setup.py实现C++扩展和python库编译

很多大的程序使用前都会需要调用setup.py进行build编译并安装自己的库，debug发现程序有时候走到了编译出的lib里面去了，在源程序改动没用，有的运算放在cuda上也是编译完成接口调用的，有必要了解一下这个setup.py是怎么工作的。

目录

目的

编写setup.py是为了实现python的C/C++扩展。比如自定义层实现自己的全新网络算法，理论上继承nn.Module编写forward函数即可自动实现反向传播，但是pytorch的函数针对特定的操作进行了优化，组合起来效率可能很低，无法充分利用GPU通道或者超负载，而且python解释器也无法优化。所以一般是用C++编写相关算法（如RoIAlign，NMS）的程序，充分利用GPU资源，然后作为扩展程序在pytorch进行导入即可，这部分就是setyp.py完成的。

cuda程序

其中主要是GPU的扩展，cuda的程序后缀.cu，头文件也是.h，是基于C++的改进，支持大多C++语法并加入一些特别的语法。

这个链接实现了一个简单的cuda的C++扩展python程序：https://oldpan.me/archives/pytorch-cuda-c-plus-plus

下面是Mask R-CNN的setup.py，以看懂这个为中心展开说明和学习：

```
# Copyright (c) Facebook, Inc. and its affiliates. All Rights Reserved.
#!/usr/bin/env python

import glob
import os

import torch
from setuptools import find_packages
from setuptools import setup
from torch.utils.cpp_extension import CUDA_HOME
from torch.utils.cpp_extension import CppExtension
from torch.utils.cpp_extension import CUDAExtension

requirements = ["torch", "torchvision"]

def get_extensions():
    this_dir = os.path.dirname(os.path.abspath(__file__))
    extensions_dir = os.path.join(this_dir, "maskrcnn_benchmark", "csrc")

    main_file = glob.glob(os.path.join(extensions_dir, "*.cpp"))
    source_cpu = glob.glob(os.path.join(extensions_dir, "cpu", "*.cpp"))
    source_cuda = glob.glob(os.path.join(extensions_dir, "cuda", "*.cu"))

    sources = main_file + source_cpu
    extension = CppExtension

    extra_compile_args = {"cxx": []}
    define_macros = []

    if torch.cuda.is_available() and CUDA_HOME is not None:
        extension = CUDAExtension
        sources += source_cuda
        define_macros += [("WITH_CUDA", None)]
        extra_compile_args["nvcc"] = [
            "-DCUDA_HAS_FP16=1",
            "-D__CUDA_NO_HALF_OPERATORS__",
            "-D__CUDA_NO_HALF_CONVERSIONS__",
            "-D__CUDA_NO_HALF2_OPERATORS__",
        ]

    sources = [os.path.join(extensions_dir, s) for s in sources]

    include_dirs = [extensions_dir]

    ext_modules = [
        extension(
            "maskrcnn_benchmark._C",
            sources,
            include_dirs=include_dirs,
            define_macros=define_macros,
            extra_compile_args=extra_compile_args,
```

```
        )
    ]

    return ext_modules

setup(
    name="maskrcnn_benchmark",
    version="0.1",
    author="fmassa",
    url="https://github.com/facebookresearch/maskrcnn-benchmark",
    description="object detection in pytorch",
    packages=find_packages(exclude=("configs", "tests",)),
    # install_requires=requirements,
    ext_modules=get_extensions(),
    cmdclass={"build_ext": torch.utils.cpp_extension.BuildExtension},
)
```

安装方式

执行的命令是python setup.py build develop，涉及包的安装的主要方式如下：

- **开发方式安装**

```
python setup.py develop
```

如果应用在开发过程中会频繁变更，每次安装还需要先将原来的版本卸掉，很麻烦。使用”develop”开发方式安装的话，应用代码不会真的被拷贝到本地Python环境的”site-packages”目录下，而是在”site-packages”目录里创建一个指向当前应用位置的链接。**这样如果当前位置的源码被改动，就会马上反映到”site-packages”里。**（mmdetection就是直接创建在site-packages的）

- **安装应用**

```
python setup.py install
```

很多方式都是这种，会将当前的Python应用安装到当前Python环境的”site-packages”目录下，这样其他程序就可以像导入标准库一样导入该应用的代码了。

setuptools

这个是发布库的主要工具，非标准库需要自行pip安装，据说是高手都用这个（网上很多说的distutils是标准库，功能不够多）。从中import的setup函数是setup.py的主要部分。

官方文档：https://setuptools.readthedocs.io/en/latest/setuptools.html#metadata

setup函数

传入的参数类型为：

name	包名称
version	包版本
author	程序的作者
author_email	程序的作者的邮箱地址
url	程序的官网地址
description	程序的简单描述
classifiers	程序的所属分类列表
packages	需要处理的包目录(通常为包含 `__init__.py` 的文件夹)
cmdclass	添加自定义命令
exclude_package_data	当 include_package_data 为 True 时该选项用于排除部分文件
ext_modules	指定扩展模块
zip_safe	不压缩包，而是以目录的形式安装

