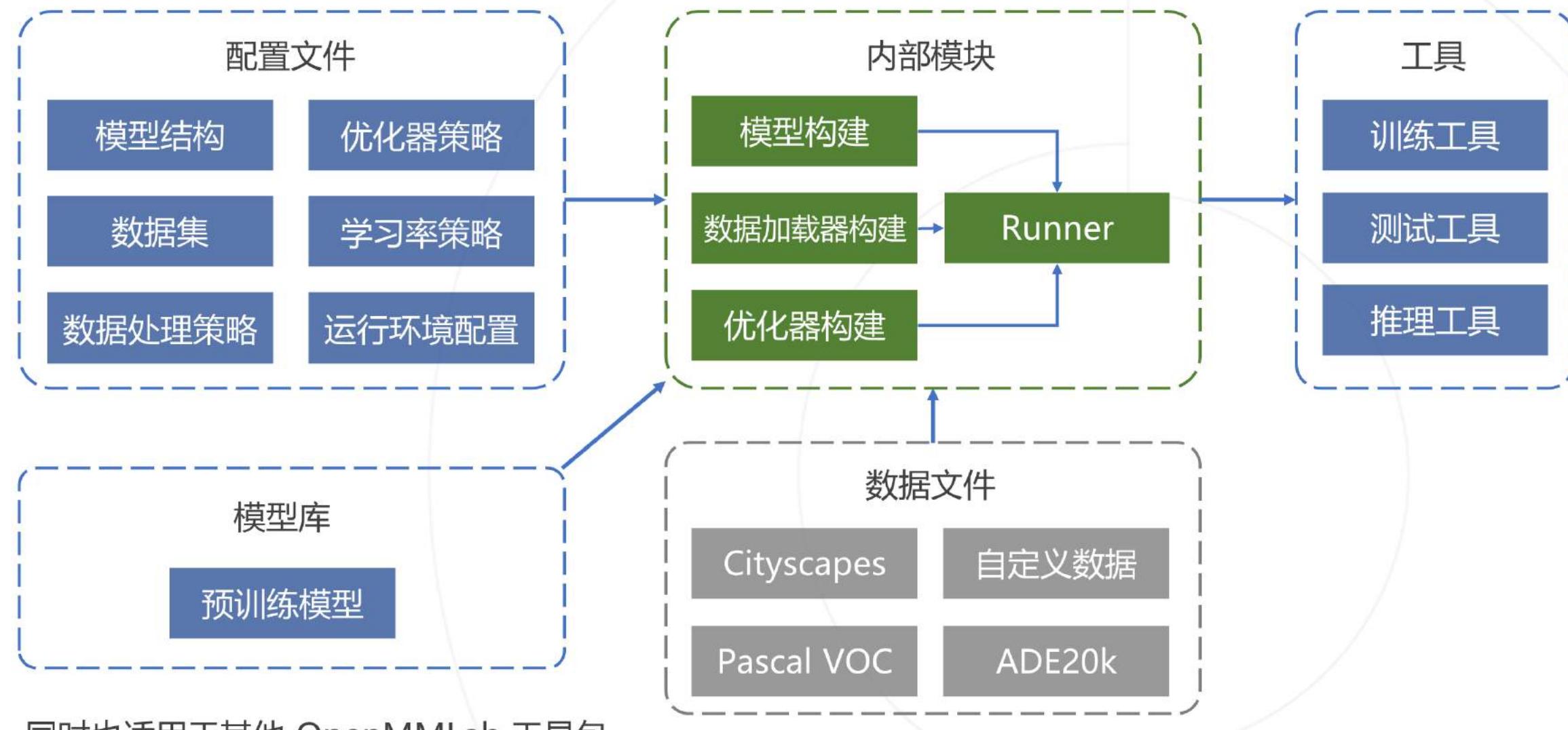




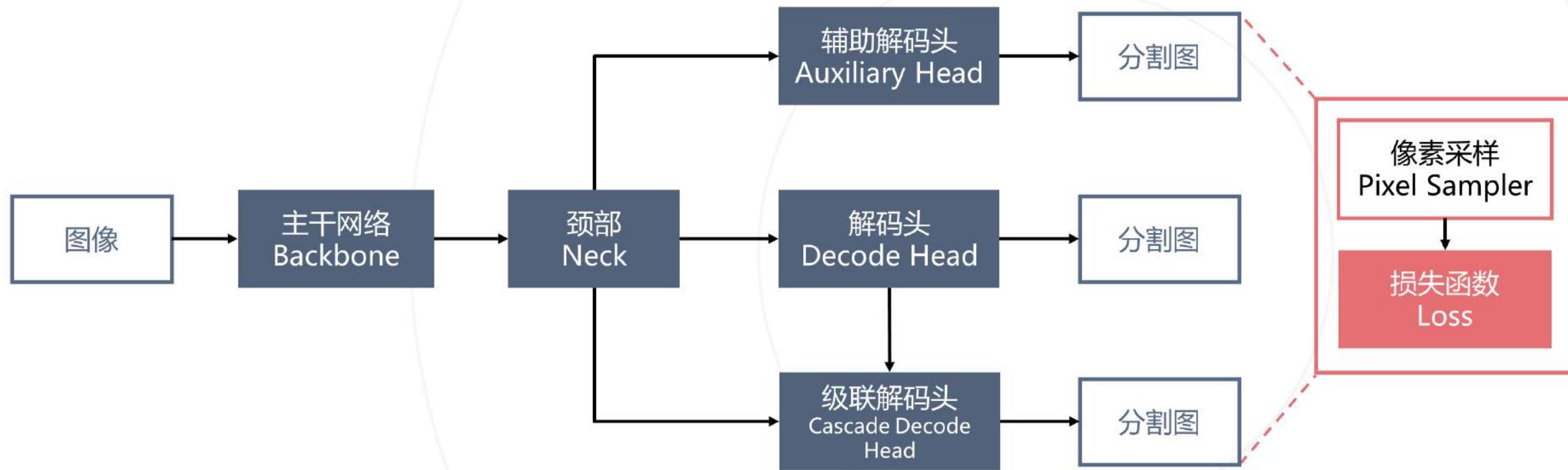


- 测试精度
- 推理速度
- 显存占用

Method	Backbone	Crop Size	Lr schd	Mem (GB)	Inf time (fps)	mIoU	mIoU(ms+flip)	download
PSPNet	R-50-D8	512x1024	40000	6.1	4.07	77.85	79.18	model log
PSPNet	R-101-D8	512x1024	40000	9.6	2.68	78.34	79.74	model log
PSPNet	R-50-D8	769x769	40000	6.9	1.76	78.26	79.88	model log
PSPNet	R-101-D8	769x769	40000	10.9	1.15	79.08	80.28	model log
PSPNet	R-50-D8	512x1024	80000	-	-	78.55	79.79	model log
PSPNet	R-101-D8	512x1024	80000	-	-	79.76	81.01	model log
PSPNet	R-50-D8	769x769	80000	-	-	79.59	80.69	model log
PSPNet	R-101-D8	769x769	80000	-	-	79.77	81.06	model log



