PaddleX

2022 年 09 月 25 日

1. 快速了解 PaddleX

本文档仅适用于 PaddleX 1.3 版本，使用本文档请确保安装的 paddlex<2.0（例如通过’pip install paddlex==1.3.11’安装 1.3.11 版本的 PaddleX），PaddleX 1.3.11 的代码位于: https://github.com/PaddlePaddle/PaddleX/tree/release/1.3。如果需要使用 PaddleX 2.0，请参考 PaddleX 2.0 安装文档: https://github.com/PaddlePaddle/PaddleX/tree/release/1.3/docs/install.md，关于 2.0 的其他使用介绍请查阅 PaddleX 2.0 文档: https://github.com/PaddlePaddle/PaddleX#paddlex-%E4%BD%BF%E7%94%A8%E6%96%87%E6%A1%A3。

PaddleX 是基于飞桨核心框架、开发套件和工具组件的深度学习全流程开发工具。具备 **全流程打通**、**融合产业实践**、**易用易集成**三大特点。

- 项目官网: http://www.paddlepaddle.org.cn/paddle/paddlex
- 项目 GitHub: https://github.com/PaddlePaddle/PaddleX
- 官方 QQ 用户群: 1045148026
- GitHub Issue 反馈: http://www.github.com/PaddlePaddle/PaddleX/issues

1. 注：本使用手册在打印为 pdf 后，可能会存在部分格式的兼容问题；
2. 注：本文档持续在 http://paddlex.readthedocs.io/进行更新。

CHAPTER 1

10 分钟快速上手使用

本文档在一个小数据集上展示了如何通过 PaddleX 进行训练。本示例同步在 AIStudio 上，可直接在线体验模型训练。

本示例代码源于 Github tutorials/train/classification/mobilenetv3_small_ssld.py，用户可自行下载至本地运行。

PaddleX 中的所有模型训练跟随以下 3 个步骤，即可快速完成训练代码开发！

> **注意**：不同模型的 transforms、datasets 和训练参数都有较大差异，更多模型训练，可直接根据文档教程获取更多模型的训练代码。模型训练教程

PaddleX 的其它用法

- 使用 VisualDL 查看训练过程中的指标变化
- 加载训练保存的模型进行预测

1. 安装 PaddleX

安装相关过程和问题可以参考 PaddleX 的安装文档。

```
pip install paddlex -i https://mirror.baidu.com/pypi/simple
```

2. 准备蔬菜分类数据集

```
wget https://bj.bcebos.com/paddlex/datasets/vegetables_cls.tar.gz
tar xzvf vegetables_cls.tar.gz
```

3. 定义训练/验证图像处理流程 transforms

因为训练时加入了数据增强操作，因此在训练和验证过程中，模型的数据处理流程需要分别进行定义。如下所示，代码在 `train_transforms` 中加入了RandomCrop和RandomHorizontalFlip两种数据增强方式，更多方法可以参考*数据增强文档*。

```
from paddlex.cls import transforms
train_transforms = transforms.Compose([
    transforms.RandomCrop(crop_size=224),
    transforms.RandomHorizontalFlip(),
    transforms.Normalize()
])
eval_transforms = transforms.Compose([
    transforms.ResizeByShort(short_size=256),
    transforms.CenterCrop(crop_size=224),
    transforms.Normalize()
])
```

4. 定义 dataset 加载图像分类数据集

定义数据集，`pdx.datasets.ImageNet` 表示读取 ImageNet 格式的分类数据集

- *paddlex.datasets.ImageNet 接口说明*
- *ImageNet 数据格式说明*

```
train_dataset = pdx.datasets.ImageNet(
    data_dir='vegetables_cls',
    file_list='vegetables_cls/train_list.txt',
    label_list='vegetables_cls/labels.txt',
    transforms=train_transforms,
    shuffle=True)
eval_dataset = pdx.datasets.ImageNet(
    data_dir='vegetables_cls',
    file_list='vegetables_cls/val_list.txt',
    label_list='vegetables_cls/labels.txt',
    transforms=eval_transforms)
```

5. 使用 MobileNetV3_small_ssld 模型开始训练

本文档中使用百度基于蒸馏方法得到的 MobileNetV3 预训练模型，模型结构与 MobileNetV3 一致，但精度更高。PaddleX 内置了 20 多种分类模型，查阅*PaddleX 模型库*了解更多分类模型。

```
num_classes = len(train_dataset.labels)
model = pdx.cls.MobileNetV3_small_ssld(num_classes=num_classes)
```

(下页继续)

(续上页)

```
model.train(num_epochs=20,
            train_dataset=train_dataset,
            train_batch_size=32,
            eval_dataset=eval_dataset,
            lr_decay_epochs=[4, 6, 8],
            save_dir='output/mobilenetv3_small_ssld',
            use_vdl=True)
```

6. 训练过程使用 VisualDL 查看训练指标变化

训练过程中，模型在训练集和验证集上的指标均会以标准输出流形式输出到命令终端。当用户设定 `use_vdl=True` 时，也会使用 VisualDL 格式将指标打点到 `save_dir` 目录下的 `vdl_log` 文件夹，在终端运行如下命令启动 visualdl 并查看可视化的指标变化情况。

```
visualdl --logdir output/mobilenetv3_small_ssld --port 8001
```

服务启动后，通过浏览器打开 https://0.0.0.0:8001 或 https://localhost:8001 即可。

如果您使用的是 AIStudio 平台进行训练，不能通过此方式启动 visualdl，请参考 AIStudio VisualDL 启动教程使用

7. 加载训练保存的模型预测

模型在训练过程中，会每间隔一定轮数保存一次模型，在验证集上评估效果最好的一轮会保存在 `save_dir` 目录下的 `best_model` 文件夹。通过如下方式可加载模型，进行预测。

- load_model 接口说明
- 分类模型 predict 接口说明

```
import paddlex as pdx
model = pdx.load_model('output/mobilenetv3_small_ssld/best_model')
result = model.predict('vegetables_cls/bocai/100.jpg')
print("Predict Result: ", result)
```

预测结果输出如下，

```
Predict Result: Predict Result: [{'score': 0.9999393, 'category': 'bocai', 'category_id
↪': 0}]
```

更多使用教程

- 1.目标检测模型训练
- 2.语义分割模型训练

- 3.实例分割模型训练
- 4.模型太大，想要更小的模型，试试模型裁剪吧!

CHAPTER 2

快速安装

以下安装过程默认用户已安装好 **paddlepaddle-gpu 或 paddlepaddle(版本大于或等于 1.8.1)**，paddlepaddle 安装方式参照飞桨官网

2.1 pip 安装

注意其中 pycocotools 在 Windows 安装较为特殊，可参考下面的 Windows 安装命令

```
pip install paddlex -i https://mirror.baidu.com/pypi/simple
```

2.2 Anaconda 安装

Anaconda 是一个开源的 Python 发行版本，其包含了 conda、Python 等 180 多个科学包及其依赖项。使用 Anaconda 可以通过创建多个独立的 Python 环境，避免用户的 Python 环境安装太多不同版本依赖导致冲突。

- 参考Anaconda 安装 PaddleX 文档

2.3 代码安装

github 代码会跟随开发进度不断更新

```
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
python setup.py install
```

2.4 pycocotools 安装问题

PaddleX 依赖 pycocotools 包，如安装 pycocotools 失败，可参照如下方式安装 pycocotools

2.4.1 Windows 系统

- Windows 安装时可能会提示 `Microsoft Visual C++ 14.0 is required`，从而导致安装出错，点击下载 VC build tools安装再执行如下 pip 命令

  注意：安装完后，需要重新打开新的终端命令窗口

```
pip install cython
pip install git+https://gitee.com/jiangjiajun/philferriere-cocoapi.git
↪#subdirectory=PythonAPI
```

2.4.2 Linux/Mac 系统

- Linux/Mac 系统下，直接使用 pip 安装如下两个依赖即可

```
pip install cython
pip install pycocotools
```

CHAPTER 3

数据标注、转换、划分

3.1 图像分类

图像分类标注是一项最基础，最简单的标注任务，用户只需将属于同一类的图片放在同一个文件夹下即可，例如下所示目录结构，

```
MyDataset/ # 图像分类数据集根目录
|--dog/ # 当前文件夹所有图片属于 dog 类别
|  |--d1.jpg
|  |--d2.jpg
|  |--...
|  |--...
|
|--...
|
|--snake/ # 当前文件夹所有图片属于 snake 类别
|  |--s1.jpg
|  |--s2.jpg
|  |--...
|  |--...
```

3.1.1 数据划分

在模型进行训练时，我们需要划分训练集，验证集和测试集，因此需要对如上数据进行划分，直接使用 paddlex 命令即可将数据集随机划分成 70% 训练集，20% 验证集和 10% 测试集

```
paddlex --split_dataset --format ImageNet --dataset_dir MyDataset --val_value 0.2 --test_
↪value 0.1
```

划分好的数据集会额外生成 `labels.txt`, `train_list.txt`, `val_list.txt`, `test_list.txt` 四个文件，之后可直接进行训练。

> 注：如您使用 PaddleX 可视化客户端进行模型训练，数据集划分功能集成在客户端内，无需自行使用命令划分

- 图像分类任务训练示例代码

3.2 目标检测

目标检测数据的标注推荐使用 LabelMe 标注工具，如您先前并无安装，那么 LabelMe 的安装可参考LabelMe 安装和启动

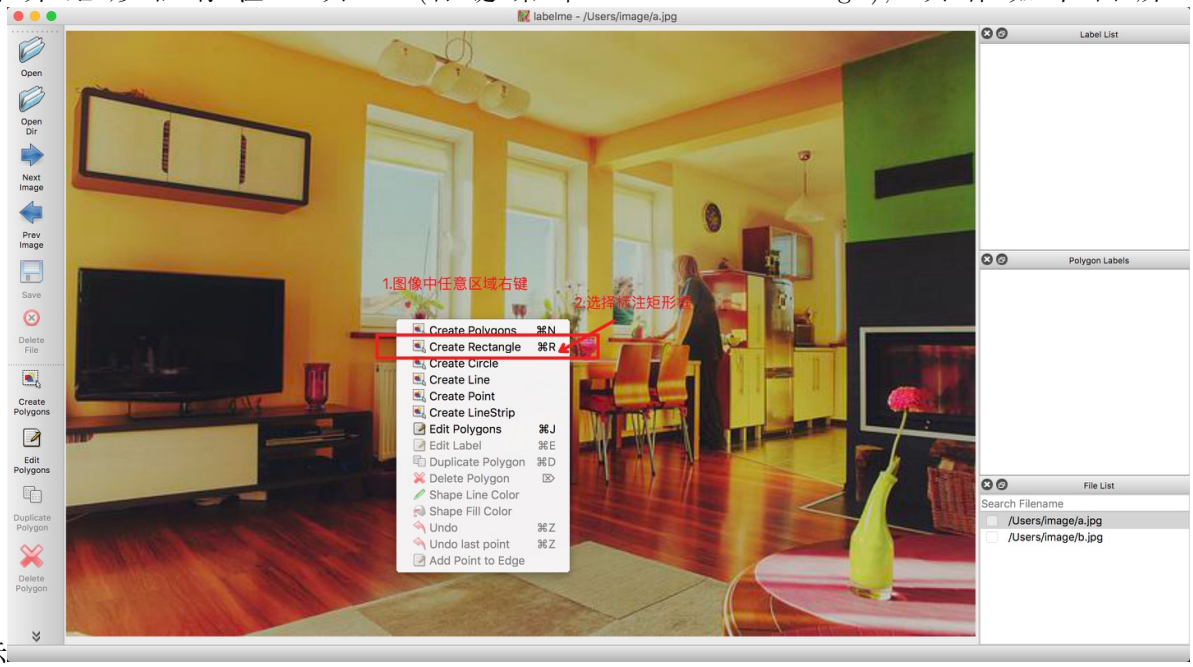
注意：LabelMe 对于中文支持不够友好，因此请不要在如下的路径以及文件名中出现中文字符!

3.2.1 准备工作

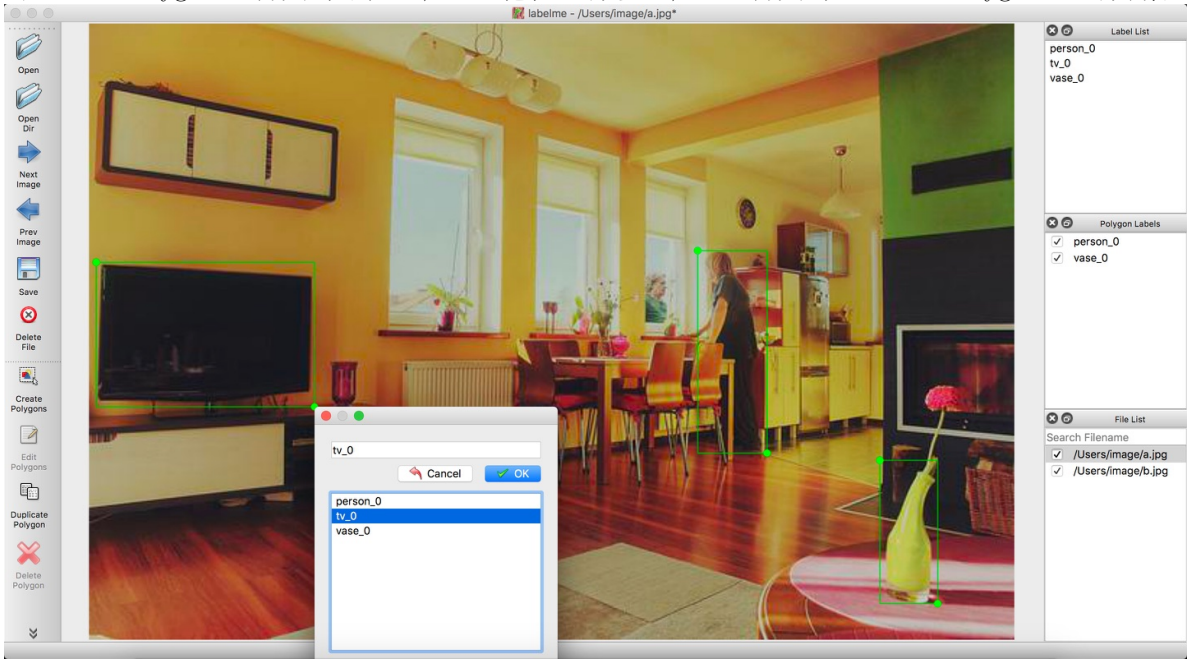
1. 将收集的图像存放于 `JPEGImages` 文件夹下，例如存储在 `D:\MyDataset\JPEGImages`
2. 创建与图像文件夹相对应的文件夹 `Annotations`，用于存储标注的 json 文件，如 `D:MyDataset\Annotations`
3. 打开 LabelMe，点击”Open Dir“按钮，选择需要标注的图像所在的文件夹打开，则”File List“对话框中会显示所有图像所对应的绝对路径，接着便可以开始遍历每张图像，进行标注工作

3.2.2 目标框标注

1. 打开矩形框标注工具 (右键菜单->Create Rectangle)，具体如下图所示



2. 使用拖拉的方式对目标物体进行标识，并在弹出的对话框中写明对应 label（当 label 已存在时点击即可，此处请注意 label 勿使用中文），具体如下图所示，当框标注错误时，可点击左侧的“Edit Polygons”再点击标注框，通过拖拉进行修改，也可再点击“Delete Polygon”进行删除。



3. 点击右侧”Save “，将标注结果保存到中创建的文件夹 Annotations 目录中

3.2.3 格式转换

LabelMe 标注后的数据还需要进行转换为 PascalVOC 或 MSCOCO 格式，才可以用于目标检测任务的训练，创建 D:\dataset_voc 目录，在 python 环境中安装 paddlex 后，使用如下命令即可

```
paddlex --data_conversion --source labelme --to PascalVOC \
        --pics D:\MyDataset\JPEGImages \
        --annotations D:\MyDataset\Annotations \
        --save_dir D:\dataset_voc
```

> 注：此文档中以 LabelMe 为示例，展示了格式转换，如您使用的是数据标注精灵工具，则可在标注完后，选择直接保存为 PascalVOC 格式

3.2.4 数据集划分

转换完数据后，为了进行训练，还需要将数据划分为训练集、验证集和测试集，同样在安装 paddlex 后，使用如下命令即可将数据划分为 70% 训练集，20% 验证集和 10% 的测试集

```
paddlex --split_dataset --format VOC --dataset_dir D:\MyDataset --val_value 0.2 --test_
↪value 0.1
```

执行上面命令行，会在 D:\MyDataset 下生成 labels.txt，train_list.txt，val_list.txt 和 test_list.txt，分别存储类别信息，训练样本列表，验证样本列表，测试样本列表

> 注：如您使用 PaddleX 可视化客户端进行模型训练，数据集划分功能集成在客户端内，无需自行使用命令划分

- 目标检测任务训练示例代码

3.3 实例分割

实例分割数据的标注推荐使用 LabelMe 标注工具，如您先前并无安装，那么 LabelMe 的安装可参考LabelMe 安装和启动

注意：LabelMe 对于中文支持不够友好，因此请不要在如下的路径以及文件名中出现中文字符!

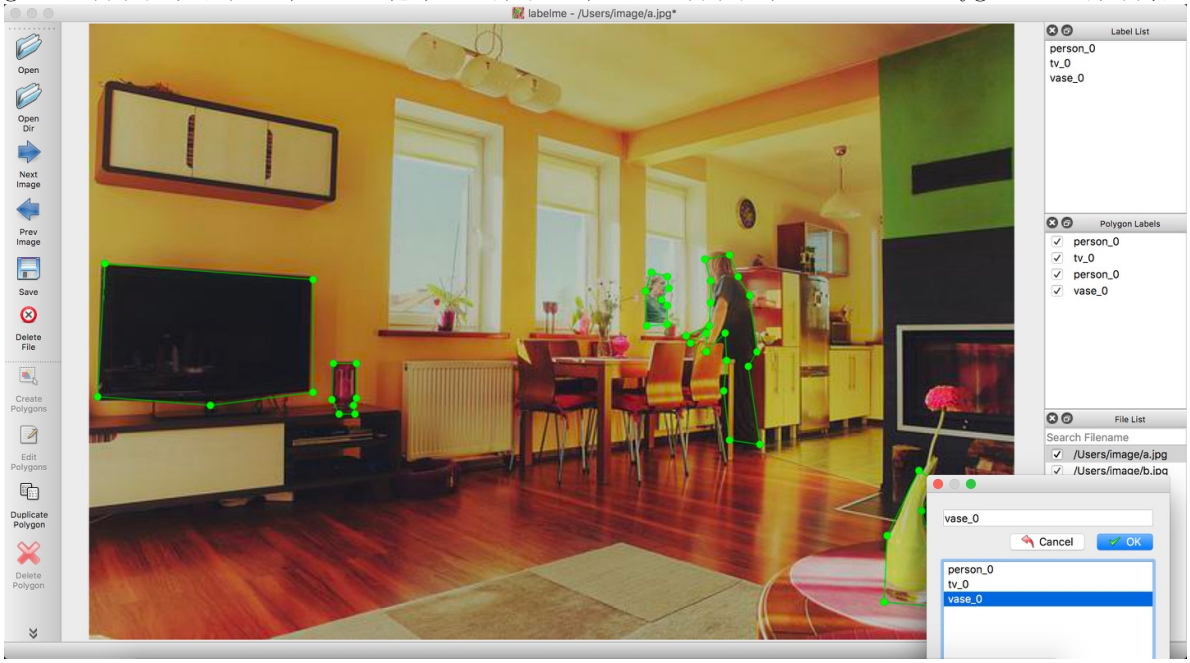
3.3.1 准备工作

1. 将收集的图像存放于 JPEGImages 文件夹下，例如存储在 D:\MyDataset\JPEGImages
2. 创建与图像文件夹相对应的文件夹 Annotations，用于存储标注的 json 文件，如 D:MyDataset\Annotations

3. 打开 LabelMe，点击”Open Dir“按钮，选择需要标注的图像所在的文件夹打开，则”File List“对话框中会显示所有图像所对应的绝对路径，接着便可以开始遍历每张图像，进行标注工作

3.3.2 目标边缘标注

1. 打开多边形标注工具（右键菜单->Create Polygon）以打点的方式圈出目标的轮廓，并在弹出的对话框中写明对应 label（当 label 已存在时点击即可，此处请注意 label 勿使用中文），具体如下提所示，当框标注错误时，可点击左侧的“Edit Polygons”再点击标注框，通过拖拉进行修改，也可再点击“Delete Polygon”进行删除。



2. 点击右侧”Save “，将标注结果保存到中创建的文件夹 Annotations 目录中

3.3.3 格式转换

LabelMe 标注后的数据还需要进行转换为 MSCOCO 格式，才可以用于实例分割任务的训练，创建保存目录 `D:\dataset_seg`，在 python 环境中安装 paddlex 后，使用如下命令即可

```
paddlex --data_conversion --source labelme --to MSCOCO \
        --pics D:\MyDataset\JPEGImages \
        --annotations D:\MyDataset\Annotations \
        --save_dir D:\dataset_coco
```

3.3.4 数据集划分

转换完数据后，为了进行训练，还需要将数据划分为训练集、验证集和测试集，同样在安装 paddlex 后，使用如下命令即可将数据划分为 70% 训练集，20% 验证集和 10% 的测试集

```
paddlex --split_dataset --format COCO --dataset_dir D:\MyDataset --val_value 0.2 --test_
↪value 0.1
```

执行上面命令行，会在 `D:\MyDataset` 下生成 `train.json`, `val.json`, `test.json`，分别存储训练样本信息，验证样本信息，测试样本信息

> 注：如您使用 PaddleX 可视化客户端进行模型训练，数据集划分功能集成在客户端内，无需自行使用命令划分

- 实例分割任务训练示例代码

3.4 语义分割

语义数据的标注推荐使用 LabelMe 标注工具，如您先前并无安装，那么 LabelMe 的安装可参考LabelMe 安装和启动，语义分割的标注与实例分割类似，流程如下

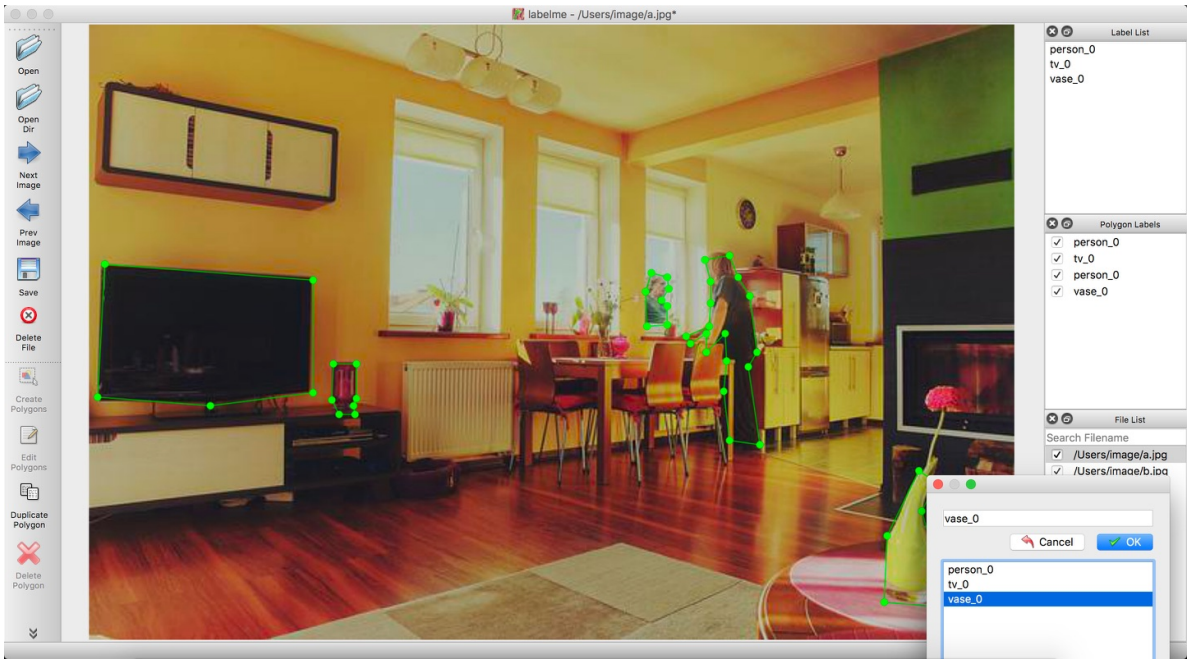
注意：LabelMe 对于中文支持不够友好，因此请不要在如下的路径以及文件名中出现中文字符!

3.4.1 准备工作

1. 将收集的图像存放于 `JPEGImages` 文件夹下，例如存储在 `D:\MyDataset\JPEGImages`
2. 创建与图像文件夹相对应的文件夹 `Annotations`，用于存储标注的 json 文件，如 `D:MyDataset\Annotations`
3. 打开 LabelMe，点击” Open Dir “按钮，选择需要标注的图像所在的文件夹打开，则” File List “对话框中会显示所有图像所对应的绝对路径，接着便可以开始遍历每张图像，进行标注工作

3.4.2 目标边缘标注

1. 打开多边形标注工具（右键菜单->Create Polygon）以打点的方式圈出目标的轮廓，并在弹出的对话框中写明对应 label（当 label 已存在时点击即可，此处请注意 label 勿使用中文），具体如下提所示，当框标注错误时，可点击左侧的 “Edit Polygons” 再点击标注框，通过拖拉进行修改，也可再点击 “Delete Polygon” 进行删除。



2. 点击右侧”Save “，将标注结果保存到中创建的文件夹 Annotations 目录中

3.4.3 格式转换

LabelMe 标注后的数据还需要进行转换为 SEG 格式，才可以用于语义分割任务的训练，创建保存目录 `D:\dataset_seg`，在 python 环境中安装 paddlex 后，使用如下命令即可

```
paddlex --data_conversion --source labelme --to SEG \
        --pics D:\MyDataset\JPEGImages \
        --annotations D:\MyDataset\Annotations \
        --save_dir D:\dataset_seg
```

3.4.4 数据集划分

转换完数据后，为了进行训练，还需要将数据划分为训练集、验证集和测试集，同样在安装 paddlex 后，使用如下命令即可将数据划分为 70% 训练集，20% 验证集和 10% 的测试集

```
paddlex --split_dataset --format SEG --dataset_dir D:\MyDataset --val_value 0.2 --test_
↪value 0.1
```

执行上面命令行，会在 `D:\MyDataset` 下生成 `train_list.txt`, `val_list.txt`, `test_list.txt`，分别存储训练样本信息，验证样本信息，测试样本信息

> 注：如您使用 PaddleX 可视化客户端进行模型训练，数据集划分功能集成在客户端内，无需自行使用命令划分

- 语义分割任务训练示例代码

3.5 LabelMe 的安装和启动

LabelMe 可用于标注目标检测、实例分割、语义分割数据集，是一款开源的标注工具。

3.5.1 1. 安装 Anaconda

推荐使用 Anaconda 安装 python 依赖，有经验的开发者可以跳过此步骤。安装 Anaconda 的方式可以参考文档。

在安装 Anaconda，并创建环境之后，再进行接下来的步骤

3.5.2 2. 安装 LabelMe

进入 Python 环境后，执行如下命令即可

```
conda activate my_paddlex
conda install pyqt
pip install labelme
```

3.5.3 3. 启动 LabelMe

进入安装了 LabelMe 的 Python 环境，执行如下命令即可启动 LabelMe

```
conda activate my_paddlex
labelme
```

CHAPTER 4

数据格式说明

4.1 图像分类 ImageNet

4.1.1 数据文件夹结构

在 PaddleX 中，图像分类支持 ImageNet 数据集格式。数据集目录 `data_dir` 下包含多个文件夹，每个文件夹中的图像均属于同一个类别，文件夹的命名即为类别名（注意路径中不要包括中文，空格）。如下为示例结构

```
MyDataset/ # 图像分类数据集根目录
|--dog/ # 当前文件夹所有图片属于 dog 类别
|   |--d1.jpg
|   |--d2.jpg
|   |--...
|   |--...
|
|--...
|
|--snake/ # 当前文件夹所有图片属于 snake 类别
|   |--s1.jpg
|   |--s2.jpg
|   |--...
|   |--...
```

4.1.2 划分训练集验证集

为了用于训练，我们需要在 MyDataset 目录下准备 train_list.txt，val_list.txt 和 labels.txt 三个文件，分别用于表示训练集列表，验证集列表和类别标签列表。点击下载图像分类示例数据集

> 注：也可使用 PaddleX 自带工具，对数据集进行随机划分，**在数据集按照上面格式组织后**，使用如下命令即可快速完成数据集随机划分，其中 val_value 表示验证集的比例，test_value 表示测试集的比例（可以为 0），剩余的比例用于训练集。

```
paddlex --split_dataset --format ImageNet --dataset_dir MyDataset --val_value␣
↪0.2 --test_value 0.1
```

labels.txt

labels.txt 用于列出所有类别，类别对应行号表示模型训练过程中类别的 id(行号从 0 开始计数)，例如 labels.txt 为以下内容

```
dog
cat
snake
```

即表示该分类数据集中共有 3 个类别，分别为 dog，cat 和 snake，在模型训练中 dog 对应的类别 id 为 0，cat 对应 1，以此类推

train_list.txt

train_list.txt 列出用于训练时的图片集合，与其对应的类别 id，示例如下

```
dog/d1.jpg 0
dog/d2.jpg 0
cat/c1.jpg 1
... ...
snake/s1.jpg 2
```

其中第一列为相对对 MyDataset 的相对路径，第二列为图片对应类别的类别 id

val_list.txt

val_list 列出用于验证时的图片集成，与其对应的类别 id，格式与 train_list.txt 一致

4.1.3 PaddleX 数据集加载

示例代码如下，

```
import paddlex as pdx
from paddlex.cls import transforms
```

(下页继续)

(续上页)

```
train_transforms = transforms.Compose([
    transforms.RandomCrop(crop_size=224), transforms.RandomHorizontalFlip(),
    transforms.Normalize()
])
eval_transforms = transforms.Compose([
    transforms.ResizeByShort(short_size=256),
    transforms.CenterCrop(crop_size=224), transforms.Normalize()
])
train_dataset = pdx.datasets.ImageNet(
                    data_dir='./MyDataset',
                    file_list='./MyDataset/train_list.txt',
                    label_list='./MyDataset/labels.txt',
                    transforms=train_transforms)
eval_dataset = pdx.datasets.ImageNet(
                    data_dir='./MyDataset',
                    file_list='./MyDataset/eval_list.txt',
                    label_list='./MyDataset/labels.txt',
                    transforms=eval_transforms)
```

4.2 目标检测 PascalVOC

4.2.1 数据集文件夹结构

在 PaddleX 中，目标检测支持 PascalVOC 数据集格式。建议用户将数据集按照如下方式进行组织，原图均放在同一目录，如 `JPEGImages`，标注的同名 xml 文件均放在同一目录，如 `Annotations`，示例如下

```
MyDataset/ # 目标检测数据集根目录
|--JPEGImages/ # 原图文件所在目录
|   |--1.jpg
|   |--2.jpg
|   |--...
|   |--...
|
|--Annotations/ # 标注文件所在目录
|   |--1.xml
|   |--2.xml
|   |--...
|   |--...
```

4.2.2 划分训练集验证集

为了用于训练，我们需要在 MyDataset 目录下准备 train_list.txt，val_list.txt 和 labels.txt 三个文件，分别用于表示训练集列表，验证集列表和类别标签列表。点击下载目标检测示例数据集

> 注：也可使用 PaddleX 自带工具，对数据集进行随机划分，**在数据集按照上面格式组织后**，使用如下命令即可快速完成数据集随机划分，其中 val_value 表示验证集的比例，test_value 表示测试集的比例（可以为 0），剩余的比例用于训练集。

```
paddlex --split_dataset --format VOC --dataset_dir MyDataset --val_value 0.2 --
↪test_value 0.1
```

labels.txt

labels.txt 用于列出所有类别，类别对应行号表示模型训练过程中类别的 id(行号从 0 开始计数)，例如 labels.txt 为以下内容

```
dog
cat
snake
```

表示该检测数据集中共有 3 个目标类别，分别为 dog，cat 和 snake，在模型训练中 dog 对应的类别 id 为 0, cat 对应 1，以此类推

train_list.txt

train_list.txt 列出用于训练时的图片集合，与其对应的标注文件，示例如下

```
JPEGImages/1.jpg Annotations/1.xml
JPEGImages/2.jpg Annotations/2.xml
... ...
```

其中第一列为原图相对 MyDataset 的相对路径，第二列为标注文件相对 MyDataset 的相对路径

val_list.txt

val_list 列出用于验证时的图片集成，与其对应的标注文件，格式与 val_list.txt 一致

4.2.3 PaddleX 数据集加载

示例代码如下，

```
import paddlex as pdx
from paddlex.det import transforms

train_transforms = transforms.Compose([
```

(下页继续)

(续上页)

```
    transforms.RandomHorizontalFlip(),
    transforms.Normalize(),
    transforms.ResizeByShort(short_size=800, max_size=1333),
    transforms.Padding(coarsest_stride=32)
])

eval_transforms = transforms.Compose([
    transforms.Normalize(),
    transforms.ResizeByShort(short_size=800, max_size=1333),
    transforms.Padding(coarsest_stride=32),
])

train_dataset = pdx.datasets.VOCDetection(
                        data_dir='./MyDataset',
                        file_list='./MyDataset/train_list.txt',
                        label_list='./MyDataset/labels.txt',
                        transforms=train_transforms)
eval_dataset = pdx.datasets.VOCDetection(
                        data_dir='./MyDataset',
                        file_list='./MyDataset/val_list.txt',
                        label_list='MyDataset/labels.txt',
                        transforms=eval_transforms)
```

4.3 实例分割 MSCOCO

4.3.1 数据集文件夹结构

在 PaddleX 中，实例分割支持 MSCOCO 数据集格式（MSCOCO 格式同样也可以用于目标检测）。建议用户将数据集按照如下方式进行组织，原图均放在同一目录，如 JPEGImages，标注文件（如 annotations.json）放在与 JPEGImages 所在目录同级目录下，示例结构如下

```
MyDataset/ # 实例分割数据集根目录
|--JPEGImages/ # 原图文件所在目录
|   |--1.jpg
|   |--2.jpg
|   |--...
|   |--...
|
|--annotations.json # 标注文件所在目录
```

4.3.2 划分训练集验证集

在 PaddleX 中，为了区分训练集和验证集，在 `MyDataset` 同级目录，使用不同的 json 表示数据的划分，例如 `train.json` 和 `val.json`。点击下载实例分割示例数据集。

> 注：也可使用 PaddleX 自带工具，对数据集进行随机划分，**在数据集按照上面格式组织后**，使用如下命令即可快速完成数据集随机划分，其中 val_value 表示验证集的比例，test_value 表示测试集的比例（可以为 0），剩余的比例用于训练集。

```
paddlex --split_dataset --format COCO --dataset_dir MyDataset --val_value 0.2 -
↪-test_value 0.1
```

MSCOCO 数据的标注文件采用 json 格式，用户可使用 Labelme, 精灵标注助手或 EasyData 等标注工具进行标注，参见数据标注工具

4.3.3 PaddleX 加载数据集

示例代码如下，

```
import paddlex as pdx
from paddlex.det import transforms

train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.Normalize(),
    transforms.ResizeByShort(short_size=800, max_size=1333),
    transforms.Padding(coarsest_stride=32)
])

eval_transforms = transforms.Compose([
    transforms.Normalize(),
    transforms.ResizeByShort(short_size=800, max_size=1333),
    transforms.Padding(coarsest_stride=32),
])

train_dataset = pdx.dataset.CocoDetection(
                    data_dir='./MyDataset/JPEGImages',
                    ann_file='./MyDataset/train.json',
                    transforms=train_transforms)
eval_dataset = pdx.dataset.CocoDetection(
                    data_dir='./MyDataset/JPEGImages',
                    ann_file='./MyDataset/val.json',
```

(下页继续)

(续上页)

```
                    transforms=eval_transforms)
```

4.4 语义分割 Seg

4.4.1 数据集文件夹结构

在 PaddleX 中，**标注文件为 png 文件**。建议用户将数据集按照如下方式进行组织，原图均放在同一目录，如 JPEGImages，标注的同名 png 文件均放在同一目录，如 Annotations，示例如下

```
MyDataset/ # 语义分割数据集根目录
|--JPEGImages/ # 原图文件所在目录
|   |--1.jpg
|   |--2.jpg
|   |--...
|   |--...
|
|--Annotations/ # 标注文件所在目录
|   |--1.png
|   |--2.png
|   |--...
|   |--...
```

语义分割的标注图像，如 1.png，为单通道图像，像素标注类别需要从 0 开始递增（一般 0 表示 background 背景），例如 0，1，2，3 表示 4 种类别，标注类别最多 255 个类别 (其中像素值 255 不参与训练和评估)。

4.4.2 划分训练集验证集

为了用于训练，我们需要在 MyDataset 目录下准备 train_list.txt，val_list.txt 和 labels.txt 三个文件，分别用于表示训练集列表，验证集列表和类别标签列表。点击下载语义分割示例数据集

> 注：也可使用 PaddleX 自带工具，对数据集进行随机划分，**在数据集按照上面格式组织后**，使用如下命令即可快速完成数据集随机划分，其中 val_value 表示验证集的比例，test_value 表示测试集的比例（可以为 0），剩余的比例用于训练集。

```
paddlex --split_dataset --format Seg --dataset_dir MyDataset --val_value 0.2 --
↪test_value 0.1
```

labels.txt

labels.txt 用于列出所有类别，类别对应行号表示模型训练过程中类别的 id(行号从 0 开始计数)，例如 labels.txt 为以下内容

```
background
human
car
```

表示该检测数据集中共有 3 个分割类别，分别为 background，human 和 car，在模型训练中 background 对应的类别 id 为 0, human 对应 1，以此类推，如不知具体类别标签，可直接在 labels.txt 逐行写 0，1，2··· 序列即可。

train_list.txt

train_list.txt 列出用于训练时的图片集合，与其对应的标注文件，示例如下

```
JPEGImages/1.jpg Annotations/1.png
JPEGImages/2.jpg Annotations/2.png
... ...
```

其中第一列为原图相对 MyDataset 的相对路径，第二列为标注文件相对 MyDataset 的相对路径

val_list.txt

val_list 列出用于验证时的图片集成，与其对应的标注文件，格式与 val_list.txt 一致

4.4.3 PaddleX 数据集加载

示例代码如下，

```
import paddlex as pdx
from paddlex.seg import transforms

train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.ResizeRangeScaling(),
    transforms.RandomPaddingCrop(crop_size=512),
    transforms.Normalize()
])

eval_transforms = transforms.Compose([
    transforms.ResizeByLong(long_size=512),
    transforms.Padding(target_size=512),
    transforms.Normalize()
])

train_dataset = pdx.datasets.SegDataset(
```

(下页继续)

(续上页)

```
                        data_dir='./MyDataset',
                        file_list='./MyDataset/train_list.txt',
                        label_list='./MyDataset/labels.txt',
                        transforms=train_transforms)
eval_dataset = pdx.datasets.SegDataset(
                        data_dir='./MyDataset',
                        file_list='./MyDataset/val_list.txt',
                        label_list='MyDataset/labels.txt',
                        transforms=eval_transforms)
```

4.5 地块检测 ChangeDet

4.5.1 数据集文件夹结构

在 PaddleX 中，**标注文件为 png 文件**。建议用户将数据集按照如下方式进行组织，同一地块不同时期的地貌原图均放在同一目录，如 JPEGImages，标注的同名 png 文件均放在同一目录，如 Annotations，示例如下

```
MyDataset/ # 语义分割数据集根目录
|--JPEGImages/ # 原图文件所在目录，包含同一物体前期和后期的图片
|  |--1_1.jpg
|  |--1_2.jpg
|  |--2_1.jpg
|  |--2_2.jpg
|  |--...
|  |--...
|
|--Annotations/ # 标注文件所在目录
|  |--1.png
|  |--2.png
|  |--...
|  |--...
```

同一地块不同时期的地貌原图，如 1_1.jpg 和 1_2.jpg，可以是 RGB 彩色图像、灰度图、或 tiff 格式的多通道图像。语义分割的标注图像，如 1.png，为单通道图像，像素标注类别需要从 0 开始递增（一般 0 表示 background 背景), 例如 0，1，2，3 表示 4 种类别，标注类别最多 255 个类别 (其中像素值 255 不参与训练和评估)。

4.5.2 划分训练集验证集

为了用于训练，我们需要在 `MyDataset` 目录下准备 `train_list.txt`, `val_list.txt` 和 `labels.txt` 三个文件，分别用于表示训练集列表，验证集列表和类别标签列表。

labels.txt

labels.txt 用于列出所有类别，类别对应行号表示模型训练过程中类别的 id(行号从 0 开始计数)，例如 labels.txt 为以下内容

```
unchanged
changed
```

表示该检测数据集中共有 2 个分割类别，分别为 `unchanged` 和 `changed`，在模型训练中 `unchanged` 对应的类别 id 为 0, `changed` 对应 1，以此类推，如不知具体类别标签，可直接在 labels.txt 逐行写 0，1，2…序列即可。

train_list.txt

train_list.txt 列出用于训练时的图片集合，与其对应的标注文件，示例如下

```
JPEGImages/1_1.jpg JPEGImages/1_2.jpg Annotations/1.png
JPEGImages/2_1.jpg JPEGImages/2_2.jpg Annotations/2.png
... ...
```

其中第一列和第二列为原图相对 `MyDataset` 的相对路径，对应同一地块不同时期的地貌图像，第三列为标注文件相对 `MyDataset` 的相对路径

val_list.txt

val_list 列出用于验证时的图片集成，与其对应的标注文件，格式与 val_list.txt 一致

4.5.3 PaddleX 数据集加载

示例代码

CHAPTER 5

模型训练

PaddleX 集成了 PaddleClas、PaddleDetection 和 PaddleSeg 三大 CV 工具套件中在工业领域应用成熟的模型，并提供了统一易用的 API 使用接口，帮助用户快速完成视觉领域的图像分类、目标检测、实例分割和语义分割模型的训练。

5.1 图像分类

5.1.1 介绍

PaddleX 共提供了 20+ 的图像分类模型，可满足开发者不同场景的需求下的使用。

- **Top1 精度**: 模型在 ImageNet 数据集上的测试精度
- **预测速度**：单张图片的预测用时（不包括预处理和后处理)
- “-”表示指标暂未更新

5.1.2 开始训练

将代码保存到本地后运行（代码下载链接位于上面的表格），**代码会自动下载训练数据并开始训练**。如保存为 `mobilenetv3_small_ssld.py`，执行如下命令即可开始训练：

```
python mobilenetv3_small_ssld.py
```

5.1.3 相关文档

- **【重要】**针对自己的机器环境和数据，调整训练参数？先了解下 PaddleX 中训练参数作用。——>> 传送门
- **【有用】**没有机器资源？使用 AIStudio 免费的 GPU 资源在线训练模型。——>> 传送门
- **【拓展】**更多图像分类模型，查阅*PaddleX 模型库*和*API 使用文档*。

5.2 目标检测

5.2.1 介绍

PaddleX 目前提供了 FasterRCNN 和 YOLOv3 两种检测结构，多种 backbone 模型，可满足开发者不同场景和性能的需求。

- **Box MMAP**: 模型在 COCO 数据集上的测试精度
- **预测速度**：单张图片的预测用时（不包括预处理和后处理）
- “-”表示指标暂未更新

5.2.2 开始训练

将代码保存到本地后运行（代码下载链接位于上面的表格），**代码会自动下载训练数据并开始训练**。如保存为 `yolov3_mobilenetv1.py`，执行如下命令即可开始训练:

```
python yolov3_mobilenetv1.py
```

5.2.3 相关文档

- **【重要】**针对自己的机器环境和数据，调整训练参数？先了解下 PaddleX 中训练参数作用。——>> 传送门
- **【有用】**没有机器资源？使用 AIStudio 免费的 GPU 资源在线训练模型。——>> 传送门
- **【拓展】**更多目标检测模型，查阅*PaddleX 模型库*和*API 使用文档*。

5.3 实例分割

5.3.1 介绍

PaddleX 目前提供了 MaskRCNN 实例分割模型结构，多种 backbone 模型，可满足开发者不同场景和性能的需求。

- **Box MMAP/Seg MMAP**: 模型在 COCO 数据集上的测试精度
- **预测速度**：单张图片的预测用时（不包括预处理和后处理)
- “-”表示指标暂未更新

5.3.2 开始训练

将代码保存到本地后运行（代码下载链接位于上面表格中），**代码会自动下载训练数据并开始训练**。如保存为 `mask_rcnn_r50_fpn.py`，执行如下命令即可开始训练：

```
python mask_rcnn_r50_fpn.py
```

5.3.3 相关文档

- **【重要】**针对自己的机器环境和数据，调整训练参数？先了解下 PaddleX 中训练参数作用。——>> 传送门
- **【有用】**没有机器资源？使用 AIStudio 免费的 GPU 资源在线训练模型。——>> 传送门
- **【拓展】**更多实例分割模型，查阅*PaddleX 模型库*和*API 使用文档*。

5.4 语义分割

5.4.1 介绍

PaddleX 目前提供了 DeepLabv3p、UNet、HRNet 和 FastSCNN 四种语义分割结构, 多种 backbone 模型，可满足开发者不同场景和性能的需求。

- **mIoU**: 模型在 CityScape 数据集上的测试精度
- **预测速度**：单张图片的预测用时（不包括预处理和后处理)
- “-”表示指标暂未更新

5.4.2 开始训练

将代码保存到本地后运行（代码下载链接位于上面的表格中），**代码会自动下载训练数据并开始训练**。如保存为 `deeplabv3p_mobilenetv2_x0.25.py`，执行如下命令即可开始训练：

```
python deeplabv3p_mobilenetv2_x0.25.py
```

5.4.3 相关文档

- **【重要】**针对自己的机器环境和数据，调整训练参数？先了解下 PaddleX 中训练参数作用。——>> 传送门
- **【有用】**没有机器资源？使用 AIStudio 免费的 GPU 资源在线训练模型。——>> 传送门
- **【拓展】**更多语义分割模型，查阅*PaddleX 模型库*和*API 使用文档*。

5.5 VisualDL 可视化训练指标

在使用 PaddleX 训练模型过程中，各个训练指标和评估指标会直接输出到标准输出流，同时也可通过 VisualDL 对训练过程中的指标进行可视化，只需在调用 `train` 函数时，将 `use_vdl` 参数设为 `True` 即可，如下代码所示，

```
model = paddlex.cls.ResNet50(num_classes=1000)
model.train(num_epochs=120, train_dataset=train_dataset,
            train_batch_size=32, eval_dataset=eval_dataset,
            log_interval_steps=10, save_interval_epochs=10,
            save_dir='./output', use_vdl=True)
```

模型在训练过程中，会在 `save_dir` 下生成 `vdl_log` 目录，通过在命令行终端执行以下命令，启动 VisualDL。

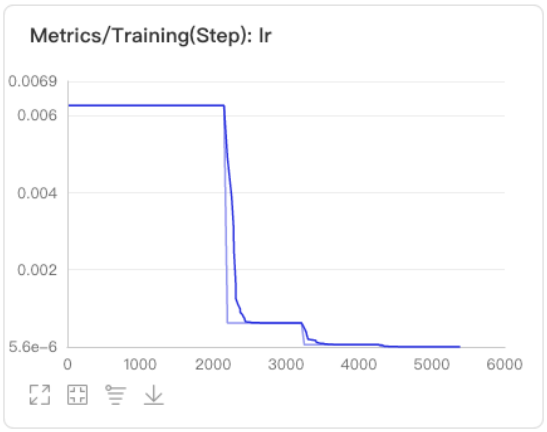
```
visualdl --logdir=output/vdl_log --port=8008
```

在浏览器打开 `http://0.0.0.0:8008` 便可直接查看随训练迭代动态变化的各个指标（0.0.0.0 表示启动 VisualDL 所在服务器的 IP，本机使用 0.0.0.0 即可）。

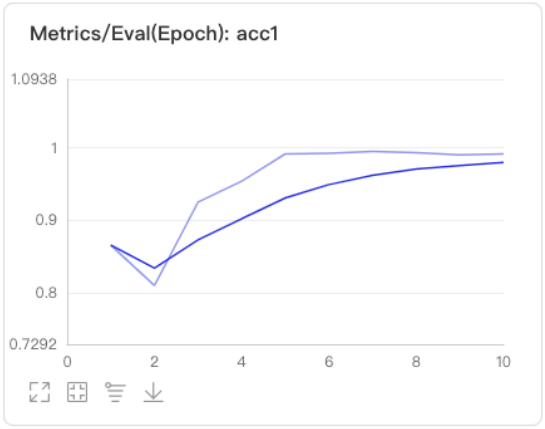
在训练分类模型过程中，使用 VisualDL 进行可视化的示例图如下所示。

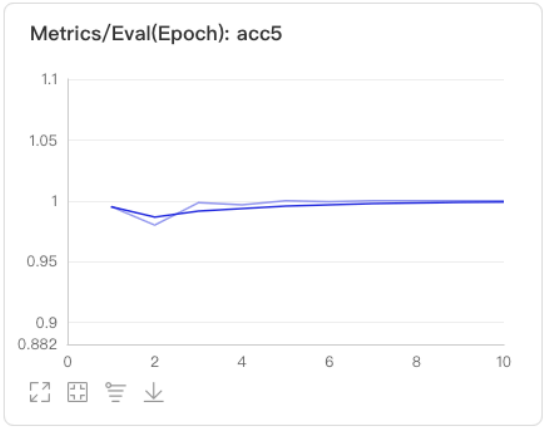
训练过程中每个 Step 的 Loss 和相应 Top1 准确率变化趋势：

训练过程中每个 Step 的学习率 lr 和相应 Top5 准确率变化趋势:



训练过程中，每次保存模型时，模型在验证数据集上的 Top1 准确率和 Top5 准确率:





CHAPTER 6

加载模型预测

PaddleX 可以使用 `paddlex.load_model` 接口加载模型（包括训练过程中保存的模型，导出的部署模型，量化模型以及裁剪的模型）进行预测，同时 PaddleX 中也内置了一系列的可视化工具函数，帮助用户方便地检查模型的效果。

注意：使用 `paddlex.load_model` 接口加载仅用于模型预测，如需要在此模型基础上继续训练，可以将该模型作为预训练模型进行训练，具体做法是在训练代码中，将 train 函数中的 `pretrain_weights` 参数指定为预训练模型路径。

6.1 图像分类

点击下载如下示例代码中的模型

```
import paddlex as pdx
test_jpg = 'mobilenetv3_small_ssld_imagenet/test.jpg'
model = pdx.load_model('mobilenetv3_small_ssld_imagenet')
result = model.predict(test_jpg)
print("Predict Result: ", result)
```

结果输出如下：

```
Predict Result: [{'category_id': 549, 'category': 'envelope', 'score': 0.29062933}]
```

测试图片如下：

- 分类模型 predict 接口说明文档

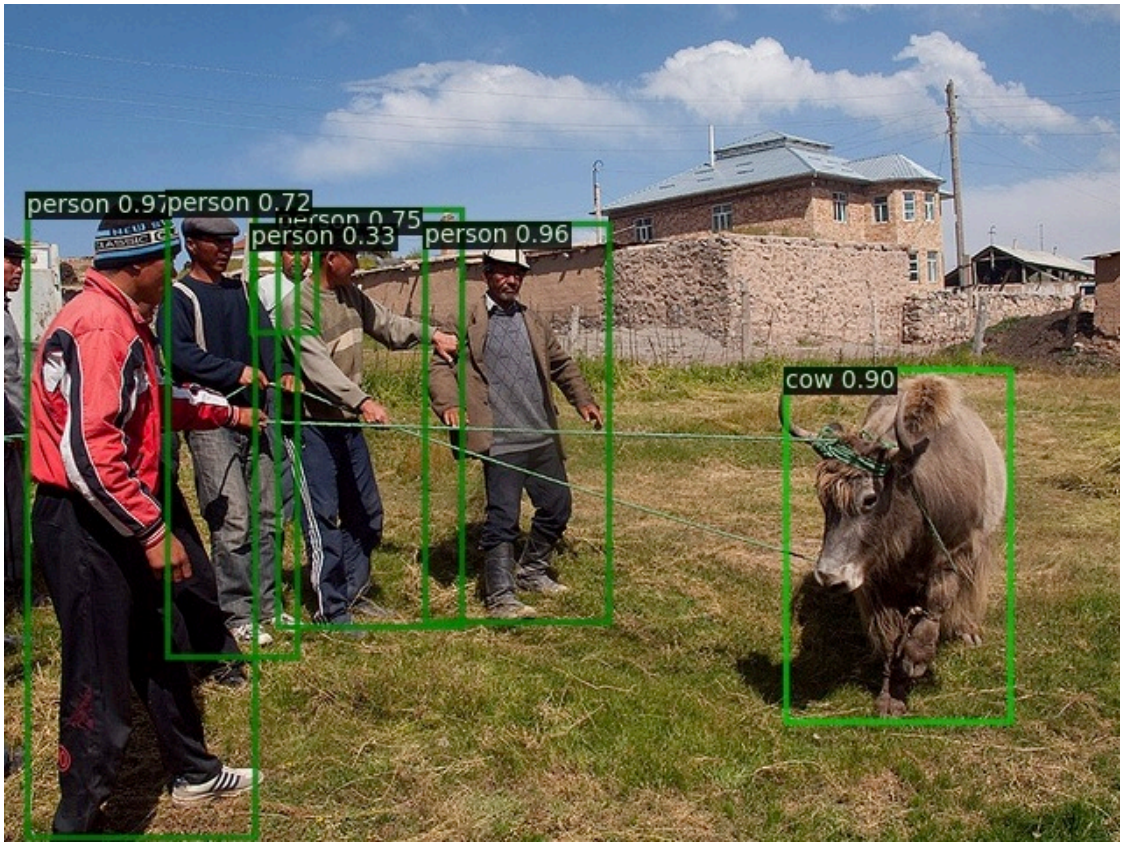
6.2 目标检测

点击下载如下示例代码中模型

```
import paddlex as pdx
test_jpg = 'yolov3_mobilenetv1_coco/test.jpg'
model = pdx.load_model('yolov3_mobilenetv1_coco')

# predict 接口并未过滤低置信度识别结果，用户根据需求按 score 值进行过滤
result = model.predict(test_jpg)

# 可视化结果存储在./visualized_test.jpg，见下图
pdx.det.visualize(test_jpg, result, threshold=0.3, save_dir='./')
```

- YOLOv3 模型 predict 接口说明文档
- 可视化 pdx.det.visualize 接口说明文档

  注意：目标检测和实例分割模型在调用 predict 接口得到的结果需用户自行过滤低置信度结果，在 paddlex.det.visualize 接口中，我们提供了 threshold 用于过滤，置信度低于此值的结果将被过滤，不会可视化。



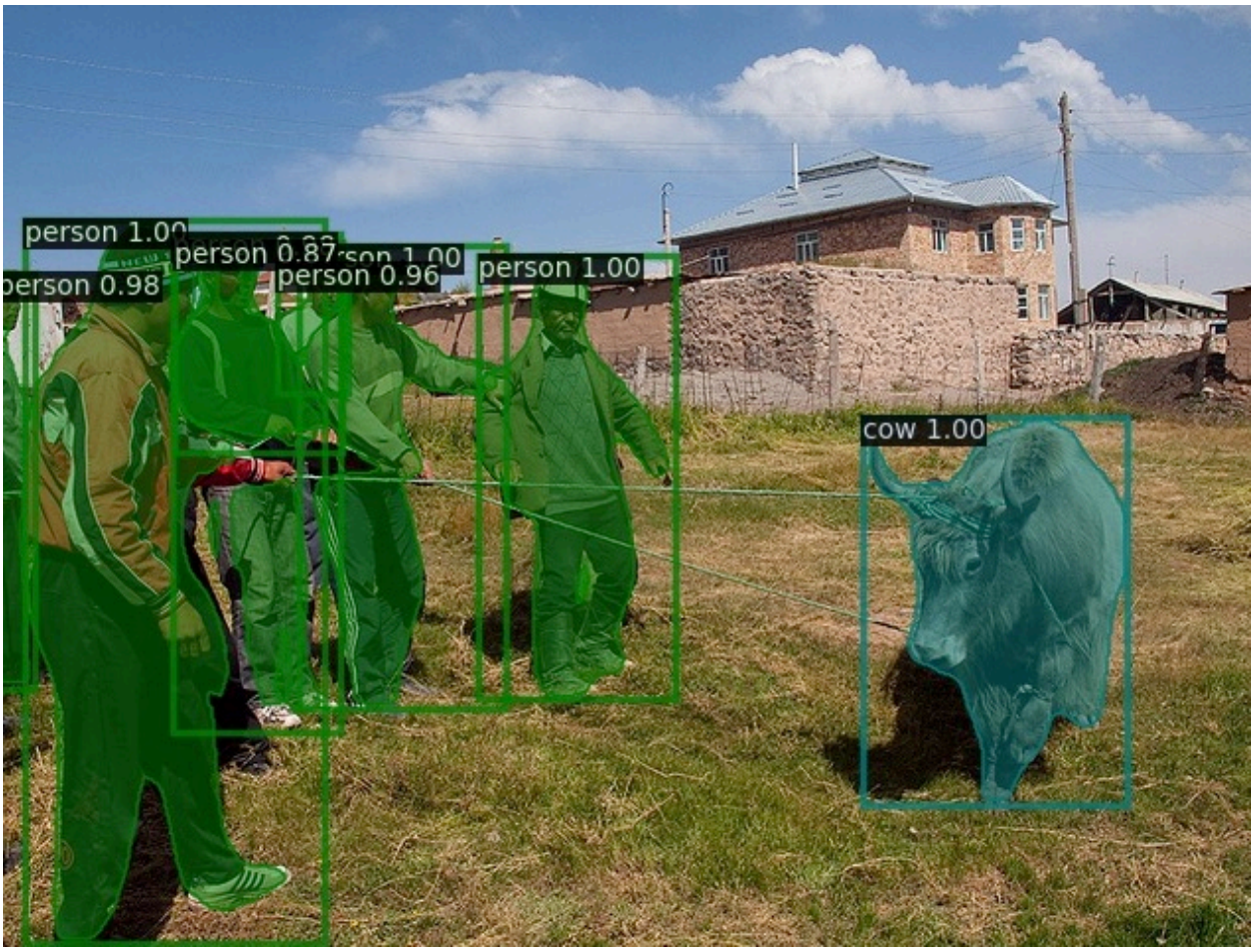
6.3 实例分割

点击下载如下示例代码中模型

```
import paddlex as pdx
test_jpg = 'mask_r50_fpn_coco/test.jpg'
model = pdx.load_model('mask_r50_fpn_coco')

# predict 接口并未过滤低置信度识别结果，用户根据需求按 score 值进行过滤
result = model.predict(test_jpg)

# 可视化结果存储在./visualized_test.jpg, 见下图
pdx.det.visualize(test_jpg, result, threshold=0.5, save_dir='./')
```

- MaskRCNN 模型 predict 接口说明文档
- 可视化 pdx.det.visualize 接口说明文档

注意：目标检测和实例分割模型在调用 predict 接口得到的结果需用户自行过滤低置信度结果，在 paddlex.det.visualize 接口中，我们提供了 threshold 用于过滤，置信度低于此值的结果将被过滤，不会可视化。

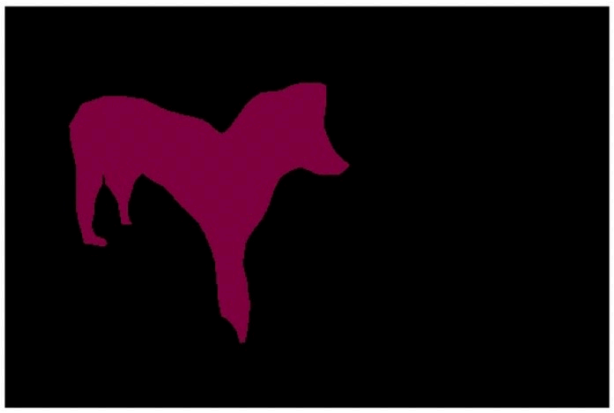


6.4 语义分割

点击下载如下示例代码中模型

```
import paddlex as pdx
test_jpg = './deeplabv3p_mobilenetv2_voc/test.jpg'
model = pdx.load_model('./deeplabv3p_mobilenetv2_voc')
result = model.predict(test_jpg)
# 可视化结果存储在./visualized_test.jpg，见下图右（左图为原图）
pdx.seg.visualize(test_jpg, result, weight=0.0, save_dir='./')
```

在上述示例代码中，通过调用 `paddlex.seg.visualize` 可以对语义分割的预测结果进行可视化，可视化的结果保存在 `save_dir` 下，见下图。其中 `weight` 参数用于调整预测结果和原图结果融合展现时的权重，0.0 时只展示预测结果 mask 的可视化，1.0 时只展示原图可视化。



6.5 公开数据集训练模型下载

PaddleX 提供了部分公开数据集上训练好的模型，用户可以直接下载后参照本文档加载使用。

PaddleX 的 `load_model` 接口可以满足用户一般的模型调研需求，如果是追求更高性能的预测部署，可以参考如下文档

- 服务端 *Python* 部署
- 服务端 C++ 部署

CHAPTER 7

训练参数调整

PaddleX 所有训练接口中，内置的参数均为根据单 GPU 卡相应 batch_size 下的较优参数，用户在自己的数据上训练模型，涉及到参数调整时，如无太多参数调优经验，则可参考如下方式

7.1 1.num_epochs 的调整

num_epochs 是模型训练迭代的总轮数 (模型对训练集全部样本过一遍即为一个 epoch)，用户可以设置较大的数值，根据模型迭代过程在验证集上的指标表现，来判断模型是否收敛，进而提前终止训练。此外也可以使用 `train` 接口中的 `early_stop` 策略，模型在训练过程会自动判断模型是否收敛自动中止。

