mmdetection源码剖析(1)--NMS

ManWingloeng 2020-07-04 原文

mmdetection源码剖析(1)--NMS

熟悉目标检测的应该都清楚**NMS**是什么算法，但是如果我们要与C++和cuda结合直接写成Pytorch的操作你们清楚怎么写吗？最近在看**mmdetection**的源码，发现其实原来写C++和cuda的扩展也不难，下面给大家讲一下。

C ++的扩展是允许用户来创建自定义PyTorch框架外的操作（operators ）的，即从PyTorch后端分离。此方法与实现本地PyTorch操作的方式*不同*。C ++扩展旨在为您节省大量与将操作与PyTorch后端集成在一起相关的样板，同时为基于PyTorch的项目提供高度的灵活性。

官方给出了一个LLTM的例子，大家也可以看一下。

NMS算法

先复习一下NMS的算法：

（1）将所有框的得分排序，选中最高分及其对应的框

（2）遍历其余的框，如果和当前最高分框的重叠面积(IOU)大于一定阈值，我们就将框删除。

（3）从未处理的框中继续选一个得分最高的，重复上述过程。

这里我给出一份纯numpy的实现：

```
def nms(bounding_boxes, Nt):
    if len(bounding_boxes) == 0:
        return [], []
    bboxes = np.array(bounding_boxes)
    x1 = bboxes[:, 0]
    y1 = bboxes[:, 1]
    x2 = bboxes[:, 2]
    y2 = bboxes[:, 3]
    scores = bboxes[:, 4]
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    order = np.argsort(scores)
    picked_boxes = []
    while order.size > 0:
        index = order[-1]
        picked_boxes.append(bounding_boxes[index])
        x11 = np.maximum(x1[index], x1[order[:-1]])
        y11 = np.maximum(y1[index], y1[order[:-1]])
        x22 = np.minimum(x2[index], x2[order[:-1]])
        y22 = np.minimum(y2[index], y2[order[:-1]])
        w = np.maximum(0.0, x22 - x11 + 1)
        h = np.maximum(0.0, y22 - y11 + 1)
        intersection = w * h
```

编写Pytorch C++扩展的步骤

需要编写下面5个文件：

nms_kernel.cu

nms_cuda.cpp

nms_ext.cpp

setup.py

nms_wrapper.py

(1) nms_kernel.cu 主要使用ATen和THC库编写nms_cuda_forward的函数，使用C++编写，涉及一些lazyInitCUDA，THCudaFree，THCCeilDiv 的操作，算法跟我们前面写的numpy差不太多。

(2) nms_cuda.cpp 是调用了nms_kernel.cu文件的nms_cuda_forward封装了一下变成nms_cuda函数。

(3) nms_ext.cpp 进一步封装nms_cuda函数为nms，并且通过PYBIND11_MODULE绑定成python可调用的函数。

```
PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
  m.def("nms", &nms, "non-maximum suppression");
}
```

通过上面那样就相当于告诉python函数名定义为nms了。

(4) setup.py 就是编译一遍nms_ext，至此你就可以通过nms_ext.nms调用cpp extension作为pytorch的操作了

```
make_cuda_ext(
    name='nms_ext',
    module='mmdet.ops.nms',
    sources=['src/nms_ext.cpp', 'src/cpu/nms_cpu.cpp'],
    sources_cuda=[
    'src/cuda/nms_cuda.cpp', 'src/cuda/nms_kernel.cu'
    ]),
```

(5) nms_wrapper.py 再次封装 nms_ext.nms，方便使用，使用实例：

```
from . import nms_ext
inds = nms_ext.nms(dets_th, iou_thr)
```