MMSegmentation 将分割模型统一拆解为如下模块，方便用户根据自己的需求进行组装和扩展。



```
model = dict(  
    分割模型的主体架构 type='EncoderDecoder' OR 'CascadeEncoderDecoder',  
    pretrained='open-mmlab://resnet50_v1c',  
    主干网络 backbone=dict(  
        type='ResNetV1c',  
        # ... more options),  
    颈部 neck = None,  
    主解码头 decode_head=dict(  
        type='PSPHead',  
        # ... more options  
        loss_decode=dict(  
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0)),  
    辅助解码头 auxiliary_head=dict(  
        type='FCNHead',  
        # ... more options  
        loss_decode=dict(  
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=0.4)),  
    损失函数 train_cfg=dict(...),  
    test_cfg=dict(...))
```

主干网络输入图像，输出多层次的特征图

```
backbone=dict(  
    type='ResNetV1c',  
    depth=50,  
    num_stages=4,  
    out_indices=(0, 1, 2, 3),  
    dilations=(1, 1, 2, 4),      增大空洞卷积倍率  
    strides=(1, 2, 1, 1),       同时移除降采样  
    norm_cfg= dict(type='SyncBN', requires_grad=True),  
    norm_eval=False,  
    style='pytorch',  
    contract_dilation=True)
```

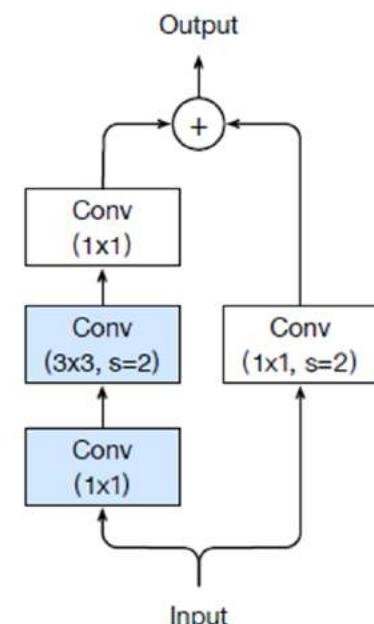
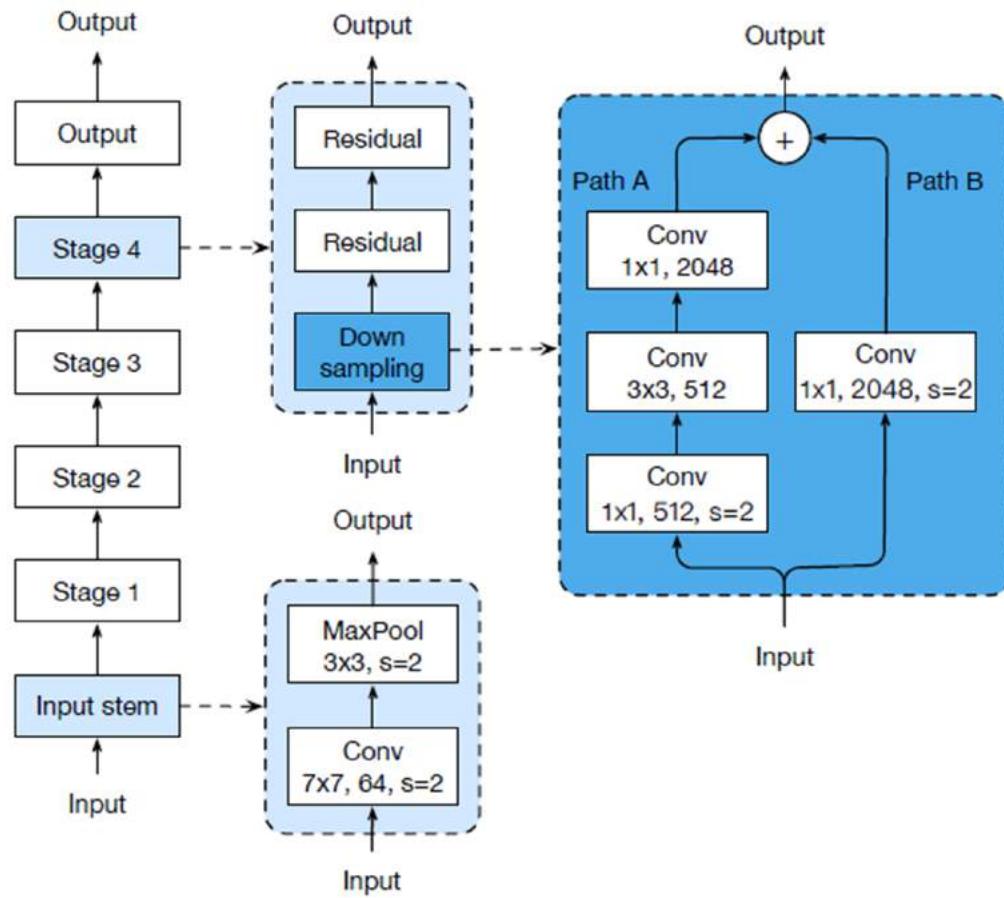
ResNet v1c 结构

输出全部级别的特征图
给颈部或者辅助解码头

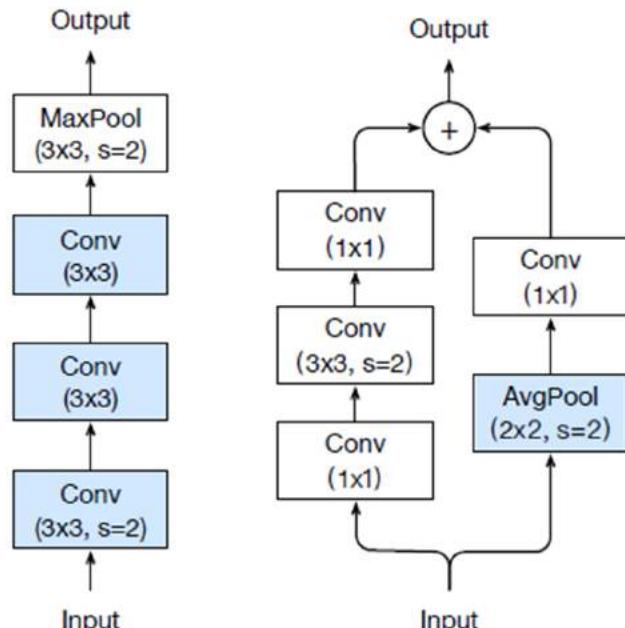
配置降采样和空洞卷积

分割模型会使用 SyncBN
以增大 batch size

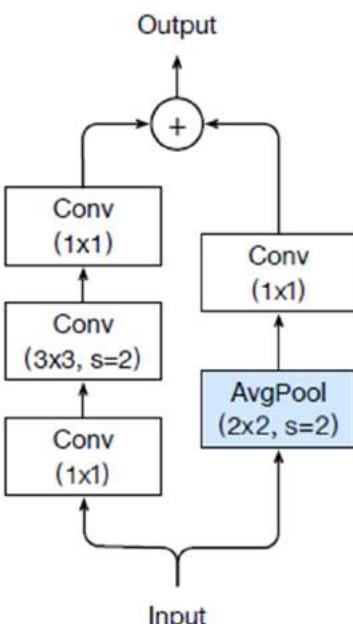
ResNet 50 层以上的模型在 BottleNeck 模块以及 stem 部分（即网络前几层）有一些变形：