7.2 2.batch_size 和 learning_rate

- Batch Size 指模型在训练过程中，前向计算一次 (即为一个 step) 所用到的样本数量
- 如若使用多卡训练，batch_size 会均分到各张卡上（因此需要让 batch size 整除卡数）
- Batch Size 跟机器的显存/内存高度相关，`batch_size` 越高，所消耗的显存/内存就越高
- PaddleX 在各个 `train` 接口中均配置了默认的 batch size(默认针对单 GPU 卡), 如若训练时提示 GPU 显存不足，则相应调低 BatchSize，如若 GPU 显存高或使用多张 GPU 卡时，可相应调高 BatchSize。
- **如若用户调整 batch size，则也注意需要对应调整其它参数，特别是 train 接口中默认的 learning_rate 值**。如在 YOLOv3 模型中，默认 `train_batch_size` 为 8，`learning_rate` 为 0.000125，当用户将模型在 2 卡机器上训练时，可以将 `train_batch_size` 调整为 16, 那么同时 `learning_rate` 也可以对应调整为 0.000125 * 2 = 0.00025

7.3 3.warmup_steps 和 warmup_start_lr

在训练模型时，一般都会使用预训练模型，例如检测模型在训练时使用 backbone 在 ImageNet 数据集上的预训练权重。但由于在自行训练时，自己的数据与 ImageNet 数据集存在较大的差异，可能会一开始由于梯度过大使得训练出现问题，这种情况下可以在刚开始训练时，让学习率以一个较小的值，慢慢增长到设定的学习率。`warmup_steps` 和 `warmup_start_lr` 就是起到这个作用，模型开始训练时，学习率会从 `warmup_start_lr` 开始，在 `warmup_steps` 个 batch 数据迭代后线性增长到设定的学习率。

> 例如 YOLOv3 的 train 接口，默认 `train_batch_size` 为 8，`learning_rate` 为 0.000125，`warmup_steps` 为 1000，`warmup_start_lr` 为 0.0；在此参数配置下表示，模型在启动训练后，在前 1000 个 step(每个 step 使用一个 batch 的数据，即 8 个样本) 内，学习率会从 0.0 开始线性增长到设定的 0.000125。

7.4 4.lr_decay_epochs 和 lr_decay_gamma

`lr_decay_epochs` 用于让学习率在模型训练后期逐步衰减，它一般是一个 list，如 [6, 8, 10]，表示学习率在第 6 个 epoch 时衰减一次，第 8 个 epoch 时再衰减一次，第 10 个 epoch 时再衰减一次。每次学习率衰减为之前的学习率 *lr_decay_gamma。

> 例如 YOLOv3 的 train 接口，默认 `num_epochs` 为 270,`learning_rate` 为 0.000125，`lr_decay_epochs` 为 [213, 240]，`lr_decay_gamma` 为 0.1; 在此参数配置下表示，模型在启动训练后，在前 213 个 epoch 中，训练时使用的学习率为 0.000125，在第 213 至 240 个 epoch 之间，训练使用的学习率为 0.000125x0.1=0.0000125，在 240 个 epoch 之后，使用的学习率为 0.000125x0.1x0.1=0.00000125

7.5 5. 参数设定时的约束

根据上述几个参数，可以了解到学习率的变化分为 WarmUp 热身阶段和 Decay 衰减阶段，

- Wamup 热身阶段：随着训练迭代，学习率从较低的值逐渐线性增长至设定的值，以 step 为单位
- Decay 衰减阶段：随着训练迭代，学习率逐步衰减，如每次衰减为之前的 0.1，以 epoch 为单位
- step 与 epoch 的关系：1 个 epoch 由多个 step 组成，例如训练样本有 800 张图像，`train_batch_size` 为 8, 那么每个 epoch 都要完整用这 800 张图片训一次模型，而每个 epoch 总共包含 800//8 即 100 个 step

在 PaddleX 中，约束 warmup 必须在 Decay 之前结束，因此各参数设置需要满足下面条件

```
warmup_steps <= lr_decay_epochs[0] * num_steps_each_epoch
```

其中 `num_steps_each_epoch` 计算方式如下，

```
num_steps_each_eposh = num_samples_in_train_dataset // train_batch_size
```

因此，如若你在启动训练时，被提示 `warmup_steps should be less than...` 时，即表示需要根据上述公式调整你的参数啦，可以调整 `lr_decay_epochs` 或者是 `warmup_steps`。

7.6 6. 如何使用多 GPU 卡进行训练

在 `import paddlex` 前配置环境变量，代码如下

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0' # 使用 0 号 GPU 卡进行训练
# 注意 paddle 或 paddlex 都需要在设置环境变量后再 import
import paddlex as pdx
```

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '' # 不使用 GPU, 使用 CPU 进行训练
import paddlex as pdx
```

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1,3' # 同时使用第 0、1、3 号 GPU 卡进行训练
import paddlex as pdx
```

7.7 相关模型接口

- 图像分类模型 train 接口
- 目标检测 FasterRCNN train 接口
- 目标检测 YOLOv3 train 接口
- 实例分割 MaskRCNN train 接口
- 语义分割 train 接口

CHAPTER 8

模型保存

8.1 训练过程保存

PaddleX 在模型训练过程中，根据 `train` 函数接口中的 `save_interval_epoch` 参数设置，每间隔相应轮数保存一次模型，模型目录中包含了 `model.pdparams`, `model.yml` 等文件。

在训练过程中保存的模型，可用于作为 pretrain_weights 继续训练模型，也可使用 `paddlex.load_model` 接口加载测试模型的预测和评估等。

8.2 部署模型导出

在前面提到的训练中保存的模型，如若要用于部署（部署可参阅 PaddleX 文档中的模型多端部署章节），需导出为部署的模型格式，部署的模型目录中包含 `__model__`，`__params__` 和 `model.yml` 三个文件。

模型部署在 Python 层面，可以使用基于高性能预测库的 python 接口 `paddlex.deploy.Predictor`，也可使用 `paddlex.load_model` 接口。

模型部署可参考文档部署模型导出

> 【总结】如若模型目录中包含 `model.pdparams`，那说明模型是训练过程中保存的，部署时需要进行导出；部署的模型目录中需包含 `__model__`，`__params__` 和 `model.yml` 三个文件。

8.3 模型部署文件说明

- `__model__`：保存了模型的网络结构信息
- `__params__`：保存了模型网络中的参数权重
- `model.yml`：在 PaddleX 中，将模型的预处理，后处理，以及类别相关信息均存储在此文件中

8.4 模型导出为 ONNX 模型

PaddleX 作为开放开源的套件，其中的大部分模型均支持导出为 ONNX 协议，满足开发者多样性的需求。

> 需要注意的是 ONNX 存在多个 OpSet 版本，下表为 PaddleX 各模型支持导出的 ONNX 协议版本。

8.4.1 如何导出

- 1. 参考文档部署模型导出，将训练保存的模型导出为部署模型
- 1. 安装 paddle2onnx `pip install paddle2onnx`，转换命令如下，通过`--opset_version` 指定版本(9/10/11)，转换使用方法参考Paddle2ONNX 说明
- 附: Paddle2ONNX 参阅 https://github.com/PaddlePaddle/paddle2onnx

CHAPTER 9

模型裁剪

模型裁剪可以更好地满足在端侧、移动端上部署场景下的性能需求，可以有效得降低模型的体积，以及计算量，加速预测性能。PaddleX 集成了 PaddleSlim 的基于敏感度的通道裁剪算法，用户可以在 PaddleX 的训练代码里轻松使用起来。

在本文档中展示了分类模型的裁剪过程，文档中代码以及更多其它模型的的裁剪代码可在 Github 中的tutorials/slim/prune目录获取。

9.1 使用方法

模型裁剪相对比我们普通训练一个模型，步骤会多出两步

- 1. 采用正常的方式训练一个模型
- 2. 对模型的参数进行敏感度分析
- 3. 根据第 2 步得到的敏感度信息，对模型进行裁剪，并以第 1 步训练好的模型作为预训练权重，继续进行训练

具体我们以图像分类模型 MobileNetV2 为例，本示例中所有代码均可在 Github 的 [tutorials/slim/prune/image_classification] 中获得。

9.1.1 第一步正常训练模型

此步骤中采用正常的代码进行模型训练，在获取本示例代码后，直接执行如下命令即可

```
python mobilenetv2_train.py
```

在训练完成后，我们以 output/mobilenetv2/best_model 保存的模型，继续接下来的步骤

9.1.2 第二步参数敏感度分析

此步骤中，我们需要加载第一步训练保存的模型，并通过不断地遍历参数，分析各参数裁剪后在验证数据集上的精度损失，以此判断各参数的敏感度。敏感度分析的代码很简单，用户可直接查看params_analysis.py。在命令行终端执行如下命令开始参数分析。

```
python params_analysis.py
```

在此步骤中，我们会得到保存的 mobilenetv2.sensi.data 文件，这个文件保存了模型中每个参数的敏感度，在后续的裁剪训练中，会根据此文件中保存的信息，对各个参数进行裁剪。同时，我们也可以对这个文件进行可视化分析，判断 eval_metric_loss 的大小设置与模型被裁剪比例的关系。(eval_metric_loss 的说明见第三步)

模型裁剪比例可视化分析代码见slim_visualize.py，执行如下命令即可

```
python slim_visualize.py
```

可视化结果如下，该图表明，当我们将 eval_metric_loss 设为 0.05 时，模型将被裁剪掉 65%；将 eval_metric_loss 设为 0.10，模型将被裁剪掉 68.0%。因此在实际使用时，我们可以根据自己的需求，去设置 eval_metric_loss 控制裁剪比例。

9.1.3 第三步模型裁剪训练

在前两步，我们得到了正常训练保存的模型 output/mobilenetv2/best_model 和基于该保存模型得到的参数敏感度信息文件 mobilenetv2.sensi.data，接下来则是进行模型裁剪训练。模型裁剪训练的代码第第一步基本一致，唯一区别在最后的 train 函数中，我们修改了 pretrain_weights，save_dir，sensitivities_file 和 eval_metric_loss 四个参数，如下所示

```
model.train(
        num_epoch=10,
        train_dataset=train_dataset,
        train_batch_size=32,
        eval_dataset=eval_dataset,
        lr_decay_epochs=[4,6,8],
        learning_rate=0.025,
        pretrain_weights='output/mobilenetv2/best_model',
        save_dir='output/mobilenetv2_prune',
        sensitivities_file='./mobilenetv2.sensi.data',
```

(下页继续)

(续上页)

```
        eval_metric_loss=0.05,
        use_vdl=True)
```

具体代码见tutorials/slim/prune/image_classification/mobilenetv2_prune_train.py，执行如下命令即可

```
python mobilenetv2_prune_train.py
```

其中修改的 4 个参数函数如下

- pretrain_weights: 预训练权重，在裁剪训练中，将其指定为第一步正常训练得到的模型路径
- save_dir: 裁剪训练过程中，模型保存的新路径
- sensitivities_file: 第二步中分析得到的各参数敏感度信息文件
- eval_metric_loss: 可用于控制模型最终被裁剪的比例，见第二步中的可视化说明

9.2 裁剪效果

在本示例的数据集上，经过裁剪训练后，模型的效果对比如下，其中预测速度不包括图像的预处理和结果的后处理。从表中可以看到，对于本示例中的简单数据集，模型裁剪掉 68% 后，模型准确度没有降低，在 CPU 的单张图片预测用时减少了 37%

CHAPTER 10

模型量化

模型量化将模型的计算从浮点型转为整型，从而加速模型的预测计算速度，在移动端/边缘端设备上降低模型的体积。

> 注：量化后的模型，通过 PaddleLite 转换为 PaddleLite 部署的模型格式后，模型体积将会大幅压缩。如若量化后的模型仍是以服务端本地部署格式 (文件包括 ___model___ 和 ___params___)，那么模型的文件大小是无法呈现参数变化情况的。

10.1 使用方法

PaddleX 中已经将量化功能作为模型导出的一个 API，代码使用方式如下，本示例代码和模型数据均可通过 GitHub 项目上代码tutorials/slim/quant/image_classification获取得到

```
import paddlex as pdx
model = pdx.load_model('mobilenetv2_vegetables')
# 加载数据集用于量化
dataset = pdx.datasets.ImageNet(
                data_dir='vegetables_cls',
                file_list='vegetables_cls/train_list.txt',
                label_list='vegetables_cls/labels.txt',
                transforms=model.test_transforms)

# 开始量化
```

(下页继续)

(续上页)

```
pdx.slim.export_quant_model(model, dataset,
                            batch_size=4,
                            batch_num=5,
                            save_dir='./quant_mobilenet',
                            cache_dir='./tmp')
```

在获取本示例代码后，执行如下命令即可完成量化和 PaddleLite 的模型导出

```
# 将 mobilenetv2 模型量化保存
python mobilenetv2_quant.py
# 将量化后的模型导出为 PaddleLite 部署格式
python paddlelite_export.py
```

10.2 量化效果

在本示例中，我们可以看到模型量化后的服务端部署模型格式 `server_mobilenet` 和 `quant_mobilenet` 两个目录中，模型参数大小并无变化。但在使用 PaddleLite 导出后，`mobilenetv2.nb` 和 `mobilenetv2_quant.nb` 大小分别为 8.8M, 2.7M，压缩至原来的 31%。

CHAPTER 11

部署模型导出

注：所有涉及到模型部署，均需要参考本文档，进行部署模型导出

在服务端部署模型时需要将训练过程中保存的模型导出为 inference 格式模型，导出的 inference 格式模型包括 `__model__`、`__params__` 和 `model.yml` 三个文件，分别表示模型的网络结构、模型权重和模型的配置文件（包括数据预处理参数等）。

> **检查你的模型文件夹**，如果里面是 `model.pdparams`，`model.pdmodel` 和 `model.yml`3 个文件时，那么就需要按照下面流程进行模型导出

在安装完 PaddleX 后，在命令行终端使用如下命令将模型导出。可直接下载小度熊分拣模型来测试本文档的流程xiaoduxiong_epoch_12.tar.gz。

```
paddlex --export_inference --model_dir=./xiaoduxiong_epoch_12 --save_dir=./inference_
↪model
```

使用 TensorRT 预测时，需固定模型的输入大小，通过`--fixed_input_shape` 来制定输入大小 [w,h]。

注意：

- 分类模型的固定输入大小请保持与训练时的输入大小一致；
- 检测模型模型中 YOLO 系列请保存 w 与 h 一致，且为 32 的倍数大小；RCNN 类无此限制，按需设定即可
- 指定 [w,h] 时，w 和 h 中间逗号隔开，不允许存在空格等其他字符。
- 需要注意的，w,h 设得越大，模型在预测过程中所需要的耗时和内存/显存占用越高；设得太小，会影响模型精度

```
paddlex --export_inference --model_dir=./xiaoduxiong_epoch_12 --save_dir=./inference_
↪model --fixed_input_shape=[640,960]
```

CHAPTER 12

PaddleHub 轻量级服务化部署

12.1 简介

借助 PaddleHub-Serving，可以将 PaddleX 的 Inference Model 进行快速部署，以提供在线预测的能力。

关于 PaddleHub-Serving 的更多信息，可参照PaddleHub-Serving。

注意：使用此方式部署，需确保自己 Python 环境中 PaddleHub 的版本高于 1.8.0，可在命令终端输入 pip show paddlehub 确认版本信息。

下面，我们按照步骤，实现将一个图像分类模型MobileNetV3_small_ssld转换成 PaddleHub 的预训练模型，并利用 PaddleHub-Serving 实现一键部署。

12.2 模型部署

12.2.1 1 部署模型准备

部署模型的格式均为目录下包含 __model__，__params__ 和 model.yml 三个文件，如若不然，则参照部署模型导出文档进行导出。

12.2.2 2 模型转换

首先，我们将 PaddleX 的 Inference Model 转换成 PaddleHub 的预训练模型，使用命令 hub convert 即可一键转换，对此命令的说明如下：

```
$ hub convert --model_dir XXXX \
              --module_name XXXX \
              --module_version XXXX \
              --output_dir XXXX
```

参数:

因此，我们仅需要一行命令即可完成预训练模型的转换。

```
 hub convert --model_dir mobilenetv3_small_ssld_imagenet_hub --module_name mobilenetv3_
↪small_ssld_imagenet_hub
```

转换成功后会打印提示信息，如下：

```
$ The converted module is stored in `MobileNetV3_small_ssld_hub_1596077881.868501`.
```

等待生成成功的提示后，我们就在输出目录中得到了一个 PaddleHub 的一个预训练模型。

12.2.3 3 模型安装

在模型转换一步中，我们得到了一个.tar.gz 格式的预训练模型压缩包，在进行部署之前需要先安装到本机，使用命令 hub install 即可一键安装，对此命令的说明如下：

```
$ hub install ${MODULE}
```

其中 ${MODULE} 为要安装的预训练模型文件路径。

因此，我们使用 hub install 命令安装：

```
hub install MobileNetV3_small_ssld_hub_1596077881.868501/mobilenetv3_small_ssld_imagenet_
↪hub.tar.gz
```

安装成功后会打印提示信息，如下：

```
$ Successfully installed mobilenetv3_small_ssld_imagenet_hub
```

12.2.4 4 模型部署

下面，我们只需要使用 hub serving 命令即可完成模型的一键部署，对此命令的说明如下：

```
$ hub serving start --modules/-m [Module1==Version1, Module2==Version2, ...] \
                    --port/-p XXXX
                    --config/-c XXXX
```

参数:

因此，我们仅需要一行代码即可完成模型的部署，如下:

```
$ hub serving start -m mobilenetv3_small_ssld_imagenet_hub
```

等待模型加载后，此预训练模型就已经部署在机器上了。

我们还可以使用配置文件对部署的模型进行更多配置，配置文件格式如下:

```
{
  "modules_info": {
    "mobilenetv3_small_ssld_imagenet_hub": {
      "init_args": {
        "version": "1.0.0"
      },
      "predict_args": {
        "batch_size": 1,
        "use_gpu": false
      }
    }
  },
  "port": 8866
}
```

| 参数 | 用途 | |-|-| |modules_info|PaddleHub Serving 预安装模型，以字典列表形式列出，key 为模型名称。其中:`init_args` 为模型加载时输入的参数，等同于 `paddlehub.Module(**init_args)`predict_args 为模型预测时输入的参数，以 `mobilenetv3_small_ssld_imagenet_hub` 为例，等同于 `mobilenetv3_small_ssld_imagenet_hub.batch_predict(**predict_args)` |port| 服务端口，默认为 8866|

12.2.5 5 测试

在第二步模型安装的同时，会生成一个客户端请求示例，存放在模型安装目录，默认为 `${HUB_HOME}/.paddlehub/modules`，对于此例，我们可以在 `~/.paddlehub/modules/mobilenetv3_small_ssld_imagenet_hub` 找到此客户端示例 `serving_client_demo.py`，代码如下:

```
# coding: utf8
import requests
import json
import cv2
import base64
```

(下页继续)

(续上页)

```
def cv2_to_base64(image):
    data = cv2.imencode('.jpg', image)[1]
    return base64.b64encode(data.tostring()).decode('utf8')


if __name__ == '__main__':
    # 获取图片的 base64 编码格式
    img1 = cv2_to_base64(cv2.imread("IMAGE_PATH1"))
    img2 = cv2_to_base64(cv2.imread("IMAGE_PATH2"))
    data = {'images': [img1, img2]}
    # 指定 content-type
    headers = {"Content-type": "application/json"}
    # 发送 HTTP 请求
    url = "http://127.0.0.1:8866/predict/mobilenetv3_small_ssld_imagenet_hub"
    r = requests.post(url=url, headers=headers, data=json.dumps(data))

    # 打印预测结果
    print(r.json()["results"])
```

使用的测试图片如下：

将代码中的 IMAGE_PATH1 改成想要进行预测的图片路径后，在命令行执行：

```
python ~/.paddlehub/module/MobileNetV3_small_ssld_hub/serving_client_demo.py
```

即可收到预测结果，如下：

```
[[{'category': 'envelope', 'category_id': 549, 'score': 0.2141510397195816}]]
```

到此，我们就完成了 PaddleX 模型的一键部署。

CHAPTER 13

CPU/GPU(加密) 部署

13.1 Python 部署

PaddleX 已经集成了基于 Python 的高性能预测接口，在安装 PaddleX 后，可参照如下代码示例，进行预测。

13.1.1 导出预测模型

可参考模型导出将模型导出为 inference 格式。

13.1.2 预测部署

预测接口说明可参考paddlex.deploy

点击下载测试图片 xiaoduxiong_test_image.tar.gz

- 单张图片预测

```
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
result = predictor.predict(image='xiaoduxiong_test_image/JPEGImages/WeChatIMG110.jpeg')
```

- 批量图片预测

```
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
image_list = ['xiaoduxiong_test_image/JPEGImages/WeChatIMG110.jpeg',
    'xiaoduxiong_test_image/JPEGImages/WeChatIMG111.jpeg']
result = predictor.batch_predict(image_list=image_list)
```

- 视频流预测

```
import cv2
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        result = predictor.predict(frame)
        vis_img = pdx.det.visualize(frame, result, threshold=0.6, save_dir=None)
        cv2.imshow('Xiaoduxiong', vis_img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
```

关于预测速度的说明：加载模型后前几张图片的预测速度会较慢，这是因为运行启动时涉及到内存显存初始化等步骤，通常在预测 20-30 张图片后模型的预测速度达到稳定。

13.1.3 预测性能对比

测试环境

- CUDA 9.0
- CUDNN 7.5
- PaddlePaddle 1.71
- GPU: Tesla P40
- AnalysisPredictor 指采用 Python 的高性能预测方式
- Executor 指采用 PaddlePaddle 普通的 Python 预测方式
- Batch Size 均为 1，耗时单位为 ms/image，只计算模型运行时间，不包括数据的预处理和后处理

性能对比

13.2 C++ 部署

13.2.1 Windows 平台部署

说明

Windows 平台下，我们使用 Visual Studio 2019 Community 进行了测试。微软从 Visual Studio 2017 开始即支持直接管理 CMake 跨平台编译项目，但是直到 2019 才提供了稳定和完全的支持，所以如果你想使用 CMake 管理项目编译构建，我们推荐你使用 Visual Studio 2019 环境下构建。

前置条件

- Visual Studio 2019
- CUDA 9.0 / CUDA 10.0, CUDNN 7+ （仅在使用 GPU 版本的预测库时需要）
- CMake 3.0+

请确保系统已经安装好上述基本软件，我们使用的是 VS2019 的社区版。

下面所有示例以工作目录为 D:\projects 演示。

Step1: 下载 PaddleX 预测代码

```
d:
mkdir projects
cd projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 C++ 预测代码在 PaddleX\deploy\cpp 目录，该目录不依赖任何 PaddleX 下其他目录。

Step2: 下载 PaddlePaddle C++ 预测库 paddle_inference

PaddlePaddle C++ 预测库针对是否使用 GPU、是否支持 TensorRT、以及不同的 CUDA 版本提供了已经编译好的预测库，目前 PaddleX 依赖于 Paddle 1.8.4，基于 Paddle 1.8.4 的 Paddle 预测库下载链接如下所示:

请根据实际情况选择下载，如若以上版本不满足您的需求，请至C++ 预测库下载列表选择符合的版本。

将预测库解压后，其所在目录（例如 D:\projects\fluid_inference\）下主要包含的内容有:

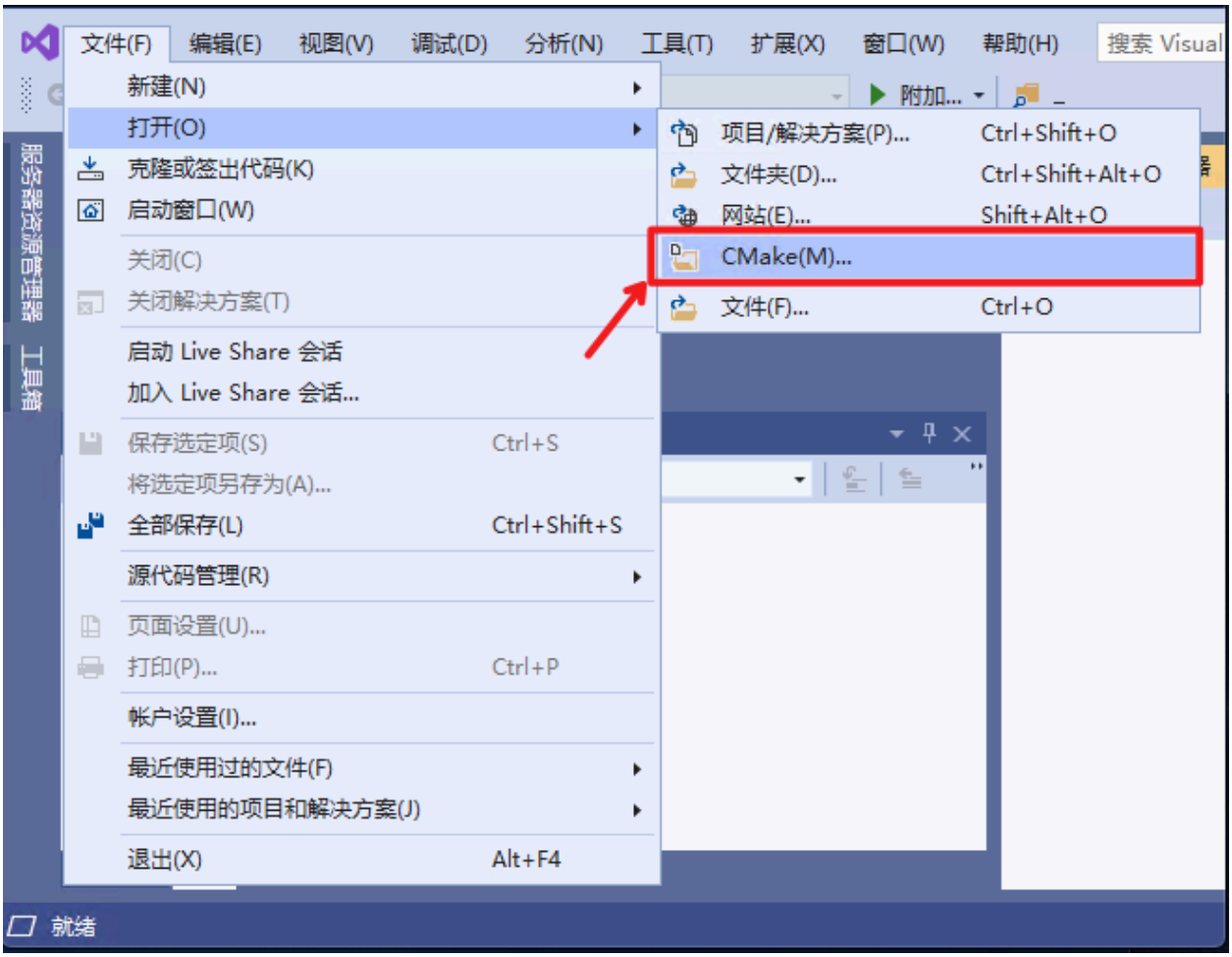
```
  \paddle\ # paddle 核心库和头文件
|
  \third_party\ # 第三方依赖库和头文件
|
  \version.txt # 版本和编译信息
```

Step3: 安装配置 OpenCV

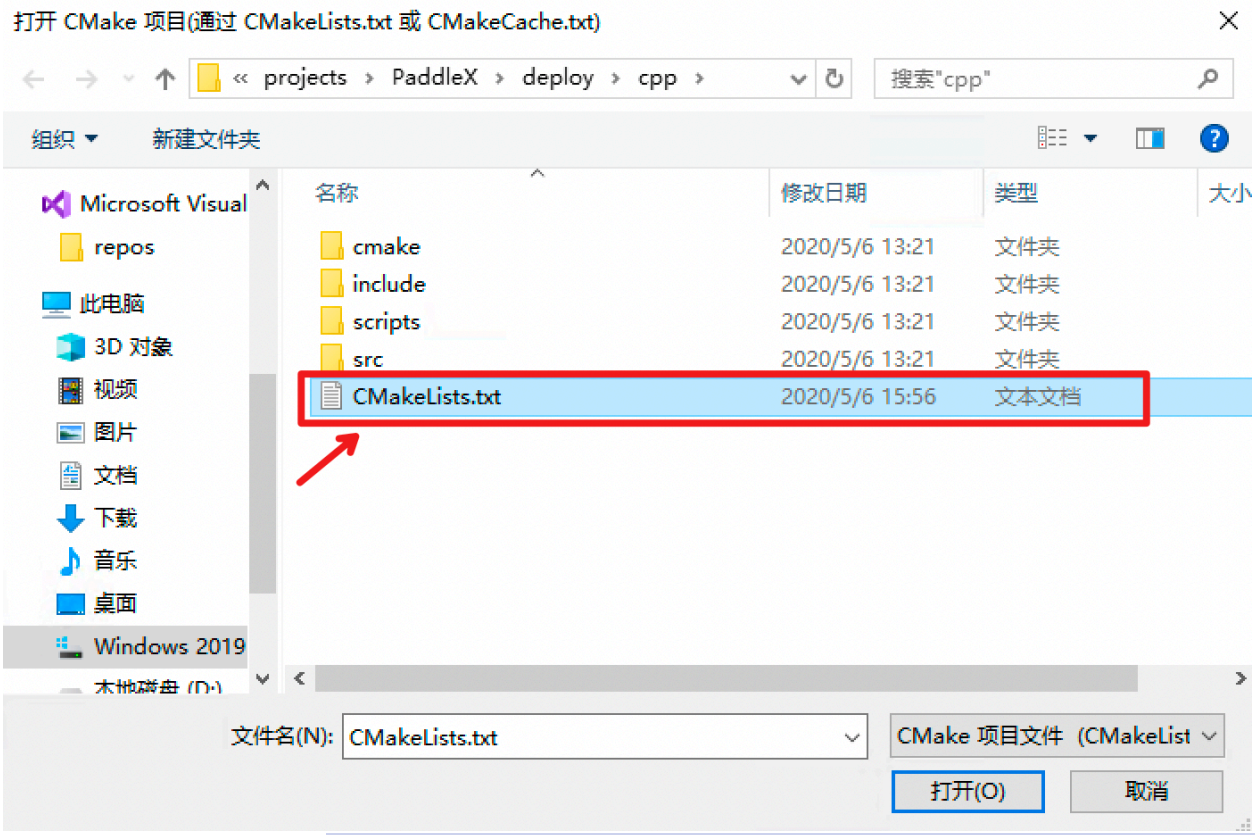
1. 在 OpenCV 官网下载适用于 Windows 平台的 3.4.6 版本，下载地址
2. 运行下载的可执行文件，将 OpenCV 解压至指定目录，例如 `D:\projects\opencv`
3. 配置环境变量，如下流程所示
    - 我的电脑-> 属性-> 高级系统设置-> 环境变量
    - 在系统变量中找到 Path（如没有，自行创建），并双击编辑
    - 新建，将 opencv 路径填入并保存，如 `D:\projects\opencv\build\x64\vc14\bin`

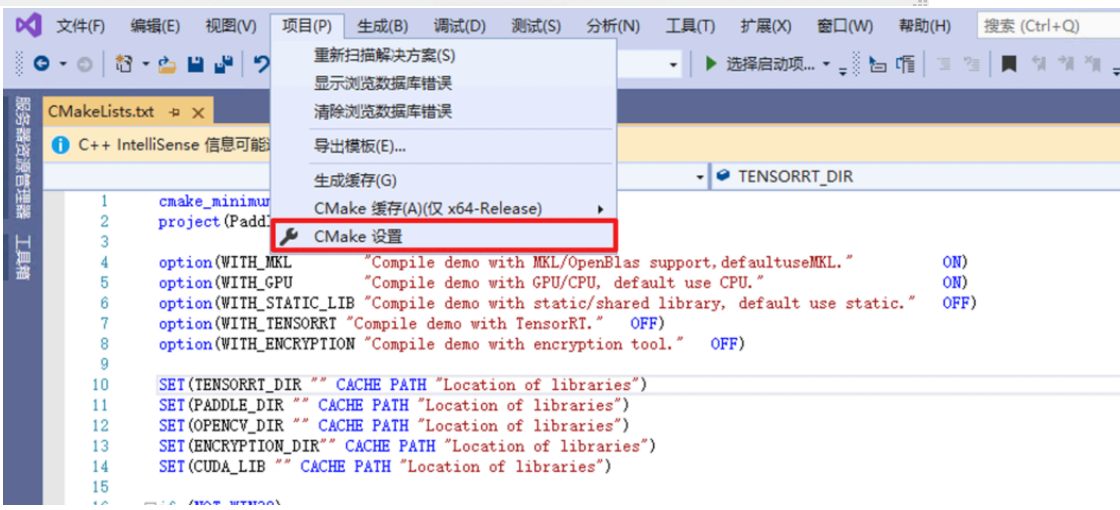
Step4: 使用 Visual Studio 2019 直接编译 CMake

1. 打开 Visual Studio 2019 Community，点击继续但无需代码
2. 点击：文件-> 打开->CMake

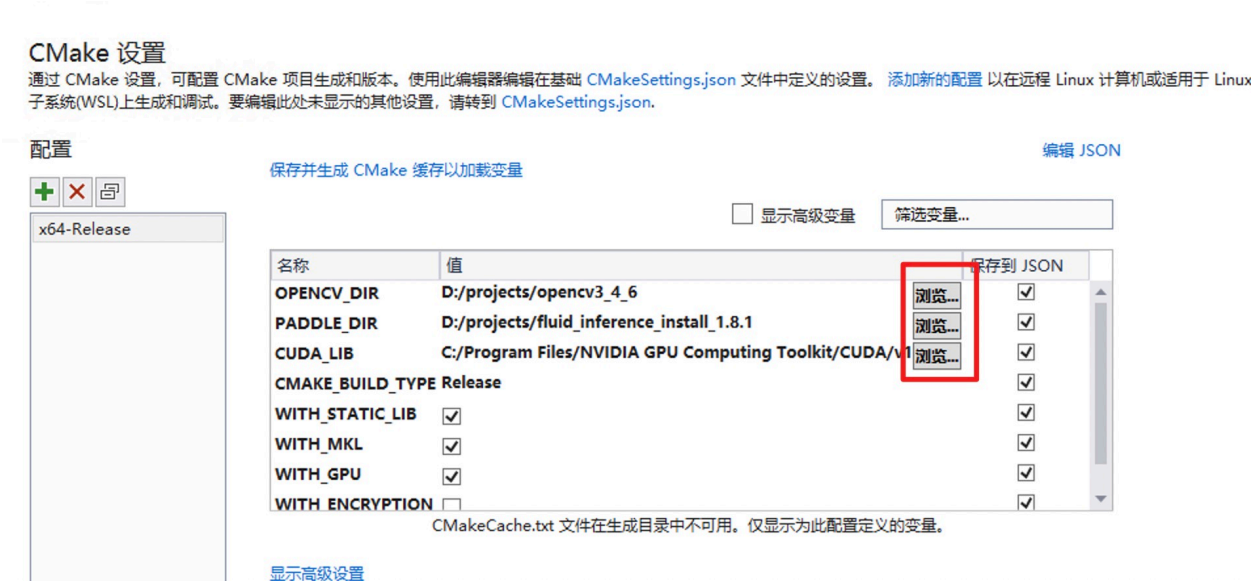


选择 C++ 预测代码所在路径（例如 `D:\projects\PaddleX\deploy\cpp`），并打开 `CMakeList.txt`:





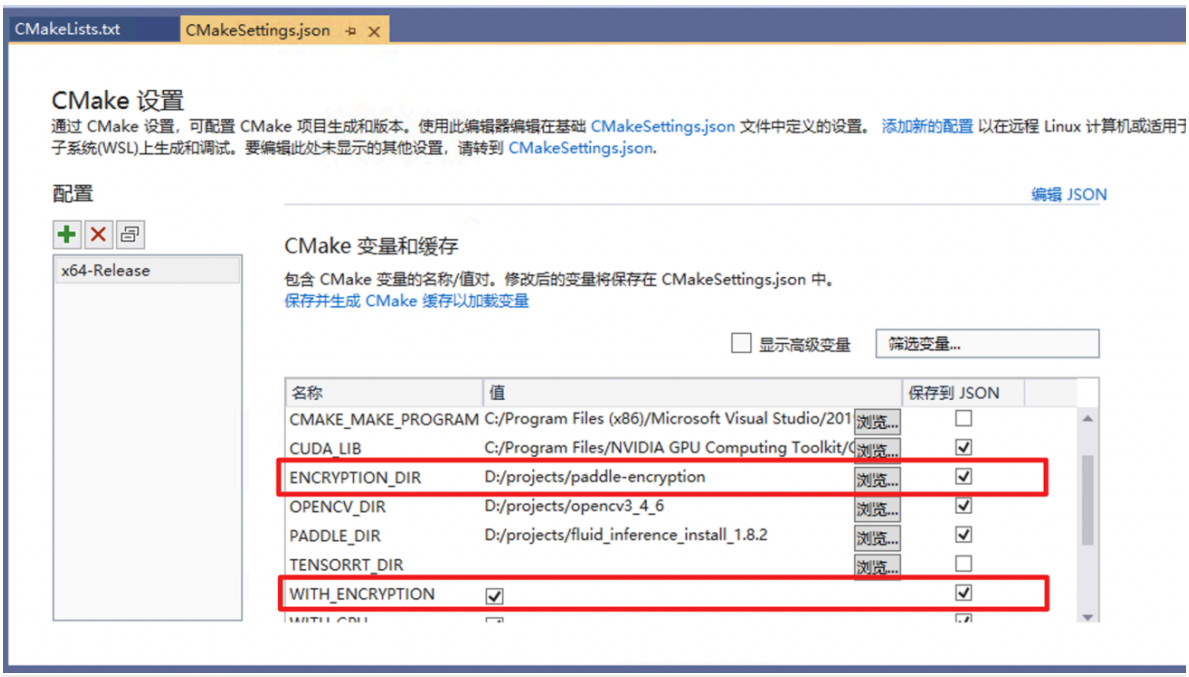
3. 点击:项目->CMake 设置
4. 点击浏览，分别设置编译选项指定 CUDA、OpenCV、Paddle 预测库的路径

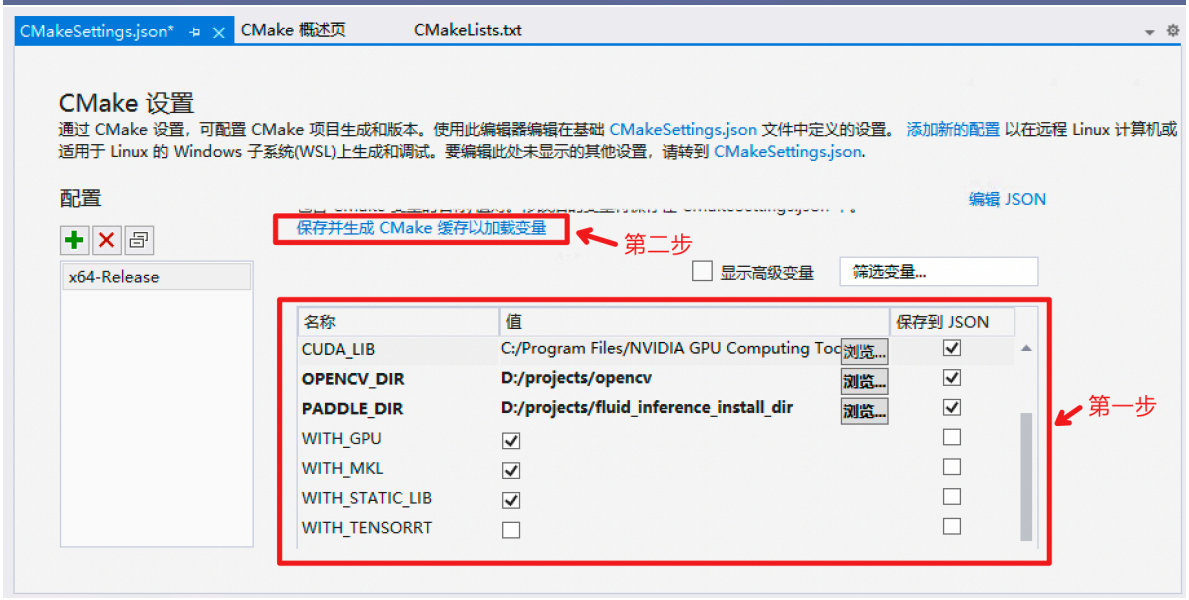


依赖库路径的含义说明如下（带 * 表示仅在使用 **GPU 版本**预测库时指定，其中 CUDA 库版本尽量与 Paddle 预测库的对齐，例如 Paddle 预测库是**使用 9.0、10.0 版本**编译的，则编译 PaddleX 预测代码时**不使用 9.2、10.1 等版本** CUDA 库）：

注意：

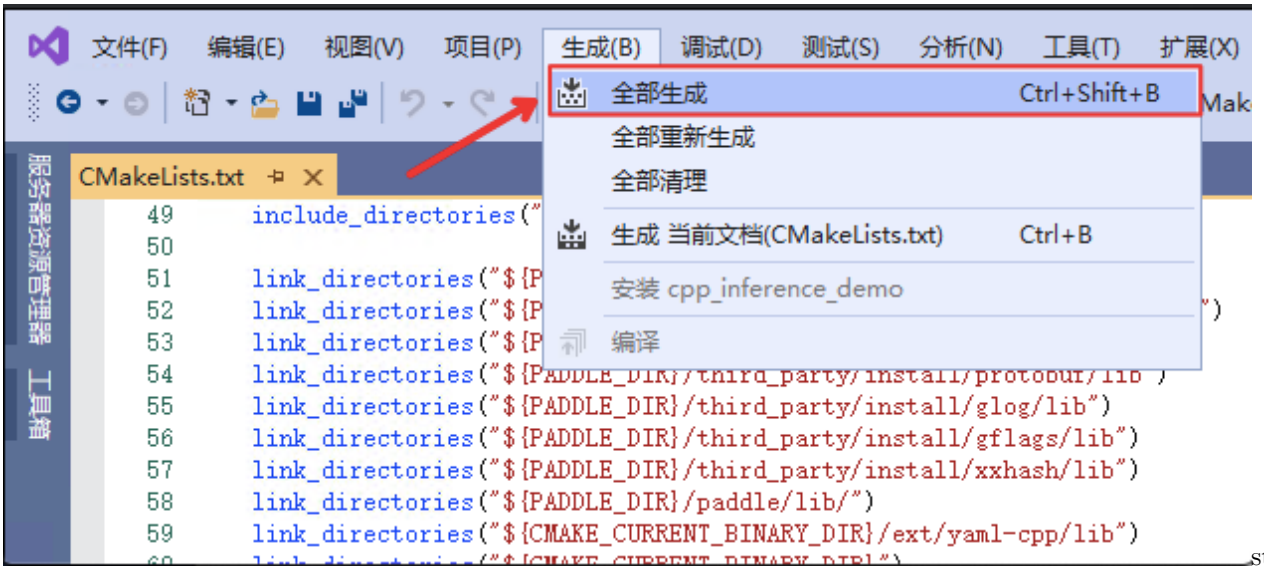
1. 如果使用 `CPU` 版预测库，请把 `WITH_GPU` 的值去掉勾
2. 如果使用的是 `openblas` 版本，请把 `WITH_MKL` 的值去掉勾
3. Windows 环境下编译会自动下载 YAML，如果编译环境无法访问外网，可手动下载：yaml-cpp.zip。YAML 文件下载后无需解压，在 `cmake/yaml.cmake` 中将 `URL https://bj.bcebos.com/paddlex/deploy/deps/yaml-cpp.zip` 中的网址，改为下载文件的路径。
4. 如果需要使用模型加密功能，需要手动下载Windows 预测模型加密工具。例如解压到 `D:/projects`，解压后目录为 `D:/projects/paddlex-encryption`。编译时需勾选 `WITH_EBNCRYPTION` 并且在 `ENCRTYPTION_DIR` 填入 `D:/projects/paddlex-encryption`。





设置完成后，点击上图中保存并生成 CMake 缓存以加载变量。

5. 点击生成-> 全部生成



step6

Step5: 预测及可视化

在加载模型前，请检查你的模型目录中文件应该包括 model.yml、__model__ 和 __params__ 三个文件。如若不满足这个条件，请参考部署模型导出将模型导出为部署格式。

上述 Visual Studio 2019 编译产出的可执行文件在 out\build\x64-Release 目录下，打开 cmd，并切换到该目录：

```
D:
cd D:\projects\PaddleX\deploy\cpp\out\build\x64-Release
```

- 编译成功后，图片预测 demo 的入口程序为 paddlex_inference\detector.exe，paddlex_inference\classifier.exe，paddlex_inference\segmenter.exe，用户可根据自己的模型类型选择，其主要命令参数说明如下：
- 编译成功后，视频预测 demo 的入口程序为 paddlex_inference\video_detector.exe，paddlex_inference\video_classifier.exe，paddlex_inference\video_segmenter.exe，用户可根据自己的模型类型选择，其主要命令参数说明如下：

注意：若系统无 GUI，则不要将 show_result 设置为 1。当使用摄像头预测时，按 ESC 键可关闭摄像头并推出预测程序。

样例

可使用小度熊识别模型中导出的 inference_model 和测试图片进行预测，例如导出到 D:\projects，模型路径为 D:\projects\inference_model。

> 关于预测速度的说明：加载模型后前几张图片的预测速度会较慢，这是因为运行启动时涉及到内存显存初始化等步骤，通常在预测 20-30 张图片后模型的预测速度达到稳定。

样例一：(使用未加密的模型对单张图像做预测)

不使用 GPU 测试图片 D:\images\xiaoduxiong.jpeg

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\inference_model --
↪image=D:\images\xiaoduxiong.jpeg --save_dir=output
```

图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例二：(使用未加密的模型对图像列表做预测)

使用 GPU 预测多个图片 D:\images\image_list.txt，image_list.txt 内容的格式如下：

```
D:\images\xiaoduxiong1.jpeg
D:\images\xiaoduxiong2.jpeg
...
D:\images\xiaoduxiongn.jpeg
```

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\inference_model --image_
↪list=D:\images\image_list.txt --use_gpu=1 --save_dir=output --batch_size=2 --thread_
↪num=2
```

图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例三：(使用加密后的模型对单张图片进行预测)

如果未对模型进行加密，请参考加密 PaddleX 模型对模型进行加密。例如加密后的模型所在目录为 D:\projects\encrypted_inference_model。

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\encrypted_inference_model --
↪image=D:\images\xiaoduxiong.jpeg --save_dir=output --key=kLAl1qOs5uRbFt0/
↪RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key 传入加密工具输出的密钥，例如 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=，图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例四：(使用未加密的模型开启摄像头预测)

```
.\paddlex_inference\video_detector.exe --model_dir=D:\projects\inference_model --use_
↪camera=1 --use_gpu=1 --save_dir=output
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。

样例五：(使用未加密的模型对视频文件做预测)

```
.\paddlex_inference\video_detector.exe --model_dir=D:\projects\inference_model --video_
↪path=D:\projects\video_test.mp4 --use_gpu=1 --show_result=1 --save_dir=output
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。如果系统有 GUI，通过将 show_result 设置为 1 在屏幕上观看可视化预测结果。

13.2.2 Linux 平台部署

说明

本文档在 Linux 平台使用 GCC 4.8.5 和 GCC 4.9.4 测试过，如果需要使用更高 G++ 版本编译使用，则需要重新编译 Paddle 预测库，请参考: 从源码编译 Paddle 预测库。

前置条件

- G++ 4.8.2 ~ 4.9.4
- CUDA 9.0 / CUDA 10.0, CUDNN 7+ （仅在使用 GPU 版本的预测库时需要）
- CMake 3.0+

请确保系统已经安装好上述基本软件，**下面所有示例以工作目录 /root/projects/演示**。

Step1: 下载代码

```
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 C++ 预测代码在/root/projects/PaddleX/deploy/cpp 目录，该目录不依赖任何 PaddleX 下其他目录。

Step2: 下载 PaddlePaddle C++ 预测库 paddle_inference

PaddlePaddle C++ 预测库针对不同的 CPU，CUDA，以及是否支持 TensorRT，提供了不同的预编译版本，目前 PaddleX 依赖于 Paddle1.8.4 版本，以下提供了多个不同版本的 Paddle 预测库:

更多和更新的版本，请根据实际情况下载: C++ 预测库下载列表

下载并解压后/root/projects/fluid_inference 目录包含内容为:

```
fluid_inference
  paddle # paddle 核心库和头文件
|
  third_party # 第三方依赖库和头文件
|
  version.txt # 版本和编译信息
```

注意: 预编译版本除 nv-jetson-cuda10-cudnn7.5-trt5 以外其它包都是基于 GCC 4.8.5 编译，使用高版本 GCC 可能存在 ABI 兼容性问题，建议降级或自行编译预测库。

Step3: 编译

编译 cmake 的命令在 scripts/build.sh 中，请根据实际情况修改主要参数，其主要内容说明如下:

```
# 是否使用 GPU(即是否使用 CUDA)
WITH_GPU=OFF
# 使用 MKL or openblas
WITH_MKL=ON
# 是否集成 TensorRT(仅 WITH_GPU=ON 有效)
WITH_TENSORRT=OFF
# TensorRT 的路径，如果需要集成 TensorRT，需修改为您实际安装的 TensorRT 路径
TENSORRT_DIR=/root/projects/TensorRT/
# Paddle 预测库路径，请修改为您实际安装的预测库路径
PADDLE_DIR=/root/projects/fluid_inference
# Paddle 的预测库是否使用静态库来编译
# 使用 TensorRT 时，Paddle 的预测库通常为动态库
WITH_STATIC_LIB=OFF
# CUDA 的 lib 路径
CUDA_LIB=/usr/local/cuda/lib64
# CUDNN 的 lib 路径
CUDNN_LIB=/usr/local/cuda/lib64

# 是否加载加密后的模型
WITH_ENCRYPTION=ON
# 加密工具的路径，如果使用自带预编译版本可不修改
sh $(pwd)/scripts/bootstrap.sh # 下载预编译版本的加密工具
ENCRYPTION_DIR=$(pwd)/paddlex-encryption

# OPENCV 路径，如果使用自带预编译版本可不修改
sh $(pwd)/scripts/bootstrap.sh  # 下载预编译版本的 opencv
```

(下页继续)

(续上页)

```
OPENCV_DIR=$(pwd)/deps/opencv3gcc4.8/

# 以下无需改动
rm -rf build
mkdir -p build
cd build
cmake .. \
    -DWITH_GPU=${WITH_GPU} \
    -DWITH_MKL=${WITH_MKL} \
    -DWITH_TENSORRT=${WITH_TENSORRT} \
    -DWITH_ENCRYPTION=${WITH_ENCRYPTION} \
    -DTENSORRT_DIR=${TENSORRT_DIR} \
    -DPADDLE_DIR=${PADDLE_DIR} \
    -DWITH_STATIC_LIB=${WITH_STATIC_LIB} \
    -DCUDA_LIB=${CUDA_LIB} \
    -DCUDNN_LIB=${CUDNN_LIB} \
    -DENCRYPTION_DIR=${ENCRYPTION_DIR} \
    -DOPENCV_DIR=${OPENCV_DIR}
make
```

注意： linux 环境下编译会自动下载 OPENCV, PaddleX-Encryption 和 YAML，如果编译环境无法访问外网，可手动下载：

- opencv3.4.6gcc4.8ffmpeg.tar.gz2
- paddlex-encryption.zip
- yaml-cpp.zip

opencv3gcc4.8.tar.bz2 文件下载后解压，然后在 script/build.sh 中指定 `OPENCE_DIR` 为解压后的路径。

paddlex-encryption.zip 文件下载后解压，然后在 script/build.sh 中指定 `ENCRYPTION_DIR` 为解压后的路径。

yaml-cpp.zip 文件下载后无需解压，在 cmake/yaml.cmake 中将 `URL https://bj.bcebos.com/paddlex/deploy/deps/yaml-cpp.zip` 中的网址，改为下载文件的路径。

修改脚本设置好主要参数后，执行 `build` 脚本：

```
sh ./scripts/build.sh
```

Step4: 预测及可视化

在加载模型前，请检查你的模型目录中文件应该包括 `model.yml`、`__model__` 和 `__params__` 三个文件。如若不满足这个条件，请参考模型导出为 Inference 文档将模型导出为部署格式。

- 编译成功后，图片预测 demo 的可执行程序分别为 `build/demo/detector`，`build/demo/classifier`，`build/demo/segmenter`，用户可根据自己的模型类型选择，其主要命令参数说明如下：
- 编译成功后，视频预测 demo 的可执行程序分别为 `build/demo/video_detector`，`build/demo/video_classifier`，`build/demo/video_segmenter`，用户可根据自己的模型类型选择，其主要命令参数说明如下：

注意：若系统无 GUI，则不要将 show_result 设置为 1。当使用摄像头预测时，按 ESC 键可关闭摄像头并推出预测程序。

样例

可使用小度熊识别模型中导出的 `inference_model` 和测试图片进行预测，导出到/root/projects，模型路径为/root/projects/inference_model。

> 关于预测速度的说明：加载模型后前几张图片的预测速度会较慢，这是因为运行启动时涉及到内存显存初始化等步骤，通常在预测 20-30 张图片后模型的预测速度达到稳定。

样例一：

不使用 GPU 测试图片 `/root/projects/images/xiaoduxiong.jpeg`

```
./build/demo/detector --model_dir=/root/projects/inference_model --image=/root/projects/
↪images/xiaoduxiong.jpeg --save_dir=output
```

图片文件可视化预测结果会保存在 `save_dir` 参数设置的目录下。

样例二：

使用 GPU 预测多个图片/root/projects/image_list.txt，image_list.txt 内容的格式如下：

```
/root/projects/images/xiaoduxiong1.jpeg
/root/projects/images/xiaoduxiong2.jpeg
...
/root/projects/images/xiaoduxiongn.jpeg
```

```
./build/demo/detector --model_dir=/root/projects/inference_model --image_list=/root/
↪projects/images_list.txt --use_gpu=1 --save_dir=output --batch_size=2 --thread_num=2
```

图片文件可视化预测结果会保存在 `save_dir` 参数设置的目录下。

样例三：

使用摄像头预测：

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --use_camera=1 --
↪use_gpu=1 --save_dir=output --save_result=1
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。

样例四：

对视频文件进行预测：

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --video_path=/
↪path/to/video_file --use_gpu=1 --save_dir=output --show_result=1 --save_result=1
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。如果系统有 GUI，通过将 show_result 设置为 1 在屏幕上观看可视化预测结果。

13.2.3 C++ 代码接口说明

头文件

include/paddlex/paddlex.h

类 PaddleX::Model

模型类，用于加载 PaddleX 训练的模型。

模型加载

```
PaddleX::Model::Init(const std::string& model_dir,
                     bool use_gpu = false,
                     bool use_trt = false,
                     bool use_mkl = true,
                     bool mkl_thread_num = 4,
                     int gpu_id = 0,
                     std::string key = "",
                     bool use_ir_optim = true)
```

参数

- model_dir: 模型目录路径
- use_gpu: 是否使用 gpu 预测
- use_trt: 是否使用 TensorRT
- use_mkl: 是否使用 MKLDNN 加速模型在 CPU 上的预测性能
- mkl_thread_num: 使用 MKLDNN 时，线程数量
- gpu_id: 使用 gpu 的 id 号

- key: 模型解密密钥，此参数用于加载加密的 PaddleX 模型时使用
- use_ir_optim: 是否加速模型后进行图优化

返回值

- 返回 true 或 false，表示模型是否加载成功

模型预测推断

分类模型单张图片预测

```
PaddleX::Model::predict(const cv::Mat& im, ClsResult* result)
```

分类模型多张图片批预测

```
PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<ClsResult>*␣
↪results)
```

目标检测/实例分割模型单张图片预测

```
PaddleX::Model::predict(const cv::Mat& im, DetResult* result)
```

目标检测/实例分割模型多张图片批预测

```
PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<DetResult>*␣
↪results)
```

语义分割模型单张图片预测

```
PaddleX::Model::predict(const cv::Mat& im, SegResult* result)
```

语义分割模型多张图片批预测

```
PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<SegResult>*␣
↪results)
```

各接口返回值为 true 或 false，用于表示是否预测成功

预测时，需传入 cv::Mat 结构体，结构需与如下示代码加载的结构体一致

```
cv::Mat im = cv::imread('test.jpg', 1);
```

当使用批预测时，注意会传入的 vector 中所有数据作为一个批次进行预测，因此 vector 越大，所需要使用的 GPU 显存会越高。

预测时，同时传入 ClsResult/DetResult/SegResult 结构体，用于存放模型的预测结果，各结构体说明如下

```
// 分类模型预测结果
class ClsResult {
 public:
  int category_id; // 类别 id
  std::string category; // 类别标签
  float score; // 预测置信度
  std::string type = "cls";
}

// 目标检测/实例分割模型预测结果
class DetResult {
 public:
  std::vector<Box> boxes; // 预测结果中的各个目标框
  int mask_resolution;
  std::string type = "det";
}

// 语义分割模型预测结果
class SegResult : public BaseResult {
 public:
  Mask<int64_t> label_map; // 预测分割中各像素的类别
  Mask<float> score_map; // 预测分割中各像素的置信度
  std::string type = "seg";
}

struct Box {
  int category_id; // 类别 id
  std::string category; // 类别标签
  float score; // 置信度
  std::vector<float> coordinate; // 4 个元素值，表示 xmin, ymin, width, height
  Mask<int> mask; // 实例分割中，用于表示 Box 内的分割结果
}

struct Mask {
  std::vector<T> data; // 分割中的 label map 或 score map
  std::vector<int> shape; // 表示分割图的 shape
}
```

预测结果可视化

目标检测/实例分割结果可视化

```
PaddleX::Visualize(const cv::Mat& img, // 原图
                                    const DetResult& result, // 预测结果
                                    const std::map<int, std::string>& labels // 各类别信息
↪<id, label_name>
                                   )
```

返回 cv::Mat 结构体，即为可视化后的结果

语义分割结果可视化

```
PaddleX::Visualize(const cv::Mat& img, // 原图
                                    const SegResult& result, // 预测结果
                    const std::map<int, std::string>& labels // 各类别信息<id, label_name>
                   )
```

返回 cv::Mat 结构体，即为可视化后的结果

代码示例

- 图像分类 PaddleX/deploy/cpp/demo/classifier.cpp
- 目标检测/实例分割 PaddleX/deploy/cpp/demo/detector.cpp
- 语义分割 PaddleX/deploy/cpp/demo/segmenter.cpp

13.3 模型加密部署

PaddleX 提供一个轻量级的模型加密部署方案，通过 PaddleX 内置的模型加密工具对推理模型进行加密，预测部署 SDK 支持直接加载密文模型并完成推理，提升 AI 模型部署的安全性。

目前加密方案已支持 Windows，Linux 系统

13.3.1 1. 方案简介

1.1 简介

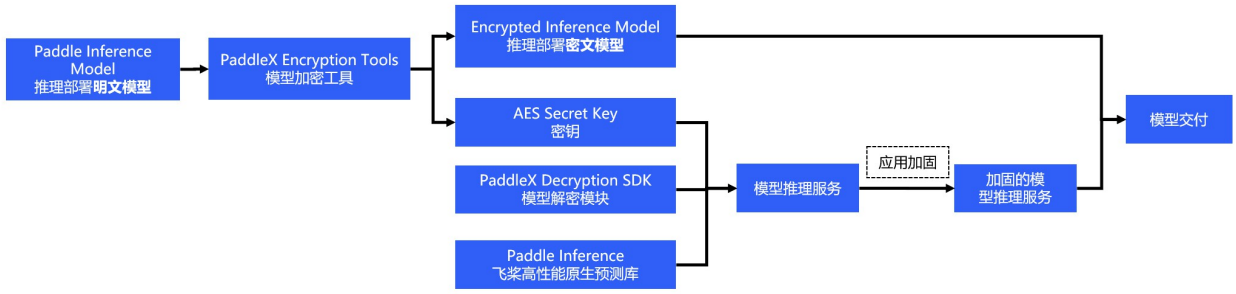
(1) 加密算法的选择和支持的库

一般使用 OpenSSL 库来支持数据的加解密，OpenSSL 提供了大量的加解密算法，包括对称加密算法（AES 等）和非对称加密算法（RSA 等）。

两种算法使用的场景不同，非对称加密算法一般应用于数字签名和密钥协商的场景下，而对称加密算法一般应用于纯数据加密场景，性能更优。在对模型的加密过程中使用对称加密算法。

以下对模型加密场景实现的说明中以开发一个 C/C++ 库为基础，采用 AES 对称加密算法，为了加解密前后能够快速判断解密是否成功，使用 AES-GCM 加解密模式，在密钥的安全性上使用长度为 256 位的密钥数据。

（2）实现模型保护的一般步骤：



下面是对提供的 C/C++ 加解密库内部实现的中文描述，参考以下步骤可以实现一套加解密库来适应自己的场景并通过内存数据加载到 Paddle Inference 预测库中

1）考虑到加密的模型文件解密后需要从内存加载数据，使用 conbine 的模式生成模型文件和参数文件。

2）项目集成 OpenSSL，使用静态库的形式。

3）实现 AES 算法接口，借助 OpenSSL 提供的 EVP 接口，在 EVP 接口中指定算法类型，算法使用对称加解密算法中的 AES，加解密模式使用 AES-GCM，密钥长度为 256 位，AES-GCM 的实现可以参考官方提供的例子自己进行封装接口：AES-GCM 实现。

4）利用 OpenSSL 库实现 SHA256 摘要算法，这部分下面有用（可选）。关于 SHA256 的 hash 计算可以参考 OpenSSL 提供的 example：OpenSSL 信息摘要例子。

5）在模型加密环节直接对 model 文件和 params 文件的数据内容进行加密后保存到新的文件，为了新的文件能够被区分和可迭代，除了加密后的数据外还添加了头部信息，比如为了判断该文件类型使用固定的魔数作为文件的开头；为了便于后面需求迭代写入版本号以示区别；为了能够在解密时判断是否采用了相同的密钥将加密时的密钥进行 SHA256 计算后存储；这三部分构成了目前加密后文件的头部信息。加密后的文件包含头部信息 + 密文信息。

6）在模型解密环节根据加密后的文件读取相关的加密数据到内存中，对内存数据使用 AES 算法进行解密，注意解密时需要采用与加密时一致的加密算法和加密的模式，以及密钥的数据和长度，否则会导致解密后数据错误。

7）集成模型预测的 C/C++ 库，在具体使用预测时一般涉及 paddle::AnalysisConfig 和 paddle::Predictor，为了能够从内存数据中直接 load 解密后的模型明文数据（避免模型解密后创建临时文件），这里需要将 AnalysisConfig 的模型加载函数从 SetModel 替换为 SetModelBuffer 来实现从内存中加载模型数据。

需要注意的是，在本方案中，密钥集成在上层预测服务的代码中。故模型的安全强度等同于代码抵御逆向调试的强度。为了保护密钥和模型的安全，开发者还需对自己的应用进行加固保护。常见的应用加固手段有：代码混淆，二进制文件加壳等等，亦或将加密机制更改为 AES 白盒加密技术来保护密钥。这类技术领域内有大量商业和开源产品可供选择，此处不一一赘述。

1.2 加密工具

Linux 版本 PaddleX 模型加密工具，编译脚本会自动下载该版本加密工具，您也可以选择手动下载。

Windows 版本 PaddleX 模型加密工具，该版本加密工具需手动下载，如果您在使用 Visual Studio 2019 编译 C++ 预测代码的过程中已经下载过该工具，此处可不必重复下载。

Linux 加密工具包含内容为：