更多参数见文档：https://setuptools.readthedocs.io/en/latest/setuptools.html#metadata

以及介绍：http://blog.konghy.cn/2018/04/29/setup-dot-py/ http://www.bjhee.com/setuptools.html

针对mask rcnn的几个参数不难看出一些简单的规则，不影响包的发布：

```
setup(
    name="maskrcnn_benchmark",     包名称
    version="0.1",                 版本号
```

```
    author="fmassa",                发布者（facebook的一个工程师github的id）
    url="https://github.com/facebookresearch/maskrcnn-benchmark",    repo的url
    description="object detection in pytorch",                       描述
    packages=find_packages(exclude=("configs", "tests",)),
    # install_requires=requirements,            安装依赖项->列表的torch两项
    ext_modules=get_extensions(),
    cmdclass={"build_ext": torch.utils.cpp_extension.BuildExtension},
)
```

其中比较麻烦的是下面几个：

- packages： python的package是包含__init__.py的文件夹；find_packages(exclude=("configs", "tests",))是递归地包含当前目录下除了configs和tests外所有文件夹的包（主要都在maskrcnn_benchmark下）。断点查看包的目录：

  ```
  ['maskrcnn_benchmark', 'maskrcnn_benchmark.structures', 'maskrcnn_benchmark.modeling', 'maskrcnn_benchmark.layers', 'maskrcnn_benchmark.(
  ```

  很长....很多看不懂的工作放在这里了

- ext_modules： 该参数用于构建 C / C++ 扩展扩展包。用于描述扩展模块的列表（列表每个元素对应一个模块），扩展模块可以设置扩展包名，头文件、源文件、链接库及其路径、宏定义和编辑参数等。

  可以看看这个函数的内部发生了什么：

```
def get_extensions():
    this_dir = os.path.dirname(os.path.abspath(__file__))
    extensions_dir = os.path.join(this_dir, "maskrcnn_benchmark", "csrc")
'''
this_dir: /py/MaskRcnn/maskrcnn-benchmark
extensions_dir: /py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc
'''
    main_file = glob.glob(os.path.join(extensions_dir, "*.cpp"))
    source_cpu = glob.glob(os.path.join(extensions_dir, "cpu", "*.cpp"))
    source_cuda = glob.glob(os.path.join(extensions_dir, "cuda", "*.cu"))
'''
可以看出source_cpu和source_cuda分别是一些写在cpu和gpu的实现，如nms，roialign等
main_file: ['/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/vision.cpp']
source_cpu: ['/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cpu/nms_cpu.cpp', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchma
source_cuda: ['/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cuda/ROIAlign_cuda.cu', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_
'''
    sources = main_file + source_cpu
    extension = CppExtension

    extra_compile_args = {"cxx": []}
    define_macros = []
    if torch.cuda.is_available() and CUDA_HOME is not None:
        extension = CUDAExtension
        sources += source_cuda
        define_macros += [("WITH_CUDA", None)]
        extra_compile_args["nvcc"] = [
            "-DCUDA_HAS_FP16=1",
            "-D__CUDA_NO_HALF_OPERATORS__",
            "-D__CUDA_NO_HALF_CONVERSIONS__",
            "-D__CUDA_NO_HALF2_OPERATORS__",
        ]
    sources = [os.path.join(extensions_dir, s) for s in sources]
'''这里的source把mainfile和cpu,cuda的源码都放进去了'''
    include_dirs = [extensions_dir]

    ext_modules = [
        extension(
            "maskrcnn_benchmark._C",
            sources,
            include_dirs=include_dirs,
            define_macros=define_macros,
            extra_compile_args=extra_compile_args,
        )
    ]
    return ext_modules
```

该对象最后是：

```
(Pdb) ext_modules
[<setuptools.extension.Extension('maskrcnn_benchmark._C') at 0x7f86a01ce128>]
(Pdb) "maskrcnn_benchmark._C"
'maskrcnn_benchmark._C'
(Pdb) sources
['/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/vision.cpp', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cpu/nms_cpu.cpp', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cpu/ROIAlign_cpu.cpp', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cuda/ROIAlign_cuda.cu', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cuda/ROIPool_cuda.cu', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cuda/nms.cu', '/py/MaskRcnn/maskrcnn-benchmark/maskrcnn_benchmark/csrc/cuda/SigmoidFocalLoss_cuda.cu']
(Pdb) define_macros
[('WITH_CUDA', None)]
(Pdb) extra_compile_args
{'cxx': [], 'nvcc': ['-DCUDA_HAS_FP16=1', '-D__CUDA_NO_HALF_OPERATORS__', '-D__CUDA_NO_HALF_CONVERSIONS__', '-D__CUDA_NO_HALF2_OPERATORS__']}
(Pdb)
```


设计cuda编程和调用的方法，暂时略过。

- cmdclass：自定义的命令，字典形式创建，此处没用到。（如果项自己重写run方法可以继承相关基类进行自定义）

作用

感觉build之后的好处和必要性是：可以全局直接调用相关的库函数，不用受目录结构的限制；链接cuda的cu文件，将放到cuda的操作通过接口链接；develope模式安装时不用像mmdetection一样去site-packages里面找lib的文件进行改动，可以源码直接操作。