(a) ResNet-B



(b) ResNet-C



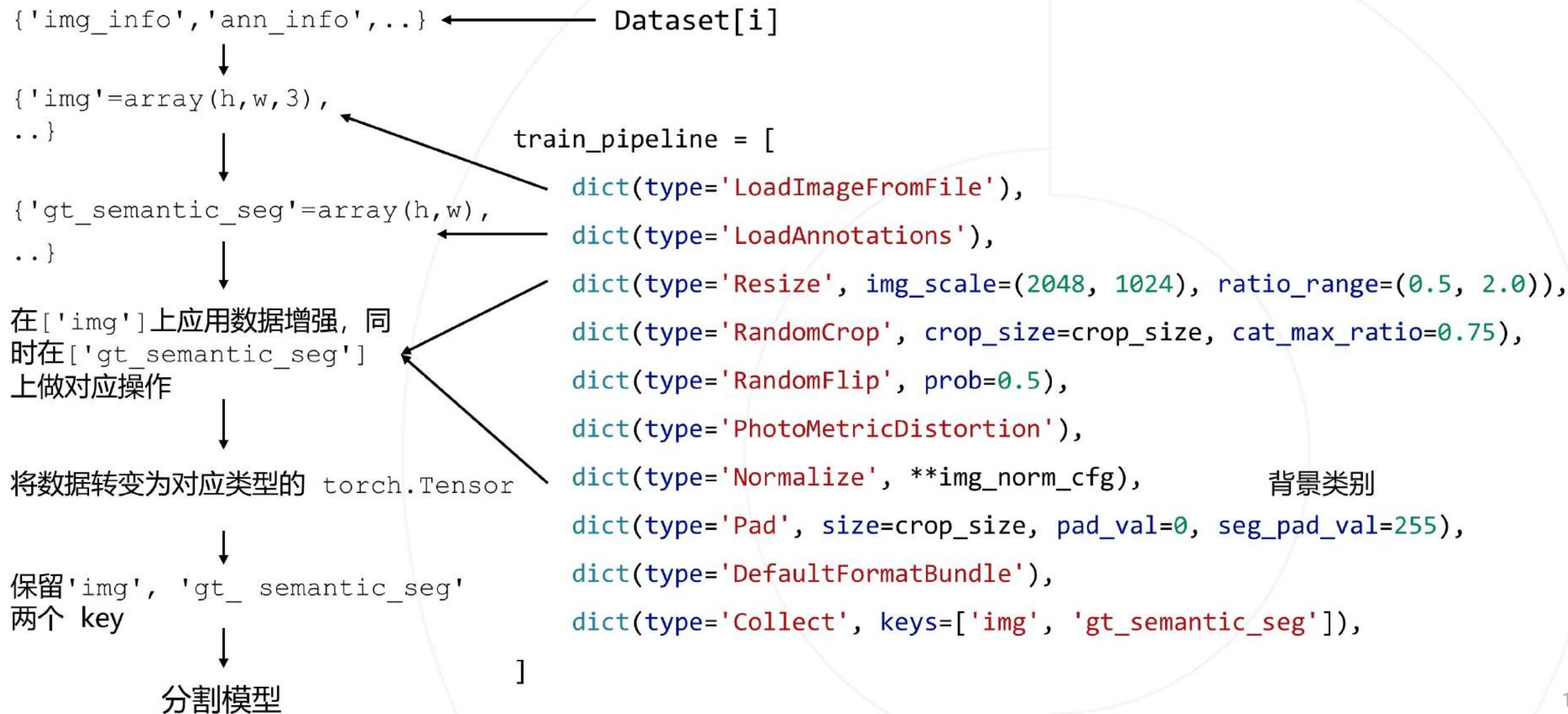
(c) ResNet-D

主解码头从特征图预测分割图

```
decode_head=dict(  
    type='PSPHead',  
    in_channels=2048,  
    in_index=3,  
    channels=512,  
    pool_scales=(1, 2, 3, 6),  
    dropout_ratio=0.1,  
    num_classes=19,  
    norm_cfg= dict(type='SyncBN', requires_grad=True),  
    align_corners=False,  
    loss_decode=dict(  
        type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0)), 15
```

	auxiliary_head=dict(辅助解码头鼓励学习更好的低层特征
使用 FCN 解码头	type='FCNHead',	不用于最终的预测
以低层次特征图为输入	in_channels=1024,	
in_index=2,		
channels=256,		
num_convs=1,		
concat_input=False,		
dropout_ratio=0.1,		
num_classes=19,		
norm_cfg=norm_cfg,		
align_corners=False,		
loss_decode=dict(
type='CrossEntropyLoss', use_sigmoid=False, loss_weight=0.4)),		
权重较小	16	

DataLoader 参数	数据集类型	dataset_type = 'CityscapesDataset'
	数据集路径	data_root = 'data/cityscapes/'
		data = dict(
	batch size	samples_per_gpu=2,
	worker 个数	workers_per_gpu=2,
	训练集配置	train=dict(
		type=dataset_type, data_root=data_root, img_dir='leftImg8bit/train', ann_dir='gtFine/train', pipeline=train_pipeline), val=dict(...), test=dict(...))



MMSegmentation 默认的学习率策略
仅在总迭代次数上有所差别

学习率策略	20k	40k	80k	160k
总迭代数	2 万次	4 万次	8 万次	16 万次

- 使用多项式下降策略
- 基于迭代次数不是轮数
- 使用 mIoU 作为评估指标

```
# optimizer
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0005)
optimizer_config = dict()

# learning policy
lr_config = dict(policy='poly', power=0.9, min_lr=1e-4, by_epoch=False)

# runtime settings
runner = dict(type='IterBasedRunner', max_iters=20000)
checkpoint_config = dict(by_epoch=False, interval=2000)
evaluation = dict(interval=2000, metric='mIoU')
```

代码实操演示

更多内容可参见 GitHub 上的代码文档以及教程；
如有问题欢迎加入我们的QQ群进行提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。