```
paddlex-encryption
  include # 头文件：paddle_model_decrypt.h（解密）和 paddle_model_encrypt.h（加密）
|
  lib # libpmodel-encrypt.so 和 libpmodel-decrypt.so 动态库
|
  tool # paddle_encrypt_tool
```

Windows 加密工具包含内容为：

```
paddlex-encryption
  include # 头文件：paddle_model_decrypt.h（解密）和 paddle_model_encrypt.h（加密）
|
  lib # pmodel-encrypt.dll 和 pmodel-decrypt.dll 动态库 pmodel-encrypt.lib 和 pmodel-
↪encrypt.lib 静态库
|
  tool # paddle_encrypt_tool.exe 模型加密工具
```

1.3 加密 PaddleX 模型

对模型完成加密后，加密工具会产生随机密钥信息（用于 AES 加解密使用），需要在后续加密部署时传入该密钥来用于解密。

> 密钥由 32 字节 key + 16 字节 iv 组成，注意这里产生的 key 是经过 base64 编码后的，这样可以扩充 key 的选取范围

Linux 平台：

```
# 假设模型在/root/projects 下
./paddlex-encryption/tool/paddle_encrypt_tool -model_dir /root/projects/paddlex_
↪inference_model -save_dir /root/projects/paddlex_encrypted_model
```

Windows 平台:

```
# 假设模型在 D:/projects 下
.\paddlex-encryption\tool\paddle_encrypt_tool.exe -model_dir D:\projects\paddlex_
↪inference_model -save_dir D:\projects\paddlex_encrypted_model
```

-model_dir 用于指定 inference 模型路径 (参考导出 inference 模型将模型导出为 inference 格式模型)，可使用导出小度熊识别模型中导出的 inference_model。加密完成后，加密过的模型会保存至指定的-save_dir 下，包含 __model__.encrypted、__params__.encrypted 和 model.yml 三个文件，同时生成密钥信息，命令输出如下图所示，密钥为 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=

```
Output: Encryption key:
        kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
Success, Encrypt __model__, __params__ to paddlex_encrypted_model(dir) success!
```

13.3.2 2. PaddleX C++ 加密部署

2.1 Linux 平台使用

参考Linux 平台编译指南编译 C++ 部署代码。编译成功后，预测 demo 的可执行程序分别为 build/demo/detector，build/demo/classifier，build/demo/segmenter，用户可根据自己的模型类型选择，其主要命令参数说明如下:

样例

可使用导出小度熊识别模型中的测试图片进行预测。

样例一:

不使用 GPU 测试图片 /root/projects/images/xiaoduxiong.jpeg

```
./build/demo/detector --model_dir=/root/projects/paddlex_encrypted_model --image=/root/
↪projects/xiaoduxiong.jpeg --save_dir=output --key=kLAl1qOs5uRbFt0/
↪RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key 传入加密工具输出的密钥，例如 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=, 图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例二:

使用 GPU 预测多个图片/root/projects/image_list.txt，image_list.txt 内容的格式如下:

```
/root/projects/images/xiaoduxiong1.jpeg
/root/projects/xiaoduxiong2.jpeg
...
/root/projects/xiaoduxiongn.jpeg
```

```
./build/demo/detector --model_dir=/root/projects/models/paddlex_encrypted_model --image_
↪list=/root/projects/images_list.txt --use_gpu=1 --save_dir=output --
↪key=kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key 传入加密工具输出的密钥，例如 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=，图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

2.2 Windows 平台使用

参考Windows 平台编译指南。需自行下载 Windows 版 PaddleX 加密工具压缩包，解压，在编译指南的编译流程基础上，在 CMake 设置中勾选 WITH_ENCRYPTION，ENCRYPTION_DIR 填写为加密工具包解压后的目录，再进行编译。参数与 Linux 版本预测部署一致。预测 demo 的入口程序为 paddlex_inference\detector.exe，paddlex_inference\classifier.exe，paddlex_inference\segmenter.exe。

样例

可使用导出小度熊识别模型中的测试图片进行预测。

样例一：

不使用 GPU 测试单张图片，例如图片为 D:\images\xiaoduxiong.jpeg，加密后的模型目录为 D:\projects\paddlex_encrypted_model

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\paddlex_encrypted_model --
↪image=D:\images\xiaoduxiong.jpeg --save_dir=output --key=kLAl1qOs5uRbFt0/
↪RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key 传入加密工具输出的密钥，例如 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=，图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例二：

使用 GPU 预测图片列表，例如图片列表为 D:\projects\image_list.txt，image_list.txt 的内容如下：

```
D:\projects\images\xiaoduxiong1.jpeg
D:\projects\images\xiaoduxiong2.jpeg
...
D:\projects\images\xiaoduxiongn.jpeg
```

加密后的模型目录例如为 D:\projects\paddlex_encrypted_model

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\paddlex_encrypted_model --image_
↪list=D:\projects\images_list.txt --use_gpu=1 --save_dir=output --key=kLAl1qOs5uRbFt0/
↪RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key 传入加密工具输出的密钥，例如 kLAl1qOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=，图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

13.3.3 附录-PaddlePaddle 其它模型加密使用

PaddleX 提供的加密方案不仅只适配于 PaddleX 训练的模型，对于 PaddlePaddle 其它模型套件或用户自行开发的模型同样适用。用户完全可以使用 PaddleX 的加密工具来加密如 PaddleDetection、PaddleSeg、PaddleClas、PaddleOCR 等套件训练的模型。

加密方法

1. 下载 PaddleX 加密工具

- Linux 版本
- Windows 版本

1. 加密模型一般我们使用 Paddle 训练导出的部署模型都可以保存为 __model__ 和 __params__ 两个文件，PaddleX 在模型保存时，还会额外再保存一个 model.yml 文件，用于存储模型的一些信息。PaddleX 在加密模型时，会读取模型目录中这三个文件，进行加密，生成 __model__.encrypted，__params__.encrypted 和 model.yml.encrypted 三个文件。因此在使用 PaddleX 加密工具加密别的套件或用户自行开发训练导出的模型时，需在模型目录中包含 __model__，__params__ 和 model.yml 三个文件。非 PaddleX 训练的模型，用户可自行创建一个 model.yml 的同名文件，加密后无需理会此文件即可。

   模型加密命令参考本文档前面步骤

1. 加载加密模型在使用 Paddle 预测库加载加密模型时，C++ 代码开发参考Paddle 部署代码模型加载函数，同时需要引入新的头文件 paddle_model_decrypt.h，相应依赖就在下载的加密工具中，CMakelist 中参考PaddleX 的依赖即可。

CHAPTER 14

Jetson 部署

14.1 Nvidia Jetson 开发板本地部署

14.1.1 说明

本文档在基于 Nvidia Jetpack 4.4 的 `Linux` 平台上使用 `GCC 7.4` 测试过，如需使用不同 G++ 版本，则需要重新编译 Paddle 预测库，请参考: NVIDIA Jetson 嵌入式硬件预测库源码编译。

14.1.2 前置条件

- G++ 7.4
- CUDA 10.0 / CUDNN 8 （仅在使用 GPU 版本的预测库时需要）
- CMake 3.0+

请确保系统已经安装好上述基本软件，**下面所有示例以工作目录 `/root/projects/演示`**。

Step1: 下载代码

```
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 `C++` 预测代码在 `PaddleX/deploy/cpp` 目录，该目录不依赖任何 `PaddleX` 下其他目录。

Step2: 下载 PaddlePaddle C++ 预测库 paddle_inference

目前 PaddlePaddle 为 Nvidia Jetson 提供了一个基于 1.6.2 版本的 C++ 预测库。

下载并解压后/root/projects/fluid_inference 目录包含内容为:

```
fluid_inference
  paddle # paddle 核心库和头文件
|
  third_party # 第三方依赖库和头文件
|
  version.txt # 版本和编译信息
```

Step3: 编译

编译 cmake 的命令在 scripts/jetson_build.sh 中，请根据实际情况修改主要参数，其主要内容说明如下:

```
# 是否使用 GPU(即是否使用 CUDA)
WITH_GPU=OFF
# 使用 MKL or openblas
WITH_MKL=OFF
# 是否集成 TensorRT(仅 WITH_GPU=ON 有效)
WITH_TENSORRT=OFF
# TensorRT 的路径，如果需要集成 TensorRT，需修改为您实际安装的 TensorRT 路径
TENSORRT_DIR=/root/projects/TensorRT/
# Paddle 预测库路径，请修改为您实际安装的预测库路径
PADDLE_DIR=/root/projects/fluid_inference
# Paddle 的预测库是否使用静态库来编译
# 使用 TensorRT 时，Paddle 的预测库通常为动态库
WITH_STATIC_LIB=OFF
# CUDA 的 lib 路径
CUDA_LIB=/usr/local/cuda/lib64
# CUDNN 的 lib 路径
CUDNN_LIB=/usr/local/cuda/lib64

# 以下无需改动
rm -rf build
mkdir -p build
cd build
cmake .. \
    -DWITH_GPU=${WITH_GPU} \
```

(下页继续)

(续上页)

```
    -DWITH_MKL=${WITH_MKL} \
    -DWITH_TENSORRT=${WITH_TENSORRT} \
    -DWITH_ENCRYPTION=${WITH_ENCRYPTION} \
    -DTENSORRT_DIR=${TENSORRT_DIR} \
    -DPADDLE_DIR=${PADDLE_DIR} \
    -DWITH_STATIC_LIB=${WITH_STATIC_LIB} \
    -DCUDA_LIB=${CUDA_LIB} \
    -DCUDNN_LIB=${CUDNN_LIB}
make
```

注意： linux 环境下编译会自动下载 YAML，如果编译环境无法访问外网，可手动下载：

- yaml-cpp.zip

yaml-cpp.zip 文件下载后无需解压，在 cmake/yaml.cmake 中将 `URL https://bj.bcebos.com/paddlex/deploy/deps/yaml-cpp.zip` 中的网址，改为下载文件的路径。

修改脚本设置好主要参数后，执行 `build` 脚本：

```
sh ./scripts/jetson_build.sh
```

Step4: 预测及可视化

在加载模型前，请检查你的模型目录中文件应该包括 `model.yml`、`__model__` 和 `__params__` 三个文件。如若不满足这个条件，请参考模型导出为 Inference 文档将模型导出为部署格式。

- 编译成功后，图片预测 demo 的可执行程序分别为 `build/demo/detector`，`build/demo/classifier`，`build/demo/segmenter`，用户可根据自己的模型类型选择，其主要命令参数说明如下：
- 编译成功后，视频预测 demo 的可执行程序分别为 `build/demo/video_detector`，`build/demo/video_classifier`，`build/demo/video_segmenter`，用户可根据自己的模型类型选择，其主要命令参数说明如下：

注意：若系统无 GUI，则不要将 show_result 设置为 1。当使用摄像头预测时，按 ESC 键可关闭摄像头并推出预测程序。

14.1.3 样例

可使用小度熊识别模型中导出的 `inference_model` 和测试图片进行预测，导出到/root/projects，模型路径为/root/projects/inference_model。

样例一：

不使用 GPU 测试图片 `/root/projects/images/xiaoduxiong.jpeg`

```
./build/demo/detector --model_dir=/root/projects/inference_model --image=/root/projects/
↪images/xiaoduxiong.jpeg --save_dir=output
```

图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例二:

使用 GPU 预测多个图片/root/projects/image_list.txt，image_list.txt 内容的格式如下:

```
/root/projects/images/xiaoduxiong1.jpeg
/root/projects/images/xiaoduxiong2.jpeg
...
/root/projects/images/xiaoduxiongn.jpeg
```

```
./build/demo/detector --model_dir=/root/projects/inference_model --image_list=/root/
↪projects/images_list.txt --use_gpu=1 --save_dir=output --batch_size=2 --thread_num=2
```

图片文件可视化预测结果会保存在 save_dir 参数设置的目录下。

样例三:

使用摄像头预测:

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --use_camera=1 --
↪use_gpu=1 --save_dir=output --save_result=1
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。

样例四:

对视频文件进行预测:

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --video_path=/
↪path/to/video_file --use_gpu=1 --save_dir=output --show_result=1 --save_result=1
```

当 save_result 设置为 1 时，可视化预测结果会以视频文件的格式保存在 save_dir 参数设置的目录下。如果系统有 GUI，通过将 show_result 设置为 1 在屏幕上观看可视化预测结果。

14.2 Nvidia Jetson 开发板 Docker 部署

本文档介绍了如何用 Docker 在 Jetson 开发板上部署 PaddleX 模型，通过 Docker 的方式部署，用户可以有效的避免可能因为系统环境导致编译或者运行的错误

提供了在 Jeston 上用于编译或者运行 PaddleX 部署代码的 Docker，主要有如下功能:

- 编译 PaddleX 部署代码: 用户可以通过 Docker 编译 PaddleX 部署代码

- 部署 PaddleX 模型：通过 Docker 使用编译好的可执行文件部署

注意：NVIDIA JetPack 在 v4.2.1 版本以上 (含 v4.2.1) 才能支持通过 Docker 部署

14.2.1 准备工作

在编译与运行之前的准备工作，主要是下载 Docker 与创建容器

Step1: 下载 Jetson 开发板 Docker

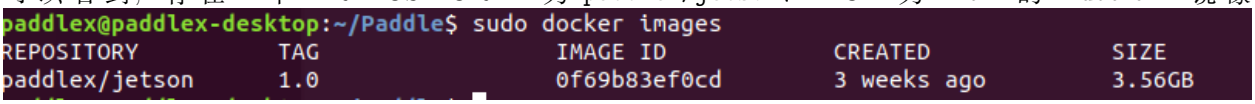
运行如下命令下载 Docker

```
sudo docker pull paddlex/jetson:1.0
```

下载成功后，通过如下命令查看 docker 的镜像

```
sudo docker images
```

可以看到，存在一个 REPOSITORY 为 paddlex/jetson、TAG 为 1.0 的 docker 镜像



Step2: 容器创建

创建容器之前，需要先准备好需要编译的部署代码与训练好的 PaddleX 部署模型

建议用户在 HOME 目录下创建 infer 文件夹，将需要部署的代码与模型拷贝到该目录下用于挂载到容器内

本文档以 PaddleX 提供的 jetson 部署代码为示例：

```
# 通过如下命令下载代码，Jetson 部署代码在 `PaddleX/deploy/cpp` 目录下面
```

git clone https://github.com/PaddlePaddle/PaddleX.git cd PaddleX git checkout release/1.3

```


# 在 HOME 目录下创建 infer 文件夹，将 cpp 文件夹拷贝到 infer 目录下面
mkdir ~/infer
cp -r PaddleX/deploy/cpp ~/infer/
```

创建容器：通过如下命令创建容器，同时将 HOME 目录下包含部署代码的 infer 文件夹挂载到容器内

```
sudo docker create -it -v ~/infer/:/infer -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=
↪$DISPLAY --net=host --name paddlex --runtime nvidia paddlex/jetson:1.0 /bin/bash
```

查看创建的容器

```
sudo docker ps -a
```

```
paddlex@paddlex-desktop:~/Paddle$ sudo docker ps -a
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
9b6f19d2e15f        paddlex/jetson:1.0     "/bin/bash"         3 seconds ago       Created                                 paddlex
paddlex@paddlex-desktop:~/Paddle$
```

创建好容器后需要运行容器

```
sudo docker start paddlex
```

14.2.2 编译

通过如下命令可以编译 infer 文件夹内的部署代码

```
sudo docker exec -it paddlex /bin/bash -c 'cd /infer/cpp && sh scripts/jetson_build.sh'
```

注意:

- cd /infer/cpp 表示进入到部署代码目录，用户需要根据实际情况自己修改

14.2.3 部署

对于图片预测，编译的可执行文件在/infer/cpp/build/demo/detector，/infer/cpp/build/demo/classifier，/infer/cpp/build/demo/segmenter, 其主要命令参数说明如下:

对于视频预测，编译的可执行文件在/infer/cpp/build/demo/video_detector，/infer/cpp/build/demo/video_classifier，/infer/cpp/build/demo/video_segmenter, 其主要命令参数说明如下:

设置 show_result 为 1 之前请执行如下命令确保容器有显示权限

```
sudo xhost +
```

注意：若系统无 GUI，则不要将 show_result 设置为 1。当使用摄像头预测时，按 ESC 键可关闭摄像头并推出预测程序。

对于使用用户编译的可执行文件进行部署的命令如下：

```
sudo docker exec -it paddlex /bin/bash -c 'cd [部署代码目录] && .build/demo/[可执行文件名] [命令参数]'
```

样例

在用户编译完部署代码后，可按如下流程运行测试模型样例

- 1) 下载 PaddleX 预训练模型及测试图片下载地址，本文档下载了 YOLOv3-MobileNetV1 模型与测试图片

- 2) 将模型导出为部署模型格式 导出部署模型步骤
- 3) 将部署模型和测试图片 copy 到 `~/infer` 文件夹
- 4) 使用如下命令，通过容器进行预测

```
sudo docker exec -it paddlex /bin/bash -c 'cd /infer/cpp && ./build/demo/detector --
↪model_dir /infer/yolov3_mobilenetv1_coco --image /infer/yolov3_mobilenetv1_coco/test.
↪jpg --use_gpu 1'
```

CHAPTER 15

Android 平台

PaddleX 的安卓端部署基于 Paddle Lite 实现，部署的流程如下，首先将训练好的模型导出为 inference model，然后对模型进行优化，最后使用 Paddle Lite 预测库进行部署，Paddle Lite 的详细介绍和使用可参考：Paddle Lite 文档

PaddleX –> Inference Model –> Paddle Lite Opt –> Paddle Lite Inference

文章简介：

- 1. 介绍如何将 PaddleX 导出为 inference model
- 2. 使用 Paddle Lite 的 OPT 模块对模型进行优化
- 3. 介绍基于 PaddleX Android SDK 的安卓 demo，以及如何快速部署训练好的模型
- 4. 介绍 PaddleX Android SDK 和二次开发

15.1 1. 将 PaddleX 模型导出为 inference 模型

参考导出 inference 模型将模型导出为 inference 格式模型。

15.2 2. 将 inference 模型优化为 Paddle Lite 模型

目前提供了两种方法将 Paddle 模型优化为 Paddle Lite 模型:

- 1. python 脚本优化模型，简单上手，目前支持最新的 Paddle Lite 2.6.1 版本

- 1. bin 文件优化模型 (linux)，支持 develop 版本 (Commit Id:11cbd50e)，**部署语义分割 `DeepLab` 模型和 `Unet` 模型时只能采用 bin 文件优化方式**。

15.2.1 2.1 使用 python 脚本优化模型

```
pip install paddlelite
python export_lite.py --model_dir /path/to/inference_model --save_file /path/to/lite_
↪model_name --place place/to/run
```

其中 `export_lite.py` 脚本请至 github 下载：https://github.com/PaddlePaddle/PaddleX/tree/release/1.3/deploy/lite/

15.2.2 2.3 使用 bin 文件优化模型 (linux)

首先下载并解压: 模型优化工具 opt

```
./opt --model_file=<model_path> \
      --param_file=<param_path> \
      --valid_targets=arm \
      --optimize_out_type=naive_buffer \
      --optimize_out=model_output_name
```

详细的使用方法和参数含义请参考: 使用 opt 转化模型

15.3 3. 移动端（Android）Demo

PaddleX 提供了基于 PaddleX Android SDK 的安卓 demo，位于`/PaddleX/deploy/lite/android/demo`，该 demo 已预置了 MobilenetV2 的模型参数，用户可直接将该 demo 导入 Android Studio 后运行体验，同时也支持用户将预置的 Mobilenetv2 模型参数替换成其他 PaddleX 导出的检测或分割模型进行预测。

15.3.1 3.1 要求

- Android Studio 3.4
- Android 手机或开发板

15.3.2 3.2 分类 Demo

3.2.1 导入工程并运行

- 打开 Android Studio，在” Welcome to Android Studio” 窗口点击” Open an existing Android Studio project”，在弹出的路径选择窗口中进入`/PaddleX/deploy/lite/android/demo` 目录，然后点击右下

角的”Open”按钮，导入工程；

- 通过 USB 连接 Android 手机或开发板；
- 载入工程后，点击菜单栏的 Run->Run ‘App’按钮，在弹出的”Select Deployment Target”窗口选择已经连接的 Android 设备，然后点击”OK”按钮；
- 运行成功后，Android 设备将加载一个名为 PaddleX Demo 的 App，默认会加载一个测试图片，同时还支持拍照和从图库选择照片进行预测；

注意：在工程构建的过程中会远程下载 Mobilenetv2 模型、yml 配置文件、测试的图片，以及 PaddleX Android SDK。

15.3.3 3.3 部署自定义模型

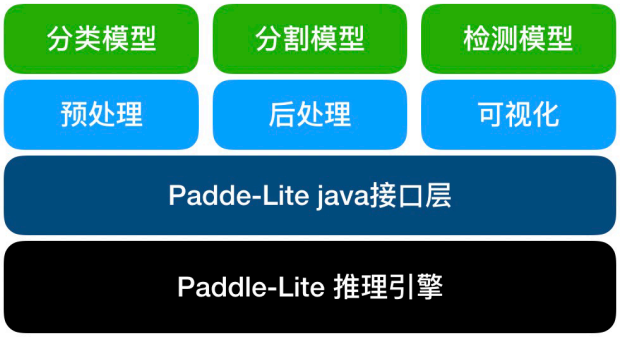
该 demo 还支持用户自定义模型来进行预测，可帮助用户快速验证自己训练好的模型，首先我们已经根据 step1~step2 描述，准备好了 Lite 模型 (.nb 文件) 和 yml 配置文件 (注意：导出 Lite 模型时需指定–place=arm)，然后在 Android Studio 的 project 视图中：

- 将.nb 文件拷贝到`/src/main/assets/model/`目录下，根据.nb 文件的名字，修改文件`/src/main/res/values/strings.xml` 中的 `MODEL_PATH_DEFAULT`；
- 将.yml 文件拷贝到`/src/main/assets/config/`目录下，根据.yml 文件的名字，修改文件`/src/main/res/values/strings.xml` 中的 `YAML_PATH_DEFAULT`；
- 可根据需要替换测试图片，将图片拷贝到`/src/main/assets/images/`目录下，根据图片文件的名字，修改文件`/src/main/res/values/strings.xml` 中的 `IMAGE_PATH_DEFAULT`；
- 将工程导入后，点击菜单栏的 Run->Run ‘App’按钮，在弹出的”Select Deployment Target”窗口选择已经连接的 Android 设备，然后点击”OK”按钮。

15.4 4. PaddleX Android SDK 和二次开发

PaddleX Android SDK 是 PaddleX 基于 Paddle Lite 开发的安卓端 AI 推理工具，以 PaddleX 导出的 Yaml 配置文件为接口，针对不同的模型实现图片的预处理，后处理，并进行可视化，开发者可集成到业务中。该 SDK 自底向上主要包括：Paddle Lite 推理引擎层，Paddle Lite 接口层以及 PaddleX 业务层。

- Paddle Lite 推理引擎层，是在 Android 上编译好的二进制包，只涉及到 Kernel 的执行，且可以单独部署，以支持极致轻量级部署。
- Paddle Lite 接口层，以 Java 接口封装了底层 c++ 推理库。
- PaddleX 业务层，封装了 PaddleX 导出模型的预处理，推理和后处理，以及可视化，支持 PaddleX 导出的检测、分割、分类模型。



架构

15.4.1 4.1 SDK 安装

首先下载并解压PaddleX Android SDK，得到 paddlex.aar 文件，将拷贝到 android 工程目录 app/libs/下面，然后为 app 的 build.gradle 添加依赖：

```
dependencies {
    implementation fileTree(include: ['*.jar','*aar'], dir: 'libs')
}
```

15.4.2 4.2 SDK 使用用例

```
import com.baidu.paddlex.Predictor;
import com.baidu.paddlex.config.ConfigParser;
import com.baidu.paddlex.postprocess.DetResult;
import com.baidu.paddlex.postprocess.SegResult;
import com.baidu.paddlex.postprocess.ClsResult;
import com.baidu.paddlex.visual.Visualize;

// Predictor
Predictor predictor = new Predictor();
// model config
ConfigParser configParser = new ConfigParser();
// Visualize
```

(下页继续)

(续上页)

```
Visualize visualize = new Visualize();
// image to predict
Mat predictMat;

// initialize
configParser.init(context, model_path, yaml_path, cpu_thread_num, cpu_power_mode);
visualize.init(configParser.getNumClasses());
predictor.init(context, configParser)

// run model
if (predictImage != null && predictor.isLoaded()) {
    predictor.setInputMat(predictMat);
    runModel();
}

// get result & visualize
if (configParser.getModelType().equalsIgnoreCase("segmenter")) {
    SegResult segResult = predictor.getSegResult();
    Mat visualizeMat  = visualize.draw(segResult, predictMat, predictor.getImageBlob());
} else if (configParser.getModelType().equalsIgnoreCase("detector")) {
    DetResult detResult = predictor.getDetResult();
    Mat visualizeMat = visualize.draw(detResult, predictMat);
} else if (configParser.getModelType().equalsIgnoreCase("classifier")) {
    ClsResult clsResult = predictor.getClsResult();
}
```

15.4.3 4.3 Result 成员变量

注意：Result 所有的成员变量以 java bean 的方式获取。

```
com.baidu.paddlex.postprocess.ClsResult
```

Fields

- **type** (String|static)：值为”cls”。
- **categoryId** (int)：类别 ID。
- **category** (String)：类别名称。
- **score** (float)：预测置信度。

com.baidu.paddlex.postprocess.DetResult

Nested classes

- **DetResult.Box** 模型预测的 box 结果。

Fields

- **type** (String|static): 值为” det”。
- **boxes** (List<DetResult.Box>): 模型预测的 box 结果。

com.baidu.paddlex.postprocess.DetResult.Box

Fields

- **categoryId** (int): 类别 ID。
- **category** (String): 类别名称。
- **score** (float): 预测框的置信度。
- **coordinate** (float[4]): 预测框的坐标值 {xmin, ymin, xmax, ymax}。

com.baidu.paddlex.postprocess.SegResult

Nested classes

- **SegResult.Mask**: 模型预测的 mask 结果。

Fields

- **type** (String|static): 值为” Seg”。
- **mask** (SegResult.Mask): 模型预测的 mask 结果。

com.baidu.paddlex.postprocess.SegResult.Mask

Fields

- **scoreData** (float[]): 模型预测在各个类别的置信度，长度为: 1 * numClass * H * W
- **scoreShape** (long[4]): scoreData 的 shape 信息，[1, numClass, H, W]

- **labelData** (long[]): 模型预测置信度最高的 label，长度为: 1 * H * W * 1
- **labelShape** (long[4]): labelData 的 shape 信息，[1, H, W, 1]

15.4.4 4.4 SDK 二次开发

- 打开 Android Studio 新建项目 (或加载已有项目)。点击菜单 File->New->Import Module，导入工程/PaddleX/deploy/lite/android/sdk, Project 视图会新增名为 sdk 的 module
- 在 app 的 build.grade 里面添加依赖:

```
dependencies {
    implementation project(':sdk')
}
```

- 源代码位于 sdk/main/java/下，修改源码进行二次开发后，点击菜单栏的 Build->Run ‘sdk’按钮可编译生成 aar，文件位于 sdk/build/outputs/aar/路径下。

CHAPTER 16

树莓派部署

16.1 树莓派

PaddleX 支持通过 Paddle-Lite 和基于 OpenVINO 的神经计算棒 (NCS2) 这两种方式在树莓派上完成预测部署。

16.1.1 硬件环境配置

对于尚未安装系统的树莓派首先需要进行系统安装、环境配置等步骤来初始化硬件环境，过程中需要的软硬件如下：

- 硬件：micro SD，显示器，键盘，鼠标
- 软件：Raspbian OS

Step1：系统安装

- 格式化 micro SD 卡为 FAT 格式，Windows 和 Mac 下建议使用SD Memory Card Formatter工具，Linux 下请参考NOOBS For Raspberry Pi
- 下载 NOOBS 版本的 Raspbian OS 下载地址并将解压后的文件复制到 SD 中，插入 SD 后给树莓派通电，然后将自动安装系统

Step2：环境配置

- 启用 VNC 和 SSH 服务：打开 LX 终端输入，输入如下命令，选择 Interfacing Option 然后选择 P2 SSH 和 P3 VNC 分别打开 SSH 与 VNC。打开后就可以通过 SSH 或者 VNC 的方式连接树莓派

```
sudo raspi-config
```

- 更换源：由于树莓派官方源速度很慢，建议在官网查询国内源 树莓派软件源。更换后执行

```
sudo apt-get update
sudo apt-get upgrade
```

16.1.2 Paddle-Lite 部署

基于 Paddle-Lite 的部署目前可以支持 PaddleX 的分类、分割与检测模型，其中检测模型仅支持 YOLOV3

部署的流程包括：PaddleX 模型转换与转换后的模型部署

说明：PaddleX 安装请参考PaddleX，Paddle-Lite 详细资料请参考Paddle-Lite

请确保系统已经安装好上述基本软件，并配置好相应环境，**下面所有示例以工作目录 `/root/projects/演示`。**

Paddle-Lite 模型转换

将 PaddleX 模型转换为 Paddle-Lite 模型，具体请参考Paddle-Lite 模型转换

Paddle-Lite 预测

Step1 下载 PaddleX 预测代码

```
mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 C++ 预测代码在 PaddleX/deploy/raspberry 目录，该目录不依赖任何 PaddleX 下其他目录，如果需要在 python 下预测部署请参考Python 预测部署。

Step2：Paddle-Lite 预编译库下载

对于 Armv7hf 的用户提供了 2.6.1 版本的 Paddle-Lite 在架构为 armv7hf 的 ArmLinux 下面的 Full 版本预编译库:Paddle-Lite(ArmLinux) 预编译库对于 Armv8 的用户提供了 2.6.3 版本的 Paddle-Lite 在架构为 armv8 的 ArmLinux 下面的 full 版本预编译库:Paddle-Lite(ArmLinux) 与编译库其他版本与 arm 架构的 Paddle-Lite 预测库请在官网Releases下载若用户需要在树莓派上自行编译 Paddle-Lite，在树莓派上 LX 终端输入

```
git clone https://github.com/PaddlePaddle/Paddle-Lite.git
cd Paddle-Lite
sudo ./lite/tools/build.sh  --arm_os=armlinux --arm_abi=armv7hf --arm_lang=gcc  --build_
↪extra=ON full_publish
```

预编库位置：`./build.lite.armlinux.armv7hf.gcc/inference_lite_lib.armlinux.armv7hf/cxx`

注意: 预测库版本需要跟 opt 版本一致，检测与分割请使用 Paddle-Lite 的 full 版本预测库，更多 Paddle-Lite 编译内容请参考Paddle-Lite 编译；更多预编译 Paddle-Lite 预测库请参考Paddle-Lite Release Note

Step3 软件依赖

提供了依赖软件的预编包或者一键编译，对于 armv7 树莓派用户不需要单独下载或编译第三方依赖软件，对于 armv8 树莓派用户需要自行编译 opencv，编译第三方依赖软件请参考：

- gflags：编译请参考 编译文档
- opencv: 编译请参考 编译文档**注意**: 对于 armv8 树莓派用户，需要自行编译 opencv

Step4: 编译

编译 `cmake` 的命令在 `scripts/build.sh` 中，修改 LITE_DIR 为 Paddle-Lite 预测库目录，若自行编译第三方依赖软件请根据 Step1 中编译软件的实际情况修改主要参数，其主要内容说明如下：

```
# Paddle-Lite 预编译库的路径
LITE_DIR=/path/to/Paddle-Lite/inference/lib
# gflags 预编译库的路径，若没有自行编译无需修改
GFLAGS_DIR=$(pwd)/deps/gflags
# opencv 预编译库的路径，若自行编译请指定到对应路径
OPENCV_DIR=$(pwd)/deps/opencv/
# arm 处理器架构 armv7-a 或者 armv8-a
ARCH=armv7-a
# Lite 预测库版本 light 或者 full
LITE=full
```

执行 `build` 脚本：

```
sh ./scripts/build.sh
```

Step5: 预测

编译成功后，分类任务的预测可执行程序为 `classifier`，分割任务的预测可执行程序为 `segmenter`，检测任务的预测可执行程序为 `detector`，其主要命令参数说明如下：

样例

样例一：单张图片分类任务测试图片 `/path/to/test_img.jpeg`

```
./build/classifier --model_dir=/path/to/nb_model
--image=/path/to/test_img.jpeg --cfg_file=/path/to/PadlleX_model.yml  --thread_num=4
```

样例二：多张图片分割任务预测多个图片`/path/to/image_list.txt`，image_list.txt 内容的格式如下：

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
./build/segmenter --model_dir=/path/to/models/nb_model --image_list=/root/projects/
↪images_list.txt --cfg_file=/path/to/PadlleX_model.yml --save_dir ./output --thread_
↪num=4
```

16.1.3 性能测试

测试环境：

硬件：Raspberry Pi 3 Model B 系统：raspbian OS 软件：paddle-lite 2.6.1

测试结果

单位 ms，num 表示 paddle-lite 下使用的线程数

从测试结果看建议用户在树莓派上使用 MobileNetV1-V3,ShuffleNetV2 这类型的小型网络

16.1.4 NCS2 部署

树莓派支持通过 OpenVINO 在 NCS2 上跑 PaddleX 模型预测，目前仅支持 PaddleX 的分类网络，基于 NCS2 的方式包含 Paddle 模型转 OpenVINO IR 以及部署 IR 在 NCS2 上进行预测两个步骤。

- 模型转换请参考：PaddleX 模型转换为 OpenVINO IR，raspbian OS 上的 OpenVINO 不支持模型转换，需要先在 host 侧转换 FP16 的 IR。
- 预测部署请参考OpenVINO 部署中 VPU 在 raspbian OS 部署的部分
- 目前仅支持 armv7 的树莓派

16.2 Python 预测部署

文档说明了在树莓派上使用 Python 版本的 Paddle-Lite 进行 PaddleX 模型好的预测部署，根据下面的命令安装 Python 版本的 Paddle-Lite 预测库，若安装不成功用户也可以下载 whl 文件进行安装Paddle-Lite_2.6.0_python，更多版本请参考Paddle-Lite Release Note

```
python -m pip install paddlelite
```

部署前需要先将 PaddleX 模型转换为 Paddle-Lite 的 nb 模型，具体请参考Paddle-Lite 模型转换 **注意**：若用户使用 2.6.0 的 Python 预测库，请下载 2.6.0 版本的 opt 转换工具转换模型

16.2.1 前置条件

- Python 3.6+
- Paddle-Lite_python 2.6.0+

请确保系统已经安装好上述基本软件，**下面所有示例以工作目录 `/root/projects/演示`**。

16.2.2 预测部署

运行/root/projects/PaddleX/deploy/raspberry/python 目录下 demo.py 文件可以进行预测，其命令参数说明如下：

注意：由于 Paddle-lite 的 python api 尚不支持 int64 数据的输入，目前树莓派在 python 下不支持部署 YoloV3，如需要请使用 C++ 代码部署 YoloV3 模型

样例

样例一：测试图片 `/path/to/test_img.jpeg`

```
cd /root/projects/python

python demo.py --model_dir /path/to/nb_model --img /path/to/test_img.jpeg --cfg_file /
↪path/to/PadlleX_model.yml --thread_num 4
```

样例二 ‘:

预测多个图片/path/to/image_list.txt，image_list.txt 内容的格式如下：

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
cd /root/projects/python

python demo.py --model_dir /path/to/models/nb_model --image_list /root/projects/images_
↪list.txt --cfg_file=/path/to/PadlleX_model.yml --thread_num 4
```

16.3 Paddle-Lite 模型转换

将 PaddleX 模型转换为 Paddle-Lite 的 nb 模型，模型转换主要包括 PaddleX 转 inference model 和 inference model 转 Paddle-Lite nb 模型

16.3.1 Step1：导出 inference 模型

PaddleX 模型转 Paddle-Lite 模型之前需要先把 PaddleX 模型导出为 inference 格式模型，导出的模型将包括 ___model___、___params___ 和 model.yml 三个文件名。具体方法请参考Inference 模型导出。

16.3.2 Step2：导出 Paddle-Lite 模型

Paddle-Lite 模型需要通过 Paddle-Lite 的 opt 工具转出模型

- 对于 armv7hf 的用户下载并解压: 模型优化工具 opt（2.6.1-linux）
- 对于 armv8 的用户下载：模型优化工具 opt（2.6.3-linux）

在 Linux 系统下运行：

```
./opt --model_file=<model_path> \
      --param_file=<param_path> \
```

(下页继续)

(续上页)

```
    --valid_targets=arm \
    --optimize_out_type=naive_buffer \
    --optimize_out=model_output_name
```

| 参数 | 说明 | | —- | —- | | –model_file | 导出 inference 模型中包含的网络结构文件: `__model__` 所在的路径 | | –param_file | 导出 inference 模型中包含的参数文件: `__params__` 所在的路径 | | –valid_targets | 指定模型可执行的 backend, 这里请指定为 `arm`| | –optimize_out_type | 输出模型类型, 目前支持两种类型: protobuf 和 naive_buffer，其中 naive_buffer 是一种更轻量级的序列化/反序列化，这里请指定为 `naive_buffer`|

若安装了 python 版本的 Paddle-Lite 也可以通过如下方式转换

```
./paddle_lite_opt --model_file=<model_path> \
    --param_file=<param_path> \
    --valid_targets=arm \
    --optimize_out_type=naive_buffer \
    --optimize_out=model_output_name
```

更多详细的使用方法和参数含义请参考: 使用 opt 转化模型, 更多 opt 预编译版本请参考Paddle-Lite Release Note

注意: opt 版本需要跟预测库版本保持一致，比如若使用 2.6.0 版本预测库，请从上面 Release Note 中下载 2.6.0 版本的 opt 转换模型

16.4 神经计算棒 2 代

PaddleX 支持在树莓派上插入 NCS2(神经计算棒 2 代) 通过 OpenVINO 部署 PadlleX 训练出来的分类模型

注意: 目前仅支持分类模型、仅支持 Armv7hf 的树莓派

16.4.1 前置条件

- OS: Raspbian OS
- PaddleX 1.0+
- OpenVINO 2020.4
- Raspbian OS: 树莓派操作操作系统下载与安装请参考树莓派系统安装与环境配置
- PaddleX: PaddleX 安装请参考PaddleX
- OpenVINO: OpenVINO 的安装请参考OpenVINO-Raspbian

注意: 安装完 OpenVINO 后需要初始化 OpenVINO 环境，并且需要对 USB 进行配置，请参考:

```
# 初始化 OpenVINO 环境
source /opt/intel/openvino/bin/setupvars.sh
# 将初始化 OpenVINO 环境的规则加入到 bashrc 中
echo "source /opt/intel/openvino/bin/setupvars.sh" >> ~/.bashrc
# 配置 USB
sh /opt/intel/openvino/install_dependencies/install_NCS_udev_rules.sh
```

16.4.2 部署流程

部署流程主要分为模型转换与转换后模型部署两个步骤，下面以 MobilnetV2 模型为例，介绍如何将 PaddleX 训练好的模型通过 OpenVINO 部署到插入 NCS2 的树莓派教程的示例项目训练 MobilenetV2 模型，请参考PaddleX 模型训练示例

16.4.3 模型转换

模型转换指的是将 PaddleX 训练出来的 Paddle 模型转换为 OpenVINO 的 IR，对于模型转换教程可以参考OpenVINO 模型转换

注意：树莓派上面安装的 OpenVINO 是不带 Model Optmizier 模块的，不能在上面进行模型转换，请在 Host 下载与树莓派一直的 OpenVINO 版本，然后进行模型转换。

以转换训练好的 MobileNetV2 为示例，请参考以下命令:

```
# 安装 paddlex
pip install paddlex

# 下载 PaddleX 代码
```

git clone https://github.com/PaddlePaddle/PaddleX.git cd PaddleX git checkout release/1.3

```
# 进入模型转换脚本文件夹
cd PaddleX/deploy/openvino/python

# 导出 inference 模型，执行命令前务必将训练好的 MobileNetV2 模型拷贝到当前目录，并命名为 MobileNetV2
paddlex --export_inference --model_dir ./MobileNetV2 --save_dir ./MobileNetV2 --fixed_
↪input_shape [224,224]
# 完成导出后会在 MobileNetV2 文件夹下面出现，__model__、__params__、model.yml 三个文件

# 转换 Paddle inference 模型到 OpenVINO IR
```

(下页继续)

(续上页)

```
python converter.py --model_dir ./MobileNetV2 --save_dir ./MobileNetV2 --fixed_input_
↪shape [224,224] --data_type FP16
# 转换成功后会在 MobileNetV2 目录下面出现 paddle2onnx.xml、paddle2onnx.mapping、
paddle2onnx.bin 三个文件
```

16.4.4 模型部署

PaddleX 支持 Python 和 C++ 两种方式在树莓派上通过 NCS2 部署：

- C++：C++ 部署教程请参考OpenVINO_Raspberry
- python：python 部署教程请参考OpenVINO_python

以转换好的 MobileNetV2 模型为示例

准备工作

```
# 在树莓派上下载 PaddleX 代码
```

git clone https://github.com/PaddlePaddle/PaddleX.git cd PaddleX git checkout release/1.3

```
# 进入 OpenVINO 部署代码
cd deploy/openvino
# 将 MobileNetV2 转好的 OpenVINO IR 以及测试图片拷贝到树莓派上面，并以及 MobileNetV2 文件夹
放到 OpenVINO 部署的代码的目录
```

C++ 部署

```
# 修改编译文件 script/build.sh，将 ARCH 参数修改为 armv7
vim script/build.sh
# 编译代码
sh script/build.sh
#OpenVINO 部署
./build/classfier --model_dir MobileNetV2/paddle2onnx.xml --image [测试图片路径] --device␣
↪MYRIAD --cfg_file MobileNetV2/model.yml --save_dir output
```

执行成功后会在 output 文件夹面保存测试图片的可视化结果

python 部署

```
进入 python 部署代码目录
cd python
#python 部署
python demo.py --model_dir ../MobileNetV2/paddle2onnx.xml --img [测试图片路径] --device␣
↪MYRIAD --cfg_file MobileNetV2/model.yml
```

(下页继续)

(续上页)

CHAPTER 17

OpenVINO 部署

17.1 OpenVINO 部署简介

PaddleX 支持将训练好的 Paddle 模型通过 OpenVINO 实现模型的预测加速，OpenVINO 详细资料与安装流程请参考OpenVINO，本文档使用 OpenVINO 2020.4 与 2021.1 测试通过。**注意**:

- 由于 PaddleX 分割模型使用了 ReSize-11 Op，OpenVINO 2021.1 版本开始支持支持 Resize-11 ，CPU 下请务必下载 OpenVINO 2021.1+ 版本
- 由于 VPU 在 OpenVINO 2021.1 版本下转换的分类模型会出现 Range layer 不支持的情况，VPU 下请务必下载 OpenVINO 2020.4 版本
- 安装 OpenVINO 过程中请务必参考 OpenVINO 官网教程，初始化 OpenVINO 使用环境，以及安装 OpenVINO 相关依赖

17.1.1 部署支持情况

下表提供了 PaddleX 在不同环境下对使用 OpenVINO 加速的支持情况

注意：其中 Raspbian OS 为树莓派操作系统。检测模型仅支持 YOLOv3

17.1.2 部署流程

PaddleX 到 OpenVINO 的部署流程可以分为如下两步:

- **模型转换**: 将 Paddle 的模型转换为 OpenVINO 的 Inference Engine

- **预测部署**: 使用 Inference Engine 进行预测

17.1.3 模型转换

模型转换请参考文档模型转换说明：由于不同软硬件平台下 OpenVINO 模型转换方法一致，故如何转换模型后续文档中不再赘述。

17.1.4 预测部署

由于不同软硬下部署 OpenVINO 实现预测的方式不完全一致，具体请参考：

Linux: 介绍了 PaddleX 在操作系统为 Linux 或者 Raspbian OS，编程语言为 C++，硬件平台为 CPU 或者 VPU 的情况下使用 OpenVINO 进行预测加速

Windows: 介绍了 PaddleX 在操作系统为 Window，编程语言为 C++，硬件平台为 CPU 或者 VPU 的情况下使用 OpenVINO 进行预测加速

Python: 介绍了 PaddleX 在 python 下使用 OpenVINO 进行预测加速

部署常见问题: 介绍了部署过程中遇到的常见问题以及其解决方案

17.2 Windows 平台

17.2.1 说明

Windows 平台下，我们使用 `Visual Studio 2019 Community` 进行了测试。微软从 `Visual Studio 2017` 开始即支持直接管理 `CMake` 跨平台编译项目，但是直到 `2019` 才提供了稳定和完全的支持，所以如果你想使用 CMake 管理项目编译构建，我们推荐你使用 `Visual Studio 2019` 环境下构建。

17.2.2 前置条件

- Visual Studio 2019
- OpenVINO 2020.4 或者 2021.1+
- CMake 3.0+

说明：

- PaddleX 安装请参考PaddleX ，OpenVINO 安装请参考OpenVINO-Windows
- CPU 下请使用 OpenVINO 2021.1+ 版本；VPU 下请使用 OpenVINO 2020.4 版本

注意：安装完 OpenVINO 后需要手动添加 OpenVINO 目录到系统环境变量，否则在运行程序时会出现找不到 dll 的情况。以安装 OpenVINO 时不改变 OpenVINO 安装目录情况下为示例，流程如下

- 我的电脑-> 属性-> 高级系统设置-> 环境变量

- 在系统变量中找到 Path（如没有，自行创建），并双击编辑
- 新建，分别将 OpenVINO 以下路径填入并保存:

  C:\Program File (x86)\IntelSWTools\openvino\inference_engine\bin\intel64\Release

  C:\Program File (x86)\IntelSWTools\openvino\inference_engine\external\tbb\bin

  C:\Program File (x86)\IntelSWTools\openvino\deployment_tools\ngraph\lib

请确保系统已经安装好上述基本软件，并配置好相应环境，**下面所有示例以工作目录为 D:\projects 演示**。

17.2.3 预测部署

文档提供了 c++ 下预测部署的方法，如果需要在 python 下预测部署请参考python 预测部署

Step1: 下载 PaddleX 预测代码

```
d:
mkdir projects
cd projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 C++ 预测代码在 PaddleX\deploy\openvino 目录，该目录不依赖任何 PaddleX 下其他目录。

Step2 软件依赖

提供了依赖软件预编译库:

- gflas
- opencv

请下载上面两个连接的预编译库。若需要自行下载请参考:

- gflags:下载地址
- opencv:下载地址

下载完 opencv 后需要配置环境变量，如下流程所示- 我的电脑-> 属性-> 高级系统设置-> 环境变量 - 在系统变量中找到 Path（如没有，自行创建），并双击编辑 - 新建，将 opencv 路径填入并保存，如 D:\projects\opencv\build\x64\vc14\bin

Step3: 使用 Visual Studio 2019 直接编译 CMake

1. 打开 Visual Studio 2019 Community，点击继续但无需代码
2. 点击：文件-> 打开->CMake 选择 C++ 预测代码所在路径（例如 D:\projects\PaddleX\deploy\openvino），并打开 CMakeList.txt
3. 点击：项目->CMake 设置
4. 点击浏览，分别设置编译选项指定 OpenVINO、Gflags、NGRAPH、OPENCV 的路径

设置完成后，点击保存并生成 CMake 缓存以加载变量。5. 点击生成-> 全部生成

Step5: 预测

上述 Visual Studio 2019 编译产出的可执行文件在 out\build\x64-Release 目录下，打开 cmd，并切换到该目录：

```
D:
cd D:\projects\PaddleX\deploy\openvino\out\build\x64-Release
```

- 编译成功后，图片预测 demo 的入口程序为 detector.exe，classifier.exe，segmenter.exe，用户可根据自己的模型类型选择，其主要命令参数说明如下：

样例

样例一：在 CPU 下做单张图片的分类任务预测测试图片 /path/to/test_img.jpeg

```
./classifier.exe --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --
↪cfg_file=/path/to/PadlleX_model.yml
```

样例二：在 CPU 下做多张图片的检测任务预测，并保存预测可视化结果预测多个图片/path/to/image_list.txt，image_list.txt 内容的格式如下：

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
./detector.exe --model_dir=/path/to/models/openvino_model --image_list=/root/projects/
↪images_list.txt --cfg_file=/path/to/PadlleX_model.yml --save_dir ./output
```

样例三: 在 VPU 下做单张图片分类任务预测测试图片 /path/to/test_img.jpeg

```
.classifier.exe --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --cfg_
↪file=/path/to/PadlleX_model.yml --device=MYRIAD
```

17.3 Linux 平台

17.3.1 前置条件

- OS: Ubuntu、Raspbian OS
- GCC* 5.4.0
- CMake 3.0+
- PaddleX 1.0+
- OpenVINO 2020.4 或者 2021.1+
- 硬件平台：CPU、VPU

说明：PaddleX 安装请参考PaddleX ，OpenVINO 安装请根据相应的系统参考OpenVINO-Linux或者OpenVINO-Raspbian

注意：CPU 下请使用 OpenVINO 2020.1+ 版本、VPU 下请使用 2020.4 版本

请确保系统已经安装好上述基本软件，并配置好相应环境，**下面所有示例以工作目录 /root/projects/演示**。

17.3.2 预测部署

文档提供了 c++ 下预测部署的方法，如果需要在 python 下预测部署请参考python 预测部署

Step1 下载 PaddleX 预测代码

```
mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
```

说明：其中 C++ 预测代码在 PaddleX/deploy/openvino 目录，该目录不依赖任何 PaddleX 下其他目录。

Step2 软件依赖

Step3 中的编译脚本会一键安装第三方依赖软件的预编译包，用户不需要单独下载或编译这些依赖软件。若需要自行编译第三方依赖软件请参考：

- gflags：编译请参考 编译文档
- opencv: 编译请参考 编译文档

Step3: 编译

编译 cmake 的命令在 scripts/build.sh 中，若在树莓派 (Raspbian OS) 上编译请修改 ARCH 参数 x86 为 armv7，若自行编译第三方依赖软件请根据 Step1 中编译软件的实际情况修改主要参数，其主要内容说明如下：

```
# openvino 预编译库的路径
OPENVINO_DIR=$INTEL_OPENVINO_DIR/inference_engine
# gflags 预编译库的路径
GFLAGS_DIR=$(pwd)/deps/gflags
# ngraph lib 预编译库的路径
NGRAPH_LIB=$INTEL_OPENVINO_DIR/deployment_tools/ngraph/lib
# opencv 预编译库的路径
OPENCV_DIR=$(pwd)/deps/opencv/
#cpu 架构 (x86 或 armv7)
ARCH=x86
```

执行 build 脚本：

```
sh ./scripts/build.sh
```

Step4: 预测

编译成功后，分类任务的预测可执行程序为 classifier，检测任务的预测可执行程序为 detector，分割任务的预测可执行程序为 segmenter，其主要命令参数说明如下：

样例

样例一：linux 系统在 CPU 下做单张图片的分类任务预测测试图片 /path/to/test_img.jpeg

```
./build/classifier --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --
↪cfg_file=/path/to/PadlleX_model.yml
```

样例二: linux 系统在 CPU 下做多张图片的检测任务预测，并保存预测可视化结果预测的多个图片/path/to/image_list.txt，image_list.txt 内容的格式如下:

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
./build/detector --model_dir=/path/to/models/openvino_model --image_list=/root/projects/
↪images_list.txt --cfg_file=/path/to/PadlleX_model.yml --save_dir ./output
```

样例三: 树莓派 (Raspbian OS) 在 VPU 下做单张图片分类任务预测测试图片 /path/to/test_img.jpeg

```
./build/classifier --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --
↪cfg_file=/path/to/PadlleX_model.yml --device=MYRIAD
```

17.3.3 性能测试

测试一：在服务器 CPU 下测试了 OpenVINO 对 PaddleX 部署的加速性能:

- CPU：Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
- OpenVINO：2020.4
- PaddleX：采用 Paddle 预测库 (1.8)，打开 mkldnn 加速，打开多线程。
- 模型来自 PaddleX tutorials，Batch Size 均为 1，耗时单位为 ms/image，只计算模型运行时间，不包括数据的预处理和后处理，20 张图片 warmup，100 张图片测试性能。

测试二: 在 PC 机上插入 VPU 架构的神经计算棒 (NCS2)，通过 Openvino 加速。

- CPU：Intel(R) Core(TM) i5-4300U 1.90GHz
- VPU：Movidius Neural Compute Stick2
- OpenVINO：2020.4
- 模型来自 PaddleX tutorials，Batch Size 均为 1，耗时单位为 ms/image，只计算模型运行时间，不包括数据的预处理和后处理，20 张图片 warmup，100 张图片测试性能。

测试三: 在树莓派 3B 上插入 VPU 架构的神经计算棒 (NCS2)，通过 Openvino 加速。

- CPU ： ARM Cortex-A72 1.2GHz 64bit
- VPU：Movidius Neural Compute Stick2
- OpenVINO 2020.4
- 模型来自 paddleX tutorials，Batch Size 均为 1，耗时单位为 ms/image，只计算模型运行时间，不包括数据的预处理和后处理，20 张图片 warmup，100 张图片测试性能。

17.4 Python 预测部署

文档说明了在 python 下基于 OpenVINO 的预测部署，部署前需要先将 paddle 模型转换为 OpenVINO 的 Inference Engine，请参考模型转换。目前 CPU 硬件上支持 PadlleX 的分类、检测、分割模型；VPU 上支持 PaddleX 的分类模型。

17.4.1 前置条件

- Python 3.6+
- OpenVINO 2021.1

说明：OpenVINO 安装请参考OpenVINO

请确保系统已经安装好上述基本软件，**下面所有示例以工作目录 /root/projects/演示**。

17.4.2 预测部署

运行/root/projects/PaddleX/deploy/openvino/python 目录下 demo.py 文件可以进行预测，其命令参数说明如下：

样例

样例一：测试图片 /path/to/test_img.jpeg

```
cd /root/projects/python

python demo.py --model_dir /path/to/openvino_model --img /path/to/test_img.jpeg --cfg_
↪file /path/to/PadlleX_model.yml
```

样例二 ‘:

预测多个图片/path/to/image_list.txt，image_list.txt 内容的格式如下：

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
cd /root/projects/python

python demo.py --model_dir /path/to/models/openvino_model --image_list /root/projects/
↪images_list.txt --cfg_file=/path/to/PadlleX_model.yml
```

17.5 OpenVINO 模型转换

将 Paddle 模型转换为 OpenVINO 的 Inference Engine

17.5.1 环境依赖

- Paddle2ONNX 0.4
- ONNX 1.6.0+
- PaddleX 1.3+
- OpenVINO 2020.4+

说明：PaddleX 安装请参考PaddleX ，OpenVINO 安装请参考OpenVINO，ONNX 请安装 1.6.0 以上版本否则会出现转模型错误, Paddle2ONNX 请安装 0.4 版本。**注意**：安装 OpenVINO 时请务必安装官网教程初始化 OpenVINO 运行环境，并安装相关依赖, 否则会出现”No module named mo”等问题

请确保系统已经安装好上述基本软件，**下面所有示例以工作目录 /root/projects/演示**。

17.5.2 导出 inference 模型

paddle 模型转 openvino 之前需要先把 paddle 模型导出为 inference 格式模型，导出的模型将包括___model___、___params___ 和 model.yml 三个文件名，导出命令如下

```
paddlex --export_inference --model_dir=/path/to/paddle_model --save_dir=./inference_
↪model --fixed_input_shape=[w,h]
```

注意：需要转 OpenVINO 模型时，导出 inference 模型请务必指定`--fixed_input_shape` 参数来固定模型的输入大小，且模型的输入大小需要与训练时一致。PaddleX 客户端在发布模型时没有固定输入大小，因此对于可视化客户端，请找到任务所在目录，从里面的 `output` 文件夹找到 `best_model` 模型目录，将此目录使用如上命令进行固定 shape 导出即可。

17.5.3 导出 OpenVINO 模型

```
mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout release/1.3
cd deploy/openvino/python

python converter.py --model_dir /path/to/inference_model --save_dir /path/to/openvino_
↪model --fixed_input_shape [w,h]
```

(下页继续)

(续上页)

转换成功后会在 save_dir 下出现后缀名为.xml、.bin、.mapping 三个文件转换参数说明如下：

注意：

- 若转换过程中出现”No module named mo”的问题，请先初始化 OpenVINO 环境 [./faq.md] 再进行模型转换
- 由于 OpenVINO 从 2021.1 版本开始支持 ONNX 的 resize-11 OP 的原因，对于 CPU 部署请下载 OpenVINO 2021.1+ 的版本
- 由于 VPU 上不支持 Range Layer，对于 VPU 部署请下载 OpenVINO 2020.4 版本进行模型转换
- YOLOv3 在通过 OpenVINO 部署时，由于 OpenVINO 对 ONNX OP 的支持限制，我们在将 YOLOv3 的 Paddle 模型导出时，对最后一层 multiclass_nms 进行了特殊处理，导出的 ONNX 模型，最终输出的 Box 结果包括背景类别（而 Paddle 模型不包含），此处在 OpenVINO 的部署代码中，我们通过后处理过滤了背景类别。

17.6 OpenVINO 部署常见问题

17.6.1 Q1 转模型过程中出现”ModuleNotFoundError: No module named ‘mo’”

原因：该问题主要是因为在安装 OpenVINO 之后未初始化 OpenVINO 环境解决方案：找到 OpenVINO 初始化环境脚本，运行后即可以解决此问题

Linux 系统初始化 OpenVINO 环境

1)root 用户安装，以 OpenVINO 2021.1 版本为例，运行如下命令即可初始化

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

2) 非 root 用户安装，以 OpenVINO 2021.1 版本、用户名为 paddlex 为例，运行如下命令即可初始化

```
source /home/paddlex/intel/openvino_2021/bin/setupvar.sh
```

Window 系统初始化 OpenVINO 环境

以 OpenVINO 2021.1 版本为例，执行如下命令即可初始化 OpenVINO 环境

```
cd C:\Program Files (x86)\Intel\openvino_2021\bin\
setupvars.bat
```

说明：更多初始化 OpenVINO 环境的细节请参考OpenVINO 官网

17.6.2 Q2 提示" convert failed, please export paddle inference model by fixed_input_shape"

原因：该问题是因为在使用 paddlex 导出 inference 模型的时候没有加入–fixed_input_shape 参数固定 shape
解决方案：导出 inference 模型的时候加入–fixed_input_shape 参数导出 inference 模型参考

17.6.3 Q3 提示""

原因：paddle 模型没有导出 inference 格式解决方案：对模型先导出 inference 格式，注意导出时需要使用–fixed_input_shape 参数固定 shape导出 inference 模型参考

CHAPTER 18

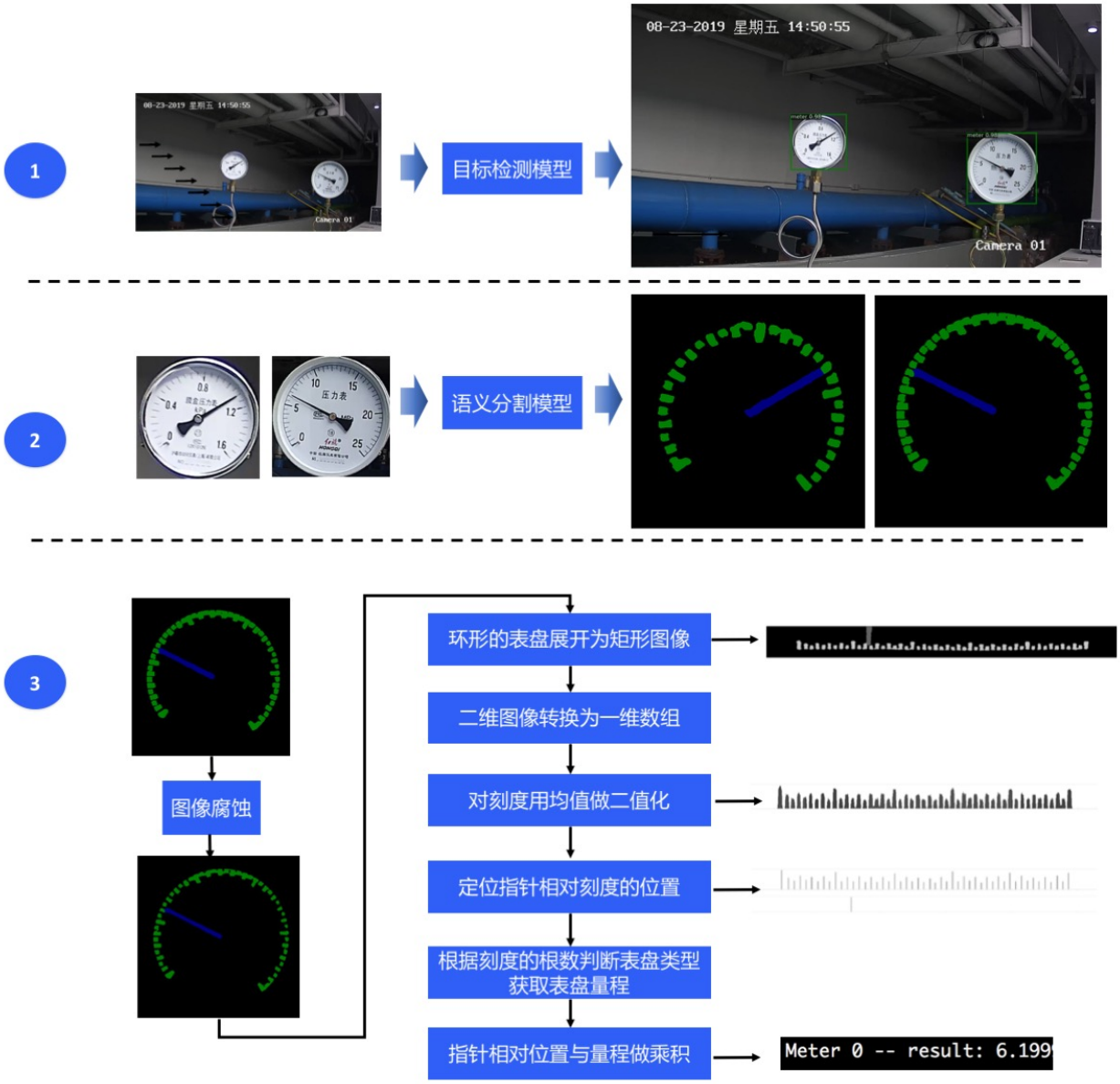
工业表计读数

本案例基于 PaddleX 实现对传统机械式指针表计的检测与自动读数功能，开放表计数据和预训练模型，并提供在 windows 系统的服务器端以及 linux 系统的 jetson 嵌入式设备上的部署指南。