稍微完整的代码如下，但是我也删减了一些，只剩下nms相关的代码，想要看完整代码可以点击下面的文件名。

nms_kernel.cu (这个估计有部分是Facebook写的)

```
// Copyright (c) Facebook, Inc. and its affiliates. All Rights Reserved.
#include <ATen/ATen.h>
#include <ATen/cuda/CUDAContext.h>
#include <ATen/DeviceGuard.h>
#include <THC/THC.h>
#include <THC/THCDeviceUtils.cuh>
#include <vector>
#include <iostream>

int const threadsPerBlock = sizeof(unsigned long long) * 8;

__device__ inline float devIoU(float const * const a, float const * const b) {
  float left = max(a[0], b[0]), right = min(a[2], b[2]);
  float top = max(a[1], b[1]), bottom = min(a[3], b[3]);
  float width = max(right - left, 0.f), height = max(bottom - top, 0.f);
  float interS = width * height;
```

```
                           const float *dev_boxes, unsigned long long *dev_mask) {
  const int row_start = blockIdx.y;
  const int col_start = blockIdx.x;
  // if (row_start > col_start) return;
  const int row_size =
        min(n_boxes - row_start * threadsPerBlock, threadsPerBlock);
  const int col_size =
        min(n_boxes - col_start * threadsPerBlock, threadsPerBlock);

  __shared__ float block_boxes[threadsPerBlock * 5];
  if (threadIdx.x < col_size) {
    block_boxes[threadIdx.x * 5 + 0] =
        dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 0];
    block_boxes[threadIdx.x * 5 + 1] =
        dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 1];
    block_boxes[threadIdx.x * 5 + 2] =
        dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 2];
    block_boxes[threadIdx.x * 5 + 3] =
        dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 3];
    block_boxes[threadIdx.x * 5 + 4] =
        dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 4];
  }
  __syncthreads();

  if (threadIdx.x < row_size) {
    const int cur_box_idx = threadsPerBlock * row_start + threadIdx.x;
    const float *cur_box = dev_boxes + cur_box_idx * 5;
    int i = 0;
    unsigned long long t = 0;
    int start = 0;
    if (row_start == col_start) {
      start = threadIdx.x + 1;
    }
    for (i = start; i < col_size; i++) {
      if (devIoU(cur_box, block_boxes + i * 5) > nms_overlap_thresh) {
        t |= 1ULL << i;
      }
    }
    const int col_blocks = THCCeilDiv(n_boxes, threadsPerBlock);
    dev_mask[cur_box_idx * col_blocks + col_start] = t;
  }
}
// boxes is a N x 5 tensor
at::Tensor nms_cuda_forward(const at::Tensor boxes, float nms_overlap_thresh) {
  // Ensure CUDA uses the input tensor device.
  at::DeviceGuard guard(boxes.device());
  using scalar_t = float;
  AT_ASSERTM(boxes.device().is_cuda(), "boxes must be a CUDA tensor");
  auto scores = boxes.select(1, 4);
  auto order_t = std::get<1>(scores.sort(0, /* descending=*/true));
  auto boxes_sorted = boxes.index_select(0, order_t);

  int boxes_num = boxes.size(0);

  const int col_blocks = THCCeilDiv(boxes_num, threadsPerBlock);

  scalar_t* boxes_dev = boxes_sorted.data_ptr<scalar_t>();

  THCState *state = at::globalContext().lazyInitCUDA(); // TODO replace with getTHCState

  unsigned long long* mask_dev = NULL;
  //THCudaCheck(THCudaMalloc(state, (void**) &mask_dev,
  //                      boxes_num * col_blocks * sizeof(unsigned long long)));

  mask_dev = (unsigned long long*) THCudaMalloc(state, boxes_num * col_blocks * sizeof(unsigned long long));

  dim3 blocks(THCCeilDiv(boxes_num, threadsPerBlock),
              THCCeilDiv(boxes_num, threadsPerBlock));
  dim3 threads(threadsPerBlock);
  nms_kernel<<<blocks, threads, 0, at::cuda::getCurrentCUDAStream()>>>(boxes_num,
                                  nms_overlap_thresh,
                          mask_dev,
                          sizeof(unsigned long long) * boxes_num * col_blocks,
                          cudaMemcpyDeviceToHost,
                          at::cuda::getCurrentCUDAStream()
                          ));

  std::vector<unsigned long long> remv(col_blocks);
  memset(&remv[0], 0, sizeof(unsigned long long) * col_blocks);

  at::Tensor keep = at::empty({boxes_num}, boxes.options().dtype(at::kLong).device(at::kCPU));
  int64_t* keep_out = keep.data_ptr<int64_t>();

  int num_to_keep = 0;
  for (int i = 0; i < boxes_num; i++) {
    int nblock = i / threadsPerBlock;
    int inblock = i % threadsPerBlock;

    if (!(remv[nblock] & (1ULL << inblock))) {
      keep_out[num_to_keep++] = i;
      unsigned long long *p = &mask_host[0] + i * col_blocks;
      for (int j = nblock; j < col_blocks; j++) {
        remv[j] |= p[j];
      }
    }
  }
  THCudaFree(state, mask_dev);
  // TODO improve this part
  return order_t.index({
      keep.narrow(/*dim=*/0, /*start=*/0, /*length=*/num_to_keep).to(
          order_t.device(), keep.scalar_type())});
}
```

nms_cuda.cpp

```
#include <torch/extension.h>
#define CHECK_CUDA(x) TORCH_CHECK(x.device().is_cuda(), #x, " must be a CUDAtensor ")
at::Tensor nms_cuda_forward(const at::Tensor boxes, float nms_overlap_thresh);
at::Tensor nms_cuda(const at::Tensor& dets, const float threshold) {
  CHECK_CUDA(dets);
  if (dets.numel() == 0)
    return at::empty({0}, dets.options().dtype(at::kLong).device(at::kCPU));
  return nms_cuda_forward(dets, threshold);
}
```