QQ

通用视觉框架OpenMMLab 实践6 图像编辑工具包 MMEditing 上

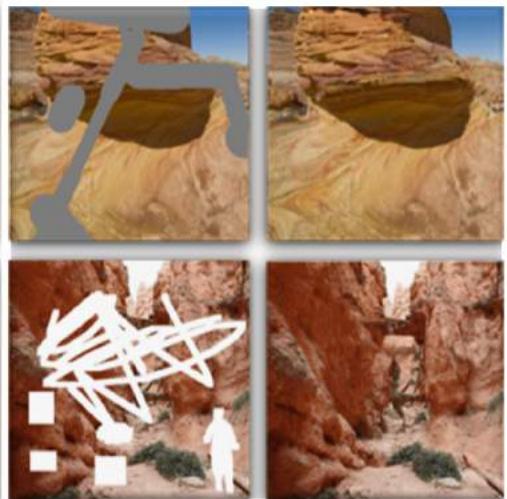
王若晖
2021年5月

④ 本节内容：

- MMEdition 项目概述
- MMEdition 中的超分辨率
 - SRCNN 模型配置解读
 - ESRGAN 模型配置解读
- 数据集与数据流水线配置解读
- 常用优化器配置解读
- 代码实践
 - 使用预训练模型对单张图像进行推理
 - 使用自定义数据集训练超分模型



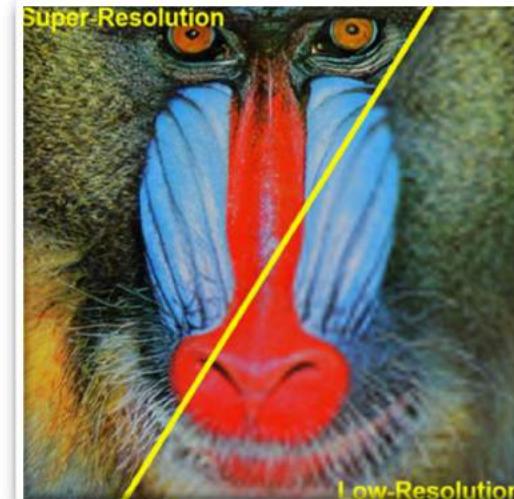
图像修复



图像抠图



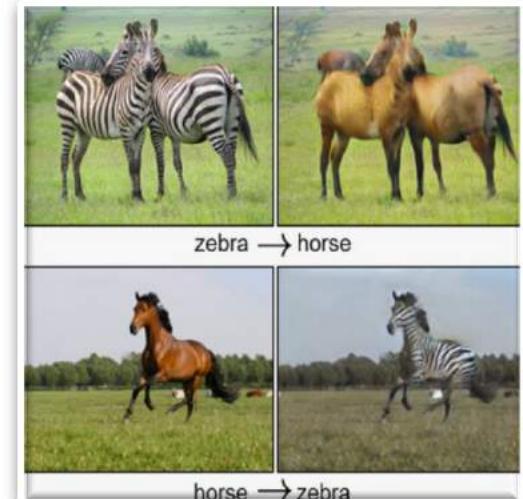
超分辨率



15 篇论文复现

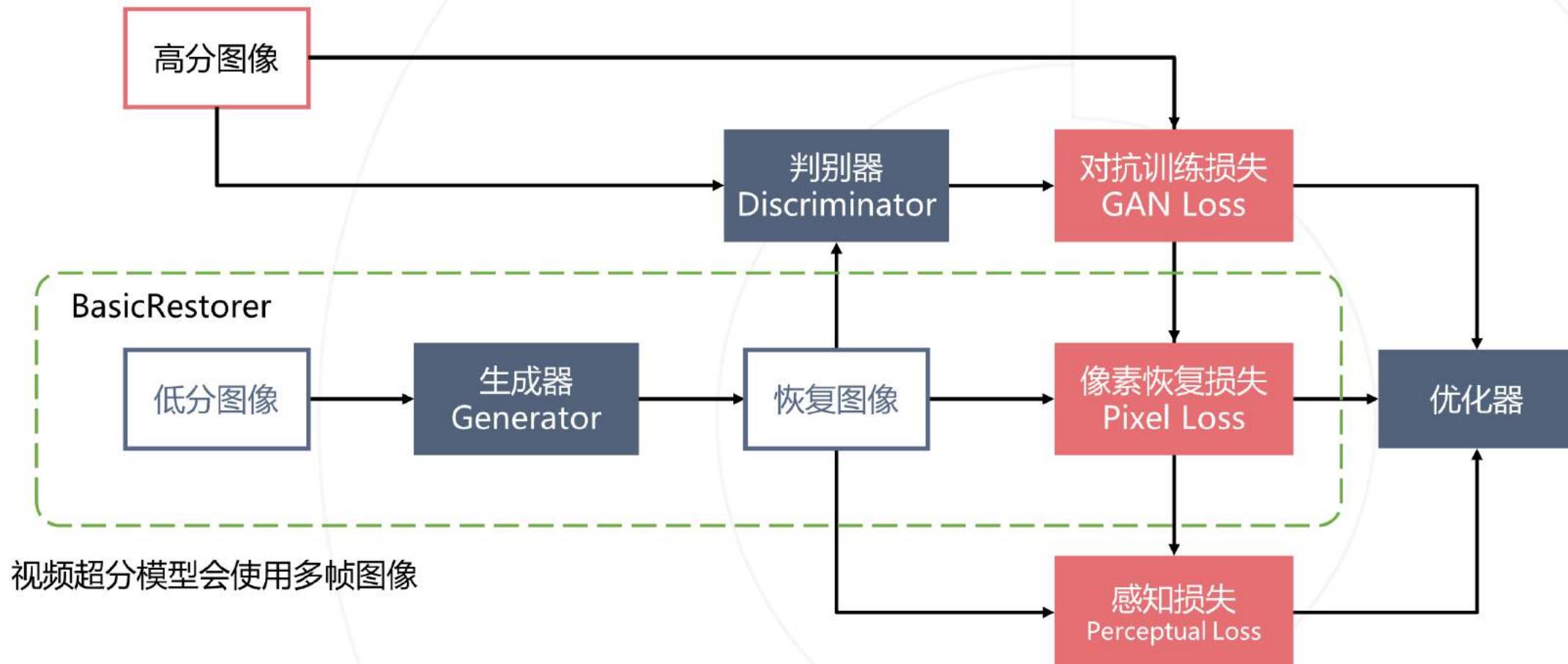
30 个预训练模型

图像转译



代码库: <https://github.com/open-mmlab/mmediting>
文档: <https://mmediting.readthedocs.io/en/latest/>

MMEditing 将超分辨率分割模型统一拆解为如下模块，方便用户根据自己的需求进行组装和扩展。



基础超分模型 BasicRestorer 使用生成器网络恢复高分图像，计算逐像素的损失函数

- SRCNN、SRResNet (SRGAN的生成器) 、 RRDBNet (ESRGAN的生成器) 属于此类

```
scale = 4
model = dict(
    基本超分模型 type='BasicRestorer',
    "生成器"网络 generator=dict(
        type='SRCNN',
        channels=(3, 64, 32, 3),
        kernel_sizes=(9, 1, 5),
        upscale_factor=scale),
    像素损失函数 pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))
```

完整的超分模型还有额外的判别器、感知损失函数、GAN损失函数等组件

➤ 完整的 SRGAN、ESRGAN 属于此类

```
model = dict(  
    type='ESRGAN',  
    generator=dict(...),  
    discriminator=dict(...),  
    pixel_loss=dict(...),  
    perceptual_loss=dict(...),  
    gan_loss=dict(...),  
    ...,  
)
```