18.1 读数流程

表计读数共分为三个步骤完成：

- 第一步，使用目标检测模型检测出图像中的表计
- 第二步，使用语义分割模型将各表计的指针和刻度分割出来
- 第三步，根据指针的相对位置和预知的量程计算出各表计的读数



MeterReader_A

- **表计检测**：由于本案例中没有面积较小的表计，所以目标检测模型选择性能更优的 **YOLOv3**。考虑到本案例主要在有 GPU 的设备上部署，所以骨干网路选择精度更高的 **DarkNet53**。
- **刻度和指针分割**：考虑到刻度和指针均为细小区域，语义分割模型选择效果更好的 **DeepLapv3**。
- **读数后处理**：首先，对语义分割的预测类别图进行图像腐蚀操作，以达到刻度细分的目的。然后把环形的表盘展开为矩形图像，根据图像中类别信息生成一维的刻度数组和一维的指针数组。接着计算刻度数组的均值，用均值对刻度数组进行二值化操作。最后定位出指针相对刻度的位置，根据刻度的根数判断表盘的类型以此获取表盘的量程，将指针相对位置与量程做乘积得到表盘的读数。

18.2 表计数据和预训练模型

本案例开放了表计测试图片，用于体验表计读数的预测推理全流程。还开放了表计检测数据集、指针和刻度分割数据集，用户可以使用这些数据集重新训练模型。

本案例开放了预先训练好的检测模型和语义分割模型，可以使用这些模型快速体验表计读数全流程，也可以直接将这些模型部署在服务器端或 jetson 嵌入式设备上进行推理预测。

18.3 快速体验表盘读数

可以使用本案例提供的预训练模型快速体验表计读数的自动预测全流程。如果不需要预训练模型，可以跳转至小节模型训练重新训练模型。

18.3.1 前置依赖

- Paddle paddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

安装的相关问题参考*PaddleX 安装*

18.3.2 测试表计读数

step 1. 下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

step 2. 预测执行文件位于 `PaddleX/examples/meter_reader/`，进入该目录:

```
cd PaddleX/examples/meter_reader/
```

预测执行文件为 `reader_infer.py`，其主要参数说明如下:

step 3. 预测

若要使用 GPU，则指定 GPU 卡号（以 0 号卡为例）:

```
export CUDA_VISIBLE_DEVICES=0
```

若不使用 GPU，则将 CUDA_VISIBLE_DEVICES 指定为空:

```
export CUDA_VISIBLE_DEVICES=
```

- 预测单张图片

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/
↪to/seg_inference_model --image /path/to/meter_test/20190822_168.jpg --save_dir ./
↪output --use_erode
```

- 预测多张图片

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/
↪to/seg_inference_model --image_dir /path/to/meter_test --save_dir ./output --use_erode
```

- 开启摄像头预测

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/
↪to/seg_inference_model --save_dir ./output --use_erode --use_camera
```

18.4 推理部署

18.4.1 Windows 系统的服务器端安全部署

c++ 部署

step 1. 下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

step 2. 将 `PaddleX\examples\meter_reader\deploy\cpp` 下的 `meter_reader` 文件夹和 `CMakeList.txt` 拷贝至 `PaddleX\deploy\cpp` 目录下，拷贝之前可以将 `PaddleX\deploy\cpp` 下原本的 `CMakeList.txt` 做好备份。

step 3. 按照 *Windows 平台部署*中的 Step2 至 Step4 完成 C++ 预测代码的编译。

step 4. 编译成功后，可执行文件在 `out\build\x64-Release` 目录下，打开 `cmd`，并切换到该目录:

```
cd PaddleX\deploy\cpp\out\build\x64-Release
```

预测程序为 paddle_inference\meter_reader.exe，其主要命令参数说明如下:

step 5. 推理预测:

用于部署推理的模型应为 inference 格式，本案例提供的预训练模型均为 inference 格式，如若是重新训练的模型，需参考部署模型导出将模型导出为 inference 格式。

- 使用未加密的模型对单张图片做预测

```
.\paddlex_inference\meter_reader.exe --det_model_dir=\path\to\det_inference_model --seg_
↪model_dir=\path\to\seg_inference_model --image=\path\to\meter_test\20190822_168.jpg --
↪use_gpu=1 --use_erode=1 --save_dir=output
```

- 使用未加密的模型对图像列表做预测图像列表 image_list.txt 内容的格式如下，因绝对路径不同，暂未提供该文件，用户可根据实际情况自行生成：

```
\path\to\images\1.jpg
\path\to\images\2.jpg
...
\path\to\images\n.jpg
```

```
.\paddlex_inference\meter_reader.exe --det_model_dir=\path\to\det_inference_model --seg_
↪model_dir=\path\to\seg_inference_model --image_list=\path\to\meter_test\image_list.txt␣
↪--use_gpu=1 --use_erode=1 --save_dir=output
```

- 使用未加密的模型开启摄像头做预测

```
.\paddlex_inference\meter_reader.exe --det_model_dir=\path\to\det_inference_model --seg_
↪model_dir=\path\to\seg_inference_model --use_camera=1 --use_gpu=1 --use_erode=1 --save_
↪dir=output
```

- 使用加密后的模型对单张图片做预测

如果未对模型进行加密，请参考加密 PaddleX 模型对模型进行加密。例如加密后的检测模型所在目录为\path\to\encrypted_det_inference_model，密钥为 yEBLDiBOdlj+5EsNNrABhfDuQGkdcreYcHcncqwdbx0=；加密后的分割模型所在目录为\path\to\encrypted_seg_inference_model，密钥为 DbVS64I9pFRo5XmQ8MNV2kSGsfEr4FKA6OH9OUhRrsY=

```
.\paddlex_inference\meter_reader.exe --det_model_dir=\path\to\encrypted_det_inference_
↪model --seg_model_dir=\path\to\encrypted_seg_inference_model --image=\path\to\test.jpg␣
↪--use_gpu=1 --use_erode=1 --save_dir=output --det_key␣
↪yEBLDiBOdlj+5EsNNrABhfDuQGkdcreYcHcncqwdbx0= --seg_key␣
↪DbVS64I9pFRo5XmQ8MNV2kSGsfEr4FKA6OH9OUhRrsY=
```

18.4.2 Linux 系统的 jetson 嵌入式设备安全部署

c++ 部署

step 1. 下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

step 2. 将 PaddleX/examples/meter_reader/deploy/cpp 下的 meter_reader 文件夹和 CMakeList.txt 拷贝至 PaddleX/deploy/cpp 目录下，拷贝之前可以将 PaddleX/deploy/cpp 下原本的 CMakeList.txt 做好备份。

step 3. 按照Nvidia Jetson 开发板部署中的 Step2 至 Step3 完成 C++ 预测代码的编译。

step 4. 编译成功后，可执行程为 build/meter_reader/meter_reader，其主要命令参数说明如下:

step 5. 推理预测:

用于部署推理的模型应为 inference 格式，本案例提供的预训练模型均为 inference 格式，如若是重新训练的模型，需参考部署模型导出将模型导出为 inference 格式。

- 使用未加密的模型对单张图片做预测

```
./build/meter_reader/meter_reader --det_model_dir=/path/to/det_inference_model --seg_
↪model_dir=/path/to/seg_inference_model --image=/path/to/meter_test/20190822_168.jpg --
↪use_gpu=1 --use_erode=1 --save_dir=output
```

- 使用未加密的模型对图像列表做预测图像列表 image_list.txt 内容的格式如下，因绝对路径不同，暂未提供该文件，用户可根据实际情况自行生成:

```
\path\to\images\1.jpg
\path\to\images\2.jpg
...
\path\to\images\n.jpg
```

```
./build/meter_reader/meter_reader --det_model_dir=/path/to/det_inference_model --seg_
↪model_dir=/path/to/seg_inference_model --image_list=/path/to/image_list.txt --use_
↪gpu=1 --use_erode=1 --save_dir=output
```

- 使用未加密的模型开启摄像头做预测

```
./build/meter_reader/meter_reader --det_model_dir=/path/to/det_inference_model --seg_
↪model_dir=/path/to/seg_inference_model --use_camera=1 --use_gpu=1 --use_erode=1 --save_
↪dir=output
```

18.5 模型训练

18.5.1 前置依赖

- Paddle paddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

安装的相关问题参考*PaddleX 安装*

18.5.2 训练

- 表盘检测的训练

```
python /path/to/PaddleX/examples/meter_reader/train_detection.py
```

- 指针和刻度分割的训练

```
python /path/to/PaddleX/examples/meter_reader/train_segmentation.py
```

运行以上脚本可以训练本案例的检测模型和分割模型。如果不需要本案例的数据和模型参数，可更换数据，选择合适的模型并调整训练参数。

CHAPTER 19

人像分割模型

本教程基于 PaddleX 核心分割模型实现人像分割，开放预训练模型和测试数据、支持视频流人像分割、提供模型 Fine-tune 到 Paddle Lite 移动端及 Nvidia Jeston 嵌入式设备部署的全流程应用指南。

19.1 预训练模型和测试数据

19.1.1 预训练模型

本案例开放了两个在大规模人像数据集上训练好的模型，以满足服务器端场景和移动端场景的需求。使用这些模型可以快速体验视频流人像分割，也可以部署到移动端或嵌入式设备进行实时人像分割，也可以用于完成模型 Fine-tuning。

- Checkpoint Parameter 为模型权重，用于 Fine-tuning 场景，包含 `__params__` 模型参数和 `model.yaml` 基础的模型配置信息。
- Inference Model 和 Quant Inference Model 为预测部署模型，包含 `__model__` 计算图结构、`__params__` 模型参数和 `model.yaml` 基础的模型配置信息。
- 其中 Inference Model 适用于服务端的 CPU 和 GPU 预测部署，Qunat Inference Model 为量化版本，适用于通过 Paddle Lite 进行移动端等端侧设备部署。

19.1.2 关于预测锯齿问题

在训练完模型后，可能会遇到预测出来结果存在『锯齿』的问题，这个可能存在的原因是由于模型在预测过程中，经历了原图缩放再放大的过程，如下流程所示，

```
原图输入 -> 预处理 transforms 将图像缩放至目标大小 -> Paddle 模型预测 -> 预测结果放大至原图大小
```

对于这种原因导致的问题，可以手动修改模型中的 model.yml 文件，将预处理中的目标大小**调整到更高**优化此问题，如在本文档中提供的人像分割 server 端模型中 model.yml 文件内容，修改 target_size 至 1024*1024（这样也会带来模型预测所需的资源更多，预测速度更慢）

```
Model: DeepLabv3p
Transforms:
- Resize:
    interp: LINEAR
    target_size:
    - 512
    - 512
```

修改为

```
Model: DeepLabv3p
Transforms:
- Resize:
    interp: LINEAR
    target_size:
    - 1024
    - 1024
```

预训练模型的存储大小和推理时长如下所示，其中移动端模型的运行环境为 cpu：骁龙 855，内存：6GB，图片大小：192*192

执行以下脚本下载全部的预训练模型：

- 下载 PaddleX 源码：

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

- 下载预训练模型的代码位于 PaddleX/examples/human_segmentation，进入该目录：

```
cd PaddleX/examples/human_segmentation
```

- 执行下载

```
python pretrain_weights/download_pretrain_weights.py
```

19.1.3 测试数据

supervise.ly发布了人像分割数据集 **Supervisely Persons**，本案例从中随机抽取一小部分数据并转化成 PaddleX 可直接加载的数据格式，运行以下代码可下载该数据、以及手机前置摄像头拍摄的人像测试视频 `video_test.mp4`.

- 下载测试数据的代码位于 `PaddleX/xamples/human_segmentation`，进入该目录并执行下载：

```
python data/download_data.py
```

19.2 快速体验视频流人像分割

19.2.1 前置依赖

- PaddlePaddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

安装的相关问题参考PaddleX 安装

- 下载 PaddleX 源码：

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

- 视频流人像分割和背景替换的执行文件均位于 `PaddleX/examples/human_segmentation`，进入该目录：

```
cd PaddleX/examples/human_segmentation
```

19.2.2 光流跟踪辅助的视频流人像分割

本案例将 DIS（Dense Inverse Search-basedmethod）光流跟踪算法的预测结果与 PaddleX 的分割结果进行融合，以此改善视频流人像分割的效果。运行以下代码进行体验，以下代码位于 `PaddleX/xamples/human_segmentation`：

- 通过电脑摄像头进行实时分割处理

```
python video_infer.py --model_dir pretrain_weights/humanseg_mobile_inference
```

- 对离线人像视频进行分割处理

```
python video_infer.py --model_dir pretrain_weights/humanseg_mobile_inference --video_
↪path data/video_test.mp4
```

视频分割结果如下所示：

19.2.3 人像背景替换

本案例还实现了人像背景替换功能，根据所选背景对人像的背景画面进行替换，背景可以是一张图片，也可以是一段视频。人像背景替换的代码位于 `PaddleX/xamples/human_segmentation`，进入该目录并执行：

- 通过电脑摄像头进行实时背景替换处理, 通过’–background_video_path’传入背景视频

```
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --background_
↪image_path data/background.jpg
```

- 对人像视频进行背景替换处理, 通过’–background_video_path’传入背景视频

```
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --video_path␣
↪data/video_test.mp4 --background_image_path data/background.jpg
```

- 对单张图像进行背景替换

```
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --image_path␣
↪data/human_image.jpg --background_image_path data/background.jpg
```

背景替换结果如下：

注意:

- 视频分割处理时间需要几分钟，请耐心等待。
- 提供的模型适用于手机摄像头竖屏拍摄场景，宽屏效果会略差一些。

19.3 模型 Fine-tune

19.3.1 前置依赖

- PaddlePaddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

安装的相关问题参考PaddleX 安装

- 下载 PaddleX 源码：

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

- 人像分割训练、评估、预测、模型导出、离线量化的执行文件均位于 PaddleX/examples/human_segmentation，进入该目录：

```
cd PaddleX/examples/human_segmentation
```

19.3.2 模型训练

使用下述命令进行基于预训练模型的模型训练，请确保选用的模型结构 model_type 与模型参数 pretrain_weights 匹配。如果不需要本案例提供的测试数据，可更换数据、选择合适的模型并调整训练参数。

```
# 指定 GPU 卡号（以 0 号卡为例）
export CUDA_VISIBLE_DEVICES=0
# 若不使用 GPU，则将 CUDA_VISIBLE_DEVICES 指定为空
# export CUDA_VISIBLE_DEVICES=
python train.py --model_type HumanSegMobile \
--save_dir output/ \
--data_dir data/mini_supervisely \
--train_list data/mini_supervisely/train.txt \
--val_list data/mini_supervisely/val.txt \
--pretrain_weights pretrain_weights/humanseg_mobile_params \
--batch_size 8 \
--learning_rate 0.001 \
--num_epochs 10 \
--image_shape 192 192
```

其中参数含义如下：

- --model_type: 模型类型，可选项为：HumanSegServer 和 HumanSegMobile
- --save_dir: 模型保存路径
- --data_dir: 数据集路径
- --train_list: 训练集列表路径
- --val_list: 验证集列表路径
- --pretrain_weights: 预训练模型路径
- --batch_size: 批大小

- --learning_rate: 初始学习率
- --num_epochs: 训练轮数
- --image_shape: 网络输入图像大小（w, h）

更多命令行帮助可运行下述命令进行查看：

```
python train.py --help
```

注意：可以通过更换--model_type 变量与对应的--pretrain_weights 使用不同的模型快速尝试。

19.3.3 评估

使用下述命令对模型在验证集上的精度进行评估：

```
python eval.py --model_dir output/best_model \
--data_dir data/mini_supervisely \
--val_list data/mini_supervisely/val.txt \
--image_shape 192 192
```

其中参数含义如下：

- --model_dir: 模型路径
- --data_dir: 数据集路径
- --val_list: 验证集列表路径
- --image_shape: 网络输入图像大小（w, h）

19.3.4 预测

使用下述命令对测试集进行预测，预测可视化结果默认保存在./output/result/文件夹中。

```
python infer.py --model_dir output/best_model \
--data_dir data/mini_supervisely \
--test_list data/mini_supervisely/test.txt \
--save_dir output/result \
--image_shape 192 192
```

其中参数含义如下：

- --model_dir: 模型路径
- --data_dir: 数据集路径
- --test_list: 测试集列表路径

- --image_shape: 网络输入图像大小（w, h）

19.3.5 模型导出

在服务端部署的模型需要首先将模型导出为 inference 格式模型，导出的模型将包括 __model__、__params__ 和 model.yml 三个文名，分别为模型的网络结构，模型权重和模型的配置文件（包括数据预处理参数等等）。在安装完 PaddleX 后，在命令行终端使用如下命令完成模型导出：

```
paddlex --export_inference --model_dir output/best_model \
--save_dir output/export
```

其中参数含义如下：

- --model_dir: 模型路径
- --save_dir: 导出模型保存路径

19.3.6 离线量化

```
python quant_offline.py --model_dir output/best_model \
--data_dir data/mini_supervisely \
--quant_list data/mini_supervisely/val.txt \
--save_dir output/quant_offline \
--image_shape 192 192
```

其中参数含义如下：

- --model_dir: 待量化模型路径
- --data_dir: 数据集路径
- --quant_list: 量化数据集列表路径，一般直接选择训练集或验证集
- --save_dir: 量化模型保存路径
- --image_shape: 网络输入图像大小（w, h）

19.4 推理部署

19.4.1 Paddle Lite 移动端部署

本案例将人像分割模型在移动端进行部署，部署流程展示如下，通用的移动端部署流程参见Paddle Lite 移动端部署。

1. 将 PaddleX 模型导出为 inference 模型

本案例使用 humanseg_mobile_quant 预训练模型，该模型已经是 inference 模型，不需要再执行模型导出步骤。如果不使用预训练模型，则执行上一章节模型训练中的模型导出将自己训练的模型导出为 inference 格式。

2. 将 inference 模型优化为 Paddle Lite 模型

下载并解压 模型优化工具 opt，进入模型优化工具 opt 所在路径后，执行以下命令：

```
./opt --model_file=<model_path> \
      --param_file=<param_path> \
      --valid_targets=arm \
      --optimize_out_type=naive_buffer \
      --optimize_out=model_output_name
```

更详细的使用方法和参数含义请参考: 使用 opt 转化模型

3. 移动端预测

PaddleX 提供了基于 PaddleX Android SDK 的安卓 demo，可供用户体验图像分类、目标检测、实例分割和语义分割，该 demo 位于 `PaddleX/deploy/lite/android/demo`，用户将模型、配置文件和测试图片拷贝至该 demo 下进行预测。

3.1 前置依赖

- Android Studio 3.4
- Android 手机或开发板

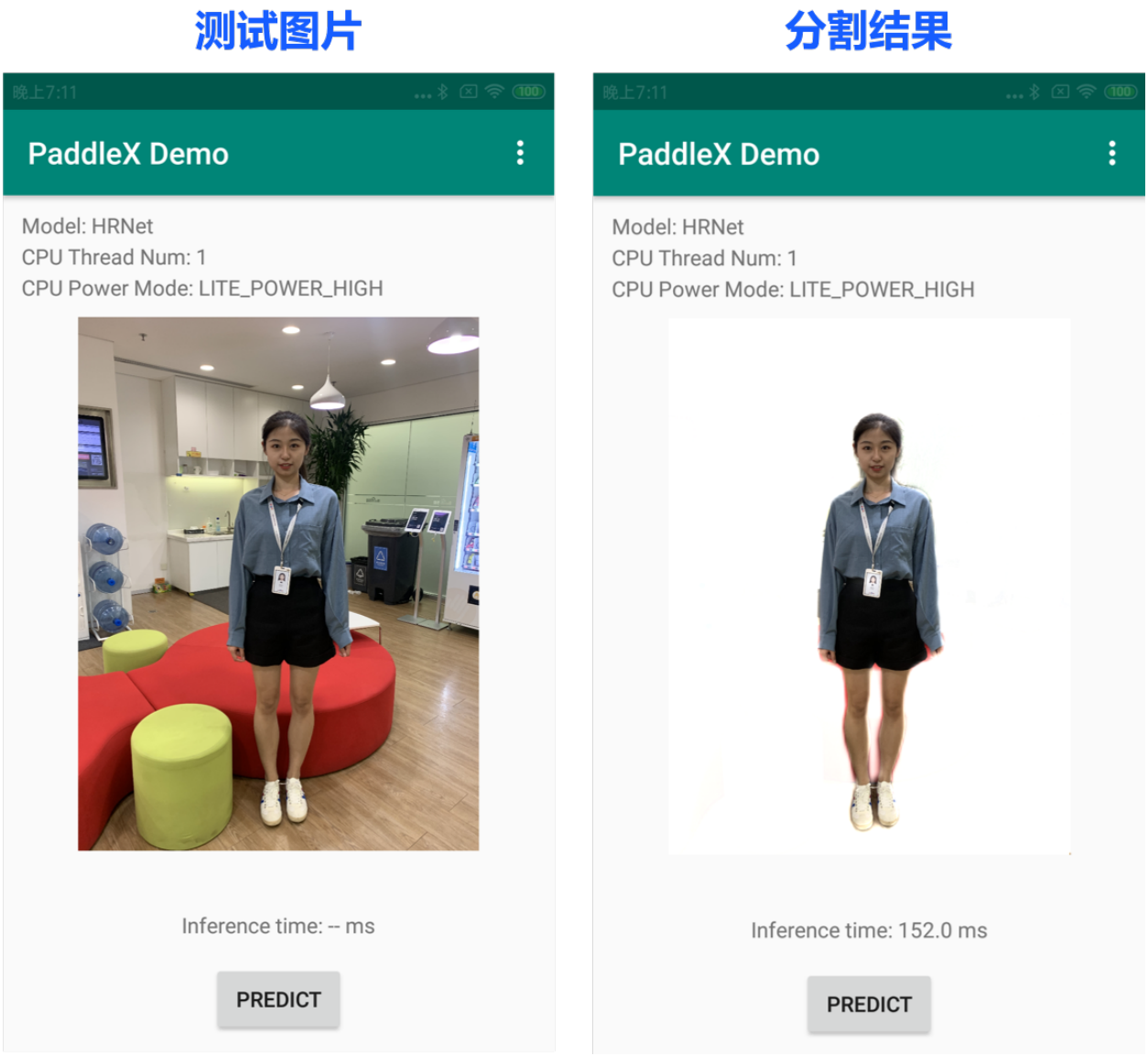
3.2 拷贝模型、配置文件和测试图片

- 将 Lite 模型(.nb 文件)拷贝到 `PaddleX/deploy/lite/android/demo/app/src/main/assets/model/` 目录下，根据.nb 文件的名字，修改文件 `PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml` 中的 `MODEL_PATH_DEFAULT`;
- 将配置文件（.yml 文件）拷贝到 `PaddleX/deploy/lite/android/demo/app/src/main/assets/config/`目录下，根据.yml 文件的名字，修改文件 `PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml` 中的 `YAML_PATH_DEFAULT`;
- 将测试图片拷贝到 `PaddleX/deploy/lite/android/demo/app/src/main/assets/images/`目录下，根据图片文件的名字，修改文件 `PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml` 中的 `IMAGE_PATH_DEFAULT`。

3.3 导入工程并运行

- 打开 Android Studio，在” Welcome to Android Studio” 窗口点击” Open an existing Android Studio project”，在弹出的路径选择窗口中进入 `PaddleX/deploy/lite/android/demo` 目录，然后点击右下角的” Open” 按钮，导入工程；
- 通过 USB 连接 Android 手机或开发板；
- 工程编译完成后，点击菜单栏的 Run->Run ‘App’ 按钮，在弹出的” Select Deployment Target” 窗口选择已经连接的 Android 设备，然后点击” OK” 按钮；
- 运行成功后，Android 设备将加载一个名为 PaddleX Demo 的 App，默认会加载一个测试图片，同时还支持拍照和从图库选择照片进行预测。

测试图片及其分割结果如下所示：



19.4.2 Nvidia Jetson 嵌入式设备部署

c++ 部署

step 1. 下载 PaddleX 源码

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

step 2. 将 PaddleX/examples/human_segmentation/deploy/cpp 下的 human_segmenter.cpp 和 CMakeList.txt 拷贝至 PaddleX/deploy/cpp 目录下，拷贝之前可以将 PaddleX/deploy/cpp 下原本的 CMakeList.txt 做好备份。

step 3. 按照Nvidia Jetson 开发板部署中的 Step2 至 Step3 完成 C++ 预测代码的编译。

step 4. 编译成功后，可执行程为 build/human_segmenter，其主要命令参数说明如下：

step 5. 推理预测

用于部署推理的模型应为 inference 格式，本案例使用 humanseg_server_inference 预训练模型，该模型已经是 inference 模型，不需要再执行模型导出步骤。如果不使用预训练模型，则执行第 2 章节模型训练中的模型导出将自己训练的模型导出为 inference 格式。

- 使用未加密的模型对单张图片做预测

待测试图片位于本案例提供的测试数据中，可以替换成自己的图片。

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --image=/path/to/
↪data/mini_supervisely/Images/pexels-photo-63776.png --use_gpu=1 --save_dir=output
```

- 使用未加密的模型开启摄像头做预测

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --use_camera=1 --
↪save_result=1 --use_gpu=1 --save_dir=output
```

- 使用未加密的模型对视频文件做预测

待测试视频文件位于本案例提供的测试数据中，可以替换成自己的视频文件。

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --video_path=/
↪path/to/data/mini_supervisely/video_test.mp4  --save_result=1 --use_gpu=1 --save_
↪dir=output
```

CHAPTER 20

RGB 遥感影像分割

本案例基于 PaddleX 实现遥感影像分割，提供滑动窗口预测方式，以避免在直接对大尺寸图片进行预测时显存不足的发生。此外，滑动窗口之间的重叠程度可配置，以此消除最终预测结果中各窗口拼接处的裂痕感。

20.1 前置依赖

- Paddle paddle >= 1.8.4
- Python >= 3.5
- PaddleX >= 1.1.4

安装的相关问题参考*PaddleX 安装*

下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

该案例所有脚本均位于 `PaddleX/examples/remote_sensing/`，进入该目录:

```
cd PaddleX/examples/remote_sensing/
```

20.2 数据准备

本案例使用 2015 CCF 大数据比赛提供的高清遥感影像，包含 5 张带标注的 RGB 图像，图像尺寸最大有 7969×7939、最小有 4011×2470。该数据集共标注了 5 类物体，分别是背景（标记为 0）、植被（标记为 1）、道路（标记为 2）、建筑（标记为 3）、水体（标记为 4）。

本案例将前 4 张图片划分入训练集，第 5 张图片作为验证集。为增加训练时的批量大小，以滑动窗口为 (1024, 1024)、步长为 (512, 512) 对前 4 张图片进行切分，加上原本的 4 张大尺寸图片，训练集一共有 688 张图片。在训练过程中直接对大图片进行验证会导致显存不足，为避免此类问题的出现，针对验证集以滑动窗口为 (769, 769)、步长为 (769，769) 对第 5 张图片进行切分，得到 40 张子图片。

运行以下脚本，下载原始数据集，并完成数据集的切分：

```
python prepare_data.py
```

20.3 模型训练

分割模型选择 Backbone 为 MobileNetv3_large_ssld 的 Deeplabv3 模型，该模型兼备高性能高精度的优点。运行以下脚本，进行模型训练：

```
python train.py
```

也可以跳过模型训练步骤，直接下载预训练模型进行后续的模型预测和评估：

```
wget https://bj.bcebos.com/paddlex/examples/remote_sensing/models/ccf_remote_model.tar.gz
tar -xvf ccf_remote_model.tar.gz
```

20.4 模型预测

直接对大尺寸图片进行预测会导致显存不足，为避免此类问题的出现，本案例提供了滑动窗口预测接口，支持有重叠和无重叠两种方式。

- 无重叠的滑动窗口预测

在输入图片上以固定大小的窗口滑动，分别对每个窗口下的图像进行预测，最后将各窗口的预测结果拼接成输入图片的预测结果。由于每个窗口边缘部分的预测效果会比中间部分的差，因此每个窗口拼接处可能会有明显的裂痕感。

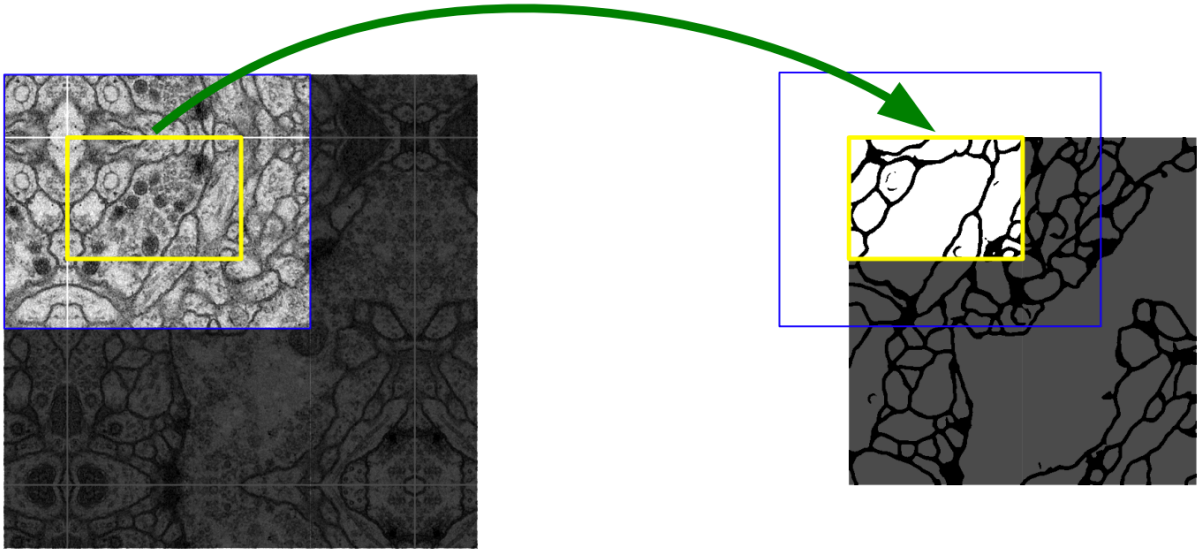
该预测方式的 API 接口详见overlap_tile_predict，**使用时需要把参数 `pad_size` 设置为 `[0, 0]`**。

- 有重叠的滑动窗口预测

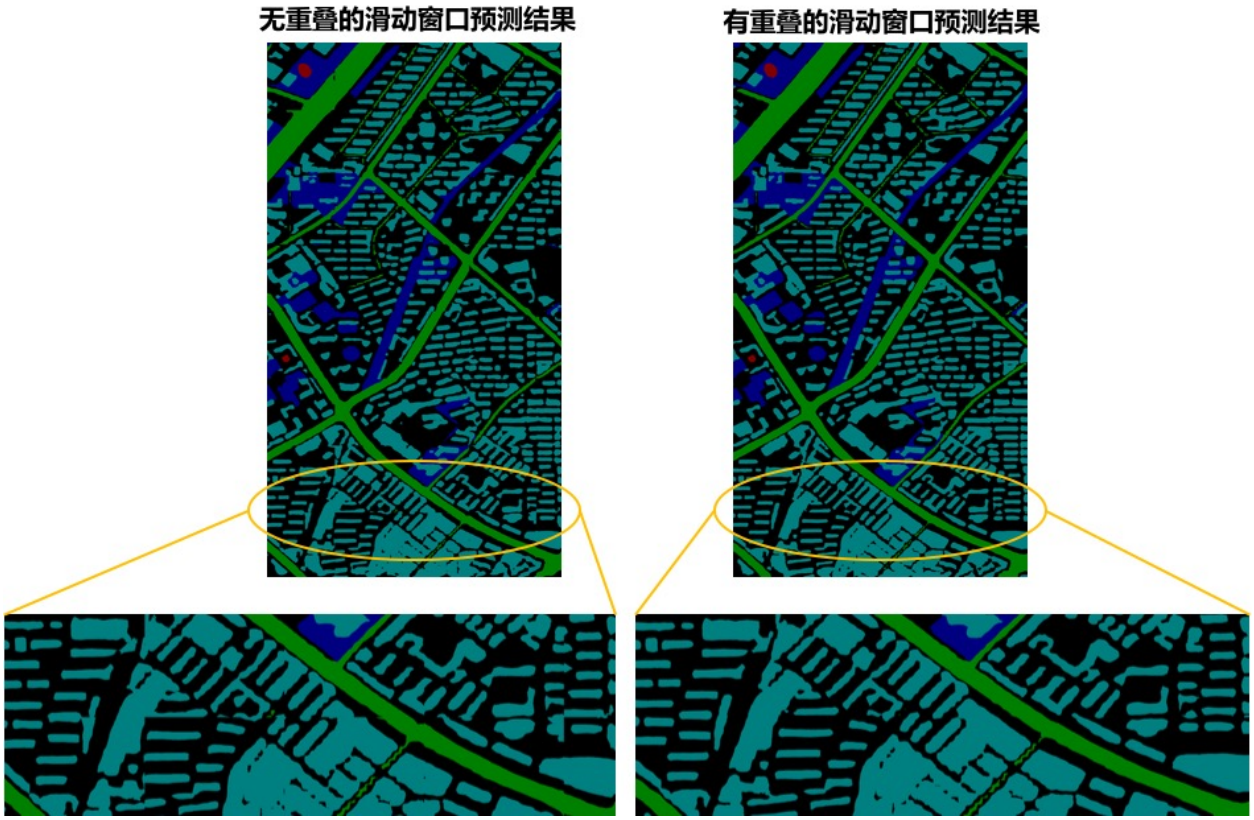
在 Unet 论文中，作者提出一种有重叠的滑动窗口预测策略（Overlap-tile strategy）来消除拼接处的裂痕感。对各滑动窗口预测时，会向四周扩展一定的面积，对扩展后的窗口进行预测，例如下图中的蓝色部分区域，

到拼接时只取各窗口中间部分的预测结果，例如下图中的黄色部分区域。位于输入图像边缘处的窗口，其扩展面积下的像素则通过将边缘部分像素镜像填补得到。

该预测方式的 API 接口说明详见overlap_tile_predict。



相比无重叠的滑动窗口预测，有重叠的滑动窗口预测策略将本案例的模型精度 miou 从 80.58% 提升至 81.52%，并且将预测可视化结果中裂痕感显著消除，可见下图中两种预测方式的效果对比。



运行以下脚本使用有重叠的滑动窗口进行预测：

```
python predict.py
```

20.5 模型评估

在训练过程中，每隔 10 个迭代轮数会评估一次模型在验证集的精度。由于已事先将原始大尺寸图片切分成小块，此时相当于使用无重叠的大图切小图预测方式，最优模型精度 miou 为 80.58%。运行以下脚本，将采用有重叠的大图切小图的预测方式，重新评估原始大尺寸图片的模型精度，此时 miou 为 81.52%。

```
python eval.py
```

CHAPTER 21

多通道遥感影像分割

遥感影像分割是图像分割领域中的重要应用场景，广泛应用于土地测绘、环境监测、城市建设等领域。遥感影像分割的目标多种多样，有诸如积雪、农作物、道路、建筑、水源等地物目标，也有例如云层的空中目标。

本案例基于 PaddleX 实现多通道遥感影像分割，涵盖数据分析、模型训练、模型预测等流程，旨在帮助用户利用深度学习技术解决多通道遥感影像分割问题。

21.1 前置依赖

- Paddle paddle >= 1.8.4
- Python >= 3.5
- PaddleX >= 1.1.4

安装的相关问题参考*PaddleX 安装*

另外还需安装 gdal, 使用 pip 安装 gdal 可能出错，推荐使用 conda 进行安装：

```
conda install gdal
```

下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

该案例所有脚本均位于 PaddleX/examples/channel_remote_sensing/，进入该目录：

```
cd PaddleX/examples/channel_remote_sensing/
```

21.2 数据准备

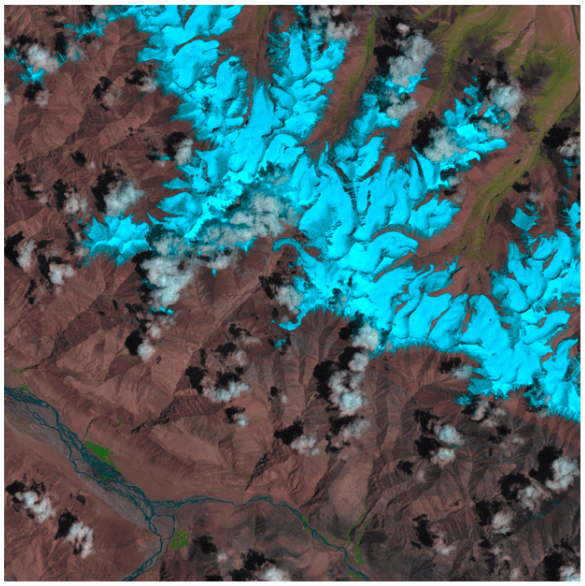
遥感影像的格式多种多样，不同传感器产生的数据格式也可能不同。PaddleX 现已兼容以下 4 种格式图片读取：

- tif
- png
- img
- npy

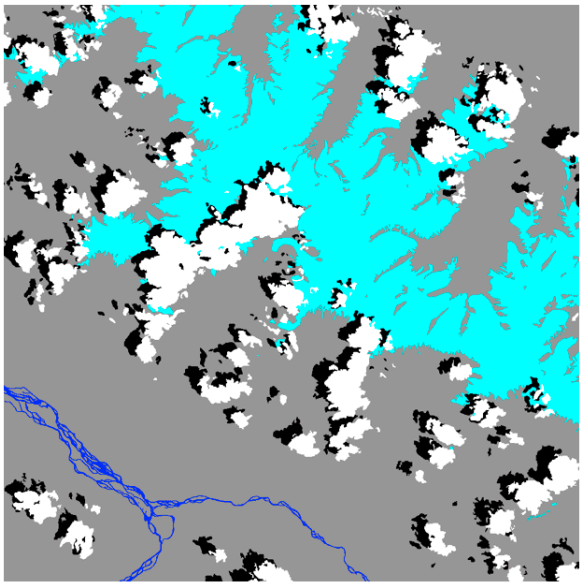
标注图要求必须为单通道的 png 格式图像，像素值即为对应的类别，像素标注类别需要从 0 开始递增。例如 0，1，2，3 表示有 4 种类别，255 用于指定不参与训练和评估的像素，标注类别最多为 256 类。

本案例使用L8 SPARCS 公开数据集进行云雪分割，该数据集包含 80 张卫星影像，涵盖 10 个波段。原始标注图片包含 7 个类别，分别是 cloud, cloud shadow, shadow over water, snow/ice, water, land 和 flooded。由于 flooded 和 shadow over water2 个类别占比仅为 1.8% 和 0.24%，我们将其进行合并，flooded 归为 land，shadow over water 归为 shadow，合并后标注包含 5 个类别。

数值、类别、颜色对应表：



彩色合成预览图



标注

执行以下命令下载并解压经过类别合并后的数据集：

```
mkdir dataset && cd dataset
wget https://paddleseg.bj.bcebos.com/dataset/remote_sensing_seg.zip
unzip remote_sensing_seg.zip
cd ..
```

其中 data 目录存放遥感影像，data_vis 目录存放彩色合成预览图，mask 目录存放标注图。

21.3 数据分析

遥感影像往往由许多波段组成，不同波段数据分布可能大相径庭，例如可见光波段和热红外波段分布十分不同。为了更深入了解数据的分布来优化模型训练效果，需要对数据进行分析。

参考文档数据分析对训练集进行统计分析，确定图像像素值的截断范围，并统计截断后的均值和方差。

21.4 模型训练

本案例选择 UNet 语义分割模型完成云雪分割，运行以下步骤完成模型训练，模型的最优精度 miou 为 78.38%。

- 设置 GPU 卡号

```
export CUDA_VISIBLE_DEVICES=0
```

- 运行以下脚本开始训练

```
python train.py --data_dir dataset/remote_sensing_seg \
--train_file_list dataset/remote_sensing_seg/train.txt \
--eval_file_list dataset/remote_sensing_seg/val.txt \
--label_list dataset/remote_sensing_seg/labels.txt \
--save_dir saved_model/remote_sensing_unet \
--num_classes 5 \
--channel 10 \
--lr 0.01 \
--clip_min_value 7172 6561 5777 5103 4291 4000 4000 4232 6934 7199 \
--clip_max_value 50000 50000 50000 50000 50000 40000 30000 18000 40000 36000 \
--mean 0.15163569 0.15142828 0.15574491 0.1716084  0.2799778  0.27652043 0.28195933 0.
↪07853807 0.56333154 0.5477584 \
--std  0.09301891 0.09818967 0.09831126 0.1057784  0.10842132 0.11062996 0.12791838 0.
↪02637859 0.0675052  0.06168227 \
--num_epochs 500 \
--train_batch_size 3
```

也可以跳过模型训练步骤，下载预训练模型直接进行模型预测：

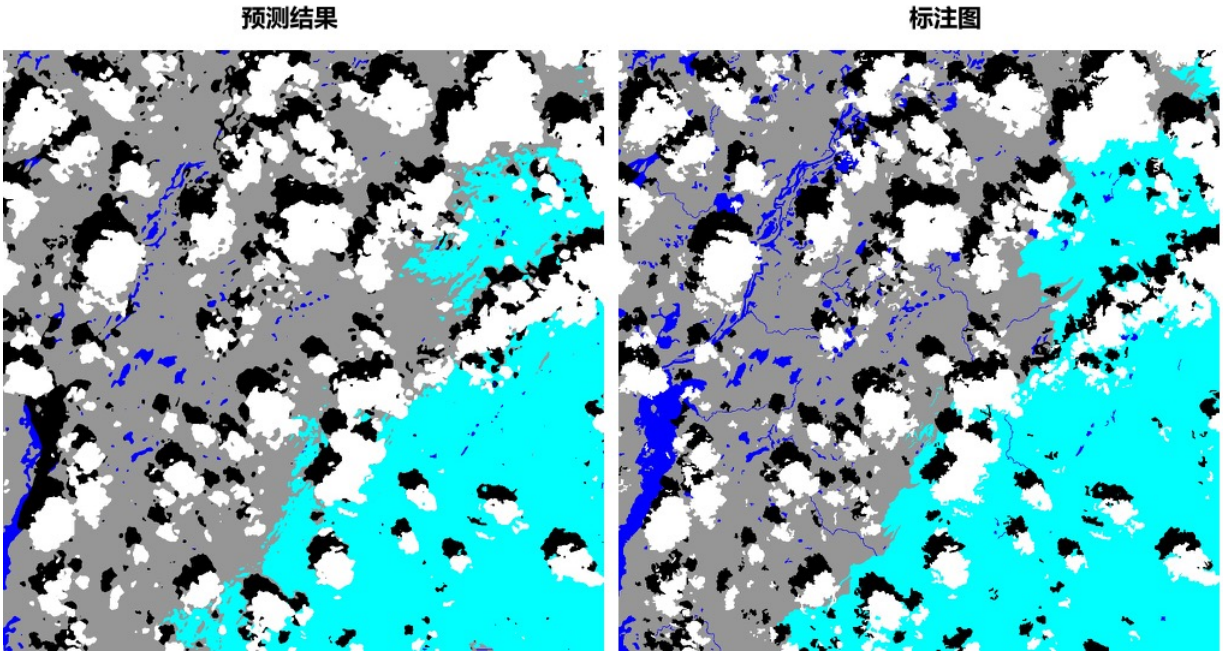
```
wget https://bj.bcebos.com/paddlex/examples/multi-channel_remote_sensing/models/l8sparcs_
↪remote_model.tar.gz
tar -xvf l8sparcs_remote_model.tar.gz
```

21.5 模型预测

运行以下脚本，对遥感图像进行预测并可视化预测结果，相应地也将对应的标注文件进行可视化，以比较预测效果。

```
export CUDA_VISIBLE_DEVICES=0
python predict.py
```

可视化效果如下所示:



数值、类别、颜色对应表:

CHAPTER 22

地块变化检测

本案例基于 PaddleX 实现地块变化检测，将同一地块的前期与后期两张图片进行拼接，而后输入给语义分割网络进行变化区域的预测。在训练阶段，使用随机缩放尺寸、旋转、裁剪、颜色空间扰动、水平翻转、竖直翻转多种数据增强策略。在验证和预测阶段，使用滑动窗口预测方式，以避免在直接对大尺寸图片进行预测时显存不足的发生。

22.1 前置依赖

- Paddle paddle >= 1.8.4
- Python >= 3.5
- PaddleX >= 1.2.2

安装的相关问题参考*PaddleX 安装*

下载 PaddleX 源码:

```
git clone https://github.com/PaddlePaddle/PaddleX
cd PaddleX
git checkout release/1.3
```

该案例所有脚本均位于 `PaddleX/examples/change_detection/`，进入该目录:

```
cd PaddleX/examples/change_detection/
```

22.2 数据准备

本案例使用Daifeng Peng 等人开放的Google Dataset, 该数据集涵盖了广州部分区域于 2006 年至 2019 年期间的房屋建筑物的变化情况，用于分析城市化进程。一共有 20 对高清图片，图片有红、绿、蓝三个波段，空间分辨率为 0.55m，图片大小有 1006x1168 至 4936x5224 不等。

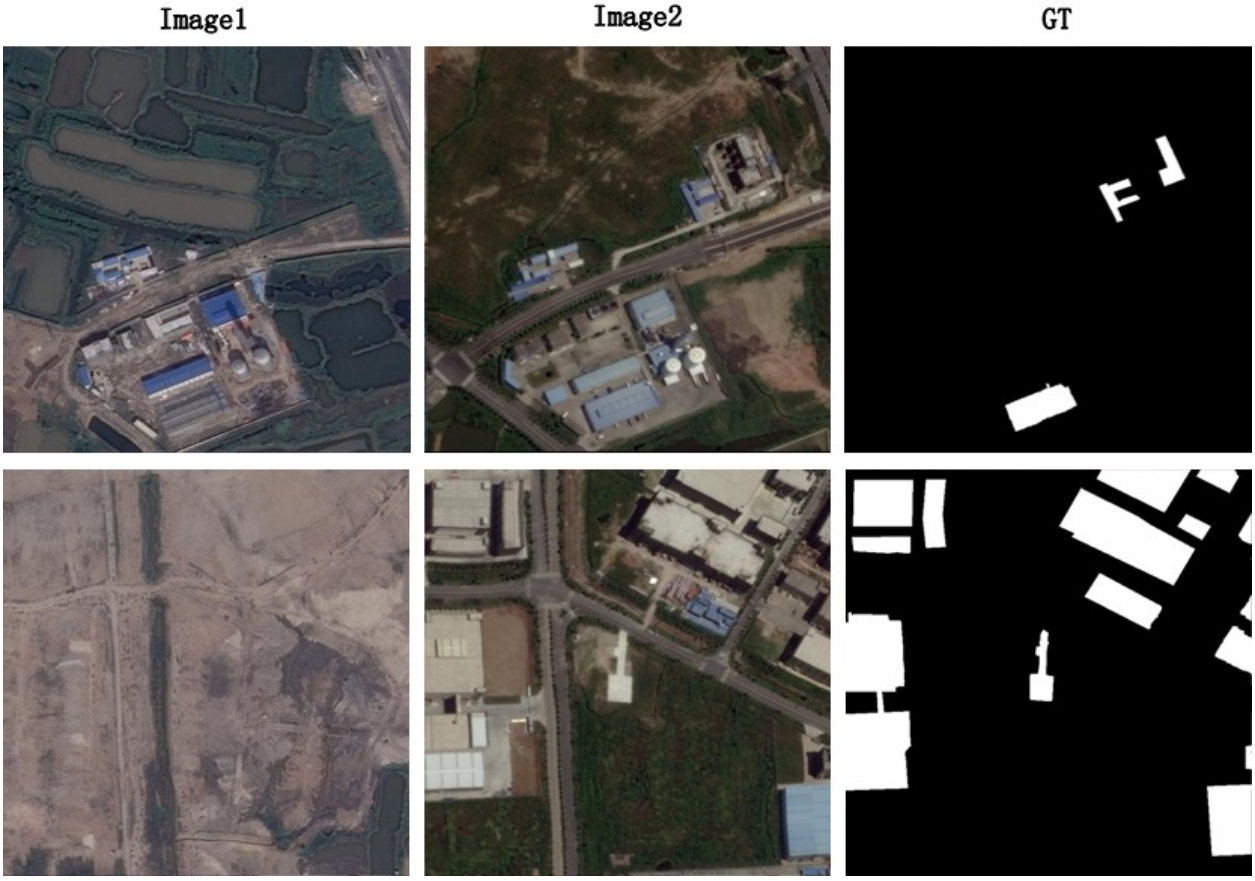
由于 Google Dataset 仅标注了房屋建筑物是否发生变化，因此本案例是二分类变化检测任务，可根据实际需求修改类别数量即可拓展为多分类变化检测。

本案例将 15 张图片划分入训练集，5 张图片划分入验证集。由于图片尺寸过大，直接训练会发生显存不足的问题，因此以滑动窗口为 (1024，1024)、步长为 (512, 512) 对训练图片进行切分，切分后的训练集一共有 743 张图片。以滑动窗口为 (769, 769)、步长为 (769，769) 对验证图片进行切分，得到 108 张子图片，用于训练过程中的验证。

运行以下脚本，下载原始数据集，并完成数据集的切分：

```
python prepare_data.py
```

切分后的数据示意如下：



注意：

- tiff 格式的图片 PaddleX 统一使用 gdal 库读取，gdal 安装可参考gdal 文档。若数据是 tiff 格式的三通道 RGB 图像，如果不想安装 gdal，需自行转成 jpeg、bmp、png 格式图片。
- label 文件需为单通道的 png 格式图片，且标注从 0 开始计数，标注 255 表示该类别不参与计算。例如本案例中，0 表示 `unchanged` 类，1 表示 `changed` 类。

22.3 模型训练

由于数据量较小，分割模型选择较好兼顾浅层细节信息和深层语义信息的 UNet 模型。运行以下脚本，进行模型训练：

```
python train.py
```

本案例使用 0,1,2,3 号 GPU 卡完成训练，可根据实际显存大小更改训练脚本中的 GPU 卡数量和 `train_batch_size` 的设置值，按 `train_batch_size` 的调整比例相应地调整学习率 `learning_rate`，例如 `train_batch_size` 由 16 减少至 8 时，`learning_rate` 则由 0.1 减少至 0.05。此外，不同数据集上能获得最优精度所对应 `learning_rate` 可能有所不同，可以尝试调整。

也可以跳过模型训练步骤，直接下载预训练模型进行后续的模型评估和预测：

```
wget https://bj.bcebos.com/paddlex/examples/change_detection/models/google_change_det_
↪model.tar.gz
tar -xvf google_change_det_model.tar.gz
```

22.4 模型评估

在训练过程中，每隔 10 个迭代轮数会评估一次模型在验证集的精度。由于已事先将原始大尺寸图片切分成小块，相当于使用无重叠的滑动窗口预测方式，最优模型精度:

category 分别对应 `unchanged` 和 `changed` 两类。

运行以下脚本，将采用有重叠的滑动窗口预测方式，重新评估原始大尺寸图片的模型精度，此时模型精度为：

```
python eval.py
```

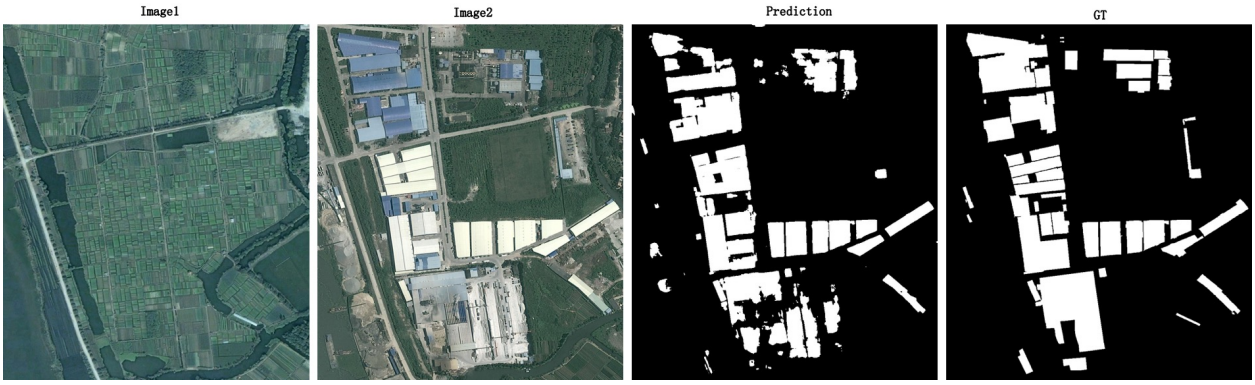
滑动窗口预测接口说明详见API 说明，已有的使用场景可参考RGB 遥感分割案例。可根据实际显存大小修改评估脚本中 `tile_size`，`pad_size` 和 `batch_size`。

22.5 模型预测

执行以下脚本，使用有重叠的滑动预测窗口对验证集进行预测。可根据实际显存大小修改评估脚本中 `tile_size`，`pad_size` 和 `batch_size`。

```
python predict.py
```

预测可视化结果如下图所示:



CHAPTER 23

工业质检

案例重构中

CHAPTER 24

介绍

PaddleX 可视化客户端基于 PaddleX 开发的可视化深度学习模型训练套件，目前支持训练视觉领域的图像分类、目标检测、实例分割和语义分割四大任务，同时支持模型裁剪、模型量化两种方式压缩模型。开发者以点选、键入的方式快速体验深度学习模型开发的全流程。可以作为您提升深度学习模型开发效率的工具。

PaddleX GUI 当前提供 Windows，Mac，Ubuntu 三种版本一键绿色安装的方式。请至飞桨官网：https://www.paddlepaddle.org.cn/paddle/paddleX 下载您需要的版本。

24.1 功能

PaddleX 可视化客户端是 PaddleX API 的衍生品，它在集成 API 功能的基础上，额外提供了可视化分析、评估等附加功能，致力于为开发者带来极致顺畅的开发体验。其拥有以下独特的功能：

24.1.1 全流程打通

PaddleX GUI 覆盖深度学习模型开发必经的 **数据处理**、**超参配置**、**模型训练及优化**、**模型发布**全流程，无需开发一行代码，即可得到高性能深度学习推理模型。

24.1.2 数据集智能分析

详细的数据结构说明，并提供 **数据标签自动校验**。支持 **可视化数据预览**、**数据分布图表展示**、**一键数据集切分**等实用功能

24.1.3 自动超参推荐

集成飞桨团队长时间产业实践经验，根据用户选择的模型类别、骨架网络等，提供多种针对性优化的 **预训练模型**，并 **提供推荐超参配置**，可 **一键开启多种优化策略**

24.1.4 可视化模型评估

集成 **可视化分析工具：VisualDL** , 以线性图表的形式展示 acc、lr 等关键参数在训练过程中的变化趋势。提供 **混淆矩阵**等实用方法，帮助快速定位问题，加速调参。模型评估报告一键导出，方便项目复盘分析。

24.1.5 模型裁剪及量化

一键启动模型裁剪、量化，在不同阶段为开发者提供模型优化的策略，满足不同环境对模型性能的需求。

24.1.6 预训练模型管理

可对历史训练模型进行保存及管理，未进行裁剪的模型可以保存为预训练模型，在后续任务中使用。

24.1.7 可视化模型测试

客户端直接展示模型预测效果，无需上线即可进行效果评估

24.1.8 模型多端部署

点选式选择模型发布平台、格式，一键导出预测模型，并匹配完善的模型预测部署说明文档，贴心助力产业端到端项目落地

CHAPTER 25

下载安装

下载地址：https://www.paddlepaddle.org.cn/paddlex

25.1 安装方式

注意：安装/解压路径请务必在不包含中文和空格的路径下，否则会导致可能无法正确训练模型

- Windows 下载后双击后选择安装路径即可
- Mac/Ubuntu 下载后解压即可

25.2 安装推荐环境

- **操作系统**：
  - Windows 10；
  - Mac OS 10.13+；
  - Ubuntu 18.04(Ubuntu 暂只支持 18.04)；

注：处理器需为 *x86_64* 架构，支持 *MKL*。

- **训练硬件**：

- **GPU**（仅 Windows 及 Linux 系统）：推荐使用支持 CUDA 的 NVIDIA 显卡，例如：GTX 1070+ 以上性能的显卡；Windows 系统 X86_64 驱动版本 >=411.31；Linux 系统 X86_64 驱动版本 >=410.48；显存 8G 以上；
- **CPU**：PaddleX 当前支持您用本地 CPU 进行训练，但推荐使用 GPU 以获得更好的开发体验。
- **内存**：建议 8G 以上
- **硬盘空间**：建议 SSD 剩余空间 1T 以上（非必须）

注：*PaddleX* 在 *Mac OS* 系统只支持 *CPU* 训练。*Windows* 系统只支持单 *GPU* 卡训练。

CHAPTER 26

使用方法

注：如果你的系统是 *Mac OS 10.15.5* 及以上，在双击客户端 *icon* 后，需要在 *Terminal* 中执行 *sudo xattr -r -d com.apple.quarantine /Users/username/PaddleX* ，并稍等几秒来启动客户端，其中 */Users/username/PaddleX* 为您保存 *PaddleX* 的文件夹路径

26.1 准备和导入数据

第一步：准备数据在开始模型训练前，您需要根据不同的任务类型，将数据标注为相应的格式。目前 PaddleX 支持【图像分类】、【目标检测】、【语义分割】、【实例分割】四种任务类型。开发者可以参考 PaddleX 使用文档中的2. 数据准备-数据标注来进行数据标注和转换工作。如若开发者自行准备数据，请注意数据格式与 PaddleX 支持四种数据格式是否一致。

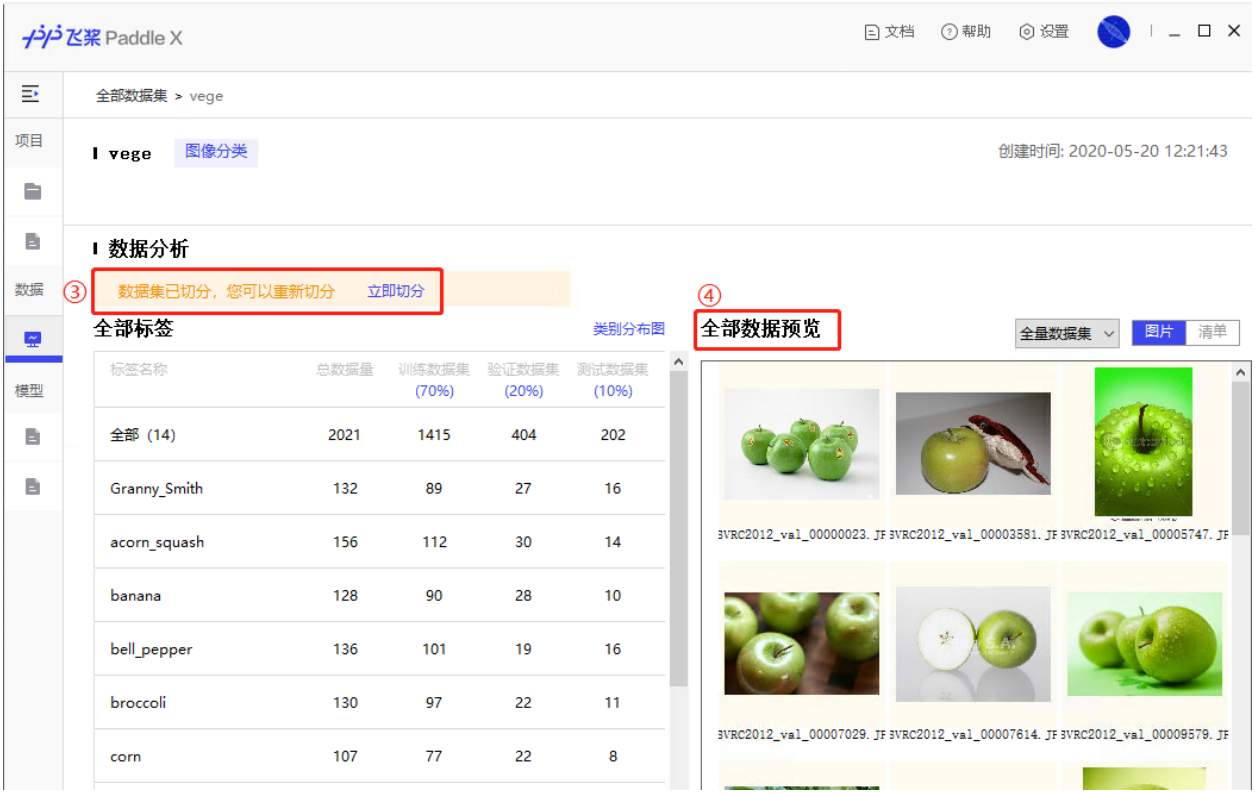
第二步：导入的据集

数据标注完成后，您需要根据不同的任务，将数据和标注文件，按照客户端提示更名并保存到正确的文件中。

在客户端新建数据集，选择与数据集匹配的任务类型，并选择数据集对应的路径，将数据集导入。

选定导入数据集后，客户端会自动校验数据及标注文件是否合规，校验成功后，您可根据实际需求，将数据集按比例划分为训练集、验证集、测试集。

您可在「数据分析」模块按规则预览您标注的数据集，双击单张图片可放大查看。

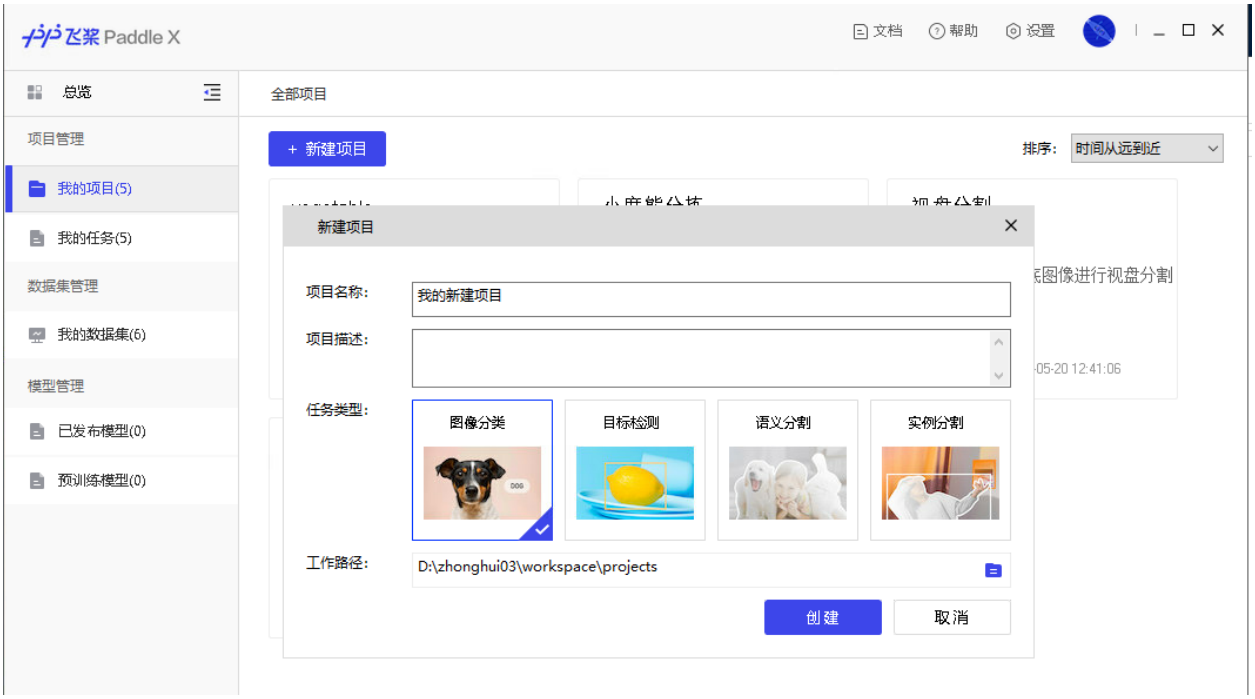


26.2 创建项目和任务

第一步：创建项目

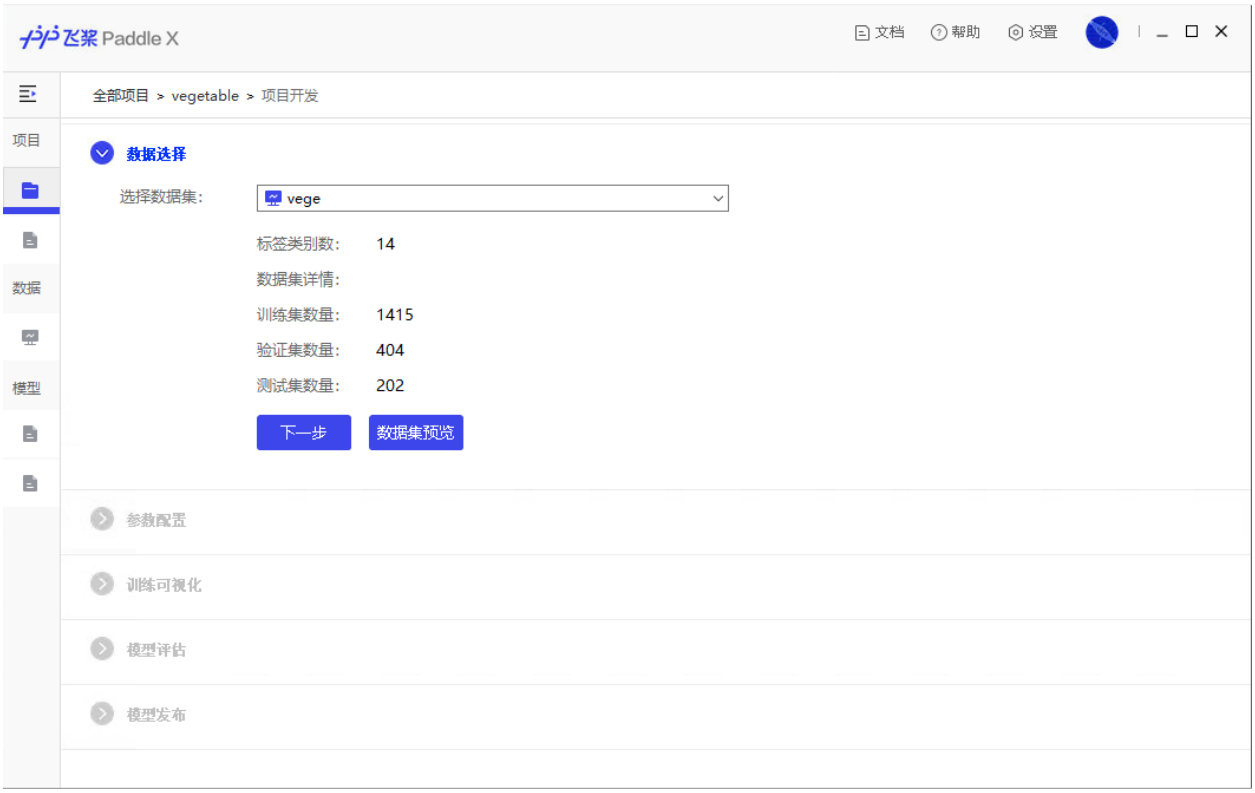
在完成数据导入后，您可以点击「新建项目」创建一个项目。

您可根据实际任务需求选择项目的任务类型，需要注意项目所采用的数据集也带有任务类型属性，两者需要进行匹配。



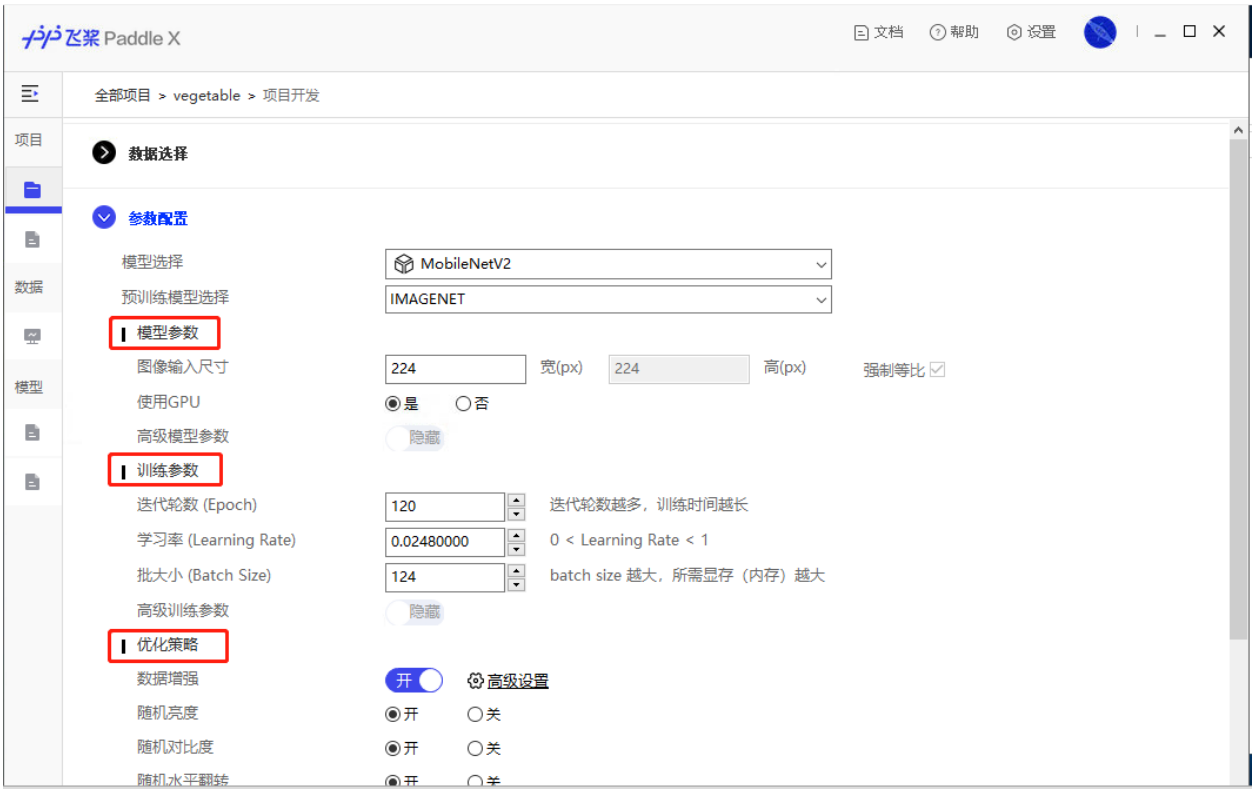
第二步：项目开发

数据选择：项目创建完成后，您需要选择已载入客户端并校验后的数据集，并点击下一步，进入参数配置页面。



参数配置：主要分为**模型参数**、**训练参数**、**优化策略**三部分。您可根据实际需求选择模型结构、骨架网络及

对应的训练参数、优化策略，使得任务效果最佳。



26.3 任务模型训练

参数配置完成后，点击启动训练，模型开始训练并进行效果评估。

训练可视化：在训练过程中，您可通过 VisualDL 查看模型训练过程参数变化、日志详情，及当前最优的训练集和验证集训练指标。模型在训练过程中通过点击”中止训练”随时中止训练过程。

模型训练结束后，可选择进入『模型剪裁分析』或者直接进入『模型评估』。

模型训练是最容易出错的步骤，经常遇到的原因为电脑无法联网下载预训练模型、显存不够。训练检测模型\实例分割模型对于显存要求较高，**建议用户通过在 Windows/Mac/Ubuntu 的命令行终端（Windows 的 Cmd 命令终端）执行 `nvidia-smi` 命令**查看显存情况，请不要使用系统自带的任务管理器查看。

26.4 任务模型裁剪训练

此步骤可选，模型裁剪训练相对比普通的任务模型训练，需要消耗更多的时间，需要在正常任务模型训练的基础上，增加『**模型裁剪分类**』和『**模型裁剪训练**』两个步骤。

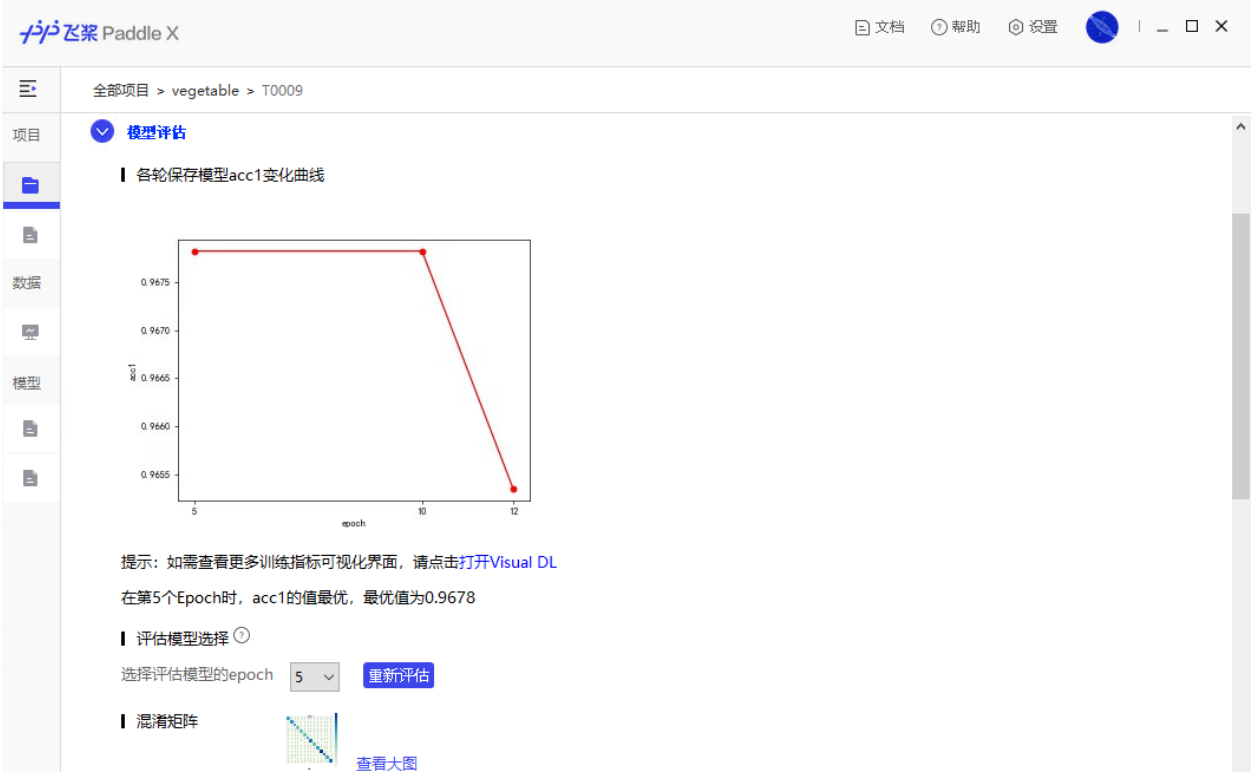
裁剪过程将对模型各卷积层的敏感度信息进行分析，根据各参数对模型效果的影响进行不同比例的裁剪，再进行精调训练获得最终裁剪后的模型。裁剪训练后的模型体积，计算量都会减少，并且可以提升模型在低性能设备的预测速度，如移动端，边缘设备，CPU。

在可视化客户端上，**用户训练好模型后**，在训练界面，

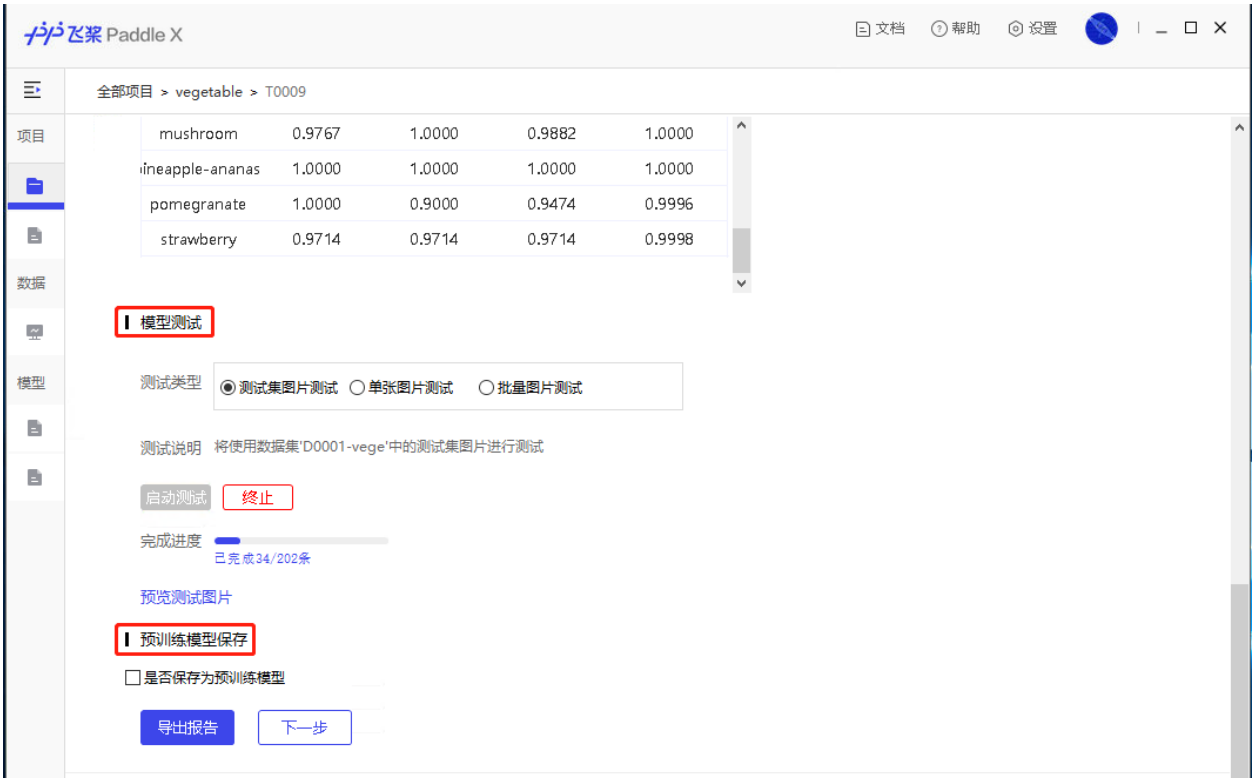
- 首先，点击『模型裁剪分析』，此过程将会消耗较长的时间
- 接着，点击『开始模型裁剪训练』，客户端会创建一个新的任务，无需修改参数，直接再启动训练即可

26.5 模型效果评估

在模型评估页面，您可查看训练后的模型效果。评估方法包括混淆矩阵、精度、召回率等。



您还可以选择『数据集切分』时留出的『测试数据集』或从本地文件夹中导入一张/多张图片，将训练后的模型进行测试。根据测试结果，您可决定是否将训练完成的模型保存为预训练模型并进入模型发布页面，或返回先前步骤调整参数配置重新进行训练。



26.6 模型发布

当模型效果满意后，您可根据实际的生产环境需求，选择将模型发布为需要的版本。

如若要部署到移动端/边缘设备，对于部分支持量化的模型，还可以根据需求选择是否量化。量化可以压缩模型体积，提升预测速度

如果您有任何问题或建议，欢迎以 issue 的形式，或加入 PaddleX 官方 QQ 群（1045148026）直接反馈您的问题和需求


飞桨 PaddleX交流群
群号：1045148026
扫一扫二维码，加入群聊。

CHAPTER 27

PaddleX 客户端常见问题

27.1 1. 训练出错，提示『训练任务异常中止，请查阅错误日志具体确认原因』?

请按照以下步骤来查找原因

- 1. 首先根据提示，找到错误日志，根据日志提示判断原因
- 2. 如无法确定原因，测 a) 尝试重新训练，看是否能正常训练；b) 调低 batchsize（同时按比例调低学习率）排除是否是显存不足原因导致
- 3. 如第 2 步仍然失败，请前往 GitHub 提 ISSUE，a) 描述清楚问题 b) 贴上训练参数截图 c) 附上错误日志 https://github.com/PaddlePaddle/PaddleX/issues
- 4. 如无 Github 帐号，则可加 QQ 群 1045148026 在线咨询工程师（咨询时请附上出错日志）

27.2 2. 没有使用 GPU，使用 CPU，错误日志仍然提示”cuda error”

部分 Windows 机型由于 GPU 型号或驱动较老，导致训练时无法使用 GPU，还会导致使用不了 CPU，针对此情况，可采取如下方式解决

- 1. 在 PaddleX 客户端安装目录下，删除”paddle”文件夹
- 2. 下载 paddlepaddle-cpu（压缩文件可在百度网盘下载，提取码 iaor，约 57M），下载解压后，将目前中的 paddle 文件夹拷贝至 PaddleX 客户端安装目录下即可

- 3. 重新启动 PaddleX 客户端，替换后客户端仅支持使用 CPU 训练模型

27.3 3. 如何升级 PaddleX 客户端

PaddleX 客户端目前需用户手动下载最新版升级，旧版本可直接删除原安装目录即可。升级前请备份好工作空间下的 3 个 workspace.*.pb 文件，避免升级过程可能导致项目信息丢失。

PaddleX 更新历史和下载地址: https://www.paddlepaddle.org.cn/paddlex/download

27.4 4. 如何卸载 PaddleX 客户端

客户端安装本质只是解压文件到相应目录，因此卸载直接删除安装目录和桌面快捷方式即可。

27.5 5. 使用客户端训练检测模型在测试图片中出现很多目标框，甚至同一物体上出现多个目标框

目标检测模型和实例分割模型，在模型预测阶段，会将所有可能的目标都输出，对于输出，我们需要可以按如下方式来处理

- 1. 观察预测出来的各个目标框，各框上应该还同时包含相应框的置信度分数
- 2. 设定一个 threshold，过滤掉低置信度的目标框上面两个步骤，在客户端的评估界面，我们可以手动输入 threshold 后，重新再预测；而对于在实际使用，例如 Python 预测部署，则根据得到结果中的’score’进行过滤

27.6 6. 使用 CPU 训练时，如何设置 CPU_NUM 多 CPU 卡进行训练

Windows 平台上由于缺少 NCCL 库，无法使用多 GPU 或多 CPU 训练。而对于 v1.1.5 版本及以下版本，当前无法设置多 CPU 进行训练（v1.1.5 版本及以下版本，请勿在环境变量中设置 CPU_NUM，可能会导致无法使用 CPU 进行模型训练）

27.7 7. 如何查看 GPU 的使用情况

Windows/Linux 上都可以在打开命令终端后（Windows 为 CMD 命令行）后，输入 nvidia-smi 命令查看 GPU 使用情况。不建议使用其它方式（如任务管理器等）。如若提示此命令不存在，可在网上搜索相应解决方法。

CHAPTER 28

PaddleX RESTful

PaddleX RESTful 是基于 PaddleX 开发的 RESTful API。

对于开发者来说通过如下指令启动 PaddleX RESTful 服务，开启 RESTful 服务后可以通过下载 Remote 版本的 GUI 或者是 web demo 连接开启 RESTful 服务的服务端完成深度学习全流程开发。

同样您还可以根据 RESTful API 来开发自己的可视化界面。

```
` paddlex_restful --start_restful --port 8081 --workspace_dir D:\Workspace `
```

注意：请确保启动 RESTful 的端口未被防火墙限制

28.1 支持 RESTful 版本的 GUI

支持 RESTful 版本的 GUI 是针对 PaddleX RESTful 开发的可视化客户端。开发者可以通过客户端连接开启 RESTful 服务的服务端，通过 GUI 远程的实现深度学习全流程：**数据处理**、**超参配置**、**模型训练及优化**、**模型发布**，无需开发一行代码，即可得到高性深度学习推理模型。

28.2 支持 RESTful 版本 Web Demo

支持 RESTful 版本 Web Demo 是针对 PaddleX RESTful 开发的网页版可视化客户端。开发者可以通过 Web Demo 连接开启 RESTful 服务的服务端，远程实现深度学习全流程：**数据处理**、**超参配置**、**模型训练及优化**、**模型发布**，无需开发一行代码，即可得到高性深度学习推理模型。

28.3 PaddleX RESTful API 二次开发

开发者可以使用 PaddleX RESTful API 进行二次开发，按照自己的需求开发可视化界面

28.3.1 介绍与使用

PaddleX RESTful 是基于 PaddleX 开发的 RESTful API。

对于开发者来说可以通过如下指令启动 PaddleX RESTful 服务

```
paddlex_restful --start_restful --port [端口号] --workspace_dir [工作空间地址]
```

对于设置 workspace 在 HOME 目录的 wk 文件夹下，RESTful 服务端口为 8080 的命令参考如下:

```
paddlex --start_restful --workspace_dir ~/wk --port 8080
```

注意：请确保启动 RESTful 的端口未被防火墙限制

开启 RESTful 服务后可以实现如下功能:

- 通过下载基于 RESTful API 的 GUI 连接开启 RESTful 服务的服务端，实现远程深度学习全流程开发。
- 通过使用 web demo 连接开启 RESTful 服务的服务端，实现远程深度学习全流程开发。
- 根据 RESTful API 来开发您自己个性化的可视化界面。

PaddleX Remote GUI

PaddleX Remote GUI 是针对 PaddleX RESTful 开发的可视化客户端。开发者可以通过客户端连接开启 RESTful 服务的服务端，通过 GUI 实现深度学习全流程：**数据处理、超参配置、模型训练及优化、模型发布**，无需开发一行代码，即可得到高性深度学习推理模型。

客户端下载地址

- MAC
- Windows

客户端使用流程

step1：安装 PaddleX

```
pip install paddlex
```

注意：若需要使用 GPU 请安装 pycuda

```
pip install pycuda
```

step2：开启 RESTful 服务

```
paddlex_restful --start_restful --port [端口号] --workspace_dir [工作空间地址]
```

setp3：根据上面的链接下载支持 RESTful 版本的 GUI

step4：运行客户端，如图所示填写开启 RESTful 后端的 ip 与端口，点击确定便可正常使用 GUI

PaddleX Web Demo

PaddleX Web Demo 是针对 PaddleX RESTful 开发的 Web 可视化客户端。Wed demo 传送门

Web DEMO 使用流程

step1：安装 paddlex

```
pip install paddlex
```

注意：若需要使用 GPU 请安装 pycuda

```
pip install pycuda
```

step2：开启 RESTful 服务

```
paddlex_restful --start_restful --port [端口号] --workspace_dir [工作空间地址]
```

step3：

- 方法 1（推荐）：在浏览器输入开启 RESTful 服务的机器与端口号如:10.3.12.4:8080，便可以使用 WEB GUI，此方法无需 step4 操作
- 方法 2：通过浏览器打开Demo文件

step4：点击设置服务器信息，填写正确的后端 ip 与端口

PaddleX RESTful API 二次开发

开发者可以使用 PaddleX RESTful API 进行二次开发，按照自己的需求开发可视化界面，详细请参考以下文档

RESTful API 二次开发简介

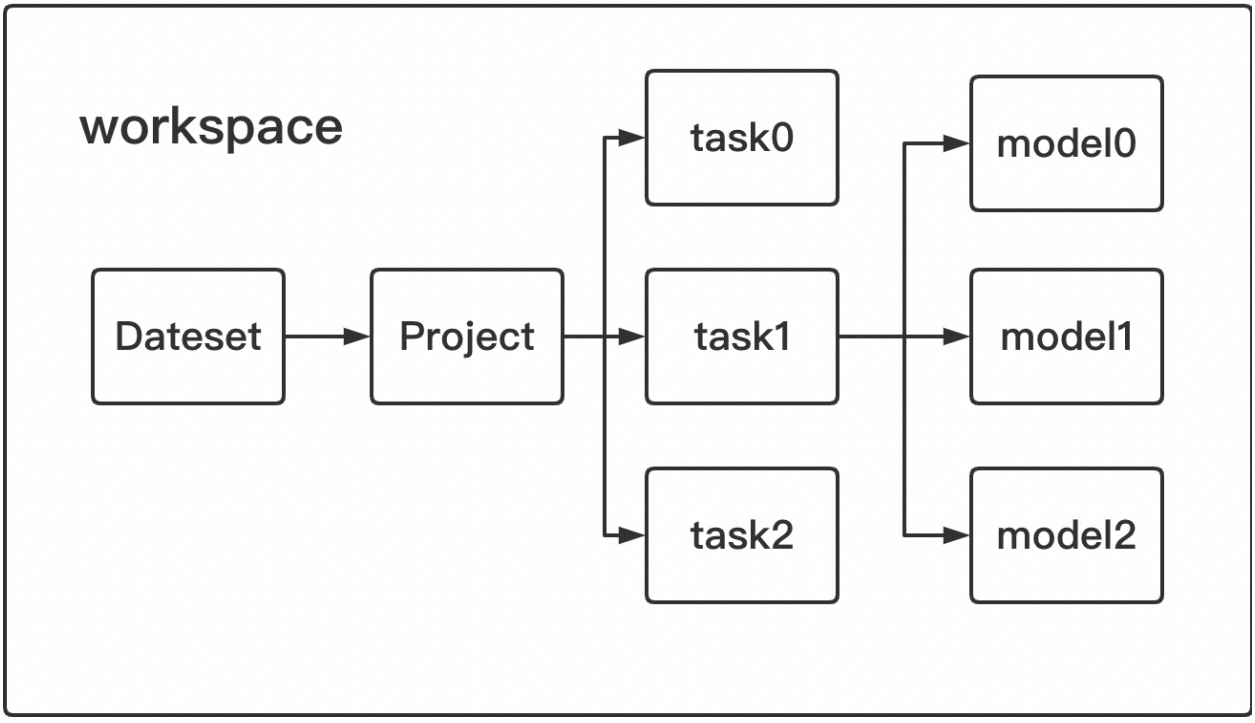
快速开始

API 参考文档

自定义数据结构

28.3.2 二次开发简介

如图，PaddleX RESTful 主要由数据集 (dataset)，项目 (project)，任务 (task)，模型 (model) 组成。上述模块数据保存在指定的工作空间 (workspace) 内，相应的结构化信息通过 protobuf 保存，workspace 的 protobuf 消息定义。



说明：后续 RESTful API 通过 `[HTTP request method] url` 来表示

流程介绍

对于通过 RESTful 接口来进行二次开发，主要的流程如下：

- 1)：指定工作空间启动 restful 服务
- 2)：创建、导入并切分数据到数据集
- 3)：创建项目，绑定数据集到该项目，并根据项目类型获取训练的默认参数
- 4)：根据默认参数，在该项目下创建训练任务，调整参数 (非必要)，开始训练模型
- 5)：对训练好的模型进行裁剪、评估、测试 (非必要)
- 6)：保存或者发布训练好的模型

工作空间

通过如下命令启动 PaddleX 的 RESTful 服务，同时会初始化工作空间，初始化工作空间主要做载入工作空间内已有的数据集、项目等模块的信息。初始化工作空间后就可以正常调用其他的 RESTful API，所有新建的数据集、项目等数据都会保存在此工作空间目录下面

```
paddlex_restful --start_restful --port [端口号] --workspace_dir [工作空间目录]
```

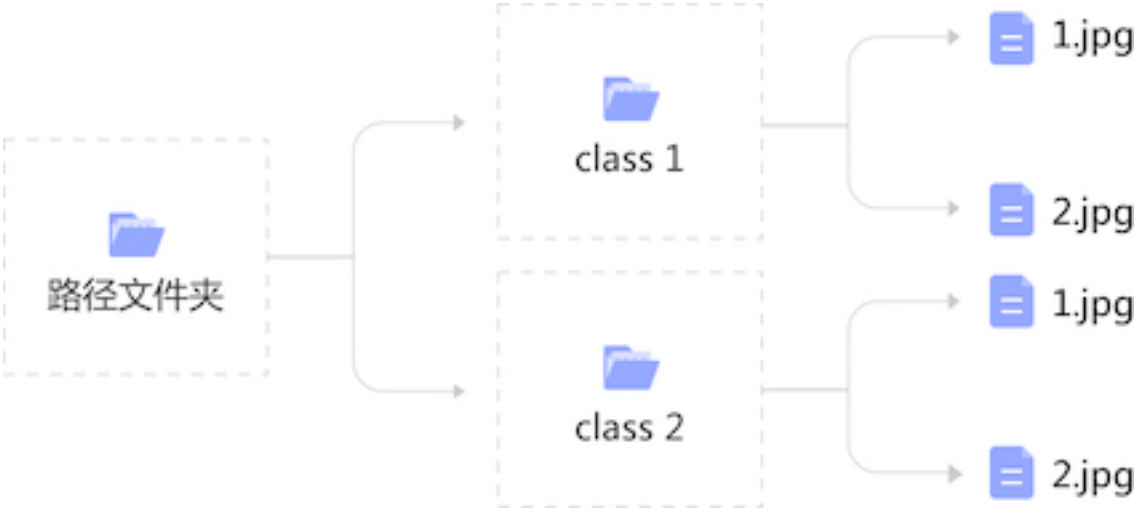
数据集

可以通过调用”[post] /dataset”接口创建数据集、创建数据集后会在工作空间内创建相应的文件夹，按照workspace protobuf 定义的变量保存数据集信息。创建数据集后可以通过”[put] \dataset”接口导入数据集，目前仅支持从路径导入并且数据集需储存在后端服务器。目前支持图像分类、目标检测、语义分割与实例分割四种数据集，具体格式如下

图像分类

如图所示

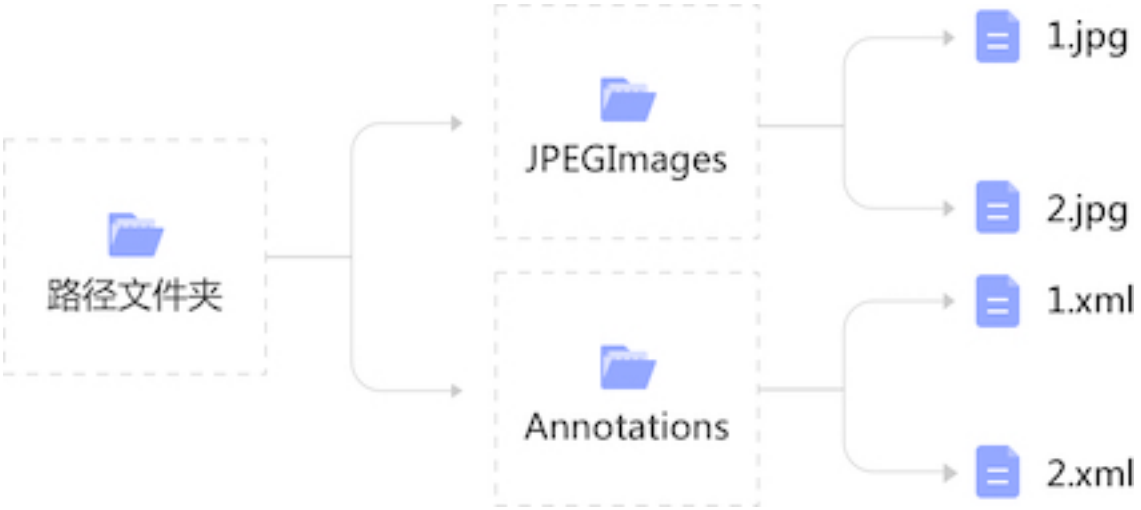
- 文件夹名为需要分类的类名，输入限定为英文字符，不可包含：空格、中文或特殊字符；
- 图片格式支持 png，jpg，jpeg，bmp 格式



目标检测

如图所示

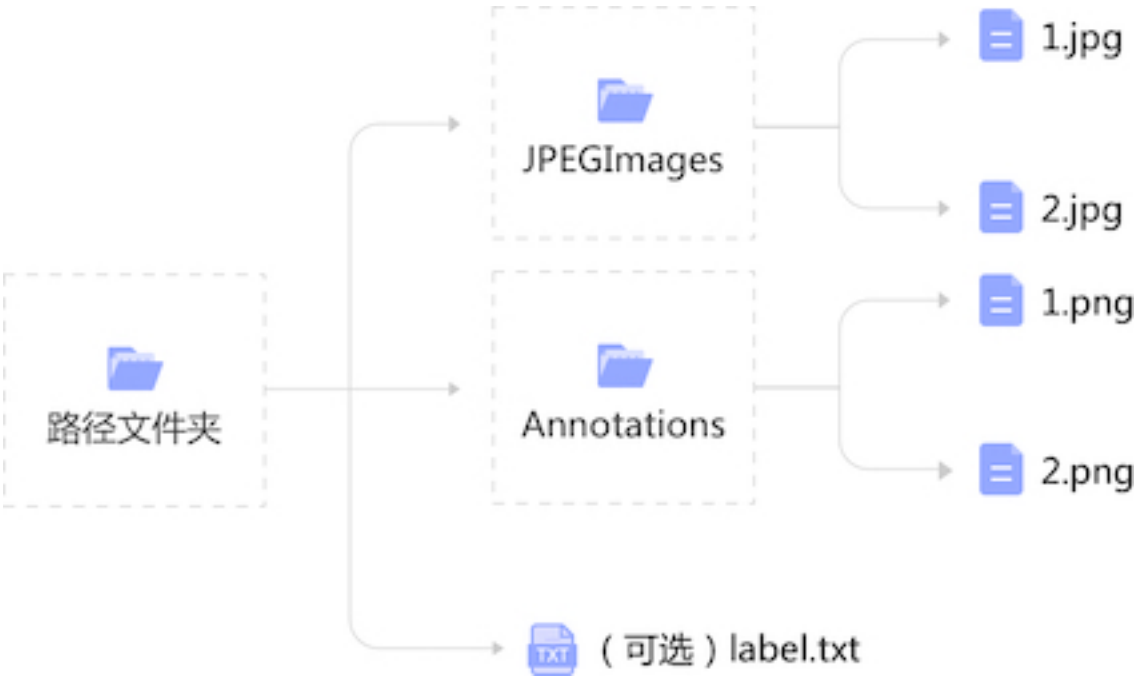
- 图片文件名需要为”JPEGImages”，标签文件夹命名需要为”Annotations”
- 图片格式支持 png，jpg，jpeg，bmp 格式；标签文件格式为.xml



语义分割

如图所示

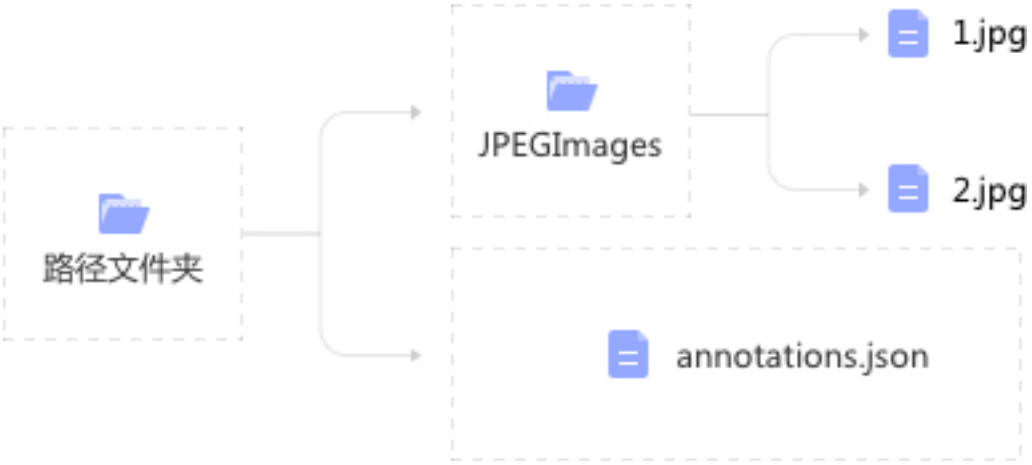
- 图片文件名需要为”JPEGImages”，标签文件夹命名需要为”Annotations”
- 图片格式支持 png，jpg，jpeg，bmp 格式
- 标注需要与图片像素严格保持一一对应，格式只可为 png。每个像素值需标注为 [0,255] 区间从 0 开始依序递增整数 ID，除 255 外，标注 ID 值的增加不能跳跃。其中 255 表示模型中需忽略的像素，0 为背景类标注。
- (可选) 可以提供一份命名为”labels.txt”的包含所有标注名的清单



实例分割

如图所示

- 图片文件名需要为”JPEGImages”，标签文件名需要为”annotations.json”
- 图片格式支持 png，jpg，jpeg，bmp 格式；标签文件格式为.json



项目

可以通过调用”[post] /project”接口创建项目，目前支持的项目类型有分类 (classification)、检测 (detection)、语义分割 (segmentation)、实例分割 (instance_segmentation)。对于新建的项目首先需要绑定项目类型对应的数据集，通过”[post] \workspace”可以实现；然后便可以在项目下创建任务，进行一系列任务相关的操作。

任务

在创建项目后，首先需要通过”[get] /project/task/params”获得默认的训练参数，可以通过调用”[post] /project/task”接口在项目中创建任务，创建好任务后可以通过 API 实现以下功能：

- 训练 (train): 进行模型的训练
- 裁剪 (prune): 对训练好的模型模型进行裁剪
- 评估 (eval): 对训练好的模型进行评估
- 预测 (predict): 用训练好的模型进行预测，目前仅支持单张图片的预测

模型

目前 PaddleX RESTful API 支持将训练评估后的模型保存为预训练模型、导出 inference 模型、导出 Padlle-Lite 模型、同时能支持模型的量化，可以通过调用”[post] /model接口来完成这些功能

28.3.3 快速开始

环境依赖

- paddlepaddle-gpu/paddlepaddle
- paddlex
- pycocotools

服务端启动 PaddleX RESTful 服务

```
paddlex_restful --start_restful --port [端口号] --workspace_dir [工作空间目录]
```

客户端请求服务端

```
import requests
url = "https://127.0.0.1:5000"
```

- url 为实际服务端 ip 与端口
- 所有的请求，通过 ret.status_code 是否为 200，判断是否正确给 Server 执行
- 在 status_code 为 200 的前提下，如果 ret.json()[' status'] 为-1，则表明出错，出错信息在 ret.json()[' message'] 里面，如果执行成功，status 是 1

创建一个 PaddleX 的训练任务

下面介绍如何通过 API 完成模型的训练、评估、预测与导出，对于每个 RESTful API 的详细介绍请参考API 接口文档，对于示例中用到自定的数据结构请参考数据结构

流程

对于通过 RESTful API 创建一个 PaddleX 的训练任务的主要流程如下

- 1)：创建并导入数据集
- 2)：创建项目并绑定数据集
- 3)：获取参数并创建任务
- 4)：开始训练
- 5)：评估任务
- 6)：导出模型

数据集操作

创建数据集

```
# dataset_type: 支持"detection"/"classification"/"segmentation"/"instance_segmentation"
params = {"name": "我的第一个数据集", "desc": "这里是数据集的描述文字", "dataset_type":
↪"detection"}
ret = requests.post(url+"/dataset", json=params)
# 获取数据集 id
did = ret.json()['id']
```

导入数据集

```
# 导入数据集
params = {'did' : did, 'path' : '/path/to/dataset'}
ret = requests.put(url+"/dataset", json=params)

# 数据集导入是一个异步操作，可以通过不断发送请求，获取导入的状态
params = {"did": did}
ret = requests.get(url+"/dataset", json=params)
# 导入状态获取，其中 DatasetStatus 为自定义枚举标量，用来表示数据集的状态，具体定义请参考数据
结构部分
import_status = DatasetStatus(ret.json['dataset_status'])
if import_status == DatasetStatus.XCOPYDONE:
    print("数据集导入成功")
elif import_status == DatasetStatus.XCOPYING:
    print("数据集正在导入中")
elif import_status == DatasetStatus.XCHECKFAIL:
    print("数据集格式校验未通过，请确定数据集格式是否正确)
```

切分数据集

```
# 当数据集导入成功后，可以对数据集进行切分
# 切分数据集按照训练集、验证集、测试集为：6：2：2 的形式切分
params = {'did' : did, 'val_split' : 0.2 , 'test_split' : 0.2}
ret = requests.put(url+"/dataset/split', json=params)

# 切分数据集后需要获取具体切分情况
```

(下页继续)

(续上页)

```
params = {'did': did}
ret = requests.get(url+"/dataset/details', json=params)
# 获取切分情况
dataset_details = ret.json()
```

项目操作

创建项目

```
# project_type: 支持 detection/classification/segmentation/instance_segmentation
params = {'name': '项目名称', 'desc': '项目描述文字', 'project_type' : 'detection'}
ret = requests.post(url+'/project', json=params)
# 获取项目 id
pid = ret.json['pid']
```

绑定数据集

```
# 修改 project 中的 did 属性
# struct 支持 project/dataset/task
params = {'struct': 'project', 'id': pid, 'attr_dict': {'did':did}}
ret = requests.put(url+'/workspace', json=params)
```

获取训练默认参数

```
params = {"pid", "P0001"}
ret = requests.get(url+"/project/task/parmas", json=params)
# 获取默认训练参数
train_params = ret.json()['train']
```

任务操作

创建任务

```
# 将训练参数 json 化
params_json = json.dumps(train_params)
# 创建任务
```

(下页继续)

(续上页)

```
params = {'pid': 'P0001', 'train':params_json}
ret = requests.post(url+'/task', json=params)
# 获取任务 id
tid = ret.json()['tid']
```

启动训练任务

```
params = {'tid' : tid}
ret = requests.post(url+'/project/task/train', json=params)

# 训练任务是一个后台异步操作，可通过如下操作，获取任务训练的最新状态
params = {'tid' : 'T0001', 'type': 'train'}
ret = requests.get(url+'/project/task/metrics', json=params)
ret.json() 获取返回值：
{'status': 1, 'train_log': 训练日志}
```

停止训练任务

通过如下操作，停止正在训练的任务

```
params = {'tid': tid, 'act': 'stop'}
ret = requests.put(url+'/project/task/train', json=params)
```

获取训练任务信息

```
params = {'tid': tid, 'type': 'train'}
ret = requests.get(url+'/project/task/metrics', json=params)
# 任务训练信息
train_log = ret.json()['train_log']
```

创建一个评估任务，并获取评估结果

```
# 获取任务状态
params = {'tid': tid}
ret = requests.get(url+'/project/task', json=params)
# 判断是否训练完成
```

(下页继续)

(续上页)

```
if TaskStatus(ret.json()['task_status']) == TaskStatus.XTRAINDONE:
    # 创建一个评估任务，可以指定 score_thresh
    params = {'tid': tid, 'score_thresh', 0.3}
    ret = requests.post(url+'/project/task/evaluate', json=params)
# 获取评估任务状态
import time
while:
    params = {'tid': tid}
    ret = requests.get(url+'/project/task/evaluate', json=params)
    # 判断是否评估完成
    if TaskStatus(ret.json()['evaluate_status']) == TaskStatus.XEVALUATED:
        break
    else:
        time.sleep(1)
# 获取评估结果
result = ret.json()['result']
```

使用模型进行预测

```
import cv2
import numpy as np
# 预测图片路径
img_path = '/path/to/img'
params = dict()
#base64 编码
with open(img_path, 'rb') as f:
    base64_data = base64.b64encode(f.read())
    base64_str = str(base64_data,'utf-8')
    params['image_data'] = base64_str
params['tid'] = tid
# 单张推理
ret = requests.post(url + 'project/task/predict', json=params)
# 获取结果保存地址
result_path = ret.json()['path']
# 判断是否预测完成
while:
    params = {'tid': tid}
    ret = requests.get(url+'/project/task/predict', json=params)
    # 判断是否评估完成
```

(下页继续)

(续上页)

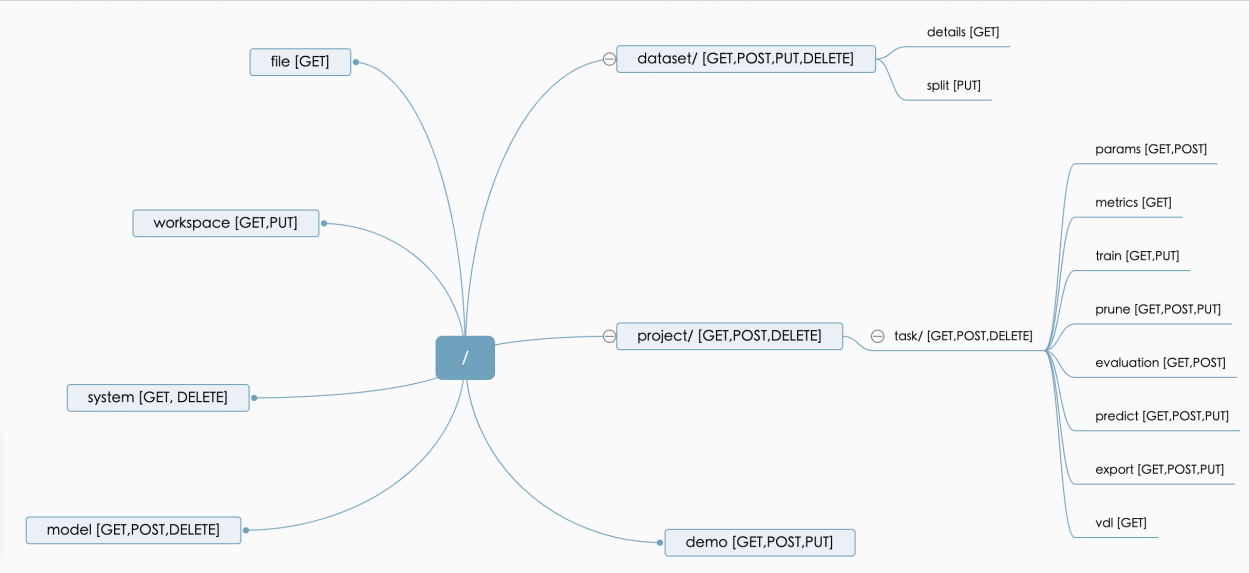
```
    if PredictStatus(ret.json()['predict_status']) == PredictStatus.XPREDONE:
        break
    else:
        time.sleep(1)
# 获取结果
params = {'path' : result_path}
ret = requests.get(url+'/file', json=params)
# 图片 based64 数据
img_data = ret.json()['img_data']
# 转换为 numpy 数据
img_data = base64.b64decode(img_data)
img_array = np.frombuffer(img_data, np.uint8)
img = cv2.imdecode(img_array, cv2.COLOR_RGB2BGR)
```

导出 inference 模型

```
# 导出 inference 模型
# 保存地址
save_dir = '/path/to/save/inference/model'
params = {'tid': tid ,'type': 'inference', 'save_dir' : save_dir}
ret = requests.post(url+'/project/task/export', json=params)
#workspace 创建 inference 模型信息
params = {'pid' : pid, 'tid': tid, 'name': 'my_inference_model', 'type': 'exported',
↪'path' : save_dir, 'exported_type': 0}
ret = requests.post(url+'/model', json=params)
```

28.3.4 RestFUL API 参考文档

API 总览



图片包含目前 PaddleX RESTful 模块提供所有的 RESTful API:

- /workspace: 工作空间相关
- /dataset：数据集操作
- /project: 项目操作
- /project/task: 任务相关操作
- /model: 模型相关操作
- /file：文件传输
- /demo: 示例

API 接口文档

说明:

- 后续例子中 HTTP 请求通过 requests 完成，url 代表运行 PaddleX RESTful 服务的主机 ip 与端口号
- 所有的请求，通过 ret.status_code 是否为 200，判断是否正确给 Server 执行
- 在 status_code 为 200 的前提下，如果 ret.json()[' status'] 为-1，则表明出错，出错信息在 ret.json()[' message'] 里面，如果执行成功，status 是 1

/workspace [GET,PUT]

主要是对工作空间的结构化信息进行操作，包括修改合和获取工作空间中数据集、项目、任务的属性

- GET 请求：获取 workspace 中数据集、项目、任务、模型的属性

- PUT 请求：修改 workspace 中数据集、项目、任务、模型的属性，对于创建项目之后我们需求通过该接口将数据集与项目绑定才能进行训练任务对于可以获取和修改的属性请参考Protobuf 结构化数据

```
methods=='GET': 获取工作目录中项目、数据集、任务的属性
        Args:
                struct(str): 结构类型，可以是'dataset', 'project' 或'task',
                id(str): 结构类型对应的 id
                attr_list(list): 需要获取的属性的列表
        Return:
                status
                if Not Args:
                        'dirname' : RESTful 服务初始化时指定的工作目录
                else:
                        attr(dict):key 为属性，value 为属性的值

        Example:
                # 获取任务 id 为'T0001' 的任务对应的 path 的值
                params = {'struct': 'task', 'id': 'T0001', 'attr_list': ['path']}
                ret = requests.get(url + '/workspace', json=params)
                if (ret.json()['status'] == 1):
                        task_path = ret.json()['attr']['path']
                else:
                        print("failed to get path)

methods=='PUT': 修改工作目录中项目、数据集、任务的属性
        Args:
                struct(str): 结构类型，可以是'dataset', 'project' 或'task'
                id(str): 结构类型对应的 id
                attr_dict(dict):key: 需要修改的属性，value: 需要修改属性的值
        Return:
                status
        Example
                # 为 id 为'P0001' 的项目绑定 id 为'D0001' 的数据集
                attr_dict = {'did': 'D0001'}
                params = {'struct': 'project', 'id': 'P0001', 'attr_dict': attr_dict}
                ret = requests.post(url + '/workspace', json=params)
```

/dataset [GET,POST,PUT,DELETE]

对数据集进行操作，包括创建、导入、查询、删除数据集的功能

- POST 请求: 创建数据集，创建时需要指定数据集类型可以是 [’ classification’， ‘detection’，

'segmentation','instance_segmentation'] 中的一种

- GET 请求: 创建好数据集后，可以通过 GET 请求获取数据集信息，目前支持获取单个或者所有数据集的信息
- PUT 请求: 导入数据集，创建数据集后并没有真实的数据与数据集关联，需要通过导入功能，将数据导入到工作空间内，需要导入的数据集必须按照
- DELETE: 删除一个数据集 DatasetStatus 定义了数据集状态，具体请参考数据集状态变量

```
methods=='GET': 获取所有数据集或者单个数据集的信息
        Args:
                did(str, optional): 数据集 id (可选), 如果存在就返回数据集 id 对应数据集的信
息
        Ruturn:
                status
                if 'did' in Args:
                        id(str): 数据集 id,
                        dataset_status(int): 数据集状态 (DatasetStatus) 枚举变量的值
                        message(str): 数据集状态信息
                        attr(dict): 数据集属性
                else:
                        datasets(list): 所有数据集属性的列表
    Example1:
        # 获取数据集 id 为'D0001' 数据集的信息
        params = {'did': 'D0001'}
        ret = requests.get(url + '/dataset', json=params)
        # 状态码转换为 Enum 类型
        ret.json()['dataset_status'] = DatasetStatus(ret.json()['dataset_status'])
    Example2:
        # 获取所有数据集信息
        ret = requests.get(url + '/dataset')
    Ruturn 中的自定数据结构:
                数据集属性 attr, 为 dict 对于其中的 key 包括如下
                attr{
                        'type'(str): 数据集类型
                        'id'(str): 数据集 id
                        'name'(str): 数据集名字
                        'path'(str): 数据集路径
                        'desc'(str): 数据集描述
                        'create_time'(str): 数据集创建时间
                        'pids'(list): 数据集绑定的项目列表
                }
```

(下页继续)

(续上页)

```
                所有数据集属性 datasets，为 list，dataset_list[idx] 包含：
                dataset_list[idx]{
                        "id": 数据集 id
                        "attr": 数据集属性
                }
                其中数据集属性 attr，为 dict 对于其中的 key 包括如下，注意与获取单个数据集的
属性区分
                attr{
                        "type": 数据集类型，
            "id": 数据集 id,
            "name": 数据集名字，
            "path": 数据集路径，
            "desc": 数据集描述，
            "create_time": 数据集创建时间
                        'dataset_status': 数据集状态 (DatasetStatus) 枚举变量的值
            'message' : 数据集状态信息

                }

methods=='POST': 创建一个新的数据集
        Args:
                name(str): 数据集名字
                desc(str): 数据集描述
                dataset_type(str): 数据集类型，可以是 ['classification', 'detection',
↪'segmentation','instance_segmentation']
        Return:
                did(str): 数据集 id
                status
        Example:
                # 新建一个分类数据集
                params = {'name' : ' 我的数据集', 'desc' : ' 数据集描述',  'dataset_type' :
 'classification'}
                ret = requests.post(url + '/dataset',  json=params)
                # 获得数据集 id
                did = ret.json()['did']

methods=='PUT': 异步，向数据集导入数据，支持分类、检测、语义分割、实例分割、摇杆分割数据集类型
        Args:
                did(str): 数据集 id
                path(str): 数据集路径
```

(下页继续)

(续上页)

```
        Return:
                status
        Example:
                # 为 id 为'D0001' 的数据集导入数据集
                params = {'did':'D0001',  'path': '/path/to/dataset'}
                ret = requests.put(url + '/dataset',  json=params)

methods=='DELETE': 删除已有的某个数据集
        Args:
                did(str): 数据集 id
        Return:
                status
        Example:
                # 删除数据集 id 为'D0001' 的数据集
                params = {'did':'D0001'}
                ret = requests.delete(url + '/dataset',  json=params)
```

/dataset/split [PUT]

按照比例切分一个已经导入数据的数据集

```
methods=='PUT': 切分某个数据集
        Args:
                did(str): 数据集 id
                val_split(float): 验证集比例
                test_split(float): 测试集比例
        Return:
                status
        Example:
                # 按照训练集, 验证集, 测试集, 7: 2: 1 的形式切分 D0001 数据集
                params = {'did':'D0001', 'val_split': 0.2, 'test_split': 0.1}
                ret = requests.put(url + '/dataset/split',  json=params)
```

/dataset/details [GET]

获取切分后数据集的具体信息

```
methods=='GET': 获取某个数据集的详细信息
        Args:
```

(下页继续)

(续上页)

```
                did(str): 数据集 id
        Return:
                details(dict): 数据集详细信息,
                status
        Example:
                # 获取数据集 id 为'D0001' 的数据集的详细信息
                params = {'did':'D0001'}
                ret = requests.get(url + '/dataset/details',  json=params)
        Ruturn 中的自定数据结构:
                数据集详细信息 (details), 其中的 key 包括如下:
                details{
                        'file_info'(dict): 全量数据集文件与标签映射表,key: 图片相对于数据集
地址的相对路径; value: 标签文件相对于数据集地址的相对路径
                        'label_info'(dict): 标签与全量数据集文件映射表,key: 标签的类别;
value: 标签文件相对于数据集地址的相对路径
                        'labels'(list): 标签列表
                        'train_files'(list): 训练集文件列表, 相对于据集地址的相对路径
                        'val_files'(list): 验证集文件列表, 相对于据集地址的相对路径
                        'test_files'(list): 测试集文件列表, 相对于据集地址的相对路径
                        'class_train_file_list(dict)': 类别与训练集映射表, key 为类别、
value 为训练图片相对于据集地址的相对路径
                        'class_val_file_list(dict)': 类别与评估集映射表, key 为类别、value
为评估图片相对于据集地址的相对路径
                        'class_test_file_list(dict)': 类别与测试集映射表, key 为类别、value
为测试图片相对于据集地址的相对路径
                }
```

/file [GET]

用于文件传输，目前支持图片、xml 格式文件、log 或者 txt 格式文件，对于图片文件如果带有 did 参数则会返回带可视化 label 的图片数据

```
methods=='GET': 获取服务端的文件, 目前支持图片、xml 格式文件、log 文件
        Args:
                'path'(str): 文件在服务端的路径
                'did'(str, optional): 可选, 数据集 id 仅在文件为图片时有效。若存在返回图片带
label 可视化。注意当前不支持分类数据集数据的标注可视化
        Return:
                # 数据为图片
```

(下页继续)

(续上页)

```
                img_data(str): base64 图片数据
                status
                # 数据为 xml 文件
                ret: 数据流
                # 数据为 log 或者 txt 文件
                ret:json 数据
        Example1:
                # 获取图片，目前支持的图片格式有:
                #{'JPEG', 'jpeg', 'JPG', 'jpg', 'BMP', 'bmp', 'PNG', 'png'}
                # 图片通过 based64 编码
                params = {'path' : '/path/to/img'}
                ret = requests.get(url + '/file', json=params)
                # 图片数据
                img_data = base64.b64decode(ret.json()['img_data'])
                # 图片解码
                #opencv
                img_array = np.frombuffer(img_data, np.uint8)
        img = cv2.imdecode(img_array, cv2.COLOR_RGB2BGR)
        #PIL
        img = Image.open(BytesIO(img_data))
        Example2:
                # 获取 xml 数据
                params = {'path' : '/path/to/xml'}
                ret = requests.get(url + '/file', json=params)
                #xml 数据
                xml_file = ret.text
```

/project [GET,POST,DELETE]

项目相关操作，包括创建、删除、查询项目

- POST 请求：创建一个项目，支持分类、检测、语义分割、实例分割
- GET 请求: 查询已经创建项目中的信息，可以是单个项目或者是所有项目
- DELETE：删除一个项目，同时会删除项目里面所有 task

```
methods=='GET': 获取指定项目 id 的信息
        Args:
                'id'(str, optional): 项目 id，可选，如果存在就返回项目 id 对应项目的信息
        Return:
```

(下页继续)

(续上页)

```
                status,
                if 'id' in Args:
                        attr(dict): 项目属性
                else:
                        projects(list): 所有项目属性
        Example1:
                # 获取 id 为 P0001 项目的信息
                params = {'id' : 'P0001'}
                ret = requests.get(url + '/project', json=params)
        Example2:
                # 获取所有项目信息
                ret = requests.get(url + '/project', json=params)
        Ruturn 中的自定数据结构:
                单个项目属性 attr(dict){
                        id(str): 项目 id,
                        name(str): 项目名字,
                        desc(str): 项目描述,
                        type(str): 项目类型,
                        did(str): 项目绑定的数据集 id,
                        path(str): 项目在服务端的路径,
                        create_time(str): 项目创建时间,
                        tasks(list): 项目中包含所有任务的任务状态变量的 int 值
                }
                多个项目的属性 projects(list), 对于 list 里面每一个元素表示了单个项目属性的
字典 project 其数据结构如下
                project(dict){
                        id(str): 项目 id,
                        attr(dict): 项目属性字典, 与单个项目属性一致
                }

methods=='POST': 创建一个项目
        Args:
                name(str): 项目名
                desc(str): 项目描述
                project_type(str): 项目类型
        Return:
                pid(str): 项目 id
                status
        Example:
                # 创建一个新的项目, project_type 支持{'classification', 'detection',
↪'segmentation', 'instance_segmentation'}
```