使用 RRDB 结构
作为生成器网络

```
generator=dict(  
    type='RRDBNet',  
    in_channels=3,  
    out_channels=3,  
    mid_channels=64,  
    num_blocks=23,  
    growth_channels=32),
```

```
discriminator=dict(type='ModifiedVGG', in_channels=3, mid_channels=64)
```

使用 VGG 结构
作为判别器网络

像素恢复损失使用 L1Loss

```
pixel_loss=dict(type='L1Loss', loss_weight=1e-2, reduction='mean'),  
perceptual_loss=dict(
```

基于 VGG 网络的最后一层
特征计算特征重构损失

```
type='PerceptualLoss',  
layer_weights={'34': 1.0},  
vgg_type='vgg19',  
perceptual_weight=1.0,  
style_weight=0,  
norm_img=False),
```

不计算风格重构损失

```
gan_loss=dict(  
type='GANLoss',  
gan_type='vanilla',  
loss_weight=5e-3,  
real_label_val=1.0,  
fake_label_val=0)
```

使用常规的对抗训练损失

DataLoader 配置

数据读取进程数
训练和测试分别
配置 batch size

```
data = dict(  
    workers_per_gpu=8,  
    train_dataloader=dict(samples_per_gpu=16, drop_last=True),  
    val_dataloader=dict(samples_per_gpu=1),  
    test_dataloader=dict(samples_per_gpu=1),  
    train/val/test=dict(  
        type= 'SRAnnotationDataset'/'SRFolderDataset',  
        lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X4_sub',  
        gt_folder='data/DIV2K/DIV2K_train_HR_sub',  
        ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  
        pipeline=train_pipeline/test_pipeline,  
        scale=scale),)
```

数据文件配置

数据子集
低分文件目录
高分文件目录
标注文件
数据处理流水线

更详细内容可参照配置文件源码



使用 Adam 优化器
对抗训练常用

```
optimizers = dict(  
    generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)),  
    discriminator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))
```

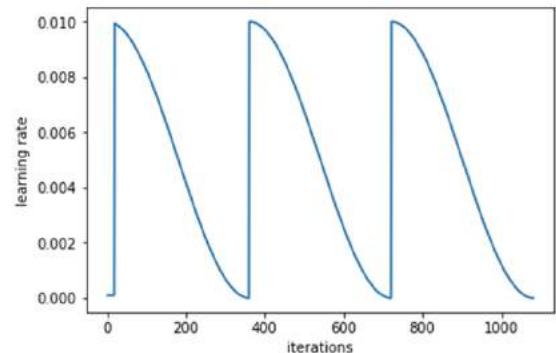
使用步长学习率策略

```
total_iters = 400000  
lr_config = dict(  
    policy='Step',  
    by_epoch=False,  
    step=[50000, 100000, 200000, 300000],  
    gamma=0.5)
```

基于迭代次数
不是轮数

使用Cosine学习率策略

```
total_iters = 1000000  
lr_config = dict(  
    policy='CosineRestart',  
    by_epoch=False,  
    periods=[250000, 250000, 250000, 250000],  
    restart_weights=[1, 1, 1, 1],  
    min_lr=1e-7)
```



代码实操演示

更多内容可参见 GitHub 上的代码文档以及教程；
如有问题欢迎加入我们的QQ群进行提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。



QQ

谢谢大家

通用视觉框架OpenMMLab 实践7 图像编辑工具包 MMEditing 下

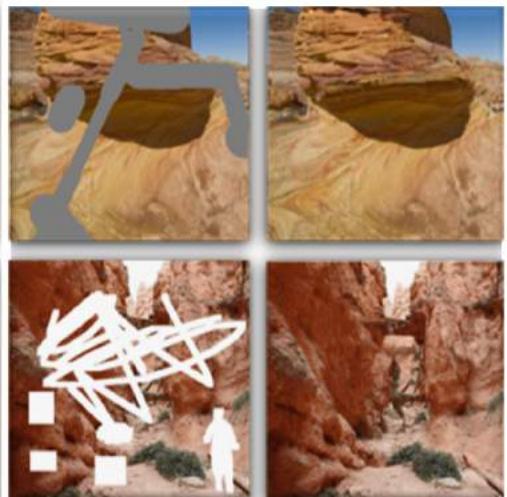
王若晖
2021年5月

④ 本节内容：

- MMEditioning 项目概述
- 配置文件解读
 - CycleGAN 与 Pix2Pix 配置文件解读
- MMEditioning 中的图像转译、图像修复与抠图
 - 使用预训练模型进行推理



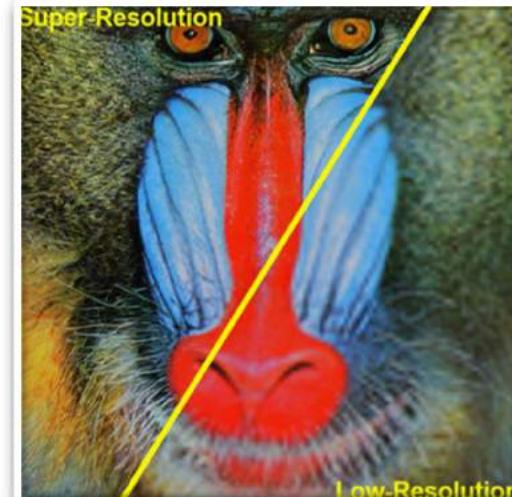
图像修复



图像抠图



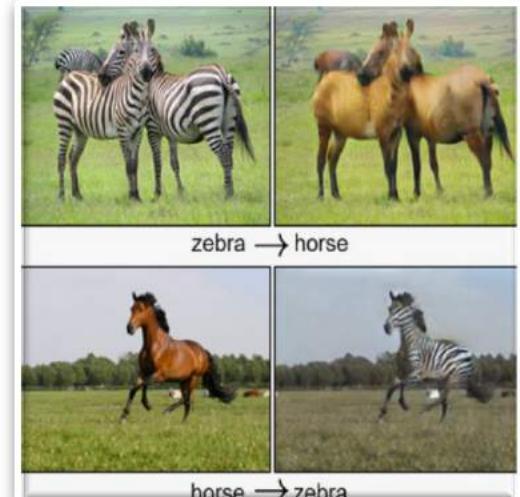
超分辨率



15 篇论文复现

30 个预训练模型

图像转译



代码库: <https://github.com/open-mmlab/mmediting>
文档: <https://mmediting.readthedocs.io/en/latest/>

Pix2Pix 模型使用生成器网络生成图像，计算对抗训练损失和像素损失函数；

CycleGAN 模型使用生成器网络生成图像，计算对抗训练损失、循环重构损失和 Identity 损失函数。

CycleGAN 模型 model = dict(

 type='CycleGAN',

 生成器网络

 generator=dict(...),

 判别器网络

 discriminator=dict(...),

 对抗训练损失

 gan_loss=dict(...),

 循环重构损失

 cycle_loss=dict(...),

 Identity 损失

 id_loss=dict(...),

Pix2Pix 模型 model = dict(

 type='Pix2Pix',

 生成器网络

 generator=dict(...),

 判别器网络

 discriminator=dict(...),

 对抗训练损失

 gan_loss=dict(...),

 像素损失

 pixel_loss=dict(...),

CycleGAN 模型

使用 ResnetGenerator 结构作为生成器网络

```
generator=dict(  
    type='ResnetGenerator',  
    in_channels=3,  
    out_channels=3,  
    base_channels=64,  
    norm_cfg=dict(type='IN'),  
    use_dropout=False,  
    num_blocks=9,  
    padding_mode='reflect',  
    init_cfg=dict(type='normal', gain=0.02)),
```

Pix2Pix 模型

使用 UnetGenerator结构作为生成器网络

```
generator=dict(  
    type='UnetGenerator',  
    in_channels=3,  
    out_channels=3,  
    num_down=8,  
    base_channels=64,  
    norm_cfg=dict(type='BN'),  
    use_dropout=True,  
    init_cfg=dict(type='normal', gain=0.02)),
```

CycleGAN 模型

使用 PatchDiscriminator 结构作为判别器网络

```
discriminator=dict(  
    type='PatchDiscriminator',  
    in_channels=3,  
    base_channels=64,  
    num_conv=3,  
    norm_cfg=dict(type='IN'),  
    init_cfg=dict(type='normal', gain=0.02)),
```

Pix2Pix 模型

使用 PatchDiscriminator 结构作为判别器网络

```
discriminator=dict(  
    type='PatchDiscriminator',  
    in_channels=6,  
    base_channels=64,  
    num_conv=3,  
    norm_cfg=dict(type='BN'),  
    init_cfg=dict(type='normal', gain=0.02)),
```

对抗训练损失

```
gan_loss=dict(  
    type='GANLoss',  
    gan_type='lsgan',  
    real_label_val=1.0,  
    fake_label_val=0.0,  
    loss_weight=1.0),  
  
cycle_loss=dict(type='L1Loss', loss_weight=10.0, reduction='mean'),  
id_loss=dict(type='L1Loss', loss_weight=0, reduction='mean')
```

CycleGAN 模型

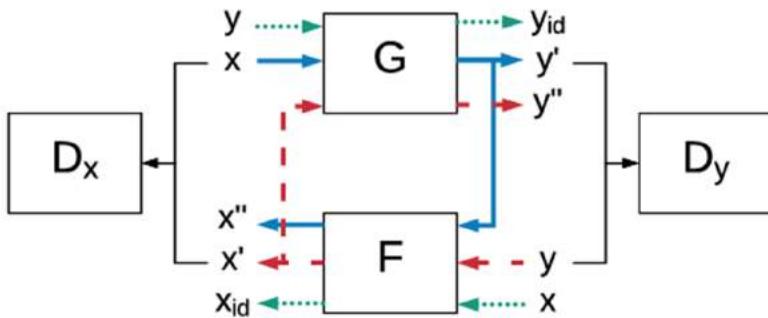
循环重构损失

Identity 损失

Pix2Pix 模型

像素损失

```
pixel_loss=dict(type='L1Loss', loss_weight=100.0, reduction='mean'))
```



```
data = dict(  
    workers_per_gpu=1,  
    train/val/test=dict(  
        type='GenerationUnpairedDataset'/'GenerationPairedDataset',  
        dataroot=data_root  
        pipeline=train_pipeline/test_pipeline,  
        test_mode=False/True),  
)
```

更详细内容可参照配置文件源码





使用 Adam 优化器
对抗训练常用

```
optimizers = dict(  
    generator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)),  
    discriminator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)))
```

CycleGAN 使用 Linear 学习率策略

```
total_iters = 400000  
lr_config = dict(  
    policy='Linear',  
    by_epoch=False,  
    target_lr=0,  
    start=133400,  
    interval=1334)
```

设置总迭代次数

指定学习率开始变化
的迭代次数和步长

Pix2Pixel 使用固定的学习率策略

```
total_iters = 80000  
lr_config = dict(  
    policy='Fixed',  
    by_epoch=False)
```

代码实操演示

更多内容可参见 GitHub 上的代码文档以及教程；
如有问题欢迎加入我们的QQ群进行提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。

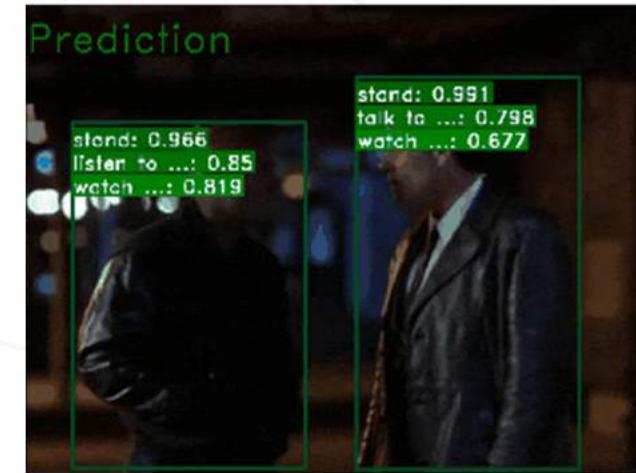


QQ

谢谢大家

通用视觉框架OpenMMLab 实践8 视频理解工具包 MMAction2

王若晖
2021年6月



全面支持

动作识别

时序检测

时空检测

算法丰富

157 个
预训练模型

21 篇
论文复现

更优更快

训练速度

模型精度

使用方便

训练工具

测试工具

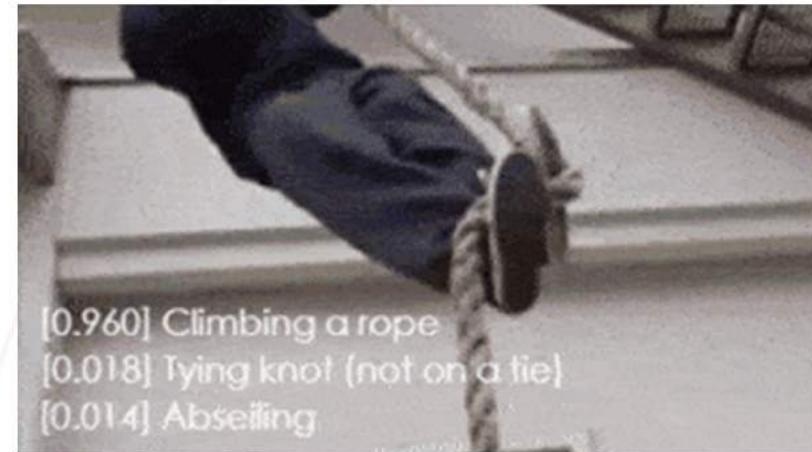
推理 API

多 IO 后端支持

④ 本节内容：

- MMAction2 项目概述
- Action Recognition 的配置文件解读
 - TSN 模型配置文件解读
 - I3D 模型配置文件解读
- 数据配置文件解读
- 优化器配置文件解读
- MMAction2 中的行为识别与时空动作检测
 - 使用预训练模型进行推理
 - 使用自定义数据训练模型