(下页继续)

(续上页)

```
                params = {'name' : ' 分类项目', 'desc': ' 一个新的项目', 'project_type' :
↪'classification'}
                ret = requests.post(url + '/project', json=params)
                # 获取项目 id
                pid = ret.json()['pid']

methods=='DELETE': 删除一个项目，以及项目相关的 task
        Args:
                pid(str): 项目 id
        Return:
                status
        Example:
                # 删除 id 为'P0001' 的项目
                params = {'pid' : 'P0001'}
                ret = requests.delete(url + '/project', json=params)
```

/project/task [GET,POST,DELETE]

任务相关，创建、获取、删除任务

- POST 请求: 创建一个任务可以是训练任务也可以是剪裁任务，剪裁任务需要指定 parent_id
- GET 请求: 获取单个任务、单个项目内所有任务、所有任务的信息，当 tid 存在时返回指定任务的信息，如果任务状态 (TaskStatus) 显示任务停止可以通过 resume 查询任务是否可以恢复训练以及保存的最大 epoch；当 pid 存在时返回该项目下面所有任务信息
- DELETE 请求: 删除任务 TaskStatus 定义了任务状态，具体请参考任务状态变量

```
methods=='GET':# 获取某个任务的信息或者所有任务的信息
        Args:
                tid(str, optional): 任务 id，可选，若存在即返回 id 对应任务的信息
                resume(str, optional): 获取是否可以恢复训练的状态，可选，需在存在 tid 的情况
下才生效
                pid(str, optional): 项目 id，可选，若存在即返回该项目 id 下所有任务信息
        Return:
                status
                if 'tid' in Args:
                        task_status(int): 任务状态 (TaskStatus) 枚举变量的值
                        message(str): 任务状态信息
                        type(str): 任务类型包括{'classification', 'detection',
↪'segmentation', 'instance_segmentation'}
```

(下页继续)

(续上页)

```
                        resumable(bool): 仅 Args 中存在 resume 时返回，任务训练是否可以恢复
                        max_saved_epochs(int): 仅 Args 中存在 resume 时返回，当前训练模型保
存的最大 epoch
                else:
                        tasks(list): 所有任务属性
        Example1:
                # 获取任务 id 为"T0001" 的信息
                params = {'tid' : 'T0001'}
                ret = requests.get(url + '/project/task', json=params)
        Example2:
                # 获取所有任务的信息
                ret = requests.get(url + '/project/task')
        Ruturn 中的自定数据结构:
                所有任务属性 (tasks)，任务属性 attr(dict) 的 list
                attr{
                        'id'(str): 任务 id
                        'name'(str): 任务名字
                        'desc'(str): 任务详细描述
                        'pid'(str): 任务所属的项目 id
                        'path'(str): 任务在工作空间的路径
                        'create_time'(str): 任务创建时间
                        'status(int)': 任务状态 (TaskStatus) 枚举变量的值
                        'type(str)': 任务类型包括{'classification', 'detection',
↪'segmentation', 'instance_segmentation'}
                }

methods=='POST':# 创建任务 (训练或者裁剪)
        Args:
                pid(str): 项目 id
                train(dict): 训练参数
                desc(str, optional): 任务描述，可选
                parent_id(str, optional): 可选，若存在即表示新建的任务为裁剪任务，parent_id
的值为裁剪任务对应的训练任务 id
        Return:
                tid(str): 任务 id
                status
        Example:
                # 在 P0001 项目下创建一个任务
                params = {'pid' : 'P0001'}
                ret = requests.post(url + '/project/task')
```

(下页继续)

(续上页)

```
                # 获取任务 id
                tid = ret.json()['tid']

methods=='DELETE':# 删除任务
        Args:
                tid (str) : 任务 id
        Return:
                status
        Example:
                # 删除 id 为 T0001 的任务
                params = {'tid' : 'T0001'}
                ret = requests.delete(url + '/project/task')
```

/project/task/params [GET,POST]

获取和设置训练参数

- GET 请求: 获取已有的参数信息或者默认的参数信息，当 tid 存在时获取该任务已有的参数信息、当 pid 存在时获取该项目的推荐参数给该任务，在创建任务之前需要先获取默认参数
- POST 请求：设置训练参数，用户修改训练参数后通过该接口设置训练参数，在 workspace 内会将训练参数保存为 params.pkl 文件

```
methods=='GET':# 获取任务 id 对应的参数，或者获取项目默认参数
        Args:
                tid (str, optional) : 获取任务对应的参数
                pid(str, optional)：获取项目对应的默认参数
                model_type(str, optional)：pid 存在下有效，对应项目下获取指定模型的默认参数
                gpu_list(list, optional):pid 存在下有效，默认值为 [0]，使用指定的 gpu 并获
取相应的默认参数
        Return:
                train(dict)：训练或者裁剪的参数
                status
        Example1:
                获取任务 id 为 T0001 的任务对应的参数
                params = {'tid' : 'T0001'}
                ret = requests.get(url + '/project/task/params',json=params)
        Example2:
                获取项目 id 为 P0001 的检测项目，PPYOLO 模型的默认参数
                params = {'pid' : 'P0001', 'model_type' : 'PPYOLO'}
```

(下页继续)

(续上页)

```
                ret = requests.get(url + '/project/task/params',json=params)
                # 获取参数 dict
                train_dict = ret.json()['train']
                ret = requests.get(url + '/project/task/params',json=params)

methods=='POST':# 设置任务参数，将前端用户设置训练参数 dict 保存在后端的 pkl 文件中
        Args:
                tid(str): 任务 id
                train(dict): 训练参数
        Return:
                status
        Example:
                # 设置训练参数
                params = {'tid': 'T0001', 'train': train_dict}
                ret = requests.post(url + '/project/task/params',json=params)
```

/project/task/train [POST,PUT]

任务训练 (包括剪裁任务训练)，主要是启动和停止任务，需要先设置好训练参数 -POST 请求：启动一个训练任务，异步操作，启动前需要设置好训练的参数 -PUT 请求：暂停一个训练任务或者重启一个暂停的训练任务，重启训练任务可以设置重启的 epoch 数

```
methods=='POST':# 异步，启动训练或者裁剪任务
        Args:
                tid(str): 任务 id
                eval_metric_loss(int, optional): 可选，裁剪任务时可用，裁剪任务所需的评估
loss
        Return:
                status
        Example:
                # 启动任务 id 为 T0001 的任务的训练
                params = {'tid':'T0001'}
                ret = requests.post(url + '/project/task/train',json=params)

methods=='PUT':# 改变任务训练的状态，即终止训练或者恢复训练
        Args:
                tid(str): 任务 id
                act(str): [stop,resume] 暂停或者恢复
                epoch(int): (resume 下可以设置) 恢复训练的起始轮数
```

(下页继续)

(续上页)

```
    Return:
                status
        Example:
                # 停止任务 id 为 T0001 的任务的训练
                params = {'tid':'T0001', 'act': 'stop'}
                ret = requests.put(url + '/project/task/train',json=params)
```

/project/task/prune [GET,POST,PUT]

创建、获取、停止剪裁任务分析；在启动剪裁任务训练之前需要通过此接口完成剪裁任务分析

- GET 请求: 获取剪裁任务状态信息
- POST 请求: 创建剪裁分析任务，异步操作
- PUT 请求: 停止正在进行的剪裁分析任务 PruneStatus 定义了裁剪分析任务状态，具体请参考裁剪分析状态变量

```
methods=='GET':# 获取剪裁任务的状态
        Args:
                tid(str): 任务 id
        Return:
                prune_status(int): 剪裁任务状态 (PruneStatus) 枚举变量的值
                status
        Example:
                # 启动任务 id 为 T0001 的任务的训练
                params = {'tid':'T0001'}
                ret = requests.get(url + '/project/task/prune',json=params)

methods=='POST':# 异步，创建一个剪裁分析，对于启动剪裁任务前需要先启动剪裁分析
        Args:
                tid(str): 任务 id
        Return:
                status
        Example:
                # 对任务 id 为 T0001 的任务启动剪裁分析任务
                params = {'tid':'T0001'}
                ret = requests.post(url + '/project/task/prune',json=params)

methods=='PUT':# 改变裁剪分析任务的状态
        Args:
```

(下页继续)

(续上页)

```
                tid(str): 任务 id
                act(str):[stop], 目前仅支持停止一个裁剪分析任务
        Return
                status
        Example:
                # 停止 T0001 的任务的裁剪分析
                params = {'tid':'T0001', 'act': 'stop'}
                ret = requests.put(url + '/project/task/prune',json=params)
```

/project/task/evaluate [GET,POST]

创建、获取一个评估任务

- POST 请求：创建一个评估任务，需要模型训练好后调用，异步操作
- GET 请求: 获取评估任务的状态

```
methods=='GET':# 获取模型评估的结果
        Args:
                tid(str): 任务 id
        Return:
                evaluate_status(int): 任务状态 (TaskStatus) 枚举变量的值
                message(str): 描述评估任务的信息
                result(dict): 如果评估成功，返回评估结果的 dict，否则为 None
                status
        Example:
                # 获取任务 id 为 T0001 的任务的评估数据
                params = {'tid':'T0001'}
                ret = requests.get(url + '/project/task/evaluate',json=params)
                result = ret.json()['result']

methods=='POST':# 异步，创建一个评估任务
        Args:
                tid(str): 任务 id
                epoch(int,optional): 需要评估的 epoch，如果为 None 则会评估训练时指标最好的
epoch
                topk(int,optional): 分类任务 topk 指标，如果为 None 默认输入为 5
                score_thresh(float): 检测任务类别的 score threshhold 值，如果为 None 默认输
入为 0.5
                overlap_thresh(float): 实例分割任务 IOU threshhold 值，如果为 None 默认输入
为 0.3
```

(下页继续)

(续上页)

```
        Return:
                status
        Example:
                # 对任务 id 为 T0001 的分类任务进行评估，topk=5
                params = {'tid':'T0001', 'epoch': None, 'topk': 5, 'score_thresh': None,
↪'overlap_thresh': None}
                ret = requests.post(url + '/project/task/evaluate',json=params)
```

/project/task/evaluate/file [GET]

评估结果生成 excel 表格

- GET 请求: 评估完成的情况下，在服务器端生成评估结果的 excel 表格

```
methods=='GET':# 评估结果生成 excel 表格
        Args:
                tid(str): 任务 id
        Return:
                path(str): 评估结果 excel 表格在服务器端的路径
                message(str): 提示信息
                status
        Example:
                # 任务 id 为 T0001 的任务在服务器端生成评估 excel 表格
                params = {'tid': 'T0001'}
                ret = requests.get(url + '/project/task/evaluate/file',json=params)
                # 显示保存路径
                print(ret.json()['path'])
```

/project/task/metrics [GET]

获取训练、评估、剪裁的日志和敏感度与模型裁剪率关系图

- GET 请求：通过 type 来确定需要获取的内容

```
methods=='GET':# 获取日志数据
        Args:
                tid(str): 任务 id
                type(str): 可以获取日志的类型，[train,eval,sensitivities,prune]，包括训练，
评估，敏感度与模型裁剪率关系图，裁剪的日志
        Return:
```

(下页继续)

(续上页)

```
                status
                if type == 'train':
                        train_log(dict): 训练日志
                elif type == 'eval':
                        eval_metrics(dict): 评估结果
                elif type == 'sensitivities':
                        sensitivities_loss_img(dict): 敏感度与模型裁剪率关系图
                elif type == 'prune':
                        prune_log(dict): 裁剪日志
        Example:
                # 获取训练日志
                paramas = {'tid': 'T0002', 'type': 'train'}
                ret = requests.get(url + '/project/task/metrics',json=params)
        Ruturn 中的自定数据结构:
                train_log(dict){
                        eta: 剩余时间,
                        train_metrics: 训练指标,
                        eval_metircs: 评估指标,
                        download_status: 下载模型状态,
                        eval_done: 是否已保存模型,
                        train_error: 训练错误原因
                }
```

/project/task/predict [GET, POST]

创建、查询、停止一个预测任务

- POST 请求: 创建一个预测任务、图片输入需要先进行 base64 编码、异步操作
- GET 请求: 获取预测任务的状态
- PUT 请求: 停止一个预测任务 PredictStatus 定义了预测任务状态变量, 具体请参考预测任务状态变量

```
methods=='GET':# 获取预测状态
        Args:
                tid(str): 任务 id
        Return:
                predict_status(int): 预测任务状态 (PredictStatus) 枚举变量的值
                message(str): 预测信息
                status
        Example:
```

(下页继续)

(续上页)

```
                # 获取预测状态
                paramas = {'tid': 'T0002'}
                ret = requests.get(url + '/project/task/predict',json=params)
                predict_status = PredictStatus(ret.json()['predict_status'])

methods=='POST':# 创建预测任务，目前仅支持单张图片的预测
        Args:
                tid(str): 任务 id
                image_data(str):base64 编码的 image 数据
                score_thresh(float, optional): 可选，检测任务时有效，检测类别的 score␣
↪threashold 值默认是 0.5
                epoch(int,float, optional): 可选，选择需要做预测的 ephoch，默认为评估指标最
好的那一个 epoch
        Return:
                path(str): 服务器上保存预测结果图片的路径
                status
        Example:
                # 对一张图片进行预测并取回预测结果
                import base64
                import cv2
                # 对图片进行 base64 编码
                img_path = 'path to img need to be predict'
                f = open(img_path, 'rb')
                base64_data = base64.b64encode(f.read())
                base64_str = str(base64_data,'utf-8')
                params[tid: 'T0001', 'image_data', base64_str]
                # 进行预测
                ret = requests.post(url + '/project/task/predict',json=params)
                result_path = ret.json()['path']
                # 判断预测是否完成
                paramas = {'tid': 'T0001'}
                ret = requests.get(url + '/project/task/predict',json=params)
                predict_status = PredictStatus(ret.json()['predict_status'])
                if (predict_status == PredictStatus.XPREDONE):
                # 取回预测结果图片
                        params = {'path' : result_path}
                        ret = requests.get(url + '/file',json=params)
                        img_data = ret.json()['img_data']

methods=='PUT':# 停止一个预测任务
```

(下页继续)

(续上页)

```
        Args:
                tid(str): 任务 id
        Return:
            status
                predict_status: 评估的状态
```

/project/task/export [GET,POST,PUT]

创建、获取、停止模型装换或者导出、支持导出 inference 和 lite 的模型

- POST 请求: 创建模型导出，可以通过 type 确定导出 inference 模型还是 lite 模型
- GET 请求: 获取导出的结果日志、或者时导出状态
- PUT 请求：停止模型的导出

```
methods=='GET':# 获取导出模型的状态
        Args:
                tid(str): 任务 id
                quant(str,optional) 可选，[log,result]，导出量模型导出状态，若值为 log 则返
回量化的日志；若值为 result 则返回量化的结果
        Return:
                status
                if quant == 'log':
                        quant_log(dict): 量化日志
                if quant == 'result'
                        quant_result(dict): 量化结果
                if quant not in Args:
                        export_status(int): 模型导出状态 (PredictStatus) 枚举变量的值
                        message(str): 模型导出提示信息
        Example:
                # 获取模型导出状态信息
                params = {'tid':'T0002'}
                ret = requests.get(url + '/project/task/export',json=params)

methods=='POST':# 导出 inference 模型或者导出 lite 模型
        Args:
                tid(str): 任务 id
                type(str): 保存模型的类别 [infer,lite]，支持 inference 模型导出和 lite 的模
型导出
                save_dir(str): 保存模型的路径
```

(下页继续)

(续上页)

```
                epoch(str,optional) 可选，指定导出的 epoch 数默认为评估效果最好的 epoch
                quant(bool,optional) 可选，type 为 infer 有效，是否导出量化后的模型，默认为
False
                model_path(str,optional) 可选，type 为 lite 时有效，inference 模型的地址
        Return:
                status
                if type == 'infer':
                        save_dir: 模型保存路径
                if type == 'lite':
                        message: 模型保存信息
        Example:
                # 导出 inference 模型
                params = {tid:'T0001', type:'infer', save_dir:'path to save', 'quant':␣
↪False}
                ret = requests.post(url + '/project/task/export',json=params)
methods=='PUT':# 停止导出模型
        Args:
                tid(str): 任务 id
        Return:
                export_status(int): 模型导出状态 (PredictStatus) 枚举变量的值
                message(str): 停止模型导出提示信息
                status
        Example:
                # 停止导出模型
                params = {tid:'T0001'}
                ret = requests.put(url + '/project/task/export',json=params)
```

/project/task/vdl [GET]

打开任务的可视化分析工具 (VisualDL)

- GET 请求: 打开任务的 vdl

```
methods=='GET':# 打开某个任务的可视化分析工具 (VisualDL)
        Args:
                tid(str): 任务 id
        Return:
                url(str):vdl 地址
                status
        Example:
```

(下页继续)

(续上页)

```
                # 打开任务 T0002 的 vdl
                params = {tid:'T0002'}
                ret = requests.get(url + '/project/task/vdl',json=params)
                # 获取打开 vdl 的 url
                url = ret.json()['url']
```

/system [GET]

获取系统信息包括 CPU、GPU 信息

- GET 请求：获取系统信息、在 paddlex 启动 restful 时会自行调用此 api、若需要使用 GPU 请确定已经安装好了 pycuda 包

```
methods=='GET':# 获取系统 GPU、CPU 信息
        Args:
                type(str):[machine_info,gpu_memory_size] 选择需要获取的系统信息
        Return:
                status
                if type=='machine_info'
                        info(dict): 服务端信息
                if type=='gpu_memory_size'
                        gpu_mem_infos(list):GPU 内存信息
        Example1:
                # 获取服务器 gpu 与 cpu 信息
                params = {'type': 'machine_info'}
                ret = requests.get(url + 'system', json=params)
                info  = ret.json()['info']
        Example2:
                # 获取服务器 gpu 显存信息
                params = {'type': 'gpu_memory_size'}
                ret = requests.get(url + 'system', json=params)
                gpu_mem_infos = ret.json()['gpu_mem_infos']
#info(dict): 服务端信息
info={
        message(str): 获取提示信息
        cpu_num(int):cpu 个数
        gpu_num(int):gpu 个数
        sysstr(str): 服务器系统信息
}
#gpu_mem_infos(list):GPU 内存信息，对于 list 里面第 i 个元素表示第 i 个 GPU 的显存信息，包含：
```

(下页继续)

(续上页)

```
        free: 可用显存
        used: 已经使用的显存
        total: 总共的显存
```

/demo [GET,POST,PUT]

创建、获取、删除 demo 项目，

- GET 请求：获取 demo 下载的进度
- POST 请求：下载和载入 demo 项目
- PUT 请求：停止下载 demo 项目

```
methods=='GET':# 获取 demo 下载进度
        Args:
                prj_type(str): 项目类型可以是 ['classification', 'detection',
↪'segmentation','instance_segmentation']
        Return:
                status
                attr(dict):demo 下载信息
        Example:
                # 分类项目示例 demo 下载
                params = {'prj_type' = 'classification'}
                ret = requests.get(url + 'demo', json=params)
#attr(dict):demo 下载信息，包含
attr={
        status(int):demo 下载状态枚举变量 DownloadStatus 的 int 值
        progress(float):demo 下载进度
}

methods=='POST':# 下载或创建 demo 工程
        Args:
                type(str):{download,load} 下载或者创建样例
                prj_type(int): 项目类型 ProjectType 枚举变量的 int 值
        Return:
                status
                if type=='load':
                        did: 数据集 id
                        pid: 项目 id
        Example1:
```

(下页继续)

(续上页)

```
                # 下载分类 demo 样例
                params = {'type': 'download', 'prj_type': 0}
                ret = requests.post(url + 'demo', json=params)
        Example2:
                # 创建分类 demo 样例工程
                params = {'type': 'load', 'prj_type': 0}
                ret = requests.post(url + 'demo', json=params)

methods=='PUT':# 停止下载或创建 demo 工程
        Args:
                prj_type(int): 项目类型 ProjectType 枚举变量的 int 值
        Return:
                status
        Example:
                # 停止分类 demo 下载或者创建
                params = {'prj_type': 0}
                ret = requests.put(url + 'demo', json=params)
```

/model [GET,POST,DELETE]

- GET 请求：获取所有或者单个模型的信息，可以是预训练模型或者 inference 模型
- POST 请求：在 workspace 中创建模型，对于 inference 模型需要先通过/project/task/export 接口现在导出 inference 模型
- DELETE 请求：删除一个模型

```
methods=='GET':# 获取一个或者所有模型的信息
        Args:
                mid(str,optional) 可选，若存在则返回某个模型的信息
                type(str,optional) 可选，[pretrained,exported]. 若存在则返回对应类型下所有
的模型信息
        Return:
                status
                if mid in Args:
                        dataset_attr(dict): 数据集属性
                        task_params(dict): 模型训练参数
                        eval_result(dict): 模型评估结果
                if type in Args and type == 'pretrained':
                        pretrained_models(list): 所有预训练模型信息
```

(下页继续)

(续上页)

```
                if type in Args and type == 'exported':
                        exported_models(list): 所有 inference 模型的信息
        Example:
                # 获取所有保存的预训练模型信息
                params = {'type': 'exported'}
                ret = requests.get(url + 'model', json=params)

methods=='POST':# 创建一个模型
        Args:
                pid(str): 项目 id
                tid(str): 任务 id
                name(str): 模型名字
                type(str): 创建模型的类型,[pretrained,exported],pretrained 代表创建预训练
模型、exported 代表创建 inference 或者 lite 模型
                source_path(str): 仅 type 为 pretrained 时有效，训练好的模型的路径
                path(str): 仅 type 为 exported 时有效，inference 或者 lite 模型的路径
                exported_type(int):0 为 inference 模型，1 为 lite 模型
                eval_results(dict, optional): 可选，仅 type 为 pretrained 时有效，模型评估
的指标
        Return:
                status
                if type == 'pretrained':
                        pmid(str): 预训练模型 id
                if type == 'exported':
                        emid(str):inference 模型 id
        Exampe:
                # 创建一个预训练模型
                params={
                        pid : 'P0001',
                        tid : 'T0001',
                        name : 'Pretrain_model',
                        type : 'pretrained',
                        source_path : '/path/to/pretrian_model',
                }
                ret = requests.post(url + 'model', json=params)

methods=='DELETE': 删除一个模型
        Args:
                type(str): 删除模型的类型，[pretrained,exported]，pretrained 代表创建预训练
模型、exported 代表创建 inference 或者 lite 模型
```

(下页继续)

(续上页)

```
                if type='pretrained':
                        pmid: 预训练模型 id
                if type='exported':
                        emid:inference 或者 lite 模型 id
        Return:
                status
        Example:
                # 删除模型 id 为 EM0001 的 inference 模型
                params = {'type': 'exported', 'emid': 'EM0001'}
                ret = requests.delete(url + 'model', json=params)
```

28.3.5 数据结构

本文档用于说明 PaddleX RESTful 模块自定义的数据结构

Protobuf 结构化数据

```
message Dataset {
    string id = 1;
    string name = 2;
    string desc = 3;
    // 'classification': 分类数据
    // 'segmentation': 分割数据
    // 'detection_voc': 检测数据（仅用于检测）
    // 'detection_coco': 检测数据（用于检测，分割，实例分割）
    string type = 4;
    string path = 5;
    string create_time = 6;
}

message Project {
    string id = 1;
    string name = 2;
    string desc = 3;
    // 'classification'
    // 'segmentation'
    // 'segmentation'
    // 'instance_segmentation'
    // 'remote_segmentation'
```

(下页继续)

(续上页)

```
    string type = 4;
    string did = 5;
    string path = 6;
    string create_time = 7;
}

message Task {
    string id = 1;
    string name = 2;
    string desc = 3;
    string pid = 4;
    string path = 5;
    string create_time = 6;
    //裁剪任务 id
    string parent_id = 7;
}

message PretrainedModel {
    string id = 1;
    string name = 2;
    string model = 3;
    string type = 4;
    // 所属项目 id
    string pid = 5;
    string tid = 6;
    string create_time = 7;
    string path = 8;
}

message ExportedModel {
    string id = 1;
    string name = 2;
    string model = 3;
    string type = 4;
    // 所属项目 id
    string pid = 5;
    string tid = 6;
    string create_time = 7;
    string path = 8;
    int32 exported_type = 9;
```

(下页继续)

(续上页)

```
}

message Workspace {
    string version = 1;
    string path = 2;
    map<string, Dataset> datasets = 3;
    map<string, Project> projects = 4;
    map<string, Task> tasks = 5;
    int32 max_dataset_id = 6;
    int32 max_project_id = 7;
    int32 max_task_id = 8;
    string current_time = 9;

    int32 max_pretrained_model_id = 10;
    map<string, PretrainedModel> pretrained_models = 11;

    int32 max_exported_model_id = 12;
    map<string, ExportedModel> exported_models = 13;
}
```

状态枚举变量

DatasetStatus(数据集状态变量)

```
DatasetStatus = Enum('DatasetStatus',
('XEMPTY', # 空数据集
'XCHECKING', # 正在验证数据集
'XCHECKFAIL', # 数据集验证失败
'XCOPYING',  # 正在导入数据集
'XCOPYDONE', # 数据集导入成功
'XCOPYFAIL', # 数据集导入失败
'XSPLITED' # 数据集已经切分
),start=0)
```

TaskStatus(任务状态变量)

```
TaskStatus = Enum('TaskStatus',
('XUNINIT',# 任务还未初始化
```

(下页继续)

(续上页)

```
'XINIT',# 任务初始化
'XDOWNLOADING',# 正在下载预训练模型
'XTRAINING',# 任务正在运行中，改状态下任务不能被删除
'XTRAINDONE',# 任务完成运行
'XEVALUATED',# 任务评估完成
'XEXPORTING',# 任务正在导出 inference 模型
'XEXPORTED',# 任务已经导出模型
'XTRAINEXIT', # 任务运行中止
'XDOWNLOADFAIL', # 任务下载失败
'XTRAINFAIL', # 任务运行失败
'XEVALUATING',# 任务评估中
'XEVALUATEFAIL', # 任务评估失败
'XEXPORTFAIL', # 任务导出模型失败
'XPRUNEING', # 裁剪分析任务运行中
'XPRUNETRAIN'# 裁剪训练任务运行中
),start=0)
```

ProjectType(项目类型)

```
ProjectType = Enum('ProjectType',
('classification',# 分类
'detection',# 检测
'segmentation',# 分割
'instance_segmentation',# 实例分割
'remote_segmentation'# 遥感分割
),start=0)
```

DownloadStatus(下载状态变量)

```
DownloadStatus = Enum('DownloadStatus',
('XDDOWNLOADING',# 下载中
'XDDOWNLOADFAIL',# 下载失败
'XDDOWNLOADDONE', 下载完成
'XDDECOMPRESSED'解压完成
),start=0)
```

PruneStatus(裁剪状态变量)

```
PruneStatus = Enum('PruneStatus',
('XSPRUNESTART',# 启动裁剪任务
'XSPRUNEING',# 正在裁剪模型
'XSPRUNEDONE',# 已完成裁剪任务
'XSPRUNEFAIL',# 裁剪任务失败
'XSPRUNEEXIT',# 裁剪任务已经停止
),start=0)
```

PredictStatus(预测任务状态变量)

```
PredictStatus = Enum('PredictStatus',
('XPRESTART',# 预测开始
'XPREDONE',# 预测完成
'XPREFAIL'# 预测失败
), start=0)
```

PretrainedModelStatus(预训练模型状态变量)

```
PretrainedModelStatus = Enum('PretrainedModelStatus',
('XPINIT', # 初始化
'XPSAVING', # 正在保存
'XPSAVEFAIL',# 保存失败
'XPSAVEDONE' # 保存完成
),start=0)
```

ExportedModelType(模型导出状态变量)

```
ExportedModelType = Enum('ExportedModelType',
('XQUANTMOBILE',# 量化后的 lite 模型
'XPRUNEMOBILE', # 裁剪后的 lite 模型
'XTRAINMOBILE',#lite 模型
'XQUANTSERVER', # 量化后的 inference 模型
'XPRUNESERVER', # 裁剪后的 inference 模型
'XTRAINSERVER'#inference 模型
),start=0)
```

28.3.6 RESTful 目录结构

介绍了 PaddleX RESTful 的整体目录结构

```
restful
|____system.py        // 处理/system 请求
|____workspace_pb2.py
|____dir.py
|____dataset        // 数据集相关
| |____datasetbase.py        // 数据集基类
| |______init__.py
| |____seg_dataset.py        // 分割数据集
| |____cls_dataset.py        // 分类数据集
| |____dataset.py        // 处理/dataset 请求
| |____operate.py        // 数据集基础操作函数
| |____utils.py        // 数据集基础工具函数
| |____ins_seg_dataset.py        // 示例分割数据集
| |____det_dataset.py        // 检测数据集
|______init__.py
|____model.py        // 处理/model 请求
|____project        // 项目相关
| |____task.py        // 处理/project/task 请求
| |______init__.py
| |____visualize.py        // 数据可视化
| |____operate.py        // 任务基础操作函数
| |____evaluate        // 模型评估
| | |____detection.py        // 检测模型评估
| | |____classification.py        // 分类模型评估
| | |______init__.py
| | |____draw_pred_result.py        // 评估与预测结果可视化
| | |____segmentation.py        // 分割模型评估
| |____train        //模型训练
| | |____detection.py        // 检测模型训练
| | |____params.py        // 模型参数
| | |____classification.py        // 分类模型训练
| | |______init__.py
| | |____params_v2.py        // 模型参数 V2 版本
| | |____segmentation.py        // 分割模型训练
| |____prune        // 模型剪裁
| | |____detection.py        // 检测模型剪裁
| | |____classification.py        // 分类模型剪裁
```

(下页继续)

(续上页)

```
| | |______init__.py
| | |____segmentation.py        // 分割模型剪裁
| |____project.py        // 处理/project 请求
|____utils.py        // 基础工具函数
|____app.py        // 创建 flask app
|____front_demo        // 前端 demo
|____workspace.py        // 处理/workspace 请求
|____demo.py        // 处理/demo 请求
|____workspace.proto        // workspace 结构化信息
|____frontend_demo // 前端 demo
| |____paddlex_restful_demo.html    //前端 demo 文件
|____templates  // flask html 文件存放目录
| |____paddlex_restful_demo.html    //前端 demo 文件
```

CHAPTER 29

API 接口说明

29.1 数据处理与增强

transforms 为 PaddleX 的模型训练提供了数据的预处理和数据增强接口。

29.1.1 paddlex.cls.transforms

对图像分类任务的数据进行操作。可以利用*Compose*类将图像预处理/增强操作进行组合。

Compose

```
paddlex.cls.transforms.Compose(transforms)
```

根据数据预处理/增强算子对输入数据进行操作。使用示例

参数

- **transforms** (list): 数据预处理/数据增强列表。

Normalize

```
paddlex.cls.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225],␣
↪min_val=[0, 0, 0], max_val=[255.0, 255.0, 255.0])
```

对图像进行标准化。

1. 像素值减去 min_val 2. 像素值除以 (max_val-min_val), 归一化到区间 [0.0, 1.0]。3. 对图像进行减均值除以标准差操作。

参数

- **mean** (list): 图像数据集的均值。默认为 [0.485, 0.456, 0.406]。长度应与图像通道数量相同。
- **std** (list): 图像数据集的标准差。默认为 [0.229, 0.224, 0.225]。长度应与图像通道数量相同。
- **min_val** (list): 图像数据集的最小值。默认值 [0, 0, 0]。长度应与图像通道数量相同。
- **max_val** (list): 图像数据集的最大值。默认值 [255.0, 255.0, 255.0]。长度应与图像通道数量相同。

ResizeByShort

```
paddlex.cls.transforms.ResizeByShort(short_size=256, max_size=-1)
```

根据图像的短边调整图像大小（resize）。

1. 获取图像的长边和短边长度。
2. 根据短边与 short_size 的比例，计算长边的目标长度，此时高、宽的 resize 比例为 short_size/原图短边长度。
3. 如果 max_size>0，调整 resize 比例：如果长边的目标长度 >max_size，则高、宽的 resize 比例为 max_size/原图长边长度。
4. 根据调整大小的比例对图像进行 resize。

参数

- **short_size** (int): 调整大小后的图像目标短边长度。默认为 256。
- **max_size** (int): 长边目标长度的最大限制。默认为-1。

CenterCrop

```
paddlex.cls.transforms.CenterCrop(crop_size=224)
```

以图像中心点扩散裁剪长宽为 `crop_size` 的正方形

1. 计算剪裁的起始点。
2. 剪裁图像。

参数

- **crop_size** (int): 裁剪的目标边长。默认为 224。

RandomCrop

```
paddlex.cls.transforms.RandomCrop(crop_size=224, lower_scale=0.08, lower_ratio=3. / 4,␣
↪upper_ratio=4. / 3)
```

对图像进行随机剪裁，模型训练时的数据增强操作。

1. 根据 lower_scale、lower_ratio、upper_ratio 计算随机剪裁的高、宽。
2. 根据随机剪裁的高、宽随机选取剪裁的起始点。
3. 剪裁图像。
4. 调整剪裁后的图像的大小到 crop_size*crop_size。

参数

- **crop_size** (int): 随机裁剪后重新调整的目标边长。默认为 224。
- **lower_scale** (float): 裁剪面积相对原面积比例的最小限制。默认为 0.08。
- **lower_ratio** (float): 宽变换比例的最小限制。默认为 3. / 4。
- **upper_ratio** (float): 宽变换比例的最大限制。默认为 4. / 3。

RandomHorizontalFlip

```
paddlex.cls.transforms.RandomHorizontalFlip(prob=0.5)
```

以一定的概率对图像进行随机水平翻转，模型训练时的数据增强操作。

参数

- **prob** (float): 随机水平翻转的概率。默认为 0.5。

RandomVerticalFlip

```
paddlex.cls.transforms.RandomVerticalFlip(prob=0.5)
```

以一定的概率对图像进行随机垂直翻转，模型训练时的数据增强操作。

参数

- **prob** (float): 随机垂直翻转的概率。默认为 0.5。

RandomRotate

```
paddlex.cls.transforms.RandomRotate(rotate_range=30, prob=0.5)
```

以一定的概率对图像在 [-rotate_range, rotaterange] 角度范围内进行旋转，模型训练时的数据增强操作。

参数

- **rotate_range** (int): 旋转度数的范围。默认为 30。
- **prob** (float): 随机旋转的概率。默认为 0.5。

RandomDistort

```
paddlex.cls.transforms.RandomDistort(brightness_range=0.9, brightness_prob=0.5, contrast_
↪range=0.9, contrast_prob=0.5, saturation_range=0.9, saturation_prob=0.5, hue_range=18,␣
↪hue_prob=0.5)
```

以一定的概率对图像进行随机像素内容变换，模型训练时的数据增强操作。

1. 对变换的操作顺序进行随机化操作。
2. 按照 1 中的顺序以一定的概率对图像进行随机像素内容变换。

【注意】如果输入是 uint8/uint16 的 RGB 图像，该数据增强必须在数据增强 Normalize 之前使用。

参数

- **brightness_range** (float): 明亮度的缩放系数范围。从 [1-`brightness_range`, 1+`brightness_range`] 中随机取值作为明亮度缩放因子 `scale`，按照公式 `image = image * scale` 调整图像明亮度。默认值为 0.9。
- **brightness_prob** (float): 随机调整明亮度的概率。默认为 0.5。
- **contrast_range** (float): 对比度的缩放系数范围。从 [1-`contrast_range`, 1+`contrast_range`] 中随机取值作为对比度缩放因子 `scale`，按照公式 `image = image * scale + (image_mean + 0.5) * (1 - scale)` 调整图像对比度。默认为 0.9。
- **contrast_prob** (float): 随机调整对比度的概率。默认为 0.5。

- **saturation_range** (float): 饱和度的缩放系数范围。从 [1-saturation_range, 1+saturation_range] 中随机取值作为饱和度缩放因子 scale，按照公式 image = gray * (1 - scale) + image * scale，其中 gray = R * 299/1000 + G * 587/1000+ B * 114/1000。默认为 0.9。
- **saturation_prob** (float): 随机调整饱和度的概率。默认为 0.5。
- **hue_range** (int): 调整色相角度的差值取值范围。从 [-hue_range, hue_range] 中随机取值作为色相角度调整差值 delta，按照公式 hue = hue + delta 调整色相角度。默认为 18，取值范围 [0, 360]。
- **hue_prob** (float): 随机调整色调的概率。默认为 0.5。

29.1.2 paddlex.det.transforms

对目标检测/实例分割任务的数据进行操作。可以利用*Compose*类将图像预处理/增强操作进行组合。

Compose

```
paddlex.det.transforms.Compose(transforms)
```

根据数据预处理/增强算子对输入数据进行操作。使用示例

参数

- **transforms** (list): 数据预处理/数据增强列表。

Normalize

```
paddlex.det.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225],␣
↪min_val=[0., 0., 0.], max_val=[255., 255., 255.])
```

对图像进行标准化。1. 像素值减去 min_val 2. 像素值除以 (max_val-min_val), 归一化到区间 [0.0, 1.0]。3. 对图像进行减均值除以标准差操作。

参数

- **mean** (list): 图像数据集的均值。默认为 [0.485, 0.456, 0.406]。长度应与图像通道数量相同。
- **std** (list): 图像数据集的标准差。默认为 [0.229, 0.224, 0.225]。长度应与图像通道数量相同。
- **min_val** (list): 图像数据集的最小值。默认值 [0, 0, 0]。长度应与图像通道数量相同。
- **max_val** (list): 图像数据集的最大值。默认值 [255.0, 255.0, 255.0]。长度应与图像通道数量相同。

ResizeByShort

```
paddlex.det.transforms.ResizeByShort(short_size=800, max_size=1333)
```

根据图像的短边调整图像大小（resize）。

1. 获取图像的长边和短边长度。
2. 根据短边与 short_size 的比例，计算长边的目标长度，此时高、宽的 resize 比例为 short_size/原图短边长度。若 short_size 为数组，则随机从该数组中挑选一个数值作为 short_size。
3. 如果 max_size>0，调整 resize 比例：如果长边的目标长度 >max_size，则高、宽的 resize 比例为 max_size/原图长边长度。
4. 根据调整大小的比例对图像进行 resize。

参数

- **short_size** (int|list): 短边目标长度。默认为 800。当需要做多尺度训练时，可以将 `short_size` 设置成数组，例如 [500, 600, 700, 800]。
- **max_size** (int): 长边目标长度的最大限制。默认为 1333。

Padding

```
paddlex.det.transforms.Padding(coarsest_stride=1)
```

将图像的长和宽 padding 至 coarsest_stride 的倍数。如输入图像为 [300, 640], `coarest_stride` 为 32，则由于 300 不为 32 的倍数，因此在图像最右和最下使用 0 值进行 padding，最终输出图像为 [320, 640]

1. 如果 coarsest_stride 为 1 则直接返回。
2. 计算宽和高与最邻近的 coarest_stride 倍数差值
3. 根据计算得到的差值，在图像最右和最下进行 padding

参数

- **coarsest_stride** (int): 填充后的图像长、宽为该参数的倍数，默认为 1。

Resize

```
paddlex.det.transforms.Resize(target_size=608, interp='LINEAR')
```

调整图像大小（resize）。

- 当目标大小（target_size）类型为 int 时，根据插值方式，将图像 resize 为 [target_size, target_size]。
- 当目标大小（target_size）类型为 list 或 tuple 时，根据插值方式，将图像 resize 为 target_size。【注意】当插值方式为“RANDOM”时，则随机选取一种插值方式进行 resize，作为模型训练时的数据增强操作。

参数

- **target_size** (int/list/tuple): 短边目标长度。默认为 608。
- **interp** (str): resize 的插值方式，与 opencv 的插值方式对应，取值范围为 [' NEAREST'，'LINEAR'，'CUBIC'，'AREA'，'LANCZOS4'，'RANDOM']。默认为" LINEAR"。

RandomHorizontalFlip

```
paddlex.det.transforms.RandomHorizontalFlip(prob=0.5)
```

以一定的概率对图像进行随机水平翻转，模型训练时的数据增强操作。

参数

- **prob** (float): 随机水平翻转的概率。默认为 0.5。

RandomDistort

```
paddlex.det.transforms.RandomDistort(brightness_range=0.5, brightness_prob=0.5, contrast_
↪range=0.5, contrast_prob=0.5, saturation_range=0.5, saturation_prob=0.5, hue_range=18,␣
↪hue_prob=0.5)
```

以一定的概率对图像进行随机像素内容变换，模型训练时的数据增强操作。

1. 对变换的操作顺序进行随机化操作。
2. 按照 1 中的顺序以一定的概率对图像进行随机像素内容变换。

【注意】如果输入是 uint8/uint16 的 RGB 图像，该数据增强必须在数据增强 Normalize 之前使用。

参数

- **brightness_range** (float): 明亮度的缩放系数范围。从 [1-`brightness_range`, 1+`brightness_range`] 中随机取值作为明亮度缩放因子 `scale`，按照公式 `image = image * scale` 调整图像明亮度。默认值为 0.5。
- **brightness_prob** (float): 随机调整明亮度的概率。默认为 0.5。

- **contrast_range** (float): 对比度的缩放系数范围。从 [1-contrast_range, 1+contrast_range] 中随机取值作为对比度缩放因子 scale，按照公式 image = image * scale + (image_mean + 0.5) * (1 - scale) 调整图像对比度。默认为 0.5。
- **contrast_prob** (float): 随机调整对比度的概率。默认为 0.5。
- **saturation_range** (float): 饱和度的缩放系数范围。从 [1-saturation_range, 1+saturation_range] 中随机取值作为饱和度缩放因子 scale，按照公式 image = gray * (1 - scale) + image * scale，其中 gray = R * 299/1000 + G * 587/1000+ B * 114/1000。默认为 0.5。
- **saturation_prob** (float): 随机调整饱和度的概率。默认为 0.5。
- **hue_range** (int): 调整色相角度的差值取值范围。从 [-hue_range, hue_range] 中随机取值作为色相角度调整差值 delta，按照公式 hue = hue + delta 调整色相角度。默认为 18，取值范围 [0, 360]。
- **hue_prob** (float): 随机调整色调的概率。默认为 0.5。

MixupImage

```
paddlex.det.transforms.MixupImage(alpha=1.5, beta=1.5, mixup_epoch=-1)
```

对图像进行 mixup 操作，模型训练时的数据增强操作，目前仅 YOLOv3 模型支持该 transform。当 label_info 中不存在 mixup 字段时，直接返回，否则进行下述操作：

1. 从随机 beta 分布中抽取出随机因子 factor。
2. 根据不同情况进行处理：
   - 当 factor>=1.0 时，去除 label_info 中的 mixup 字段，直接返回。
   - 当 factor<=0.0 时，直接返回 label_info 中的 mixup 字段，并在 label_info 中去除该字段。
   - 其余情况，执行下述操作：（1）原图像乘以 factor，mixup 图像乘以 (1-factor)，叠加 2 个结果。（2）拼接原图像标注框和 mixup 图像标注框。（3）拼接原图像标注框类别和 mixup 图像标注框类别。（4）原图像标注框混合得分乘以 factor，mixup 图像标注框混合得分乘以 (1-factor)，叠加 2 个结果。
3. 更新 im_info 中的 augment_shape 信息。

参数

- **alpha** (float): 随机 beta 分布的下限。默认为 1.5。
- **beta** (float): 随机 beta 分布的上限。默认为 1.5。
- **mixup_epoch** (int): 在前 mixup_epoch 轮使用 mixup 增强操作；当该参数为-1 时，该策略不会生效。默认为-1。

RandomExpand

```
paddlex.det.transforms.RandomExpand(ratio=4., prob=0.5, fill_value=[123.675, 116.28, 103.
↪53])
```

随机扩张图像，模型训练时的数据增强操作。

1. 随机选取扩张比例（扩张比例大于 1 时才进行扩张）。
2. 计算扩张后图像大小。
3. 初始化像素值为输入填充值的图像，并将原图像随机粘贴于该图像上。
4. 根据原图像粘贴位置换算出扩张后真实标注框的位置坐标。
5. 根据原图像粘贴位置换算出扩张后真实分割区域的位置坐标。

参数

- **ratio** (float): 图像扩张的最大比例。默认为 4.0。
- **prob** (float): 随机扩张的概率。默认为 0.5。
- **fill_value** (list): 扩张图像的初始填充值（0-255）。默认为 [123.675, 116.28, 103.53]。

【注意】该数据增强必须在数据增强 Resize、ResizeByShort 之前使用。

RandomCrop

```
paddlex.det.transforms.RandomCrop(aspect_ratio=[.5, 2.], thresholds=[.0, .1, .3, .5, .7,␣
↪.9], scaling=[.3, 1.], num_attempts=50, allow_no_crop=True, cover_all_box=False)
```

随机裁剪图像，模型训练时的数据增强操作。

1. 若 allow_no_crop 为 True，则在 thresholds 加入’ no_crop’。
2. 随机打乱 thresholds。
3. 遍历 thresholds 中各元素：(1) 如果当前 thresh 为’ no_crop’，则返回原始图像和标注信息。(2) 随机取出 aspect_ratio 和 scaling 中的值并由此计算出候选裁剪区域的高、宽、起始点。(3) 计算真实标注框与候选裁剪区域 IoU，若全部真实标注框的 IoU 都小于 thresh，则继续第 3 步。(4) 如果 cover_all_box 为 True 且存在真实标注框的 IoU 小于 thresh，则继续第 3 步。(5) 筛选出位于候选裁剪区域内的真实标注框，若有效框的个数为 0，则继续第 3 步，否则进行第 4 步。
4. 换算有效真值标注框相对候选裁剪区域的位置坐标。
5. 换算有效分割区域相对候选裁剪区域的位置坐标。

【注意】该数据增强必须在数据增强 Resize、ResizeByShort 之前使用。

参数

- **aspect_ratio** (list): 裁剪后短边缩放比例的取值范围，以 [min, max] 形式表示。默认值为 [.5, 2.]。
- **thresholds** (list): 判断裁剪候选区域是否有效所需的 IoU 阈值取值列表。默认值为 [.0, .1, .3, .5, .7, .9]。
- **scaling** (list): 裁剪面积相对原面积的取值范围，以 [min, max] 形式表示。默认值为 [.3, 1.]。
- **num_attempts** (int): 在放弃寻找有效裁剪区域前尝试的次数。默认值为 50。
- **allow_no_crop** (bool): 是否允许未进行裁剪。默认值为 True。
- **cover_all_box** (bool): 是否要求所有的真实标注框都必须在裁剪区域内。默认值为 False。

CLAHE

```
paddlex.det.transforms.CLAHE(clip_limit=2., tile_grid_size=(8, 8))
```

对图像进行对比度增强。

【注意】该数据增强只适用于灰度图。

参数

- **clip_limit** (int|float): 颜色对比度的阈值，默认值为 2.。
- **tile_grid_size** (list|tuple): 进行像素均衡化的网格大小。默认值为 (8, 8)。

29.1.3 paddlex.seg.transforms

对用于分割任务的数据进行操作。可以利用*Compose*类将图像预处理/增强操作进行组合。

Compose

```
paddlex.seg.transforms.Compose(transforms)
```

根据数据预处理/数据增强列表对输入数据进行操作。使用示例

参数

- **transforms** (list): 数据预处理/数据增强列表。

RandomHorizontalFlip

```
paddlex.seg.transforms.RandomHorizontalFlip(prob=0.5)
```

以一定的概率对图像进行水平翻转，模型训练时的数据增强操作。

参数

- **prob** (float): 随机水平翻转的概率。默认值为 0.5。

RandomVerticalFlip

```
paddlex.seg.transforms.RandomVerticalFlip(prob=0.1)
```

以一定的概率对图像进行垂直翻转，模型训练时的数据增强操作。

参数

- **prob** (float): 随机垂直翻转的概率。默认值为 0.1。

Resize

```
paddlex.seg.transforms.Resize(target_size, interp='LINEAR')
```

调整图像大小（resize）。

- 当目标大小（target_size）类型为 int 时，根据插值方式，将图像 resize 为 [target_size, target_size]。
- 当目标大小（target_size）类型为 list 或 tuple 时，根据插值方式，将图像 resize 为 target_size, target_size 的输入应为 [w, h] 或（w, h）。

参数

- **target_size** (int|list|tuple): 目标大小
- **interp** (str): resize 的插值方式，与 opencv 的插值方式对应，可选的值为 [' NEAREST' ，'LINEAR' ，'CUBIC' ，'AREA' ，'LANCZOS4']，默认为" LINEAR"。

ResizeByLong

```
paddlex.seg.transforms.ResizeByLong(long_size)
```

对图像长边 resize 到固定值，短边按比例进行缩放。

参数

- **long_size** (int): resize 后图像的长边大小。

ResizeRangeScaling

```
paddlex.seg.transforms.ResizeRangeScaling(min_value=400, max_value=600)
```

对图像长边随机 resize 到指定范围内，短边按比例进行缩放，模型训练时的数据增强操作。

参数

- **min_value** (int): 图像长边 resize 后的最小值。默认值 400。
- **max_value** (int): 图像长边 resize 后的最大值。默认值 600。

ResizeStepScaling

```
paddlex.seg.transforms.ResizeStepScaling(min_scale_factor=0.75, max_scale_factor=1.25,␣
↪scale_step_size=0.25)
```

对图像按照某一个比例 resize，这个比例以 scale_step_size 为步长，在 [min_scale_factor, max_scale_factor] 随机变动，模型训练时的数据增强操作。

参数

- **min_scale_factor** (float), resize 最小尺度。默认值 0.75。
- **max_scale_factor** (float), resize 最大尺度。默认值 1.25。
- **scale_step_size** (float), resize 尺度范围间隔。默认值 0.25。

Normalize

```
paddlex.seg.transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5], min_val=[0,␣
↪0, 0], max_val=[255.0, 255.0, 255.0])
```

对图像进行标准化。

1. 像素值减去 min_val 2. 像素值除以 (max_val-min_val), 归一化到区间 [0.0, 1.0]。3. 对图像进行减均值除以标准差操作。

参数

- **mean** (list): 图像数据集的均值。默认值 [0.5, 0.5, 0.5]。长度应与图像通道数量相同。
- **std** (list): 图像数据集的标准差。默认值 [0.5, 0.5, 0.5]。长度应与图像通道数量相同。
- **min_val** (list): 图像数据集的最小值。默认值 [0, 0, 0]。长度应与图像通道数量相同。
- **max_val** (list): 图像数据集的最大值。默认值 [255.0, 255.0, 255.0]。长度应与图像通道数量相同。

Padding

```
paddlex.seg.transforms.Padding(target_size, im_padding_value=[127.5, 127.5, 127.5],␣
↪label_padding_value=255)
```

对图像或标注图像进行 padding，padding 方向为右和下。根据提供的值对图像或标注图像进行 padding 操作。

参数

- **target_size** (int|list|tuple): padding 后图像的大小。
- **im_padding_value** (list): 图像 padding 的值。默认为 [127.5, 127.5, 127.5]。长度应与图像通道数量相同。
- **label_padding_value** (int): 标注图像 padding 的值。默认值为 255（仅在训练时需要设定该参数）。

RandomPaddingCrop

```
paddlex.seg.transforms.RandomPaddingCrop(crop_size=512, im_padding_value=[127.5, 127.5,␣
↪127.5], label_padding_value=255)
```

对图像和标注图进行随机裁剪，当所需要的裁剪尺寸大于原图时，则进行 padding 操作，模型训练时的数据增强操作。

参数

- **crop_size**（int|list|tuple): 裁剪图像大小。默认为 512。

- **im_padding_value** (list): 图像 padding 的值。默认为 [127.5, 127.5, 127.5]。长度应与图像通道数量相同。
- **label_padding_value** (int): 标注图像 padding 的值。默认值为 255。

RandomBlur

```
paddlex.seg.transforms.RandomBlur(prob=0.1)
```

以一定的概率对图像进行高斯模糊，模型训练时的数据增强操作。

参数

- **prob** (float): 图像模糊概率。默认为 0.1。

RandomRotate

```
paddlex.seg.transforms.RandomRotate(rotate_range=15, im_padding_value=[127.5, 127.5, 127.
↪5], label_padding_value=255)
```

对图像进行随机旋转，模型训练时的数据增强操作。目前支持多通道的 RGB 图像，例如支持多张 RGB 图像沿通道轴做 concatenate 后的图像数据，不支持通道数量不是 3 的倍数的图像数据。

在旋转区间 [-rotate_range, rotate_range] 内，对图像进行随机旋转，当存在标注图像时，同步进行，并对旋转后的图像和标注图像进行相应的 padding。

参数

- **rotate_range** (float): 最大旋转角度。默认为 15 度。
- **im_padding_value** (list): 图像 padding 的值。默认为 [127.5, 127.5, 127.5]。长度应与图像通道数量相同。
- **label_padding_value** (int): 标注图像 padding 的值。默认为 255。

RandomScaleAspect

```
paddlex.seg.transforms.RandomScaleAspect(min_scale=0.5, aspect_ratio=0.33)
```

裁剪并 resize 回原始尺寸的图像和标注图像，模型训练时的数据增强操作。

按照一定的面积比和宽高比对图像进行裁剪，并 reszie 回原始图像的图像，当存在标注图时，同步进行。

参数

- **min_scale** (float)：裁取图像占原始图像的面积比，取值 [0，1]，为 0 时则返回原图。默认为 0.5。
- **aspect_ratio** (float)：裁取图像的宽高比范围，非负值，为 0 时返回原图。默认为 0.33。

RandomDistort

```
paddlex.seg.transforms.RandomDistort(brightness_range=0.5, brightness_prob=0.5, contrast_
↪range=0.5, contrast_prob=0.5, saturation_range=0.5, saturation_prob=0.5, hue_range=18,␣
↪hue_prob=0.5)
```

以一定的概率对图像进行随机像素内容变换，模型训练时的数据增强操作。

【注意】如果输入是 uint8/uint16 的 RGB 图像，该数据增强必须在数据增强 Normalize 之前使用。如果输入是由多张 RGB 图像数据沿通道方向做拼接而成的图像数据，则会把每 3 个通道数据视为一张 RGB 图像数据，依次对每 3 个通道数据做随机像素内容变化。

1. 对变换的操作顺序进行随机化操作。
2. 按照 1 中的顺序以一定的概率对图像进行随机像素内容变换。

参数

- **brightness_range** (float): 明亮度的缩放系数范围。从 [1-brightness_range, 1+brightness_range] 中随机取值作为明亮度缩放因子 `scale`，按照公式 `image = image * scale` 调整图像明亮度。默认值为 0.5。
- **brightness_prob** (float): 随机调整明亮度的概率。默认为 0.5。
- **contrast_range** (float): 对比度的缩放系数范围。从 [1-contrast_range, 1+contrast_range] 中随机取值作为对比度缩放因子 `scale`，按照公式 `image = image * scale + (image_mean + 0.5) * (1 - scale)` 调整图像对比度。默认为 0.5。
- **contrast_prob** (float): 随机调整对比度的概率。默认为 0.5。
- **saturation_range** (float): 饱和度的缩放系数范围。从 [1-saturation_range, 1+saturation_range] 中随机取值作为饱和度缩放因子 `scale`，按照公式 `image = gray * (1 - scale) + image * scale`，其中 `gray = R * 299/1000 + G * 587/1000+ B * 114/1000`。默认为 0.5。
- **saturation_prob** (float): 随机调整饱和度的概率。默认为 0.5。
- **hue_range** (int): 调整色相角度的差值取值范围。从 [-hue_range, hue_range] 中随机取值作为色相角度调整差值 `delta`，按照公式 `hue = hue + delta` 调整色相角度。默认为 18，取值范围 [0, 360]。
- **hue_prob** (float): 随机调整色调的概率。默认为 0.5。

Clip

```
paddlex.seg.transforms.Clip(min_val=[0, 0, 0], max_val=[255.0, 255.0, 255.0])
```

对图像上超出一定范围的数据进行截断。

参数

- **min_val** (list): 裁剪的下限，小于 min_val 的数值均设为 min_val. 默认值 0。
- **max_val** (list): 裁剪的上限，大于 max_val 的数值均设为 max_val. 默认值 255.0。

29.1.4 数据增强与 imgaug 支持

数据增强操作可用于在模型训练时，增加训练样本的多样性，从而提升模型的泛化能力。

PaddleX 内置增强操作

PaddleX 对于图像分类、目标检测、实例分割和语义分割内置了部分常见的数据增强操作，如下表所示，

imgaug 增强库的支持

PaddleX 目前已适配 imgaug 图像增强库，用户可以直接在 PaddleX 构造 `transforms` 时，调用 imgaug 的方法，如下示例，

```
import paddlex as pdx
from paddlex.cls import transforms
import imgaug.augmenters as iaa
train_transforms = transforms.Compose([
    # 随机在 [0.0 3.0] 中选值对图像进行模糊
    iaa.blur.GaussianBlur(sigma=(0.0, 3.0)),
    transforms.RandomCrop(crop_size=224),
    transforms.Normalize()
])
```

除了上述用法，`Compose` 接口中也支持 imgaug 的 `Someof`、`Sometimes`、`Sequential`、`Oneof` 等操作，开发者可以通过这些方法随意组合出增强流程。由于 imgaug 对于标注信息 (目标检测框和实例分割 mask) 与 PaddleX 模型训练逻辑有部分差异，**目前在检测和分割中，只支持 pixel-level 的增强方法，(即在增强时，不对图像的大小和方向做改变) 其它方法仍在适配中**，详情可见下表，

需要注意的是，imgaug 的基础方法中，如 `imgaug.augmenters.blur` 仅为图像处理操作，并无概率设置，而在 CV 模型训练中，增强操作往往是以一定概率应用在样本上，因此我们可以通过 imgaug 的 `Someof`、`Sometimes`、`Sequential`、`Oneof` 等操作来组合实现，如下代码所示，

- Someof 执行定义增强方法列表中的部分方法
- Sometimes 以一定概率执行定义的增强方法列表
- Sequential 按顺序执行定义的增强方法列表

```
image imgaug.augmenters as iaa
from paddlex.cls import transforms
# 以 0.6 的概率对图像样本进行模糊
img_augmenters = iaa.Sometimes(0.6, [
    iaa.blur.GaussianBlur(sigma=(0.0, 3.0))
])
train_transforms = transforms.Compose([
    img_augmenters,
    transforms.RandomCrop(crop_size=224),
    transforms.Normalize()
])
```

29.2 数据集读取

29.2.1 paddlex.datasets.ImageNet

用于图像分类模型

```
paddlex.datasets.ImageNet(data_dir, file_list, label_list, transforms=None, num_workers=
'auto' , buffer_size=8, parallel_method='process', shuffle=False)
```

读取 ImageNet 格式的分类数据集，并对样本进行相应的处理。ImageNet 数据集格式的介绍可查看文档:数据集格式说明

示例：代码文件

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和类别 id 的文件路径（文本内每行路径为相对 data_dir 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.cls.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.cls.transforms。
- **num_workers** (int|str)：数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 num_workers: 如果 CPU 核数的一半

大于 8，则 num_workers 为 8，否则为 CPU 核数的一半。

- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 8。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.2.2 paddlex.datasets.VOCDetection

用于目标检测模型

```
paddlex.datasets.VOCDetection(data_dir, file_list, label_list, transforms=None, num_
↪workers= 'auto' , buffer_size=100, parallel_method='process', shuffle=False)
```

读取 PascalVOC 格式的检测数据集，并对样本进行相应的处理。PascalVOC 数据集格式的介绍可查看文档:数据集格式说明

示例：代码文件

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和对应标注文件的文件路径（文本内每行路径为相对 data_dir 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.det.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.det.transforms。
- **num_workers** (int|str)：数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 num_workers: 如果 CPU 核数的一半大于 8，则 num_workers 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

add_negative_samples(self, image_dir)

将背景图片加入训练

- **image_dir** (str): 背景图片所在的文件夹目录。

示例：代码文件

29.2.3 paddlex.datasets.CocoDetection

用于实例分割/目标检测模型

```
paddlex.datasets.CocoDetection(data_dir, ann_file, transforms=None, num_workers='auto',␣
↪buffer_size=100, parallel_method='process', shuffle=False)
```

读取 MSCOCO 格式的检测数据集，并对样本进行相应的处理，该格式的数据集同样可以应用到实例分割模型的训练中。MSCOCO 数据集格式的介绍可查看文档:数据集格式说明

示例：代码文件

参数

- **data_dir** (str): 数据集所在的目录路径。
- **ann_file** (str): 数据集的标注文件，为一个独立的 json 格式文件。
- **transforms** (paddlex.det.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.det.transforms。
- **num_workers** (int|str)：数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 `num_workers`: 如果 CPU 核数的一半大于 8，则 `num_workers` 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

add_negative_samples(self, image_dir)

将背景图片加入训练

- **image_dir** (str): 背景图片所在的文件夹目录。

示例：代码文件

29.2.4 paddlex.datasets.SegDataset

用于语义分割模型

```
paddlex.datasets.SegDataset(data_dir, file_list, label_list, transforms=None, num_
↪workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

读取语义分割任务数据集，并对样本进行相应的处理。语义分割任务数据集格式的介绍可查看文档:数据集格式说明

示例：代码文件

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和对应标注文件的文件路径（文本内每行路径为相对 data_dir 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.seg.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.seg.transforms。
- **num_workers** (int|str)：数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 num_workers: 如果 CPU 核数的一半大于 8，则 num_workers 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.2.5 paddlex.datasets.EasyDataCls

用于图像分类模型