Action Recognition



Spatio-temporal Action Detection

GroundTruth



Prediction



Supported methods for Action Recognition:

▼ (click to collapse)

- TSN (ECCV'2016)
- TSM (ICCV'2019)
- TSM Non-Local (ICCV'2019)
- R(2+1)D (CVPR'2018)
- I3D (CVPR'2017)
- I3D Non-Local (CVPR'2018)
- SlowOnly (ICCV'2019)
- SlowFast (ICCV'2019)
- CSN (ICCV'2019)
- TIN (AAAI'2020)
- TPN (CVPR'2020)
- C3D (CVPR'2014)
- X3D (CVPR'2020)
- OmniSource (ECCV'2020)
- MultiModality: Audio (ArXiv'2020)
- TANet (ArXiv'2020)
- TRN (CVPR'2015)

Supported methods for Temporal Action Detection:

▼ (click to collapse)

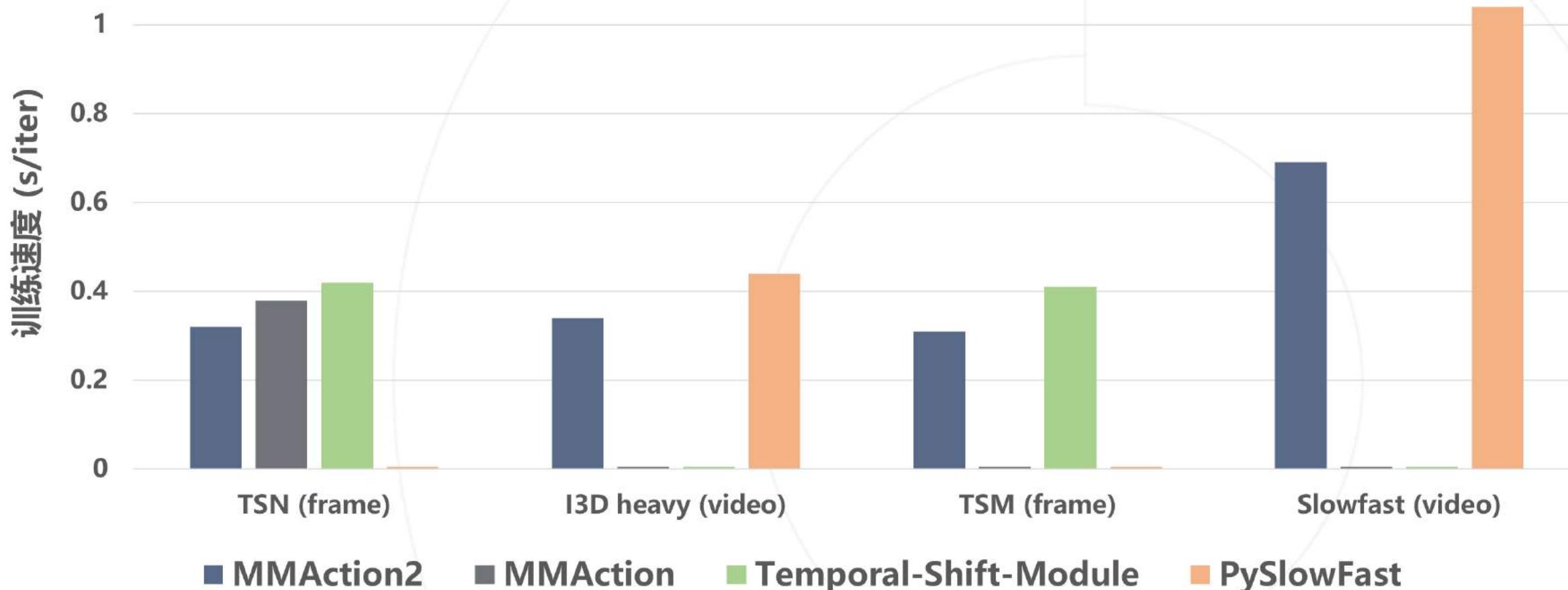
- BSN (ECCV'2018)
- BMN (ICCV'2019)
- SSN (ICCV'2017)

Supported methods for Spatial Temporal Action Detection:

▼ (click to collapse)

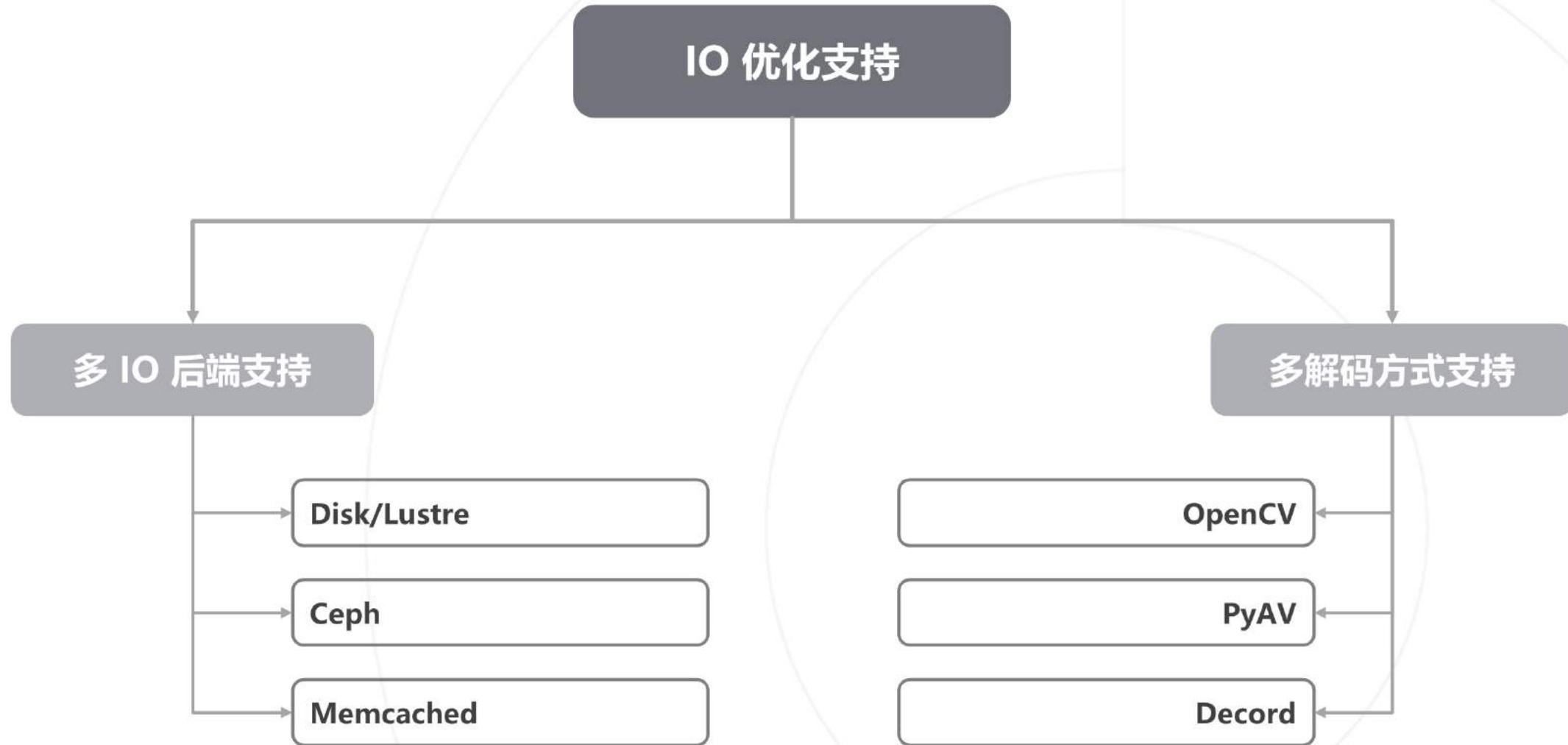
- SlowOnly+Fast R-CNN (ICCV'2019)
- SlowFast+Fast R-CNN (ICCV'2019)
- Long-Term Feature Bank (CVPR'2019)