```
paddlex.datasets.EasyDataCls(data_dir, file_list, label_list, transforms=None, num_
↪workers='auto', buffer_size=8, parallel_method='process', shuffle=False)
```

读取 EasyData 平台标注图像分类数据集，并对样本进行相应的处理。

参数

- **data_dir** (str): 数据集所在的目录路径。

- **file_list** (str): 描述数据集图片文件和对应标注文件的文件路径（文本内每行路径为相对 `data_dir` 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.seg.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.cls.transforms。
- **num_workers** (int|str): 数据集中样本在预处理过程中的线程或进程数。默认为'auto'。当设为'auto'时，根据系统的实际 CPU 核数设置 `num_workers`: 如果 CPU 核数的一半大于 8，则 `num_workers` 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 8。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持'thread'线程和'process'进程两种方式。默认为'process'（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.2.6 paddlex.datasets.EasyDataDet

用于**目标检测/实例分割模型**

```
paddlex.datasets.EasyDataDet(data_dir, file_list, label_list, transforms=None, num_
↪workers= 'auto' , buffer_size=100, parallel_method='process', shuffle=False)
```

读取 EasyData 目标检测/实例分割格式数据集，并对样本进行相应的处理，该格式的数据集同样可以应用到实例分割模型的训练中。

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和对应标注文件的文件路径（文本内每行路径为相对 `data_dir` 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.det.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.det.transforms。
- **num_workers** (int|str): 数据集中样本在预处理过程中的线程或进程数。默认为'auto'。当设为'auto'时，根据系统的实际 CPU 核数设置 `num_workers`: 如果 CPU 核数的一半大于 8，则 `num_workers` 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。

- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.2.7 paddlex.datasets.EasyDataSeg

用于语义分割模型

```
paddlex.datasets.EasyDataSeg(data_dir, file_list, label_list, transforms=None, num_
↪workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

读取 EasyData 语义分割任务数据集，并对样本进行相应的处理。

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和对应标注文件的文件路径（文本内每行路径为相对 data_dir 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.seg.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.seg.transforms。
- **num_workers** (int|str): 数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 num_workers: 如果 CPU 核数的一半大于 8，则 num_workers 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.2.8 paddlex.datasets.ChangeDetDataset

用于完成变化检测的语义分割模型

```
paddlex.datasets.ChangeDetDataset(data_dir, file_list, label_list, transforms=None, num_
↪workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

读取用于完成变化检测的语义分割数据集，并对样本进行相应的处理。地块检测数据集格式的介绍可查看文档:数据集格式说明

示例：代码文件

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片 1 文件、图片 2 文件和对应标注文件的文件路径（文本内每行路径为相对 `data_dir` 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。
- **transforms** (paddlex.seg.transforms): 数据集中每个样本的预处理/增强算子，详见paddlex.seg.transforms。
- **num_workers** (int|str)：数据集中样本在预处理过程中的线程或进程数。默认为’auto’。当设为’auto’时，根据系统的实际 CPU 核数设置 `num_workers`: 如果 CPU 核数的一半大于 8，则 `num_workers` 为 8，否则为 CPU 核数的一半。
- **buffer_size** (int): 数据集中样本在预处理过程中队列的缓存长度，以样本数为单位。默认为 100。
- **parallel_method** (str): 数据集中样本在预处理过程中并行处理的方式，支持’thread’线程和’process’进程两种方式。默认为’process’（Windows 和 Mac 下会强制使用 thread，该参数无效）。
- **shuffle** (bool): 是否需要对数据集中样本打乱顺序。默认为 False。

29.3 数据集工具

29.3.1 数据集分析

paddlex.datasets.analysis.Seg

```
paddlex.datasets.analysis.Seg(data_dir, file_list, label_list)
```

构建统计分析语义分类数据集的分析器。

参数

- **data_dir** (str): 数据集所在的目录路径。
- **file_list** (str): 描述数据集图片文件和类别 id 的文件路径（文本内每行路径为相对 `data_dir` 的相对路径）。
- **label_list** (str): 描述数据集包含的类别信息文件路径。

analysis

```
analysis(self)
```

Seg 分析器的分析接口，完成以下信息的分析统计：

- 图像数量
- 图像最大和最小的尺寸
- 图像通道数量
- 图像各通道的最小值和最大值
- 图像各通道的像素值分布
- 图像各通道归一化后的均值和方差
- 标注图中各类别的数量及比重

代码示例

统计信息示例

cal_clipped_mean_std

```
cal_clipped_mean_std(self, clip_min_value, clip_max_value, data_info_file)
```

Seg 分析器用于计算图像截断后的均值和方差的接口。

参数

- **clip_min_value** (list): 截断的下限，小于 min_val 的数值均设为 min_val。
- **clip_max_value** (list): 截断的上限，大于 max_val 的数值均设为 max_val。
- **data_info_file** (str): 在 analysis() 接口中保存的分析结果文件（名为 `train_information.pkl`）的路径。

代码示例

计算结果示例

29.3.2 数据集生成

paddlex.det.paste_objects

```
paddlex.det.paste_objects(templates, background, save_dir='dataset_clone')
```

将目标物体粘贴在背景图片上生成新的图片和标注文件

参数

- **templates** (list|tuple)：可以将多张图像上的目标物体同时粘贴在同一个背景图片上，因此 templates 是一个列表，其中每个元素是一个 dict，表示一张图片的目标物体。一张图片的目标物体有 `image` 和 `annos` 两个关键字，`image` 的键值是图像的路径，或者是解码后的排列格式为（H, W, C）且类型为 uint8 且为 BGR 格式的数组。图像上可以有多个目标物体，因此 `annos` 的键值是一个列表，列表中每个元素是一个 dict，表示一个目标物体的信息。该 dict 包含 `polygon` 和 `category` 两个关键字，其中 `polygon` 表示目标物体的边缘坐标，例如 [[0, 0], [0, 1], [1, 1], [1, 0]]，`category` 表示目标物体的类别，例如’dog’。
- **background** (dict)：背景图片可以有真值，因此 background 是一个 dict，包含 `image` 和 `annos` 两个关键字，`image` 的键值是背景图像的路径，或者是解码后的排列格式为（H, W, C）且类型为 uint8 且为 BGR 格式的数组。若背景图片上没有真值，则 `annos` 的键值是空列表 []，若有，则 `annos` 的键值是由多个 dict 组成的列表，每个 dict 表示一个物体的信息，包含 `bbox` 和 `category` 两个关键字，`bbox` 的键值是物体框左上角和右下角的坐标，即 [x1, y1, x2, y2]，`category` 表示目标物体的类别，例如’dog’。
- **save_dir** (str)：新图片及其标注文件的存储目录。默认值为 `dataset_clone`。

代码示例

```
import paddlex as pdx
templates = [{'image': 'dataset/JPEGImages/budaodian-10.jpg',
              'annos': [{'polygon': [[146, 169], [909, 169], [909, 489], [146, 489]],
                         'category': 'lou_di'},
                        {'polygon': [[146, 169], [909, 169], [909, 489], [146, 489]],
                         'category': 'lou_di'}]}]
background = {'image': 'dataset/JPEGImages/budaodian-12.jpg', 'annos': []}
pdx.det.paste_objects(templates, background, save_dir='dataset_clone')
```

29.4 视觉模型集

PaddleX 目前支持 四种视觉任务解决方案 ，包括图像分类、目标检测、实例分割和语义分割。对于每种视觉任务，PaddleX 又提供了 1 种或多种模型，用户可根据需求及应用场景选取。

29.4.1 Image Classification

paddlex.cls.ResNet50

```
paddlex.cls.ResNet50(num_classes=1000, input_channel=3)
```

构建 ResNet50 分类器，并实现其训练、评估和预测。

参数

- **num_classes** (int): 类别数。默认为 1000。
- **input_channel** (int): 输入图像的通道数量。默认为 3。

train

```
train(self, num_epochs, train_dataset, train_batch_size=64, eval_dataset=None, save_
↪interval_epochs=1, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET
↪', optimizer=None, learning_rate=0.025, warmup_steps=0, warmup_start_lr=0.0, lr_decay_
↪epochs=[30, 60, 90], lr_decay_gamma=0.1, use_vdl=False, sensitivities_file=None, eval_
↪metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_checkpoint=None)
```

参数

- **num_epochs** (int): 训练迭代轮数。
- **train_dataset** (paddlex.datasets): 训练数据读取器。
- **train_batch_size** (int): 训练数据 batch 大小。同时作为验证数据 batch 大小。默认值为 64。
- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **save_interval_epochs** (int): 模型保存间隔（单位：迭代轮数）。默认为 1。
- **log_interval_steps** (int): 训练日志输出间隔（单位：迭代步数）。默认为 2。
- **save_dir** (str): 模型保存路径。
- **pretrain_weights** (str): 若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为 None，则不使用预训练模型。默认为’IMAGENET’。若模型为’ResNet50_vd’，则默认下载百度自研 10 万类预训练模型，即默认为’BAIDU10W’。
- **optimizer** (paddle.fluid.optimizer): 优化器。当该参数为 None 时，使用默认优化器：fluid.layers.piecewise_decay 衰减策略，fluid.optimizer.Momentum 优化方法。
- **learning_rate** (float): 默认优化器的初始学习率。默认为 0.025。
- **warmup_steps** (int): 默认优化器的 warmup 步数，学习率将在设定的步数内，从 warmup_start_lr 线性增长至设定的 learning_rate，默认为 0。
- **warmup_start_lr**(float): 默认优化器的 warmup 起始学习率，默认为 0.0。
- **lr_decay_epochs** (list): 默认优化器的学习率衰减轮数。默认为 [30, 60, 90]。
- **lr_decay_gamma** (float): 默认优化器的学习率衰减率。默认为 0.1。
- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认值为 False。

- **sensitivities_file** (str): 若指定为路径时，则加载路径下敏感度信息进行裁剪；若为字符串’DEFAULT’，则自动下载在 ImageNet 图片数据上获得的敏感度信息进行裁剪；若为 None，则不进行裁剪。默认为 None。
- **eval_metric_loss** (float): 可容忍的精度损失。默认为 0.05。
- **early_stop** (bool): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 `early_stop_patience` 个 epoch 内连续下降或持平，则终止训练。默认值为 5。
- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, return_details=False)
```

参数

- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **batch_size** (int): 验证数据批大小。默认为 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **return_details** (bool): 是否返回详细信息，默认 False。

返回值

- **dict**: 当 return_details 为 False 时，返回 dict, 包含关键字：’acc1’、’acc5’，分别表示最大值的 accuracy、前 5 个最大值的 accuracy。
- **tuple** (metrics, eval_details): 当 `return_details` 为 True 时，增加返回 dict，包含关键字：’true_labels’、’pred_scores’，分别代表真实类别 id、每个类别的预测得分。

predict

```
predict(self, img_file, transforms=None, topk=1)
```

分类模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `ResNet50.test_transforms` 和 `ResNet50.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `predict` 接口时，用户需要再重新定义 test_transforms 传入给 `predict` 接口。

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。

- **transforms** (paddlex.cls.transforms): 数据预处理操作。
- **topk** (int): 预测时前 k 个最大值。

返回值

- **list**: 其中元素均为字典。字典的关键字为’category_id’、’category’、’score’，分别对应预测类别 id、预测类别标签、预测得分。

batch_predict

```
batch_predict(self, img_file_list, transforms=None, topk=1)
```

分类模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `ResNet50.test_transforms` 和 `ResNet50.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `batch_predict` 接口时，用户需要再重新定义 test_transforms 传入给 `batch_predict` 接口。

参数

- **img_file_list** (list|tuple): 对列表（或元组）中的图像同时进行预测，列表中的元素可以是图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.cls.transforms): 数据预处理操作。
- **topk** (int): 预测时前 k 个最大值。

返回值

- **list**: 每个元素都为列表，表示各图像的预测结果。在各图像的预测列表中，其中元素均为字典。字典的关键字为’category_id’、’category’、’score’，分别对应预测类别 id、预测类别标签、预测得分。

其它分类模型

PaddleX 提供了共计 22 种分类模型，所有分类模型均提供同 `ResNet50` 相同的训练 `train`，评估 `evaluate` 和预测 `predict` 接口，各模型效果可参考模型库。

29.4.2 Object Detection

paddlex.det.PPYOLO

```
paddlex.det.PPYOLO(num_classes=80, backbone='ResNet50_vd_ssld', with_dcn_v2=True,␣
↪anchors=None, anchor_masks=None, use_coord_conv=True, use_iou_aware=True, use_spp=True,
↪ use_drop_block=True, scale_x_y=1.05, ignore_threshold=0.7, label_smooth=False, use_
↪iou_loss=True, use_matrix_nms=True, nms_score_threshold=0.01, nms_topk=1000, nms_keep_
↪topk=100, nms_iou_threshold=0.45, train_random_shapes=[320, 352, 384, 416, 448, 480,␣
↪512, 544, 576, 608], input_channel=3)
```

(下页继续)

(续上页)

构建 PPYOLO 检测器。**注意在 PPYOLO，num_classes 不需要包含背景类，如目标包括 human、dog 两种，则 num_classes 设为 2 即可，这里与 FasterRCNN/MaskRCNN 有差别**

参数

- **num_classes** (int): 类别数。默认为 80。
- **backbone** (str): PPYOLO 的 backbone 网络，取值范围为 [' ResNet50_vd_ssld']。默认为' ResNet50_vd_ssld'。
- **with_dcn_v2** (bool): Backbone 是否使用 DCNv2 结构。默认为 True。
- **anchors** (list|tuple): anchor 框的宽度和高度，为 None 时表示使用默认值 [[10, 13], [16, 30], [33, 23], [30, 61], [62, 45], [59, 119], [116, 90], [156, 198], [373, 326]]。
- **anchor_masks** (list|tuple): 在计算 PPYOLO 损失时，使用 anchor 的 mask 索引，为 None 时表示使用默认值 [[6, 7, 8], [3, 4, 5], [0, 1, 2]]。
- **use_coord_conv** (bool): 是否使用 CoordConv。默认值为 True。
- **use_iou_aware** (bool): 是否使用 IoU Aware 分支。默认值为 True。
- **use_spp** (bool): 是否使用 Spatial Pyramid Pooling 结构。默认值为 True。
- **use_drop_block** (bool): 是否使用 Drop Block。默认值为 True。
- **scale_x_y** (float): 调整中心点位置时的系数因子。默认值为 1.05。
- **use_iou_loss** (bool): 是否使用 IoU loss。默认值为 True。
- **use_matrix_nms** (bool): 是否使用 Matrix NMS。默认值为 True。
- **ignore_threshold** (float): 在计算 PPYOLO 损失时，IoU 大于 `ignore_threshold` 的预测框的置信度被忽略。默认为 0.7。
- **nms_score_threshold** (float): 检测框的置信度得分阈值，置信度得分低于阈值的框应该被忽略。默认为 0.01。
- **nms_topk** (int): 进行 NMS 时，根据置信度保留的最大检测框数。默认为 1000。
- **nms_keep_topk** (int): 进行 NMS 后，每个图像要保留的总检测框数。默认为 100。
- **nms_iou_threshold** (float): 进行 NMS 时，用于剔除检测框 IOU 的阈值。默认为 0.45。
- **label_smooth** (bool): 是否使用 label smooth。默认值为 False。
- **train_random_shapes** (list|tuple): 训练时从列表中随机选择图像大小。默认值为 [320, 352, 384, 416, 448, 480, 512, 544, 576, 608]。
- **input_channel** (int): 输入图像的通道数量。默认为 3。

train

```
train(self, num_epochs, train_dataset, train_batch_size=8, eval_dataset=None, save_
↪interval_epochs=20, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET
↪', optimizer=None, learning_rate=1.0/8000, warmup_steps=1000, warmup_start_lr=0.0, lr_
↪decay_epochs=[213, 240], lr_decay_gamma=0.1, metric=None, use_vdl=False, sensitivities_
↪file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_
↪checkpoint=None, use_ema=True, ema_decay=0.9998)
```

PPYOLO 模型的训练接口，函数内置了 piecewise 学习率衰减策略和 momentum 优化器。

参数

- **num_epochs** (int): 训练迭代轮数。
- **train_dataset** (paddlex.datasets): 训练数据读取器。
- **train_batch_size** (int): 训练数据 batch 大小。目前检测仅支持单卡评估，训练数据 batch 大小与显卡数量之商为验证数据 batch 大小。默认值为 8。
- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **save_interval_epochs** (int): 模型保存间隔（单位：迭代轮数）。默认为 20。
- **log_interval_steps** (int): 训练日志输出间隔（单位：迭代次数）。默认为 2。
- **save_dir** (str): 模型保存路径。默认值为’output’。
- **pretrain_weights** (str): 若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为字符串’COCO’，则自动下载在 COCO 数据集上预训练的模型权重；若为 None，则不使用预训练模型。默认为’IMAGENET’。
- **optimizer** (paddle.fluid.optimizer): 优化器。当该参数为 None 时，使用默认优化器：fluid.layers.piecewise_decay 衰减策略，fluid.optimizer.Momentum 优化方法。
- **learning_rate** (float): 默认优化器的学习率。默认为 1.0/8000。
- **warmup_steps** (int): 默认优化器进行 warmup 过程的步数。默认为 1000。
- **warmup_start_lr** (int): 默认优化器 warmup 的起始学习率。默认为 0.0。
- **lr_decay_epochs** (list): 默认优化器的学习率衰减轮数。默认为 [213, 240]。
- **lr_decay_gamma** (float): 默认优化器的学习率衰减率。默认为 0.1。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [’COCO’，‘VOC’]。默认值为 None。
- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认值为 False。
- **sensitivities_file** (str): 若指定为路径时，则加载路径下敏感度信息进行裁剪；若为字符串’DEFAULT’，则自动下载在 PascalVOC 数据上获得的敏感度信息进行裁剪；若为 None，

则不进行裁剪。默认为 None。

- **eval_metric_loss** (float): 可容忍的精度损失。默认为 0.05。
- **early_stop** (bool): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 early_stop_patience 个 epoch 内连续下降或持平，则终止训练。默认值为 5。
- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。
- **use_ema** (bool): 是否使用指数衰减计算参数的滑动平均值。默认值为 True。
- **ema_decay** (float): 指数衰减率。默认值为 0.9998。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_
↪details=False)
```

PPYOLO 模型的评估接口，模型评估后会返回在验证集上的指标 box_map(metric 指定为'VOC'时) 或 box_mmap(metric 指定为 COCO 时)。

参数

- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **batch_size** (int): 验证数据批大小。默认为 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **metric** (bool): 训练过程中评估的方式，取值范围为 ['COCO'，'VOC']。默认为 None，根据用户传入的 Dataset 自动选择，如为 VOCDetection，则 metric 为'VOC'；如为 COCODetection，则 metric 为'COCO'默认为 None，如为 EasyData 类型数据集，同时也会使用'VOC'。
- **return_details** (bool): 是否返回详细信息。默认值为 False。

返回值

- **tuple** (metrics, eval_details) | **dict** (metrics): 当 return_details 为 True 时，返回 (metrics, eval_details)，当 return_details 为 False 时，返回 metrics。metrics 为 dict，包含关键字：'bbox_mmap'或者'bbox_map'，分别表示平均准确率平均值在各个阈值下的结果取平均值的结果（mmAP）、平均准确率平均值（mAP）。eval_details 为 dict，包含 bbox 和 gt 两个关键字。其中关键字 bbox 的键值是一个列表，列表中每个元素代表一个预测结果，一个预测结果是一个由图像 id，预测框类别 id，预测框坐标，预测框得分组成的列表。而关键字 gt 的键值是真实标注框的相关信息。

predict

```
predict(self, img_file, transforms=None)
```

PPYOLO 模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 PPYOLO.test_transforms 和 PPYOLO.eval_transforms 中。如未在训练时定义 eval_dataset，那在调用预测 predict 接口时，用户需要再重新定义 test_transforms 传入给 predict 接口

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 预测结果列表，列表中每个元素均为一个 dict，key 包括’bbox’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

batch_predict

```
batch_predict(self, img_file_list, transforms=None)
```

PPYOLO 模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 PPYOLO.test_transforms 和 PPYOLO.eval_transforms 中。如未在训练时定义 eval_dataset，那在调用预测 batch_predict 接口时，用户需要再重新定义 test_transforms 传入给 batch_predict 接口

参数

- **img_file_list** (str|np.ndarray): 对列表（或元组）中的图像同时进行预测，列表中的元素是预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 每个元素都为列表，表示各图像的预测结果。在各图像的预测结果列表中，每个元素均为一个 dict，key 包括’bbox’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

paddlex.det.YOLOv3

```
paddlex.det.YOLOv3(num_classes=80, backbone='MobileNetV1', anchors=None, anchor_
↪masks=None, ignore_threshold=0.7, nms_score_threshold=0.01, nms_topk=1000, nms_keep_
↪topk=100, nms_iou_threshold=0.45, label_smooth=False, train_random_shapes=[320, 352,␣
↪384, 416, 448, 480, 512, 544, 576, 608], input_channel=3)
```

构建 YOLOv3 检测器。**注意在 YOLOv3，num_classes 不需要包含背景类，如目标包括 human、dog 两种，则 num_classes 设为 2 即可，这里与 FasterRCNN/MaskRCNN 有差别**

参数

- **num_classes** (int): 类别数。默认为 80。
- **backbone** (str): YOLOv3 的 backbone 网络，取值范围为 [' DarkNet53' ， 'ResNet34' ， 'MobileNetV1' ， 'MobileNetV3_large']。默认为' MobileNetV1'。
- **anchors** (list|tuple): anchor 框的宽度和高度，为 None 时表示使用默认值 [[10, 13], [16, 30], [33, 23], [30, 61], [62, 45], [59, 119], [116, 90], [156, 198], [373, 326]]。
- **anchor_masks** (list|tuple): 在计算 YOLOv3 损失时，使用 anchor 的 mask 索引，为 None 时表示使用默认值 [[6, 7, 8], [3, 4, 5], [0, 1, 2]]。
- **ignore_threshold** (float): 在计算 YOLOv3 损失时，IoU 大于 ignore_threshold 的预测框的置信度被忽略。默认为 0.7。
- **nms_score_threshold** (float): 检测框的置信度得分阈值，置信度得分低于阈值的框应该被忽略。默认为 0.01。
- **nms_topk** (int): 进行 NMS 时，根据置信度保留的最大检测框数。默认为 1000。
- **nms_keep_topk** (int): 进行 NMS 后，每个图像要保留的总检测框数。默认为 100。
- **nms_iou_threshold** (float): 进行 NMS 时，用于剔除检测框 IoU 的阈值。默认为 0.45。
- **label_smooth** (bool): 是否使用 label smooth。默认值为 False。
- **train_random_shapes** (list|tuple): 训练时从列表中随机选择图像大小。默认值为 [320, 352, 384, 416, 448, 480, 512, 544, 576, 608]。
- **input_channel** (int): 输入图像的通道数量。默认为 3。

train

```
train(self, num_epochs, train_dataset, train_batch_size=8, eval_dataset=None, save_
↪interval_epochs=20, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET
↪', optimizer=None, learning_rate=1.0/8000, warmup_steps=1000, warmup_start_lr=0.0, lr_
↪decay_epochs=[213, 240], lr_decay_gamma=0.1, metric=None, use_vdl=False, sensitivities_
↪file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_
↪checkpoint=None)
```

(下页继续)

(续上页)

YOLOv3 模型的训练接口，函数内置了 piecewise 学习率衰减策略和 momentum 优化器。

参数

- **num_epochs** (int): 训练迭代轮数。
- **train_dataset** (paddlex.datasets): 训练数据读取器。
- **train_batch_size** (int): 训练数据 batch 大小。目前检测仅支持单卡评估，训练数据 batch 大小与显卡数量之商为验证数据 batch 大小。默认值为 8。
- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **save_interval_epochs** (int): 模型保存间隔（单位：迭代轮数）。默认为 20。
- **log_interval_steps** (int): 训练日志输出间隔（单位：迭代次数）。默认为 2。
- **save_dir** (str): 模型保存路径。默认值为’output’。
- **pretrain_weights** (str): 若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为字符串’COCO’，则自动下载在 COCO 数据集上预训练的模型权重；若为 None，则不使用预训练模型。默认为’IMAGENET’。
- **optimizer** (paddle.fluid.optimizer): 优化器。当该参数为 None 时，使用默认优化器：fluid.layers.piecewise_decay 衰减策略，fluid.optimizer.Momentum 优化方法。
- **learning_rate** (float): 默认优化器的学习率。默认为 1.0/8000。
- **warmup_steps** (int): 默认优化器进行 warmup 过程的步数。默认为 1000。
- **warmup_start_lr** (int): 默认优化器 warmup 的起始学习率。默认为 0.0。
- **lr_decay_epochs** (list): 默认优化器的学习率衰减轮数。默认为 [213, 240]。
- **lr_decay_gamma** (float): 默认优化器的学习率衰减率。默认为 0.1。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [’COCO’，‘VOC’]。默认值为 None。
- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认值为 False。
- **sensitivities_file** (str): 若指定为路径时，则加载路径下敏感度信息进行裁剪；若为字符串’DEFAULT’，则自动下载在 PascalVOC 数据上获得的敏感度信息进行裁剪；若为 None，则不进行裁剪。默认为 None。
- **eval_metric_loss** (float): 可容忍的精度损失。默认为 0.05。
- **early_stop** (bool): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 early_stop_patience 个 epoch 内连续下降或持平，则终止训练。默认值为 5。

- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_
↪details=False)
```

YOLOv3 模型的评估接口，模型评估后会返回在验证集上的指标 `box_map`(metric 指定为' VOC' 时) 或 `box_mmap`(metric 指定为 `COCO` 时)。

参数

- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **batch_size** (int): 验证数据批大小。默认为 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [' COCO' ，'VOC']。默认为 None，根据用户传入的 Dataset 自动选择，如为 VOCDetection，则 `metric` 为' VOC' ；如为 COCODetection，则 `metric` 为' COCO' 默认为 None，如为 EasyData 类型数据集，同时也会使用' VOC' 。
- **return_details** (bool): 是否返回详细信息。默认值为 False。

返回值

- **tuple** (metrics, eval_details) | **dict** (metrics): 当 `return_details` 为 True 时，返回 (metrics, eval_details)，当 `return_details` 为 False 时，返回 metrics。metrics 为 dict，包含关键字:' bbox_mmap' 或者' bbox_map '；分别表示平均准确率平均值在各个阈值下的结果取平均值的结果 (mmAP)、平均准确率平均值 (mAP)。eval_details 为 dict，包含 bbox 和 gt 两个关键字。其中关键字 bbox 的键值是一个列表，列表中每个元素代表一个预测结果，一个预测结果是一个由图像 id，预测框类别 id, 预测框坐标，预测框得分组成的列表。而关键字 gt 的键值是真实标注框的相关信息。

predict

```
predict(self, img_file, transforms=None)
```

YOLOv3 模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `YOLOv3.test_transforms` 和 `YOLOv3.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `predict` 接口时，用户需要再重新定义 `test_transforms` 传入给 `predict` 接口

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 预测结果列表，列表中每个元素均为一个 dict，key 包括’bbox’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

batch_predict

```
batch_predict(self, img_file_list, transforms=None)
```

YOLOv3 模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 YOLOv3.test_transforms 和 YOLOv3.eval_transforms 中。如未在训练时定义 eval_dataset，那在调用预测 batch_predict 接口时，用户需要再重新定义 test_transforms 传入给 batch_predict 接口

参数

- **img_file_list** (str|np.ndarray): 对列表（或元组）中的图像同时进行预测，列表中的元素是预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 每个元素都为列表，表示各图像的预测结果。在各图像的预测结果列表中，每个元素均为一个 dict，key 包括’bbox’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

paddlex.det.FasterRCNN

```
paddlex.det.FasterRCNN(num_classes=81, backbone='ResNet50', with_fpn=True, aspect_
↪ratios=[0.5, 1.0, 2.0], anchor_sizes=[32, 64, 128, 256, 512], with_dcn=False, rpn_cls_
↪loss='SigmoidCrossEntropy', rpn_focal_loss_alpha=0.25, rpn_focal_loss_gamma=2, rcnn_
↪bbox_loss='SmoothL1Loss', rcnn_nms='MultiClassNMS', keep_top_k=100, nms_threshold=0.5,␣
↪score_threshold=0.05, softnms_sigma=0.5, bbox_assigner='BBoxAssigner', fpn_num_
↪channels=256, input_channel=3, rpn_batch_size_per_im=256, rpn_fg_fraction=0.5, test_
↪pre_nms_top_n=None, test_post_nms_top_n=1000)
```

构建 FasterRCNN 检测器。**注意在 FasterRCNN 中，num_classes 需要设置为类别数 + 背景类，如目标包括 human、dog 两种，则 num_classes 需设为 3，多的一种为背景 background 类别**

参数

- **num_classes** (int): 包含了背景类的类别数。默认为 81。
- **backbone** (str): FasterRCNN 的 backbone 网络，取值范围为 [' ResNet18' , 'ResNet50' , 'ResNet50_vd' , 'ResNet101' , 'ResNet101_vd' , 'HRNet_W18' , 'ResNet50_vd_ssld']。默认为' ResNet50'。
- **with_fpn** (bool): 是否使用 FPN 结构。默认为 True。
- **aspect_ratios** (list): 生成 anchor 高宽比的可选值。默认为 [0.5, 1.0, 2.0]。
- **anchor_sizes** (list): 生成 anchor 大小的可选值。默认为 [32, 64, 128, 256, 512]。
- **with_dcn** (bool): backbone 网络中是否使用 deformable convolution network v2。默认为 False。
- **rpn_cls_loss** (str): RPN 部分的分类损失函数，取值范围为 [' SigmoidCrossEntropy' , 'SigmoidFocalLoss']。当遇到模型误检了很多背景区域时，可以考虑使用' SigmoidFocalLoss' ，并调整适合的 `rpn_focal_loss_alpha` 和 `rpn_focal_loss_gamma`。默认为' SigmoidCrossEntropy'。
- **rpn_focal_loss_alpha** (float)：当 RPN 的分类损失函数设置为' SigmoidFocalLoss' 时，用于调整正样本和负样本的比例因子，默认为 0.25。当 PN 的分类损失函数设置为' SigmoidCrossEntropy' 时，`rpn_focal_loss_alpha` 的设置不生效。
- **rpn_focal_loss_gamma** (float): 当 RPN 的分类损失函数设置为' SigmoidFocalLoss' 时，用于调整易分样本和难分样本的比例因子，默认为 2。当 RPN 的分类损失函数设置为' SigmoidCrossEntropy' 时，`rpn_focal_loss_gamma` 的设置不生效。
- **rcnn_bbox_loss** (str): RCNN 部分的位置回归损失函数，取值范围为 [' SmoothL1Loss' , 'CIoULoss']。默认为' SmoothL1Loss'。
- **rcnn_nms** (str): RCNN 部分的非极大值抑制的计算方法，取值范围为 [' MultiClassNMS' , 'MultiClassSoftNMS' ,' MultiClassCiouNMS']。默认为' MultiClassNMS' 。当选择' MultiClassNMS' 时，可以将 `keep_top_k` 设置成 100、`nms_threshold` 设置成 0.5、`score_threshold` 设置成 0.05。当选择' MultiClassSoftNMS' 时，可以将 `keep_top_k` 设置为 300、`score_threshold` 设置为 0.01、`softnms_sigma` 设置为 0.5。当选择' MultiClassCiouNMS' 时，可以将 `keep_top_k` 设置为 100、`score_threshold` 设置成 0.05、`nms_threshold` 设置成 0.5。
- **keep_top_k** (int): RCNN 部分在进行非极大值抑制计算后，每张图像保留最多保存 `keep_top_k` 个检测框。默认为 100。
- **nms_threshold** (float): RCNN 部分在进行非极大值抑制时，用于剔除检测框所需的 IoU 阈值。当 `rcnn_nms` 设置为 `MultiClassSoftNMS` 时，`nms_threshold` 的设置不生效。默认为 0.5。
- **score_threshold** (float): RCNN 部分在进行非极大值抑制前，用于过滤掉低置信度边界框所需的置信度阈值。默认为 0.05。

- **softnms_sigma** (float): 当 `rcnn_nms` 设置为 `MultiClassSoftNMS` 时，用于调整被抑制的检测框的置信度，调整公式为 `score = score * weights, weights = exp(-(iou * iou) / softnms_sigma)`。默认设为 0.5。
- **bbox_assigner** (str): 训练阶段，RCNN 部分生成正负样本的采样方式。可选范围为 ['BBoxAssigner'，'LibraBBoxAssigner']。当目标物体的区域只占原始图像的一小部分时，可以考虑采用LibraRCNN中提出的 IoU-balanced Sampling 采样方式来获取更多的难分负样本，设置为'LibraBBoxAssigner'即可。默认为'BBoxAssigner'。
- **fpn_num_channels** (int): FPN 部分特征层的通道数量。默认为 256。
- **input_channel** (int): 输入图像的通道数量。默认为 3。
- **rpn_batch_size_per_im** (int): 训练阶段，RPN 部分每张图片的正负样本的数量总和。默认为 256。
- **rpn_fg_fraction** (float): 训练阶段，RPN 部分每张图片的正负样本数量总和中正样本的占比。默认为 0.5。
- **test_pre_nms_top_n** (int): 预测阶段，RPN 部分做非极大值抑制计算的候选框的数量。若设置为 None, 有 FPN 结构的话，`test_pre_nms_top_n` 会被设置成 6000, 无 FPN 结构的话，`test_pre_nms_top_n` 会被设置成 1000。默认为 None。
- **test_post_nms_top_n** (int): 预测阶段，RPN 部分做完非极大值抑制后保留的候选框的数量。默认为 1000。

train

```
train(self, num_epochs, train_dataset, train_batch_size=2, eval_dataset=None, save_
↪interval_epochs=1, log_interval_steps=2,save_dir='output', pretrain_weights='IMAGENET',
↪ optimizer=None, learning_rate=0.0025, warmup_steps=500, warmup_start_lr=1.0/1200, lr_
↪decay_epochs=[8, 11], lr_decay_gamma=0.1, metric=None, use_vdl=False, early_stop=False,
↪ early_stop_patience=5, resume_checkpoint=None)
```

FasterRCNN 模型的训练接口，函数内置了 `piecewise` 学习率衰减策略和 `momentum` 优化器。

参数

- **num_epochs** (int): 训练迭代轮数。
- **train_dataset** (paddlex.datasets): 训练数据读取器。
- **train_batch_size** (int): 训练数据 batch 大小。目前检测仅支持单卡评估，训练数据 batch 大小与显卡数量之商为验证数据 batch 大小。默认为 2。
- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **save_interval_epochs** (int): 模型保存间隔（单位：迭代轮数）。默认为 1。
- **log_interval_steps** (int): 训练日志输出间隔（单位：迭代次数）。默认为 2。

- **save_dir** (str): 模型保存路径。默认值为’output’。
- **pretrain_weights** (str): 若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为字符串’COCO’，则自动下载在 COCO 数据集上预训练的模型权重（注意：暂未提供 ResNet18 的 COCO 预训练模型）；为 None，则不使用预训练模型。默认为’IMAGENET’。
- **optimizer** (paddle.fluid.optimizer): 优化器。当该参数为 None 时，使用默认优化器：fluid.layers.piecewise_decay 衰减策略，fluid.optimizer.Momentum 优化方法。
- **learning_rate** (float): 默认优化器的初始学习率。默认为 0.0025。
- **warmup_steps** (int): 默认优化器进行 warmup 过程的步数。默认为 500。
- **warmup_start_lr** (int): 默认优化器 warmup 的起始学习率。默认为 1.0/1200。
- **lr_decay_epochs** (list): 默认优化器的学习率衰减轮数。默认为 [8, 11]。
- **lr_decay_gamma** (float): 默认优化器的学习率衰减率。默认为 0.1。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [’COCO’，‘VOC’]。默认值为 None。
- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认值为 False。
- **early_stop** (float): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 `early_stop_patience` 个 epoch 内连续下降或持平，则终止训练。默认值为 5。
- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_
↪details=False)
```

FasterRCNN 模型的评估接口，模型评估后会返回在验证集上的指标 box_map(metric 指定为’VOC’时) 或 box_mmap(metric 指定为 COCO 时)。

参数

- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **batch_size** (int): 验证数据批大小。默认为 1。当前只支持设置为 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [’COCO’，‘VOC’]。默认为 None，根据用户传入的 Dataset 自动选择，如为 VOCDetection，则 `metric` 为’VOC’；如为 COCODetection，则 `metric` 为’COCO’。

- **return_details** (bool): 是否返回详细信息。默认值为 False。

返回值

- **tuple** (metrics, eval_details) | **dict** (metrics): 当 `return_details` 为 True 时，返回 (metrics, eval_details)，当 `return_details` 为 False 时，返回 metrics。metrics 为 dict，包含关键字：'bbox_mmap'或者'bbox_map'，分别表示平均准确率平均值在各个 IoU 阈值下的结果取平均值的结果（mmAP）、平均准确率平均值（mAP）。eval_details 为 dict，包含 bbox 和 gt 两个关键字。其中关键字 bbox 的键值是一个列表，列表中每个元素代表一个预测结果，一个预测结果是一个由图像 id，预测框类别 id, 预测框坐标，预测框得分组成的列表。而关键字 gt 的键值是真实标注框的相关信息。

predict

```
predict(self, img_file, transforms=None)
```

FasterRCNN 模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `FasterRCNN.test_transforms` 和 `FasterRCNN.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `predict` 接口时，用户需要再重新定义 test_transforms 传入给 `predict` 接口。

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 预测结果列表，列表中每个元素均为一个 dict，key 包括'bbox'，'category'，'category_id'，'score'，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

batch_predict

```
batch_predict(self, img_file_list, transforms=None)
```

FasterRCNN 模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `FasterRCNN.test_transforms` 和 `FasterRCNN.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `batch_predict` 接口时，用户需要再重新定义 test_transforms 传入给 `batch_predict` 接口。

参数

- **img_file_list** (list|tuple): 对列表（或元组）中的图像同时进行预测，列表中的元素是预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。

- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 每个元素都为列表，表示各图像的预测结果。在各图像的预测结果列表中，每个元素均为一个 dict，key 包括’bbox’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、类别、类别 id、置信度，其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。

29.4.3 Instance Segmentation

MaskRCNN

```
paddlex.det.MaskRCNN(num_classes=81, backbone='ResNet50', with_fpn=True, aspect_
↪ratios=[0.5, 1.0, 2.0], anchor_sizes=[32, 64, 128, 256, 512], with_dcn=False, rpn_cls_
↪loss='SigmoidCrossEntropy', rpn_focal_loss_alpha=0.25, rpn_focal_loss_gamma=2, rcnn_
↪bbox_loss='SmoothL1Loss', rcnn_nms='MultiClassNMS', keep_top_k=100, nms_threshold=0.5,␣
↪score_threshold=0.05, softnms_sigma=0.5, bbox_assigner='BBoxAssigner', fpn_num_
↪channels=256, input_channel=3, rpn_batch_size_per_im=256, rpn_fg_fraction=0.5, test_
↪pre_nms_top_n=None, test_post_nms_top_n=1000)
```

构建 MaskRCNN 检测器。**注意在 MaskRCNN 中，num_classes 需要设置为类别数 + 背景类，如目标包括 human、dog 两种，则 num_classes 需设为 3，多的一种为背景 background 类别**

参数

- **num_classes** (int): 包含了背景类的类别数。默认为 81。
- **backbone** (str): MaskRCNN 的 backbone 网络，取值范围为 [’ ResNet18’，‘ResNet50’，‘ResNet50_vd’，‘ResNet101’，‘ResNet101_vd’，‘HRNet_W18’，‘ResNet50_vd_ssld’]。默认为’ ResNet50’。
- **with_fpn** (bool): 是否使用 FPN 结构。默认为 True。
- **aspect_ratios** (list): 生成 anchor 高宽比的可选值。默认为 [0.5, 1.0, 2.0]。
- **anchor_sizes** (list): 生成 anchor 大小的可选值。默认为 [32, 64, 128, 256, 512]。
- **with_dcn** (bool): backbone 网络中是否使用 deformable convolution network v2。默认为 False。
- **rpn_cls_loss** (str): RPN 部分的分类损失函数，取值范围为 [’ SigmoidCrossEntropy’，‘SigmoidFocalLoss’]。当遇到模型误检了很多背景区域时，可以考虑使用’ SigmoidFocalLoss’，并调整适合的 `rpn_focal_loss_alpha` 和 `rpn_focal_loss_gamma`。默认为’ SigmoidCrossEntropy’。

- **rpn__focal__loss__alpha** (float)：当 RPN 的分类损失函数设置为’SigmoidFocalLoss’时，用于调整正样本和负样本的比例因子，默认为 0.25。当 PN 的分类损失函数设置为’SigmoidCrossEntropy’时，`rpn_focal_loss_alpha` 的设置不生效。
- **rpn__focal__loss__gamma** (float)：当 RPN 的分类损失函数设置为’SigmoidFocalLoss’时，用于调整易分样本和难分样本的比例因子，默认为 2。当 RPN 的分类损失函数设置为’SigmoidCrossEntropy’时，`rpn_focal_loss_gamma` 的设置不生效。
- **rcnn__bbox__loss** (str)：RCNN 部分的位置回归损失函数，取值范围为 [’SmoothL1Loss’, ‘CIoULoss’]。默认为’SmoothL1Loss’。
- **rcnn__nms** (str)：RCNN 部分的非极大值抑制的计算方法，取值范围为 [’MultiClassNMS’, ‘MultiClassSoftNMS’,’MultiClassCiouNMS’]。默认为’MultiClassNMS’。当选择’MultiClassNMS’时，可以将 `keep_top_k` 设置成 100、`nms_threshold` 设置成 0.5、`score_threshold` 设置成 0.05。当选择’MultiClassSoftNMS’时，可以将 `keep_top_k` 设置为 300、`score_threshold` 设置为 0.01、`softnms_sigma` 设置为 0.5。当选择’MultiClassCiouNMS’时，可以将 `keep_top_k` 设置为 100、`score_threshold` 设置成 0.05、`nms_threshold` 设置成 0.5。
- **keep__top__k** (int)：RCNN 部分在进行非极大值抑制计算后，每张图像保留最多保存 `keep_top_k` 个检测框。默认为 100。
- **nms__threshold** (float)：RCNN 部分在进行非极大值抑制时，用于剔除检测框所需的 IoU 阈值。当 `rcnn_nms` 设置为 `MultiClassSoftNMS` 时，`nms_threshold` 的设置不生效。默认为 0.5。
- **score__threshold** (float)：RCNN 部分在进行非极大值抑制前，用于过滤掉低置信度边界框所需的置信度阈值。默认为 0.05。
- **softnms__sigma** (float)：当 `rcnn_nms` 设置为 `MultiClassSoftNMS` 时，用于调整被抑制的检测框的置信度，调整公式为 `score = score * weights, weights = exp(-(iou * iou) / softnms_sigma)`。默认设为 0.5。
- **bbox__assigner** (str)：训练阶段，RCNN 部分生成正负样本的采样方式。可选范围为 [’BBoxAssigner’, ‘LibraBBoxAssigner’]。当目标物体的区域只占原始图像的一小部分时，可以考虑采用LibraRCNN中提出的 IoU-balanced Sampling 采样方式来获取更多的难分负样本，设置为’LibraBBoxAssigner’即可。默认为’BBoxAssigner’。
- **fpn__num__channels** (int)：FPN 部分特征层的通道数量。默认为 256。
- **input__channel** (int)：输入图像的通道数量。默认为 3。
- **rpn__batch__size__per__im** (int)：训练阶段，RPN 部分每张图片的正负样本的数量总和。默认为 256。
- **rpn__fg__fraction** (float)：训练阶段，RPN 部分每张图片的正负样本数量总和中正样本的占比。默认为 0.5。
- **test__pre__nms__top__n** (int)：预测阶段，RPN 部分做非极大值抑制计算的候选框的数量。若设置为 None, 有 FPN 结构的话，`test_pre_nms_top_n` 会被设置成 6000, 无 FPN 结

构的话，test_pre_nms_top_n 会被设置成 1000。默认为 None。

- **test__post__nms__top__n** (int)：预测阶段，RPN 部分做完非极大值抑制后保留的候选框的数量。默认为 1000。

train

```
train(self, num_epochs, train_dataset, train_batch_size=1, eval_dataset=None, save_
↪interval_epochs=1, log_interval_steps=20, save_dir='output', pretrain_weights='IMAGENET
↪', optimizer=None, learning_rate=1.0/800, warmup_steps=500, warmup_start_lr=1.0 / 2400,
↪ lr_decay_epochs=[8, 11], lr_decay_gamma=0.1, metric=None, use_vdl=False, early_
↪stop=False, early_stop_patience=5, resume_checkpoint=None)
```

MaskRCNN 模型的训练接口，函数内置了 piecewise 学习率衰减策略和 momentum 优化器。

参数

- **num__epochs** (int)：训练迭代轮数。
- **train__dataset** (paddlex.datasets)：训练数据读取器。
- **train__batch__size** (int)：训练数据 batch 大小。目前检测仅支持单卡评估，训练数据 batch 大小与显卡数量之商为验证数据 batch 大小。默认为 1。
- **eval__dataset** (paddlex.datasets)：验证数据读取器。
- **save__interval__epochs** (int)：模型保存间隔（单位：迭代轮数）。默认为 1。
- **log__interval__steps** (int)：训练日志输出间隔（单位：迭代次数）。默认为 2。
- **save__dir** (str)：模型保存路径。默认值为’output’。
- **pretrain__weights** (str)：若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为字符串’COCO’，则自动下载在 COCO 数据集上预训练的模型权重（注意：暂未提供 ResNet18 和 HRNet_W18 的 COCO 预训练模型）；若为 None，则不使用预训练模型。默认为 None。
- **optimizer** (paddle.fluid.optimizer)：优化器。当该参数为 None 时，使用默认优化器：fluid.layers.piecewise_decay 衰减策略，fluid.optimizer.Momentum 优化方法。
- **learning__rate** (float)：默认优化器的初始学习率。默认为 0.00125。
- **warmup__steps** (int)：默认优化器进行 warmup 过程的步数。默认为 500。
- **warmup__start__lr** (int)：默认优化器 warmup 的起始学习率。默认为 1.0/2400。
- **lr__decay__epochs** (list)：默认优化器的学习率衰减轮数。默认为 [8, 11]。
- **lr__decay__gamma** (float)：默认优化器的学习率衰减率。默认为 0.1。
- **metric** (bool)：训练过程中评估的方式，取值范围为 [’COCO’，‘VOC’]。默认值为 None。

- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认值为 False。
- **early_stop** (float): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 `early_stop_patience` 个 epoch 内连续下降或持平，则终止训练。默认值为 5。
- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_
↪details=False)
```

MaskRCNN 模型的评估接口，模型评估后会返回在验证集上的指标 box_mmap(metric 指定为 COCO 时) 和相应的 seg_mmap。

参数

- **eval_dataset** (paddlex.datasets): 验证数据读取器。
- **batch_size** (int): 验证数据批大小。默认为 1。当前只支持设置为 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **metric** (bool): 训练过程中评估的方式，取值范围为 [' COCO' ，'VOC']。默认为 None，根据用户传入的 Dataset 自动选择，如为 VOCDetection，则 `metric` 为' VOC' ；如为 COCODetection，则 `metric` 为' COCO'。
- **return_details** (bool): 是否返回详细信息。默认值为 False。

返回值

- **tuple** (metrics, eval_details) | **dict** (metrics): 当 `return_details` 为 True 时，返回 (metrics, eval_details)，当 return_details 为 False 时，返回 metrics。metrics 为 dict，包含关键字:' bbox_mmap' 和' segm_mmap' 或者' bbox_map '和' segm_map'，分别表示预测框和分割区域平均准确率平均值在各个 IoU 阈值下的结果取平均值的结果（mmAP）、平均准确率平均值（mAP）。eval_details 为 dict，包含 `bbox`、`mask` 和 `gt` 三个关键字。其中关键字 `bbox` 的键值是一个列表，列表中每个元素代表一个预测结果，一个预测结果是一个由图像 id，预测框类别 id, 预测框坐标，预测框得分组成的列表。关键字 `mask` 的键值是一个列表，列表中每个元素代表各预测框内物体的分割结果，分割结果由图像 id、预测框类别 id、表示预测框内各像素点是否属于物体的二值图、预测框得分。而关键字 gt 的键值是真实标注框的相关信息。

predict

```
predict(self, img_file, transforms=None)
```

MaskRCNN 模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `MaskRCNN.test_transforms` 和 `MaskRCNN.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `predict` 接口时，用户需要再重新定义 test_transforms 传入给 `predict` 接口。

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 预测结果列表，列表中每个元素均为一个 dict，key’ bbox’，‘mask’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、Mask 信息，类别、类别 id、置信度。其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。Mask 信息为原图大小的二值图，1 表示像素点属于预测类别，0 表示像素点是背景。

batch_predict

```
batch_predict(self, img_file_list, transforms=None)
```

MaskRCNN 模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `MaskRCNN.test_transforms` 和 `MaskRCNN.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `batch_predict` 接口时，用户需要再重新定义 test_transforms 传入给 `batch_predict` 接口。

参数

- **img_file_list** (list|tuple): 对列表（或元组）中的图像同时进行预测，列表中的元素可以是预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.det.transforms): 数据预处理操作。

返回值

- **list**: 每个元素都为列表，表示各图像的预测结果。在各图像的预测结果列表中，每个元素均为一个 dict，包含关键字:’ bbox’，‘mask’，‘category’，‘category_id’，‘score’，分别表示每个预测目标的框坐标信息、Mask 信息，类别、类别 id、置信度。其中框坐标信息为 [xmin, ymin, w, h]，即左上角 x, y 坐标和框的宽和高。Mask 信息为原图大小的二值图，1 表示像素点属于预测类别，0 表示像素点是背景。

29.4.4 Semantic Segmentation

paddlex.seg.DeepLabv3p

```
paddlex.seg.DeepLabv3p(num_classes=2, backbone='MobileNetV2_x1.0', output_stride=16,␣
↪aspp_with_sep_conv=True, decoder_use_sep_conv=True, encoder_with_aspp=True, enable_
↪decoder=True, use_bce_loss=False, use_dice_loss=False, class_weight=None, ignore_
↪index=255, pooling_crop_size=None, input_channel=3)
```

构建 DeepLabv3p 分割器。

参数

- **num_classes** (int): 类别数。
- **backbone** (str): DeepLabv3+ 的 backbone 网络，实现特征图的计算，取值范围为 [' Xception65', 'Xception41', 'MobileNetV2_x0.25', 'MobileNetV2_x0.5', 'MobileNetV2_x1.0' , 'MobileNetV2_x1.5' , 'MobileNetV2_x2.0' , 'MobileNetV3_large_x1_0_ssld']，默认值为' MobileNetV2_x1.0'。
- **output_stride** (int): backbone 输出特征图相对于输入的下采样倍数，一般取值为 8 或 16。默认 16。
- **aspp_with_sep_conv** (bool): aspp 模块是否采用 separable convolutions。默认 True。
- **decoder_use_sep_conv** (bool): decoder 模块是否采用 separable convolutions。默认 True。
- **encoder_with_aspp** (bool): 是否在 encoder 阶段采用 aspp 模块。默认 True。
- **enable_decoder** (bool): 是否使用 decoder 模块。默认 True。
- **use_bce_loss** (bool): 是否使用 bce loss 作为网络的损失函数，只能用于两类分割。可与 dice loss 同时使用。默认 False。
- **use_dice_loss** (bool): 是否使用 dice loss 作为网络的损失函数，只能用于两类分割，可与 bce loss 同时使用，当 `use_bce_loss` 和 `use_dice_loss` 都为 False 时，使用交叉熵损失函数。默认 False。
- **class_weight** (list/str): 交叉熵损失函数各类损失的权重。当 `class_weight` 为 list 的时候，长度应为 `num_classes`。当 `class_weight` 为 str 时，weight.lower() 应为' dynamic'，这时会根据每一轮各类像素的比重自行计算相应的权重，每一类的权重为：每类的比例 * num_classes。class_weight 取默认值 None 是，各类的权重 1，即平时使用的交叉熵损失函数。
- **ignore_index** (int): label 上忽略的值，label 为 `ignore_index` 的像素不参与损失函数的计算。默认 255。

- **pooling_crop_size** (int)：当 backbone 为 `MobileNetV3_large_x1_0_ssld` 时，需设置为训练过程中模型输入大小，格式为 [W，H]。例如模型输入大小为 [512，512]，则 `pooling_crop_size` 应该设置为 [512, 512]。在 encoder 模块中获取图像平均值时被用到，若为 None，则直接求平均值；若为模型输入大小，则使用 `avg_pool` 算子得到平均值。默认值 None。
- **input_channel** (int)：输入图像通道数。默认值 3。

train

```
train(self, num_epochs, train_dataset, train_batch_size=2, eval_dataset=None, eval_batch_
↪size=1, save_interval_epochs=1, log_interval_steps=2, save_dir='output', pretrain_
↪weights='IMAGENET', optimizer=None, learning_rate=0.01, lr_decay_power=0.9, use_
↪vdl=False, sensitivities_file=None, eval_metric_loss=0.05, early_stop=False, early_
↪stop_patience=5, resume_checkpoint=None):
```

DeepLabv3p 模型的训练接口，函数内置了 `polynomial` 学习率衰减策略和 `momentum` 优化器。

参数

- **num_epochs** (int)：训练迭代轮数。
- **train_dataset** (paddlex.datasets)：训练数据读取器。
- **train_batch_size** (int)：训练数据 batch 大小。同时作为验证数据 batch 大小。默认 2。
- **eval_dataset** (paddlex.datasets)：评估数据读取器。
- **save_interval_epochs** (int)：模型保存间隔（单位：迭代轮数）。默认为 1。
- **log_interval_steps** (int)：训练日志输出间隔（单位：迭代次数）。默认为 2。
- **save_dir** (str)：模型保存路径。默认’output’
- **pretrain_weights** (str)：若指定为路径时，则加载路径下预训练模型；若为字符串’IMAGENET’，则自动下载在 ImageNet 图片数据上预训练的模型权重；若为字符串’COCO’，则自动下载在 COCO 数据集上预训练的模型权重（注意：暂未提供 Xception41、MobileNetV2_x0.25、MobileNetV2_x0.5、MobileNetV2_x1.5、MobileNetV2_x2.0 的 COCO 预训练模型）；若为字符串’CITYSCAPES’，则自动下载在 CITYSCAPES 数据集上预训练的模型权重（注意：暂未提供 Xception41、MobileNetV2_x0.25、MobileNetV2_x0.5、MobileNetV2_x1.5、MobileNetV2_x2.0 的 CITYSCAPES 预训练模型）；若为 None，则不使用预训练模型。默认’IMAGENET’。
- **optimizer** (paddle.fluid.optimizer)：优化器。当该参数为 None 时，使用默认的优化器：使用 fluid.optimizer.Momentum 优化方法，polynomial 的学习率衰减策略。
- **learning_rate** (float)：默认优化器的初始学习率。默认 0.01。
- **lr_decay_power** (float)：默认优化器学习率衰减指数。默认 0.9。

- **use_vdl** (bool): 是否使用 VisualDL 进行可视化。默认 False。
- **sensitivities_file** (str): 若指定为路径时，则加载路径下敏感度信息进行裁剪；若为字符串’DEFAULT’，则自动下载在 Cityscapes 图片数据上获得的敏感度信息进行裁剪；若为 None，则不进行裁剪。默认为 None。
- **eval_metric_loss** (float): 可容忍的精度损失。默认为 0.05。
- **early_stop** (bool): 是否使用提前终止训练策略。默认值为 False。
- **early_stop_patience** (int): 当使用提前终止训练策略时，如果验证集精度在 `early_stop_patience` 个 epoch 内连续下降或持平，则终止训练。默认值为 5。
- **resume_checkpoint** (str): 恢复训练时指定上次训练保存的模型路径。若为 None，则不会恢复训练。默认值为 None。

evaluate

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, return_details=False):
```

DeepLabv3p 模型评估接口。

参数

- **eval_dataset** (paddlex.datasets): 评估数据读取器。
- **batch_size** (int): 评估时的 batch 大小。默认 1。
- **epoch_id** (int): 当前评估模型所在的训练轮数。
- **return_details** (bool): 是否返回详细信息。默认 False。

返回值

- **dict**: 当 `return_details` 为 False 时，返回 dict。包含关键字：’miou’、’category_iou’、’macc’、‘category_acc’ 和’kappa’，分别表示平均 IoU、各类别 IoU、平均准确率、各类别准确率和 kappa 系数。
- **tuple** (metrics, eval_details)：当 `return_details` 为 True 时，增加返回 dict (eval_details)，包含关键字：’confusion_matrix’，表示评估的混淆矩阵。

predict

```
predict(self, img_file, transforms=None):
```

DeepLabv3p 模型预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `DeepLabv3p.test_transforms` 和 `DeepLabv3p.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `predict` 接口时，用户需要再重新定义 test_transforms 传入给 `predict` 接口。

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.seg.transforms): 数据预处理操作。

返回值

- **dict**: 包含关键字’label_map’和’score_map’，‘label_map’存储预测结果灰度图，像素值表示对应的类别，’score_map’存储各类别的概率，shape=(h, w, num_classes)。

batch_predict

```
batch_predict(self, img_file_list, transforms=None):
```

DeepLabv3p 模型批量预测接口。需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `DeepLabv3p.test_transforms` 和 `DeepLabv3p.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `batch_predict` 接口时，用户需要再重新定义 test_transforms 传入给 `batch_predict` 接口。

参数

- **img_file_list** (list|tuple): 对列表（或元组）中的图像同时进行预测，列表中的元素可以是预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **transforms** (paddlex.seg.transforms): 数据预处理操作。

返回值

- **dict**: 每个元素都为列表，表示各图像的预测结果。各图像的预测结果用字典表示，包含关键字’label_map’和’score_map’，‘label_map’存储预测结果灰度图，像素值表示对应的类别，’score_map’存储各类别的概率，shape=(h, w, num_classes)。

overlap_tile_predict

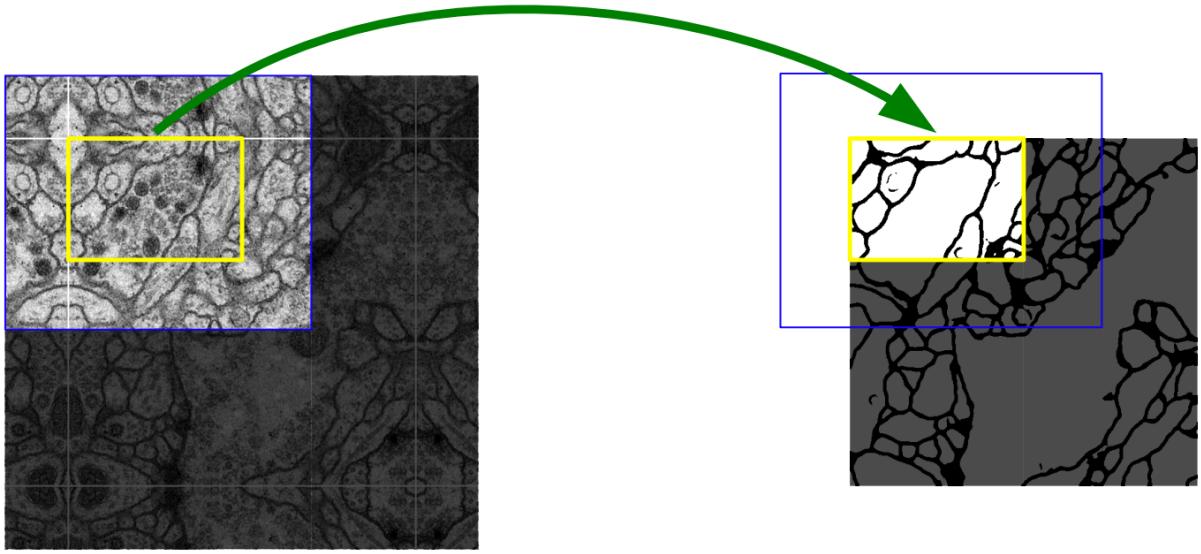
```
overlap_tile_predict(self, img_file, tile_size=[512, 512], pad_size=[64, 64], batch_
↪size=32, transforms=None)
```

DeepLabv3p 模型的滑动预测接口，支持有重叠和无重叠两种方式。

无重叠的滑动窗口预测：在输入图片上以固定大小的窗口滑动，分别对每个窗口下的图像进行预测，最后将各窗口的预测结果拼接成输入图片的预测结果。**使用时需要把参数 `pad_size` 设置为 `[0, 0]`**。

有重叠的滑动窗口预测：在 Unet 论文中，作者提出一种有重叠的滑动窗口预测策略（Overlap-tile strategy）来消除拼接处的裂痕感。对各滑动窗口预测时，会向四周扩展一定的面积，对扩展后的窗口进行预测，例如下图中的蓝色部分区域，到拼接时只取各窗口中间部分的预测结果，例如下

图中的黄色部分区域。位于输入图像边缘处的窗口，其扩展面积下的像素则通过将边缘部分像素镜像填补得到。



需要注意的是，只有在训练过程中定义了 eval_dataset，模型在保存时才会将预测时的图像处理流程保存在 `DeepLabv3p.test_transforms` 和 `DeepLabv3p.eval_transforms` 中。如未在训练时定义 eval_dataset，那在调用预测 `overlap_tile_predict` 接口时，用户需要再重新定义 test_transforms 传入给 `overlap_tile_predict` 接口。

参数

- **img_file** (str|np.ndarray): 预测图像路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **tile_size** (list|tuple): 滑动窗口的大小，该区域内用于拼接预测结果，格式为（W，H）。默认值为 [512, 512]。
- **pad_size** (list|tuple): 滑动窗口向四周扩展的大小，扩展区域内不用于拼接预测结果，格式为（W，H）。默认值为 [64, 64]。
- **batch_size** (int)：对窗口进行批量预测时的批量大小。默认值为 32。
- **transforms** (paddlex.seg.transforms): 数据预处理操作。

返回值

- **dict**: 包含关键字’label_map’和’score_map’，‘label_map’存储预测结果灰度图，像素值表示对应的类别，’score_map’存储各类别的概率，shape=(h, w, num_classes)。

paddlex.seg.UNet