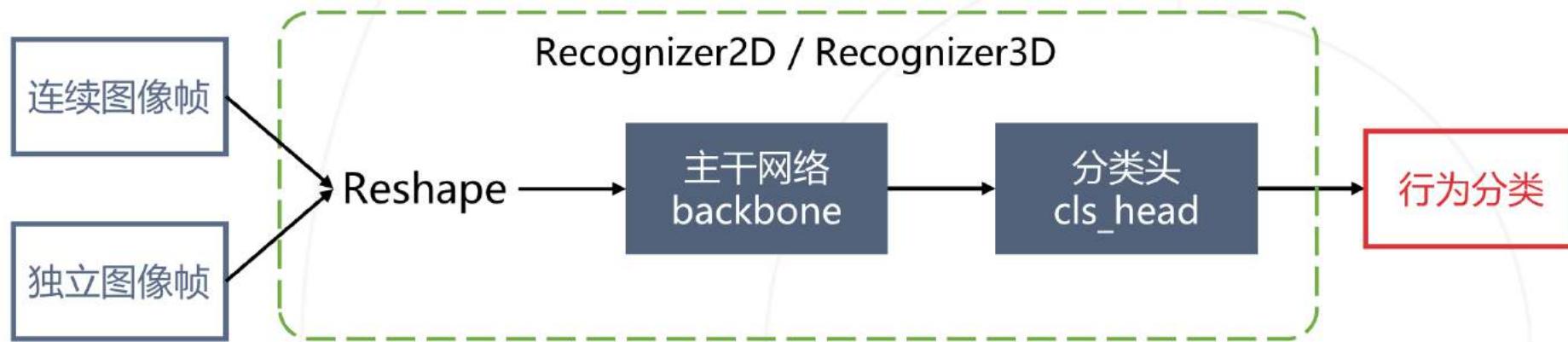
不同模型的训练时间 (越短越快)



MMAction2 的模型精度 普遍优于 其他代码框架

模型	数据集	分辨率	MMAction2	其他代码
TSN	Kinetics400	Short-side 320	70.91	70.6
I3D	Kinetics400	Short-side 256	73.27	72.6
TPN	Kinetics400	Short-side 320	76.20	75.49
CSN	Kinetics400	Short-side 320	80.76	80.6
TSM	Sth-v1	Height 100	49.28	48.61
TIN	Sth-v2	Height 240	56.70	56.38
TSM	Sth-v2	Height 240	62.04	60.98





更多模型配置文件可参考：

https://github.com/open-mmlab/mmaction2/tree/master/configs/_base_/models

2D ResNet-50
作为主干网络

TSN 的头部
使用平均共识函数将多个分
段的特征平均
再传入线性层计算分类概率

```
model = dict(  
    type='Recognizer2D',  
    backbone=dict(  
        type='ResNet',  
        pretrained='torchvision://resnet50',  
        depth=50,  
        norm_eval=False),  
    cls_head=dict(  
        type='TSNHead',  
        num_classes=400,  
        in_channels=2048,  
        spatial_type='avg',  
        consensus=dict(type='AvgConsensus', dim=1),  
        dropout_ratio=0.4,  
        init_std=0.01),
```

```
model = dict(  
    type='Recognizer3D',  
    backbone=dict(  
        type='ResNet3d',  
        pretrained2d=True,  
        pretrained='torchvision://resnet50',  
        depth=50,  
        conv_cfg=dict(type='Conv3d'),  
        norm_eval=False,  
        inflate=((1, 1, 1), (1, 0, 1, 0), (1, 0,  
        1, 0, 1, 0), (0, 1, 0)),  
        zero_init_residual=False),  
    neck=None,  
    cls_head=dict(  
        type='I3DHead',  
        num_classes=400,  
        in_channels=2048,  
        spatial_type='avg',  
        dropout_ratio=0.5,  
        init_std=0.01))
```

膨胀的
3D ResNet-50
作为主干网络

只膨胀设置为 1
的层，其余层做
T 个2D卷积

I3D 的头部
把特征中的 THW 通过
average pooling 压缩
成 1 维，再传入线性层
计算分类概率

```
cls_head=dict(  
    type='I3DHead',  
    num_classes=400,  
    in_channels=2048,  
    spatial_type='avg',  
    dropout_ratio=0.5,  
    init_std=0.01))
```

数据子集 {
batch size
数据读取进程数
数据子集
数据集类型
类别标注文件
数据集根目录
模态
数据处理流水线}

```
data = dict(  
    videos_per_gpu=8,  
    workers_per_gpu=4,  
    train/val/test=dict(  
        type='RawframeDataset'/'VideoDataset'...,  
        ann_file='annotation.txt',  
        data_prefix='data/kinetics400/rawframes_train',  
        modality='RGB'/'Flow',  
        pipeline=train_pipeline)  
)
```

some/directory-1 163 1
some/directory-2 122 1
some/directory-3 258 2
some/directory-4 234 2
some/directory-5 295 3
some/directory-6 121 3

图像帧目录

帧数量 动作分类

更详细内容可参照配置文件源码



更详细内容可参照配置文件源码

sgd_50e.py 配置文件

SGD 优化器

```
# optimizer
optimizer = dict(
    type='SGD',
    lr=0.01, # this lr is used for 8 gpus
    momentum=0.9,
    weight_decay=0.0001)
optimizer_config = dict(grad_clip=dict(max_norm=40, norm_type=2))

# learning policy
lr_config = dict(policy='step', step=[20, 40])
total_epochs = 50
```

当梯度的总体 norm 超过
max_norm 时按照进行归一化

步长下降策略

更详细内容可参照配置文件源码

代码实操演示

更多内容可参见 GitHub 上的代码文档以及教程；
如有问题欢迎加入我们的QQ群进行提问。



OpenMMLab 社区

群号：144762544



扫一扫二维码，加入群聊。



QQ

谢谢大家