```
paddlex.seg.UNet(num_classes=2, upsample_mode='bilinear', use_bce_loss=False, use_dice_
↪loss=False, class_weight=None, ignore_index=255, input_channel=3)
```

构建 UNet 分割器。

参数

- **num_classes** (int): 类别数。
- **upsample_mode** (str): UNet decode 时采用的上采样方式，取值为’bilinear’时利用双线行差值进行上菜样，当输入其他选项时则利用反卷积进行上菜样，默认为’bilinear’。
- **use_bce_loss** (bool): 是否使用 bce loss 作为网络的损失函数，只能用于两类分割。可与 dice loss 同时使用。默认 False。
- **use_dice_loss** (bool): 是否使用 dice loss 作为网络的损失函数，只能用于两类分割，可与 bce loss 同时使用。当 use_bce_loss 和 use_dice_loss 都为 False 时，使用交叉熵损失函数。默认 False。
- **class_weight** (list/str): 交叉熵损失函数各类损失的权重。当 `class_weight` 为 list 的时候，长度应为 `num_classes`。当 `class_weight` 为 str 时，weight.lower() 应为’dynamic’，这时会根据每一轮各类像素的比重自行计算相应的权重，每一类的权重为：每类的比例 * num_classes。class_weight 取默认值 None 是，各类的权重 1，即平时使用的交叉熵损失函数。
- **ignore_index** (int): label 上忽略的值，label 为 `ignore_index` 的像素不参与损失函数的计算。默认 255。
- **input_channel** (int): 输入图像通道数。默认值 3。

- train 训练接口说明同*DeepLabv3p 模型 train 接口*
- evaluate 评估接口说明同*DeepLabv3p 模型 evaluate 接口*
- predict 预测接口说明同*DeepLabv3p 模型 predict 接口*
- batch_predict 批量预测接口说明同*DeepLabv3p 模型 predict 接口*
- overlap_tile_predict 滑动窗口预测接口同*DeepLabv3p 模型 poverlap_tile_predict 接口*

paddlex.seg.HRNet

```
paddlex.seg.HRNet(num_classes=2, width=18, use_bce_loss=False, use_dice_loss=False,␣
↪class_weight=None, ignore_index=255, input_channel=3)
```

构建 HRNet 分割器。

参数

- **num_classes** (int): 类别数。
- **width** (int|str): 高分辨率分支中特征层的通道数量。默认值为 18。可选择取值为 [18, 30, 32, 40, 44, 48, 60, 64, ‘18_small_v1’]。’18_small_v1’ 是 18 的轻量级版本。

- **use_bce_loss** (bool): 是否使用 bce loss 作为网络的损失函数，只能用于两类分割。可与 dice loss 同时使用。默认 False。
- **use_dice_loss** (bool): 是否使用 dice loss 作为网络的损失函数，只能用于两类分割，可与 bce loss 同时使用。当 use_bce_loss 和 use_dice_loss 都为 False 时，使用交叉熵损失函数。默认 False。
- **class_weight** (list|str): 交叉熵损失函数各类损失的权重。当 `class_weight` 为 list 的时候，长度应为 `num_classes`。当 `class_weight` 为 str 时，weight.lower() 应为' dynamic'，这时会根据每一轮各类像素的比重自行计算相应的权重，每一类的权重为：每类的比例 * num_classes。class_weight 取默认值 None 是，各类的权重 1，即平时使用的交叉熵损失函数。
- **ignore_index** (int): label 上忽略的值，label 为 `ignore_index` 的像素不参与损失函数的计算。默认 255。
- **input_channel** (int): 输入图像通道数。默认值 3。

- train 训练接口说明同*DeepLabv3p 模型 train 接口*
- evaluate 评估接口说明同*DeepLabv3p 模型 evaluate 接口*
- predict 预测接口说明同*DeepLabv3p 模型 predict 接口*
- batch_predict 批量预测接口说明同*DeepLabv3p 模型 predict 接口*
- overlap_tile_predict 滑动窗预测接口同*DeepLabv3p 模型 poverlap_tile_predict 接口*

paddlex.seg.FastSCNN

```
paddlex.seg.FastSCNN(num_classes=2, use_bce_loss=False, use_dice_loss=False, class_
↪weight=None, ignore_index=255, multi_loss_weight=[1.0], input_channel=3)
```

构建 FastSCNN 分割器。

参数

- **num_classes** (int): 类别数。
- **use_bce_loss** (bool): 是否使用 bce loss 作为网络的损失函数，只能用于两类分割。可与 dice loss 同时使用。默认 False。
- **use_dice_loss** (bool): 是否使用 dice loss 作为网络的损失函数，只能用于两类分割，可与 bce loss 同时使用。当 use_bce_loss 和 use_dice_loss 都为 False 时，使用交叉熵损失函数。默认 False。
- **class_weight** (list/str): 交叉熵损失函数各类损失的权重。当 `class_weight` 为 list 的时候，长度应为 `num_classes`。当 `class_weight` 为 str 时，weight.lower() 应为' dynamic'，这时会根据每一轮各类像素的比重自行计算相应的权重，每一类的权重为：每类的比例 *

num_classes。class_weight 取默认值 None 是，各类的权重 1，即平时使用的交叉熵损失函数。

- **ignore_index** (int): label 上忽略的值，label 为 `ignore_index` 的像素不参与损失函数的计算。默认 255。
- **multi_loss_weight** (list): 多分支上的 loss 权重。默认计算一个分支上的 loss，即默认值为 [1.0]。也支持计算两个分支或三个分支上的 loss，权重按 [fusion_branch_weight, higher_branch_weight, lower_branch_weight] 排列，fusion_branch_weight 为空间细节分支和全局上下文分支融合后的分支上的 loss 权重，higher_branch_weight 为空间细节分支上的 loss 权重，lower_branch_weight 为全局上下文分支上的 loss 权重，若 higher_branch_weight 和 lower_branch_weight 未设置则不会计算这两个分支上的 loss。
- **input_channel** (int): 输入图像通道数。默认值 3。

- train 训练接口说明同*DeepLabv3p 模型 train 接口*
- evaluate 评估接口说明同*DeepLabv3p 模型 evaluate 接口*
- predict 预测接口说明同*DeepLabv3p 模型 predict 接口*
- batch_predict 批量预测接口说明同*DeepLabv3p 模型 predict 接口*
- overlap_tile_predict 滑动窗预测接口同*DeepLabv3p 模型 poverlap_tile_predict 接口*

29.5 模型压缩

29.5.1 paddlex.slim.prune.analysis

计算参数敏感度

```
paddlex.slim.prune.analysis(model, dataset, batch_size, save_file='model.sensi.data')
```

此函数接口与 `paddlex.slim.cal_params_sensitivites` 接口功能一致，仅修改了函数名，参数名，顺序和默认值，推荐使用此接口。

使用示例参考教程-模型裁剪训练

29.5.2 paddlex.slim.cal_params_sensitivities

此函数接口与 `paddlex.slim.prune.analysis` 功能一致，推荐使用 `paddlex.slim.prune.analysis` 接口**计算参数敏感度**

```
paddlex.slim.cal_params_sensitivities(model, save_file, eval_dataset, batch_size=8)
```

计算模型中可剪裁参数在验证集上的敏感度，并将敏感度信息保存至文件 `save_file`

1. 获取模型中可剪裁卷积 Kernel 的名称。
2. 计算每个可剪裁卷积 Kernel 不同剪裁率下的敏感度。

【注意】卷积的敏感度是指按照剪裁率将模型剪裁后模型精度的损失。选择合适的敏感度，对应地也能确定最终模型需要剪裁的参数列表和各剪裁参数对应的剪裁率。

查看使用示例

参数

- **model** (paddlex.cls.models/paddlex.det.models/paddlex.seg.models): paddlex 加载的模型。
- **save_file** (str): 计算的得到的 sensetives 文件存储路径。
- **eval_dataset** (paddlex.datasets): 评估数据集的读取器。
- **batch_size** (int): 评估时的 batch_size 大小。

29.5.3 paddlex.slim.export_quant_model

导出量化模型

```
paddlex.slim.export_quant_model(model, test_dataset, batch_size=2, batch_num=10, save_
↪dir='./quant_model', cache_dir='./temp')
```

导出量化模型，该接口实现了 Post Quantization 量化方式，需要传入测试数据集，并设定 `batch_size` 和 `batch_num`。量化过程中会以数量为 `batch_size` * `batch_num` 的样本数据的计算结果为统计信息完成模型的量化。

参数

- **model**(paddlex.cls.models/paddlex.det.models/paddlex.seg.models): paddlex 加载的模型。
- **test_dataset**(paddlex.dataset): 测试数据集。
- **batch_size**(int): 进行前向计算时的批数据大小。
- **batch_num**(int): 进行向前计算时批数据数量。
- **save_dir**(str): 量化后模型的保存目录。
- **cache_dir**(str): 量化过程中的统计数据临时存储目录。

使用示例

点击下载如下示例中的模型，数据集

```
import paddlex as pdx
model = pdx.load_model('vegetables_mobilenet')
test_dataset = pdx.datasets.ImageNet(
                    data_dir='vegetables_cls',
```

(下页继续)

(续上页)

```
                    file_list='vegetables_cls/train_list.txt',
                    label_list='vegetables_cls/labels.txt',
                    transforms=model.eval_transforms)
pdx.slim.export_quant_model(model, test_dataset, save_dir='./quant_mobilenet')
```

29.6 预测结果可视化

PaddleX 提供了一系列模型预测和结果分析的可视化函数。

29.6.1 paddlex.det.visualize

目标检测/实例分割预测结果可视化

```
paddlex.det.visualize(image, result, threshold=0.5, save_dir='./', color=None)
```

将目标检测/实例分割模型预测得到的 Box 框和 Mask 在原图上进行可视化。

参数

- **image** (str|np.ndarray): 原图文件路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **result** (str): 模型预测结果。
- **threshold**(float): score 阈值，将 Box 置信度低于该阈值的框过滤不进行可视化。默认 0.5
- **save_dir**(str): 可视化结果保存路径。若为 None，则表示不保存，该函数将可视化的结果以 np.ndarray 的形式返回；若设为目录路径，则将可视化结果保存至该目录下。默认值为’./’。
- **color**(list|tuple|np.array): 各类别的 BGR 颜色值组成的数组，形状为 Nx3（N 为类别数量），数值范围为 [0, 255]。例如针对 2 个类别的 [[255, 0, 0], [0, 255, 0]]。若为 None，则自动生成各类别的颜色。默认值为 None。

使用示例

点击下载如下示例中的模型

```
import paddlex as pdx
model = pdx.load_model('xiaoduxiong_epoch_12')
result = model.predict('./xiaoduxiong_epoch_12/xiaoduxiong.jpeg')
pdx.det.visualize('./xiaoduxiong_epoch_12/xiaoduxiong.jpeg', result, save_dir='./')
# 预测结果保存在./visualize_xiaoduxiong.jpeg
```

29.6.2 paddlex.seg.visualize

语义分割模型预测结果可视化

```
paddlex.seg.visualize(image, result, weight=0.6, save_dir='./', color=None)
```

将语义分割模型预测得到的 Mask 在原图上进行可视化。

参数

- **image** (str|np.ndarray): 原图文件路径或 numpy 数组 (HWC 排列，BGR 格式)。
- **result** (str): 模型预测结果。
- **weight**(float): mask 可视化结果与原图权重因子，weight 表示原图的权重。默认 0.6。
- **save_dir**(str): 可视化结果保存路径。若为 None，则表示不保存，该函数将可视化的结果以 np.ndarray 的形式返回；若设为目录路径，则将可视化结果保存至该目录下。默认值为’./’。
- **color** (list): 各类别的 BGR 颜色值组成的列表。例如两类时可设置为 [255, 255, 255, 0, 0, 0]。默认值为 None，则使用默认生成的颜色列表。

使用示例

点击下载如下示例中的模型和测试图片

```
import paddlex as pdx
model = pdx.load_model('cityscape_deeplab')
result = model.predict('city.png')
pdx.det.visualize('city.png', result, save_dir='./')
# 预测结果保存在./visualize_city.png
```

29.6.3 paddlex.det.draw_pr_curve

目标检测/实例分割准确率-召回率可视化

```
paddlex.det.draw_pr_curve(eval_details_file=None, gt=None, pred_bbox=None, pred_
↪mask=None, iou_thresh=0.5, save_dir='./')
```

将目标检测/实例分割模型评估结果中各个类别的准确率和召回率的对应关系进行可视化，同时可视化召回率和置信度阈值的对应关系。

注：PaddleX 在训练过程中保存的模型目录中，均包含 `eval_result.json` 文件，可将此文件路径传给 `eval_details_file` 参数，设定 `iou_threshold` 即可得到对应模型在验证集上的 PR 曲线图。

参数

- **eval_details_file** (str): 模型评估结果的保存路径，包含真值信息和预测结果。默认值为 None。
- **gt** (list): 数据集的真值信息。默认值为 None。
- **pred_bbox** (list): 模型在数据集上的预测框。默认值为 None。
- **pred_mask** (list): 模型在数据集上的预测 mask。默认值为 None。
- **iou_thresh** (float): 判断预测框或预测 mask 为真阳时的 IoU 阈值。默认值为 0.5。
- **save_dir** (str): 可视化结果保存路径。默认值为'./'。

注意： `eval_details_file` 的优先级更高，只要 `eval_details_file` 不为 None，就会从 `eval_details_file` 提取真值信息和预测结果做分析。当 `eval_details_file` 为 None 时，则用 `gt`、`pred_mask`、`pred_mask` 做分析。

使用示例

点击下载如下示例中的模型和数据集

方式一：分析训练过程中保存的模型文件夹中的评估结果文件 `eval_details.json`，例如模型中的 `eval_details.json`。

```
import paddlex as pdx
eval_details_file = 'insect_epoch_270/eval_details.json'
pdx.det.draw_pr_curve(eval_details_file, save_dir='./insect')
```

方式二：分析模型评估函数返回的评估结果。

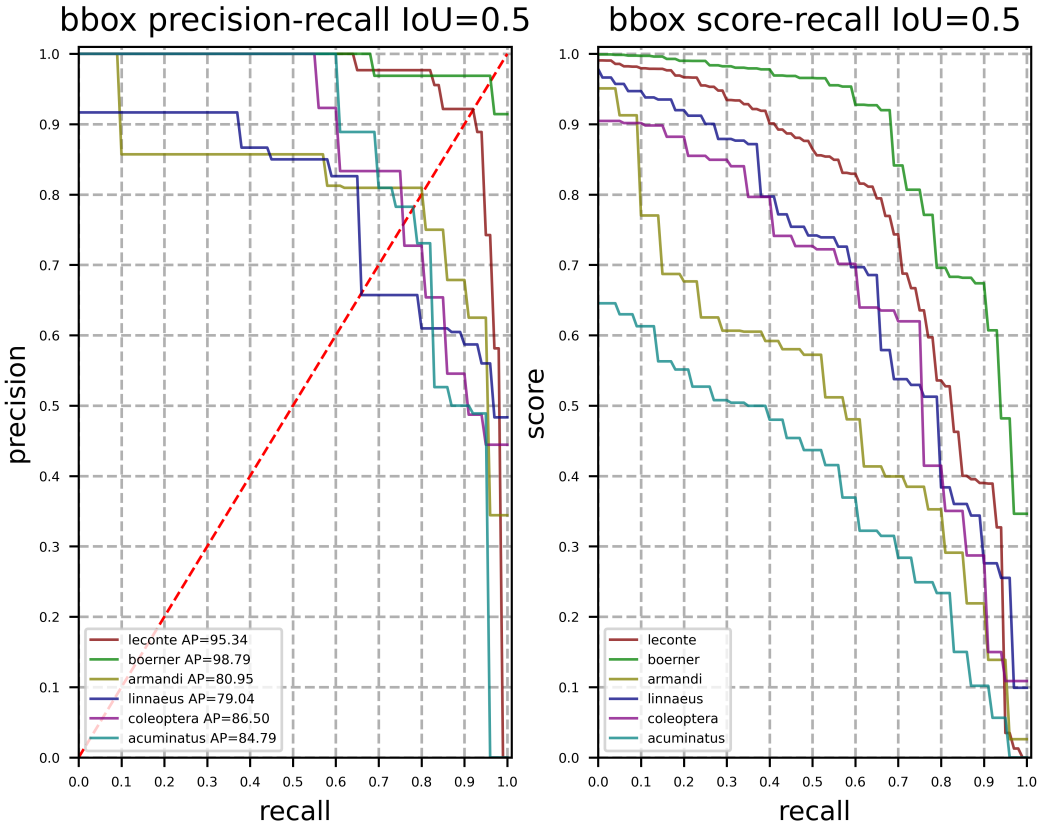
```
import os
# 选择使用 0 号卡
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

from paddlex.det import transforms
import paddlex as pdx

model = pdx.load_model('insect_epoch_270')
eval_dataset = pdx.datasets.VOCDetection(
    data_dir='insect_det',
    file_list='insect_det/val_list.txt',
    label_list='insect_det/labels.txt',
    transforms=model.eval_transforms)
metrics, evaluate_details = model.evaluate(eval_dataset, batch_size=8, return_
↪details=True)
```

(下页继续)

(续上页)

```
gt = evaluate_details['gt']
bbox = evaluate_details['bbox']
pdx.det.draw_pr_curve(gt=gt, pred_bbox=bbox, save_dir='./insect')
```

预测框的各个类别的准确率和召回率的对应关系、召回率和置信度阈值的对应关系可视化如下:



29.6.4 paddlex.slim.visualzie

模型剪裁比例可视化分析

```
paddlex.slim.visualize(model, sensitivities_file, save_dir='./')
```

利用此接口，可以分析在不同的 `eval_metric_loss` 参数下，模型被剪裁的比例情况。可视化结果纵轴为 eval_metric_loss 参数值，横轴为对应的模型被剪裁的比例。`eval_metric_loss` 即卷积的敏感度，是指按照剪裁率将模型剪裁后模型精度的损失。

参数

- **model** (paddlex.cv.models): 使用 PaddleX 加载的模型。
- **sensitivities_file** (str): 模型各参数在验证集上计算得到的参数敏感度信息文件。
- **save_dir**(str): 可视化结果保存路径，默认为当前目录

使用示例

点击下载示例中的模型和sensitivities_file

```
import paddlex as pdx
model = pdx.load_model('vegetables_mobilenet')
pdx.slim.visualize(model, 'mobilenetv2.sensitivities', save_dir='./')
# 可视化结果保存在./sensitivities.png
```

29.6.5 paddlex.transforms.visualize

数据预处理/增强过程可视化

```
paddlex.transforms.visualize(dataset,
                             img_count=3,
                             save_dir='vdl_output')
```

对数据预处理/增强中间结果进行可视化。可使用 VisualDL 查看中间结果：

1. VisualDL 启动方式: visualdl –logdir vdl_output/image_transforms –port 8001
2. 浏览器打开 https://0.0.0.0:8001 ，在页面上面点击『样本数据-图像』即可。其中 0.0.0.0 为本机访问，如为远程服务, 改成相应机器 IP

参数

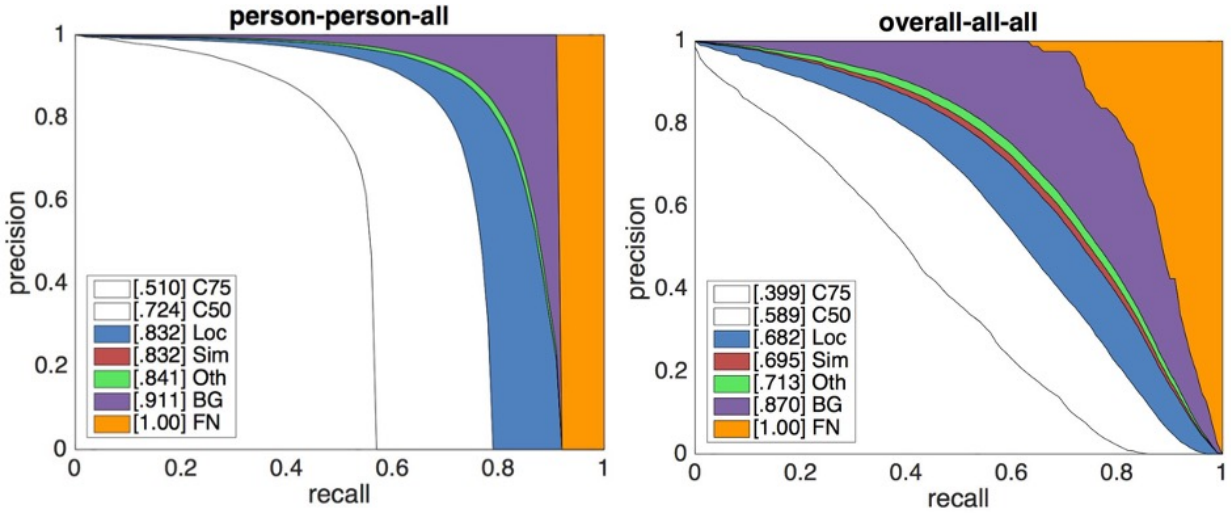
- **dataset** (paddlex.datasets): 数据集读取器。
- **img_count** (int): 需要进行数据预处理/增强的图像数目。默认为 3。
- **save_dir** (str): 日志保存的路径。默认为’vdl_output’。

29.6.6 paddlex.det.coco_error_analysis

分析模型预测错误的原因

```
paddlex.det.coco_error_analysis(eval_details_file=None, gt=None, pred_bbox=None, pred_
↪mask=None, save_dir='./output')
```

逐个分析模型预测错误的原因，并将分析结果以图表的形式展示。分析结果图表示例如下：



左图显示的是 `person` 类的分析结果，有图显示的是所有类别整体的分析结果。

分析图表展示了 7 条 Precision-Recall（PR）曲线，每一条曲线表示的 Average Precision （AP）比它左边那条高，原因是逐步放宽了评估要求。以 `person` 类为例，各条 PR 曲线的评估要求解释如下：

- C75: 在 IoU 设置为 0.75 时的 PR 曲线, AP 为 0.510。
- C50: 在 IoU 设置为 0.5 时的 PR 曲线，AP 为 0.724。C50 与 C75 之间的白色区域面积代表将 IoU 从 0.75 放宽至 0.5 带来的 AP 增益。
- Loc: 在 IoU 设置为 0.1 时的 PR 曲线，AP 为 0.832。Loc 与 C50 之间的蓝色区域面积代表将 IoU 从 0.5 放宽至 0.1 带来的 AP 增益。蓝色区域面积越大，表示越多的检测框位置不够精准。
- Sim: 在 Loc 的基础上，如果检测框与真值框的类别不相同，但两者同属于一个亚类，则不认为该检测框是错误的，在这种评估要求下的 PR 曲线, AP 为 0.832。Sim 与 Loc 之间的红色区域面积越大，表示子类间的混淆程度越高。
- Oth: 在 Sim 的基础上，如果检测框与真值框的亚类不相同，则不认为该检测框是错误的，在这种评估要求下的 PR 曲线，AP 为 0.841。Oth 与 Sim 之间的绿色区域面积越大，表示亚类间的混淆程度越高。
- BG: 在 Oth 的基础上，背景区域上的检测框不认为是错误的，在这种评估要求下的 PR 曲线，AP 为 91.1。BG 与 Oth 之间的紫色区域面积越大，表示背景区域被误检的数量越多。
- FN: 在 BG 的基础上，漏检的真值框不认为是错误的，在这种评估要求下的 PR 曲线，AP 为 1.00。FN 与 BG 之间的橙色区域面积越大，表示漏检的真值框数量越多。

更为详细的说明参考COCODataset 官网给出分析工具说明

参数

- **eval_details_file** (str): 模型评估结果的保存路径，包含真值信息和预测结果。默认值为 None。
- **gt** (list): 数据集的真值信息。默认值为 None。
- **pred_bbox** (list): 模型在数据集上的预测框。默认值为 None。
- **pred_mask** (list): 模型在数据集上的预测 mask。默认值为 None。
- **save_dir** (str): 可视化结果保存路径。默认值为'./output'。

注意： `eval_details_file` 的优先级更高，只要 `eval_details_file` 不为 None，就会从 `eval_details_file` 提取真值信息和预测结果做分析。当 `eval_details_file` 为 None 时，则用 `gt`、`pred_mask`、`pred_mask` 做分析。

使用示例

点击下载如下示例中的模型和数据集

方式一：分析训练过程中保存的模型文件夹中的评估结果文件 `eval_details.json`，例如模型中的 `eval_details.json`。

```
import paddlex as pdx
eval_details_file = 'insect_epoch_270/eval_details.json'
pdx.det.coco_error_analysis(eval_details_file, save_dir='./insect')
```

方式二：分析模型评估函数返回的评估结果。

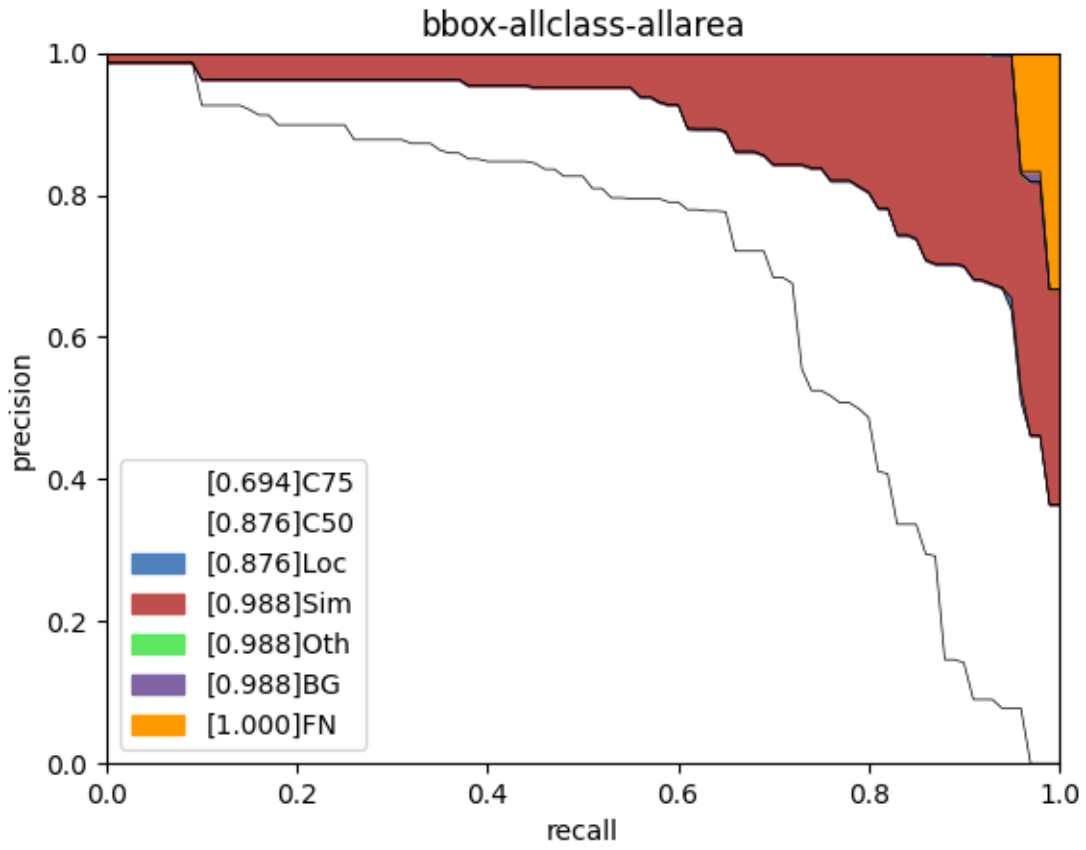
```
import os
# 选择使用 0 号卡
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

from paddlex.det import transforms
import paddlex as pdx

model = pdx.load_model('insect_epoch_270')
eval_dataset = pdx.datasets.VOCDetection(
    data_dir='insect_det',
    file_list='insect_det/val_list.txt',
    label_list='insect_det/labels.txt',
    transforms=model.eval_transforms)
metrics, evaluate_details = model.evaluate(eval_dataset, batch_size=8, return_
↪details=True)
gt = evaluate_details['gt']
```

(下页继续)

(续上页)

```
bbox = evaluate_details['bbox']
pdx.det.coco_error_analysis(gt=gt, pred_bbox=bbox, save_dir='./insect')
```

所有类别整体的分析结果示例如下：



29.7 模型可解释性

目前 PaddleX 支持对于图像分类的结果以可视化的方式进行解释，支持 LIME 和 NormLIME 两种可解释性算法。

29.7.1 paddlex.interpret.lime

LIME 可解释性结果可视化

```
paddlex.interpret.lime(img_file,
                       model,
```

(下页继续)

(续上页)

```
                        num_samples=3000,
                        batch_size=50,
                        save_dir='./')
```

使用 LIME 算法将模型预测结果的可解释性可视化。LIME 表示与模型无关的局部可解释性，可以解释任何模型。LIME 的思想是以输入样本为中心，在其附近的空间中进行随机采样，每个采样通过原模型得到新的输出，这样得到一系列的输入和对应的输出，LIME 用一个简单的、可解释的模型（比如线性回归模型）来拟合这个映射关系，得到每个输入维度的权重，以此来解释模型。

注意：可解释性结果可视化目前只支持分类模型。

参数

- **img_file** (str): 预测图像路径。
- **model** (paddlex.cv.models): paddlex 中的模型。
- **num_samples** (int): LIME 用于学习线性模型的采样数，默认为 3000。
- **batch_size** (int): 预测数据 batch 大小，默认为 50。
- **save_dir** (str): 可解释性可视化结果（保存为 png 格式文件）和中间文件存储路径。

可视化效果

使用示例

对预测可解释性结果可视化的过程可参见代码。

29.7.2 paddlex.interpret.normlime

NormLIME 可解释性结果可视化

```
paddlex.interpret.normlime(img_file,
                           model,
                           dataset=None,
                           num_samples=3000,
                           batch_size=50,
                           save_dir='./',
                           normlime_weights_file=None)
```

使用 NormLIME 算法将模型预测结果的可解释性可视化。NormLIME 是利用一定数量的样本来出一个全局的解释。由于 NormLIME 计算量较大，此处采用一种简化的方式：使用一定数量的测试样本（目前默认使用所有测试样本），对每个样本进行特征提取，映射到同一个特征空间；然后以此特征做为输入，以模型输出做为输出，使用线性回归对其进行拟合，得到一个全局的输入和输出的关系。之后，对一测试样本进行解释时，使用 NormLIME 全局的解释，来对 LIME 的结果进行滤波，使最终的可视化结果更加稳定。

注意：可解释性结果可视化目前只支持分类模型。

参数

- **img_file** (str): 预测图像路径。
- **model** (paddlex.cv.models): paddlex 中的模型。
- **dataset** (paddlex.datasets): 数据集读取器，默认为 None。
- **num_samples** (int): LIME 用于学习线性模型的采样数，默认为 3000。
- **batch_size** (int): 预测数据 batch 大小，默认为 50。
- **save_dir** (str): 可解释性可视化结果（保存为 png 格式文件）和中间文件存储路径。
- **normlime_weights_file** (str): NormLIME 初始化文件名，若不存在，则计算一次，保存于该路径；若存在，则直接载入。

注意： `dataset` 读取的是一个数据集，该数据集不宜过大，否则计算时间会较长，但应包含所有类别的数据。NormLIME 可解释性结果可视化目前只支持分类模型。

使用示例

对预测可解释性结果可视化的过程可参见代码。

CHAPTER 30

PaddleX 模型库

30.1 图像分类模型

表中模型准确率均为在 ImageNet 数据集上测试所得，表中符号-表示相关指标暂未测试，预测速度测试环境如下所示:

- CPU 的评估环境基于骁龙 855（SD855）。
- GPU 评估环境基于 T4 机器，在 FP32+TensorRT 配置下运行 500 次测得（去除前 10 次的 warmup 时间）。

30.1.1 移动端系列

30.1.2 其他系列

30.2 目标检测模型

表中模型精度 BoxAP 通过 `evaluate()` 接口测试 MSCOCO 验证集得到，符号-表示相关指标暂未测试，预测时间在以下环境测试所的:

- 测试环境:
  - CUDA 9.0
  - CUDNN 7.5

  - PaddlePaddle v1.6
  - TensorRT-5.1.2.2
  - GPU 分别为: Tesla V100

- 测试方式:
  - 为了方便比较不同模型的推理速度，输入采用同样大小的图片，为 3x640x640。
  - Batch Size=1
  - 去掉前 10 轮 warmup 时间，测试 100 轮的平均时间，单位 ms/image，包括输入数据拷贝至 GPU 的时间、计算时间、数据拷贝至 CPU 的时间。
  - 采用 Fluid C++ 预测引擎，开启 FP32 TensorRT 配置。
  - 测试时开启了 FLAGS_cudnn_exhaustive_search=True，使用 exhaustive 方式搜索卷积计算算法。

30.3 实例分割模型

表中模型精度 BoxAP/MaskAP 通过 `evaluate()` 接口测试 MSCOCO 验证集得到，符号-表示相关指标暂未测试，预测时间在以下环境测试所的

- 测试环境:
  - CUDA 9.0
  - CUDNN 7.5
  - PaddlePaddle v1.6
  - TensorRT-5.1.2.2
  - GPU 分别为: Tesla V100

- 测试方式:
  - 为了方便比较不同模型的推理速度，输入采用同样大小的图片，为 3x640x640。
  - Batch Size=1
  - 去掉前 10 轮 warmup 时间，测试 100 轮的平均时间，单位 ms/image，包括输入数据拷贝至 GPU 的时间、计算时间、数据拷贝至 CPU 的时间。
  - 采用 Fluid C++ 预测引擎，开启 FP32 TensorRT 配置。
  - 测试时开启了 FLAGS_cudnn_exhaustive_search=True，使用 exhaustive 方式搜索卷积计算算法。

30.4 语义分割模型

以下指标均在 MSCOCO 验证集上测试得到，表中符号-表示相关指标暂未测试。

以下指标均在 Cityscapes 验证集上测试得到，表中符号-表示相关指标暂未测试。

CHAPTER 31

PaddleX 指标及日志

PaddleX 在模型训练、评估过程中，都会有相应的日志和指标反馈，本文档用于说明这些日志和指标的含义。

31.1 训练通用统计信息

PaddleX 所有模型在训练过程中，输出的日志信息都包含了 6 个通用的统计信息，用于辅助用户进行模型训练，例如**分割模型**的训练日志，如下图所示。

```
[TRAIN] Epoch=4/20, Step=62/66, loss=0.007226, lr=0.008215, time_each_step=0.41s, eta=0:9:44
[TRAIN] Epoch=4/20, Step=64/66, loss=0.012199, lr=0.008201, time_each_step=0.41s, eta=0:9:43
[TRAIN] Epoch=4/20, Step=66/66, loss=0.003387, lr=0.008187, time_each_step=0.41s, eta=0:9:42
[TRAIN] Epoch 4 finished, loss=0.007828, lr=0.008414 .
```

各字段含义如下:

不同模型的日志中除了上述通用字段外，还有其它字段，这些字段含义可见文档后面对各任务模型的描述。

31.2 评估通用统计信息

PaddleX 所有模型在训练过程中会根据用户设定的 `save_interval_epochs` 参数，每间隔一定轮数进行评估和保存。例如**分类模型**的评估日志，如下图所示。

```
Start to evaluating(total_samples=240, total_steps=8)...
##################################################################################################
[EVAL] Finished, Epoch=2, acc1=0.258333, acc5=0.7875 .
Model saved in output/mobilenetv2/best_model.
Model saved in output/mobilenetv2/epoch_2.
Current evaluated best model in eval_dataset is epoch_2, acc1=0.25833333333333336
```

上图中第 1 行表明验证数据集中样本数为 240，需要迭代 8 步才能评估完所有验证数据；第 5 行用于表明第 2 轮的模型已经完成保存操作；第 6 行则表明当前保存的模型中，第 2 轮的模型在验证集上指标最优（分类任务看 `acc1`，此时 `acc1` 值为 0.258333），最优模型会保存在 `best_model` 目录中。

31.3 分类特有统计信息

31.3.1 训练日志字段

分类任务的训练日志除了通用统计信息外，还包括 `acc1` 和 `acc5` 两个特有字段。

> 注：acck 准确率是针对一张图片进行计算的：把模型在各个类别上的预测得分按从高往低进行排序，取出前 k 个预测类别，若这 k 个预测类别包含了真值类，则认为该图片分类正确。

```
[TRAIN] Epoch=2/10, Step=22/26, loss=0.370639, acc1=0.8125, acc5=1.0, lr=0.025, time_each_step=0.11s, eta=0:1:14
[TRAIN] Epoch=2/10, Step=24/26, loss=1.782637, acc1=0.71875, acc5=1.0, lr=0.025, time_each_step=0.11s, eta=0:1:14
[TRAIN] Epoch=2/10, Step=26/26, loss=0.462437, acc1=0.90625, acc5=1.0, lr=0.025, time_each_step=0.11s, eta=0:1:14
[TRAIN] Epoch 2 finished, loss=1.240256, acc1=0.759615, acc5=0.998798, lr=0.025 .
```

上图中第 1 行中的 `acc1` 表示参与当前迭代步数的训练样本的平均 top1 准确率，值越高代表模型越优；`acc5` 表示参与当前迭代步数的训练样本的平均 top5（若类别数 n 少于 5，则为 topn）准确率，值越高代表模型越优。第 4 行中的 `loss` 表示整个训练集的平均损失函数值，`acc1` 表示整个训练集的平均 top1 准确率，`acc5` 表示整个训练集的平均 top5 准确率。

31.3.2 评估日志字段

```
Start to evaluating(total_samples=240, total_steps=8)...
##################################################################################################
[EVAL] Finished, Epoch=2, acc1=0.258333, acc5=0.7875 .
Model saved in output/mobilenetv2/best_model.
Model saved in output/mobilenetv2/epoch_2.
Current evaluated best model in eval_dataset is epoch_2, acc1=0.25833333333333336
```

上图中第 3 行中的 `acc1` 表示整个验证集的平均 top1 准确率，`acc5` 表示整个验证集的平均 top5 准确率。

31.4 检测特有统计信息

31.4.1 训练日志字段

YOLOv3

YOLOv3 的训练日志只包括训练通用统计信息（见上文训练通用统计信息）。

```
[TRAIN] Epoch=1/270, Step=204/211, loss=65.632935, lr=2.5e-05, time_each_step=0.41s, eta=7:24:50
[TRAIN] Epoch=1/270, Step=206/211, loss=49.226215, lr=2.6e-05, time_each_step=0.41s, eta=7:22:34
[TRAIN] Epoch=1/270, Step=208/211, loss=66.177971, lr=2.6e-05, time_each_step=0.41s, eta=7:30:9
[TRAIN] Epoch=1/270, Step=210/211, loss=61.262436, lr=2.6e-05, time_each_step=0.42s, eta=7:35:28
[TRAIN] Epoch 1 finished, loss=597.357239, lr=1.3e-05 .
```

上图中第 5 行 `loss` 表示整个训练集的平均损失函数 loss 值。

FasterRCNN

FasterRCNN 的训练日志除了通用统计信息外，还包括 `loss_cls`、`loss_bbox`、`loss_rpn_cls` 和 `loss_rpn_bbox`，这些字段的含义如下:

```
2020-04-26 17:22:45 [INFO]      [TRAIN] Epoch=1/12, Step=216/221, loss=0.716083, loss_cls=0.379841, loss_bbox=0.272749, loss_rpn_cls=0.044854, loss_rpn_bbox=0.018638, lr=0.
001013, time_each_step=0.17s, eta=0:9:0
2020-04-26 17:22:46 [INFO]      [TRAIN] Epoch=1/12, Step=218/221, loss=0.536624, loss_cls=0.255749, loss_bbox=0.253681, loss_rpn_cls=0.011562, loss_rpn_bbox=0.015631, lr=0.
001014, time_each_step=0.17s, eta=0:9:0
2020-04-26 17:22:46 [INFO]      [TRAIN] Epoch=1/12, Step=220/221, loss=0.610345, loss_cls=0.308021, loss_bbox=0.248639, loss_rpn_cls=0.021446, loss_rpn_bbox=0.03224, lr=0.0
01016, time_each_step=0.17s, eta=0:9:0
2020-04-26 17:22:46 [INFO]      [TRAIN] Epoch 1 finished, loss=0.797818, loss_cls=0.39108, loss_bbox=0.305496, loss_rpn_cls=0.078625, loss_rpn_bbox=0.022617, lr=0.000925 .
```

上图中第 1 行 `loss`, `loss_cls`、`loss_bbox`、`loss_rpn_clss`、`loss_rpn_bbox` 都是参与当前迭代步数的训练样本的损失值，而第 7 行是针整个训练集的损失函数值。

MaskRCNN

MaskRCNN 的训练日志除了通用统计信息外，还包括 `loss_cls`、`loss_bbox`、`loss_mask`、`loss_rpn_cls` 和 `loss_rpn_bbox`，这些字段的含义如下:

```
2020-04-26 17:53:16 [INFO]      [TRAIN] Epoch=1/12, Step=216/221, loss=1.049532, loss_cls=0.451329, loss_bbox=0.364285, loss_mask=0.199765, loss_rpn_cls=0.022088, loss_rpn_
bbox=0.012065, lr=0.000775, time_each_step=0.21s, eta=0:11:15
2020-04-26 17:53:16 [INFO]      [TRAIN] Epoch=1/12, Step=218/221, loss=0.849116, loss_cls=0.306857, loss_bbox=0.280559, loss_mask=0.237476, loss_rpn_cls=0.010742, loss_rpn_
bbox=0.013482, lr=0.000778, time_each_step=0.21s, eta=0:11:14
2020-04-26 17:53:16 [INFO]      [TRAIN] Epoch=1/12, Step=220/221, loss=1.07725, loss_cls=0.367705, loss_bbox=0.345688, loss_mask=0.31945, loss_rpn_cls=0.01434, loss_rpn_bbo
x=0.030067, lr=0.000782, time_each_step=0.21s, eta=0:11:14
2020-04-26 17:53:17 [INFO]      [TRAIN] Epoch 1 finished, loss=1.618055, loss_cls=0.575324, loss_bbox=0.383326, loss_mask=0.457685, loss_rpn_cls=0.182465, loss_rpn_bbox=0.0
19253, lr=0.0006 .
```

上图中第 1 行 `loss`, `loss_cls`、`loss_bbox`、`loss_mask`、`loss_rpn_clss`、`loss_rpn_bbox` 都是参与当前迭代步数的训练样本的损失值，而第 7 行是针整个训练集的损失函数值。

31.4.2 评估日志字段

检测可以使用两种评估标准：VOC 评估标准和 COCO 评估标准。

VOC 评估标准

```
Start to evaluating(total_samples=245, total_steps=31)...
##################################################################
[EVAL] Finished, Epoch=2, bbox_map=9.2085 .
```

注：map 为平均准确率的平均值，即 IoU(Intersection Over Union) 取 0.5 时各个类别的准确率-召回率曲线下面积的平均值。

上图中第 3 行 bbox_map 表示检测任务中整个验证集的平均准确率平均值。

COCO 评估标准

注：COCO 评估指标可参见COCO 官网解释。PaddleX 主要反馈 mmAP，即 AP at IoU=.50:.05:.95 这项指标，为在各个 IoU 阈值下平均准确率平均值（mAP）的平均值。

COCO 格式的数据集不仅可以用于训练目标检测模型，也可以用于训练实例分割模型。在目标检测中，PaddleX 主要反馈针对检测框的 bbox_mmAP 指标；在实例分割中，还包括针对 Mask 的 seg_mmAP 指标。如下所示，第一张日志截图为目标检测的评估结果，第二张日志截图为实例分割的评估结果。

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.404
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.862
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.292
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.402
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.406
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.274
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.521
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.532
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.400
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.532
2020-04-26 17:22:55 [INFO]      [EVAL] Finished, Epoch=1, bbox_mmap=0.404178 .
```

上图中红框标注的 bbox_mmap 表示整个验证集的检测框平均准确率平均值。

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.347
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.737
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.255
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.041
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.352
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.241
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.457
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.468
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.440
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.470
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *segm*
DONE (t=0.68s).
Accumulating evaluation results...
DONE (t=0.08s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.466
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.730
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.533
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.005
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.475
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.299
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.579
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.587
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.200
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.593
2020-04-26 17:15:51 [INFO]      [EVAL] Finished, Epoch=1, bbox_mmap=0.347233, segm_mmap=0.466408 .
```

上图中红框标注的 bbox_mmap 和 seg_mmap 分别表示整个验证集的检测框平均准确率平均值、Mask 平均准确率平均值。

31.5 分割特有统计信息

31.5.1 训练日志字段

语义分割的训练日志只包括训练通用统计信息（见上文训练通用统计信息）。

```
[TRAIN] Epoch=4/20, Step=62/66, loss=0.007226, lr=0.008215, time_each_step=0.41s, eta=0:9:44
[TRAIN] Epoch=4/20, Step=64/66, loss=0.012199, lr=0.008201, time_each_step=0.41s, eta=0:9:43
[TRAIN] Epoch=4/20, Step=66/66, loss=0.003387, lr=0.008187, time_each_step=0.41s, eta=0:9:42
[TRAIN] Epoch 4 finished, loss=0.007828, lr=0.008414 .
```

31.5.2 评估日志字段

语义分割的评估日志包括了 miou、category_iou、macc、category_acc、kappa，这些字段的含义如下：

```
2020-04-26 17:58:50 [INFO]      Start to evaluating(total_samples=76, total_steps=19)...
100%|########################################################################################################################| 19/19 [00:05<00:00,  3.73it/s]
2020-04-26 17:58:55 [INFO]      [EVAL] Finished, Epoch=4, miou=0.901114, category_iou=[0.9961154 0.8061131], macc=0.996177, category_acc=[0.99748952 0.92140862], kappa=0.89
0705 .
```

CHAPTER 32

PaddleX 可解释性

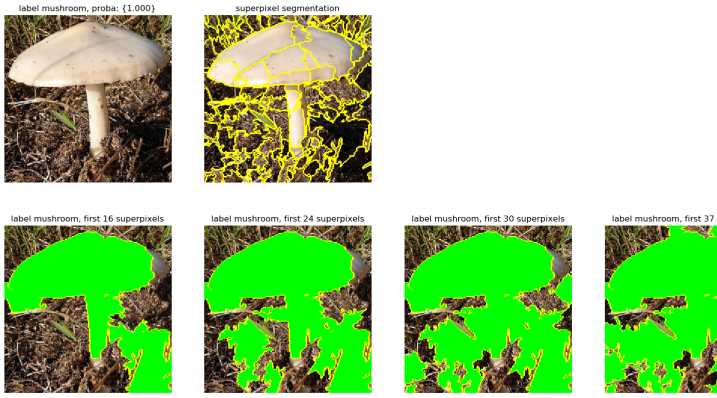
目前深度学习普遍存在一个问题：模型目前还是黑盒，几乎无法感知到它的内部工作状态，预测结果的可信度一直遭到质疑。为此，PadlleX 提供了 2 种对图像分类预测结果进行可解释性研究的算法：LIME 和 NormLIME。

32.1 LIME

LIME 全称 Local interpretable model-agnostic explanations，表示一种与模型无关的局部可解释性。其实现步骤主要如下：

1. 获取图像的超像素。
2. 以输入样本为中心，在其附近的空间中进行随机采样，每个采样即对样本中的超像素进行随机遮掩（每个采样的权重和该采样与原样本的距离成反比）。
3. 每个采样通过预测模型得到新的输出，这样得到一系列的输入 `X` 和对应的输出 `Y`。
4. 将 `X` 转换为超像素特征 `F`，用一个简单的、可解释的模型 `Model`（这里使用岭回归）来拟合 `F` 和 `Y` 的映射关系。
5. `Model` 将得到 `F` 每个输入维度的权重（每个维度代表一个超像素），以此来解释模型。

LIME 的使用方式可参见代码示例和api 介绍。在使用时，参数中的 `num_samples` 设置尤为重要，其表示上述步骤 2 中的随机采样的个数，若设置过小会影响可解释性结果的稳定性，若设置过大则将在上述步骤 3 耗费较长时间；参数 `batch_size` 则表示在计算上述步骤 3 时，若设置过小将在上述步骤 3 耗费较长时间，而上限则根据机器配置决定。



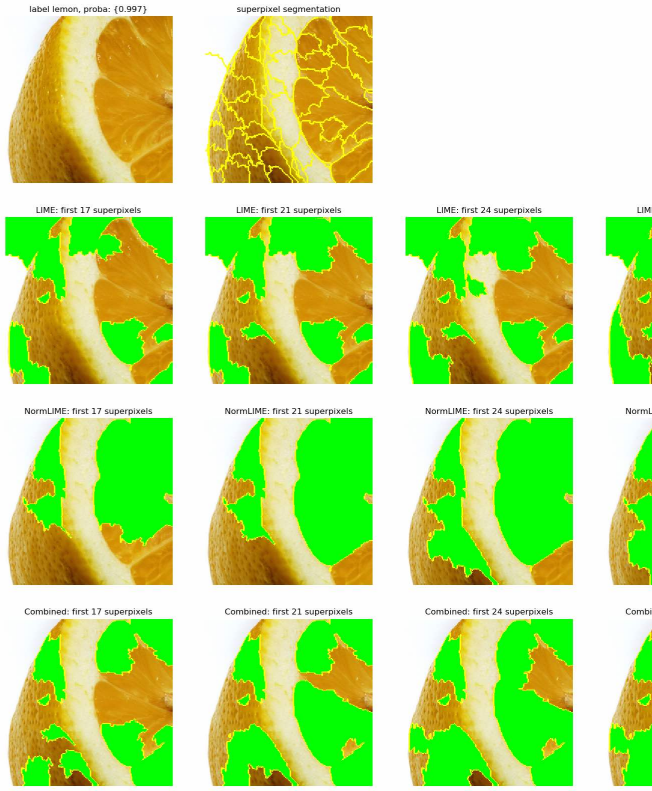
最终 LIME 可解释性算法的可视化结果如下所示：
中绿色区域代表起正向作用的超像素，红色区域代表起反向作用的超像素，”First n superpixels” 代表前 n 个权重比较大的超像素（由上述步骤 5 计算所得结果）。

32.2 NormLIME

NormLIME 是在 LIME 上的改进，LIME 的解释是局部性的，是针对当前样本给的特定解释，而 NormLIME 是利用一定数量的样本对当前样本的一个全局性的解释，有一定的降噪效果。其实现步骤如下所示：

1. 下载 Kmeans 模型参数和 ResNet50_vc 网络前三层参数。（ResNet50_vc 的参数是在 ImageNet 上训练所得网络的参数；使用 ImageNet 图像作为数据集，每张图像从 ResNet50_vc 的第三层输出提取对应超象素位置上的平均特征和质心上的特征，训练将得到此处的 Kmeans 模型）
2. 使用测试集中的数据计算 normlime 的权重信息（如无测试集，可用验证集代替）：对每张图像的处理：(1) 获取图像的超像素。(2) 使用 ResNet50_vc 获取第三层特征，针对每个超像素位置，组合质心特征和均值特征 `F`。(3) 把 `F` 作为 Kmeans 模型的输入，计算每个超像素位置的聚类中心。(4) 使用训练好的分类模型，预测该张图像的 `label`。对所有图像的处理：(1) 以每张图像的聚类中心信息组成的向量（若某聚类中心出现在盖章途中设置为 1，反之为 0）为输入，预测的 `label` 为输出，构建逻辑回归函数 `regression_func`。(2) 由 `regression_func` 可获得每个聚类中心不同类别下的权重，并对权重进行归一化。
3. 使用 Kmeans 模型获取需要可视化图像的每个超像素的聚类中心。
4. 对需要可视化的图像的超像素进行随机遮掩构成新的图像。
5. 对每张构造的图像使用预测模型预测 label。
6. 根据 normlime 的权重信息，每个超像素可获不同的权重，选取最高的权重为最终的权重，以此来解释模型。

NormLIME 的使用方式可参见代码示例和api 介绍。在使用时，参数中的 `num_samples` 设置尤为重要，其表示上述步骤 2 中的随机采样的个数，若设置过小会影响可解释性结果的稳定性，若设置过大则将在上述步骤 3 耗费较长时间；参数 `batch_size` 则表示在计算上述步骤 3 时，预测的 batch size，若设置过小将在上述步骤 3 耗费较长时间，而上限则根据机器配置决定；而 `dataset` 则是由测试集或验证集构造的数据。



最终 NormLIME 可解释性算法的可视化结果如下所示：
中绿色区域代表起正向作用的超像素，红色区域代表起反向作用的超像素，" First n superpixels" 代表前 n 个权重比较大的超像素（由上述步骤 5 计算所得结果）。图中最后一行代表把 LIME 和 NormLIME 对应超像素权重相乘的结果。

CHAPTER 33

无联网模型训练

PaddleX 在模型训练时，存在以下两种情况需要进行联网下载

1. 训练模型时，用户没有配置自定义的预训练模型权重 `pretrain_weights`，此时 PaddleX 会自动联网下载在标准数据集上的预训练模型；2. 模型裁剪训练时，用户没有配置自定义的参数敏感度信息文件 `sensitivities_file`，并将 `sensitivities_file` 配置成了’DEFAULT’字符串，此时 PaddleX 会自动联网下载模型在标准数据集上计算得到的参数敏感度信息文件。

33.1 PaddleX Python API 离线训练

通过如下代码先下载好 PaddleX 的所有预训练模型，下载完共约 7.5G

```
from paddlex.cv.models.utils.pretrain_weights import image_pretrain
from paddlex.cv.models.utils.pretrain_weights import coco_pretrain
from paddlex.cv.models.utils.pretrain_weights import cityscapes_pretrain
import paddlehub as hub

save_dir = '/home/work/paddlex_pretrain'
for name, url in image_pretrain.items():
    hub.download(name, save_dir)
for name, url in coco_pretrain.items():
    hub.download(name, save_dir)
for name, url in cityscapes_pretrain.items():
    hub.download(name, save_dir)
```

用户在可联网的机器上，执行如上代码，所有的预训练模型将会下载至指定的 save_dir（代码示例中为/home/work/paddlex_pretrain），之后在通过 Python 代码使用 PaddleX 训练代码时，只需要在 import paddlex 的同时，配置如下参数，模型在训练时便会优先在此目录下寻找已经下载好的预训练模型。

```
import paddlex as pdx
pdx.pretrain_dir = '/home/work/paddlex_pretrain'
```

33.2 PaddleX GUI 离线训练

PaddleX GUI 在打开后，需要用户设定工作空间，假设当前用户设定的工作空间为 D:\PaddleX_Workspace，为了离线训练，用户需手动下载如下所有文件（下载后无需再做解压操作）至 D:\PaddleX_Workspace\pretrain 目录，之后在训练模型时，便不再需要联网

```
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet18_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet34_pretrained.tar
http://paddle-imagenet-models-name.bj.bcebos.com/ResNet50_pretrained.tar
http://paddle-imagenet-models-name.bj.bcebos.com/ResNet101_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet50_vd_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet101_vd_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet50_vd_ssld_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet101_vd_ssld_pretrained.tar
http://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV1_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV2_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV2_x0_5_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV2_x2_0_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV2_x0_25_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV2_x1_5_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV3_small_x1_0_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV3_large_x1_0_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV3_small_x1_0_ssld_pretrained.
↪tar
https://paddle-imagenet-models-name.bj.bcebos.com/MobileNetV3_large_x1_0_ssld_pretrained.
↪tar
https://paddle-imagenet-models-name.bj.bcebos.com/DarkNet53_ImageNet1k_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/DenseNet121_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/DenseNet161_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/DenseNet201_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/ResNet50_cos_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/Xception41_deeplab_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/Xception65_deeplab_pretrained.tar
```

（下页继续）

(续上页)

```
https://paddle-imagenet-models-name.bj.bcebos.com/ShuffleNetV2_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W18_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W30_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W32_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W40_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W44_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W48_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W60_C_pretrained.tar
https://paddle-imagenet-models-name.bj.bcebos.com/HRNet_W64_C_pretrained.tar
http://paddle-imagenet-models-name.bj.bcebos.com/AlexNet_pretrained.tar
https://paddlemodels.bj.bcebos.com/object_detection/yolov3_darknet.tar
https://paddlemodels.bj.bcebos.com/object_detection/yolov3_mobilenet_v1.tar
https://bj.bcebos.com/paddlex/models/yolov3_mobilenet_v3.tar
https://paddlemodels.bj.bcebos.com/object_detection/yolov3_r34.tar
https://paddlemodels.bj.bcebos.com/object_detection/yolov3_r50vd_dcn.tar
https://bj.bcebos.com/paddlex/pretrained_weights/faster_rcnn_r18_fpn_1x.tar
https://paddlemodels.bj.bcebos.com/object_detection/faster_rcnn_r50_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/faster_rcnn_r50_vd_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/faster_rcnn_r101_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/faster_rcnn_r101_vd_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/faster_rcnn_hrnetv2p_w18_2x.tar
https://bj.bcebos.com/paddlex/pretrained_weights/mask_rcnn_r18_fpn_1x.tar
https://paddlemodels.bj.bcebos.com/object_detection/mask_rcnn_r50_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/mask_rcnn_r50_vd_fpn_2x.tar
https://paddlemodels.bj.bcebos.com/object_detection/mask_rcnn_r101_fpn_1x.tar
https://paddlemodels.bj.bcebos.com/object_detection/mask_rcnn_r101_vd_fpn_1x.tar
https://bj.bcebos.com/paddlex/pretrained_weights/mask_rcnn_hrnetv2p_w18_2x.tar
https://paddleseg.bj.bcebos.com/models/unet_coco_v3.tgz
https://bj.bcebos.com/v1/paddleseg/deeplab_mobilenet_x1_0_coco.tgz
https://paddleseg.bj.bcebos.com/models/xception65_coco.tgz
https://paddlemodels.bj.bcebos.com/object_detection/ppyolo_2x.pdparams
https://paddleseg.bj.bcebos.com/models/deeplabv3p_mobilenetv3_large_cityscapes.tar.gz
https://paddleseg.bj.bcebos.com/models/mobilenet_cityscapes.tgz
https://paddleseg.bj.bcebos.com/models/xception65_bn_cityscapes.tgz
https://paddleseg.bj.bcebos.com/models/hrnet_w18_bn_cityscapes.tgz
https://paddleseg.bj.bcebos.com/models/fast_scnn_cityscape.tar
```

CHAPTER 34

更新日志

v1.3.0 2020.12.20

- 模型更新
  - 图像分类模型 ResNet50_vd 新增 10 万分类预训练模型
  - 目标检测模型 FasterRCNN 新增模型裁剪支持
  - 目标检测模型新增多通道图像训练支持
- 模型部署更新
  - 修复 OpenVINO 部署 C++ 代码中部分 Bug
  - 树莓派部署新增 Arm V8 支持
- 产业案例更新
- 新增工业质检产业案例，提供基于 GPU 和 CPU 两种部署场景下的工业质检方案，及与质检相关的优化策略 详情链接
- **新增 RestFUL API 模块**新增 RestFUL API 模块，开发者可通过此模块快速开发基于 PaddleX 的训练平台
- 增加基于 RestFUL API 的 HTML Demo 详情链接
- 增加基于 RestFUL API 的 Remote 版可视化客户端 详情链接 新增模型通过 OpenVINO 的部署方案详情链接

v1.2.0 2020.09.07

- 模型更新

  - 新增产业最实用目标检测模型 PP-YOLO，深入考虑产业应用对精度速度的双重面诉求，COCO 数据集精度 45.2%，Tesla V100 预测速度 72.9FPS。详情链接
  - FasterRCNN、MaskRCNN、YOLOv3、DeepLabv3p 等模型新增内置 COCO 数据集预训练模型，适用于小数据集的微调训练。
  - 目标检测模型 FasterRCNN 和 MaskRCNN 新增 backbone HRNet_W18，适用于对细节预测要求较高的应用场景。详情链接
  - 语义分割模型 DeepLabv3p 新增 backbone MobileNetV3_large_ssld，模型体积 9.3MB，Cityscapes 数据集精度仍保持有 73.28%。详情链接

- 模型部署更新
  - 新增模型通过 OpenVINO 预测加速的部署方案，CPU 上相比 mkldnn 加速库预测速度提升 1.5 ~ 2 倍左右。详情链接
  - 新增模型在树莓派上的部署方案，进一步丰富边缘侧的部署方案。详情链接
  - 优化 PaddleLite Android 部署的数据预处理和后处理代码性能，预处理速度提升 10 倍左右，后处理速度提升 4 倍左右。
  - 优化 Paddle 服务端 C++ 代码部署代码，增加 use_mkl 等参数，CPU 上相比未开启 mkldnn 预测速度提升 10 ~ 50 倍左右。

- 产业案例更新
  - 新增大尺寸 RGB 图像遥感分割案例，提供滑动窗口预测接口，不仅能避免显存不足的发生，而且能通过配置重叠程度消除最终预测结果中各窗口拼接处的裂痕感。详情链接
  - 新增多通道遥感影像分割案例，打通语义分割任务对任意通道数量的数据分析、模型训练、模型部署全流程。详情链接

- 其它
  - 新增数据集切分功能，支持通过命令行一键切分 ImageNet、PascalVOC、MSCOCO 和语义分割数据集详情链接

v1.1.0 2020.07.12

- 模型更新
- 新增语义分割模型 HRNet、FastSCNN
- 目标检测 FasterRCNN、实例分割 MaskRCNN 新增 backbone HRNet
- 目标检测/实例分割模型新增 COCO 数据集预训练模型
- 集成 X2Paddle，PaddleX 所有分类模型和语义分割模型支持导出为 ONNX 协议
- 模型部署更新
- 模型加密增加支持 Windows 平台
- 新增 Jetson、Paddle Lite 模型部署预测方案

- C++ 部署代码新增 batch 批预测，并采用 OpenMP 对预处理进行并行加速
- 新增 2 个 PaddleX 产业案例
- 人像分割案例
- 工业表计读数案例
- 新增数据格式转换功能，LabelMe、精灵标注助手和 EasyData 平台标注的数据转为 PaddleX 支持加载的数据格式
- PaddleX 文档更新，优化文档结构

v1.0.0 2020.05.20

- 增加模型 C++ 部署和 Python 部署代码
- 增加模型加密部署方案
- 增加分类模型的 OpenVINO 部署方案
- 增加模型可解释性的接口

v0.1.8 2020.05.17

- 修复部分代码 Bug
- 新增 EasyData 平台数据标注格式支持
- 支持 imgaug 数据增强库的 pixel-level 算子