nms_ext.cpp

```
#include <torch/extension.h>
#ifdef WITH_CUDA
at::Tensor nms_cuda(const at::Tensor& dets, const float threshold);
#endif
at::Tensor nms(const at::Tensor& dets, const float threshold){
  if (dets.device().is_cuda()) {
#ifdef WITH_CUDA
    return nms_cuda(dets, threshold);
#else
    AT_ERROR("nms is not compiled with GPU support");
#endif
  }
#  return nms_cpu(dets, threshold);
}
PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
  m.def("nms", &nms, "non-maximum suppression");
}
```

setup.py

```
def make_cuda_ext(name, module, sources, sources_cuda=[]):
        extension = CUDAExtension
        extra_compile_args['nvcc'] = [
            '-D__CUDA_NO_HALF_OPERATORS__',
            '-D__CUDA_NO_HALF_CONVERSIONS__',
            '-D__CUDA_NO_HALF2_OPERATORS__',
        ]
        sources += sources_cuda
    else:
        print(f'Compiling {name} without CUDA')
        extension = CppExtension
        # raise EnvironmentError('CUDA is required to compile MMDetection!')

    return extension(
        name=f'{module}.{name}',
        sources=[os.path.join(*module.split('.'), p) for p in sources],
        define_macros=define_macros,
        extra_compile_args=extra_compile_args)

if __name__ == '__main__':
    write_version_py()
    setup(
        name='mmdet',
        version=get_version(),
        description='Open MMLab Detection Toolbox and Benchmark',
        long_description=readme(),
        author='OpenMMLab',
        author_email='chenkaidev@gmail.com',
        keywords='computer vision, object detection',
        url='https://github.com/open-mmlab/mmdetection',
        packages=find_packages(exclude=('configs', 'tools', 'demo')),
        package_data={'mmdet.ops': ['*/*.so']},
        classifiers=[
            'Development Status :: 4 - Beta',
            'License :: OSI Approved :: Apache Software License',
            'Operating System :: OS Independent',
            'Programming Language :: Python :: 3',
            'Programming Language :: Python :: 3.5',
            'Programming Language :: Python :: 3.6',
            'Programming Language :: Python :: 3.7',
        ],
        license='Apache License 2.0',
        setup_requires=parse_requirements('requirements/build.txt'),
        tests_require=parse_requirements('requirements/tests.txt'),
        install_requires=parse_requirements('requirements/runtime.txt'),
        extras_require={
            'all': parse_requirements('requirements.txt'),
            'tests': parse_requirements('requirements/tests.txt'),
            'build': parse_requirements('requirements/build.txt'),
            'optional': parse_requirements('requirements/optional.txt'),
        },
        ext_modules=[
            make_cuda_ext(
                name='compiling_info',
                module='mmdet.ops.utils',
                sources=['src/compiling_info.cpp']),
            make_cuda_ext(
                name='nms_ext',
                module='mmdet.ops.nms',
                sources=['src/nms_ext.cpp', 'src/cpu/nms_cpu.cpp'],
                sources_cuda=[
                    'src/cuda/nms_cuda.cpp', 'src/cuda/nms_kernel.cu'
                ]),
        ],
        cmdclass={'build_ext': BuildExtension},
        zip_safe=False)
```

nms_wrapper.py

```
from . import nms_ext
def nms(dets, iou_thr, device_id=None):
        device = 'cpu' if device_id is None else f'cuda:{device_id}'
        dets_th = torch.from_numpy(dets).to(device)
    else:
        raise TypeError('dets must be either a Tensor or numpy array, '
                        f'but got {type(dets)}')
    # execute cpu or cuda nms
    if dets_th.shape[0] == 0:
        inds = dets_th.new_zeros(0, dtype=torch.long)
    else:
        if dets_th.is_cuda:
            inds = nms_ext.nms(dets_th, iou_thr)
        else:
            inds = nms_ext.nms(dets_th, iou_thr)

    if is_numpy:
        inds = inds.cpu().numpy()
    return dets[inds, :], inds
```

总结

想要编写一个c++和cuda的扩展给Pytorch使用，其实主要就4步：

使用ATEN和THC编写前向代码cu文件A

封装成一个cpp文件B

再把B封装一遍并且使用PYBIND11_MODULE绑定函数名function

通过make_cuda_ext（这个是mmdetection自定义的函数）把组件setup一遍

通过绑定的函数名function就可以在Pytorch中调用了。

mmdetection源码剖析(1)--NMS的更多相关文章

1. jQuery之Deferred源码剖析
   一.前言 大约在夏季,我们谈过ES6的Promise(详见here),其实在ES6前jQuery早就有了Promise,也就是我们所知道的Deferred对象,宗旨当然也和ES6的Promise一样, ...
2. Nodejs事件引擎libuv源码剖析之：高效线程池(threadpool)的实现
   声明:本文为原创博文,转载请注明出处. Nodejs编程是全异步的,这就意味着我们不必每次都阻塞等待该次操作的结果,而事件完成(就绪)时会主动回调通知我们.在网络编程中,一般都是基于Reactor线程 ...
3. Apache Spark源码剖析
   Apache Spark源码剖析(全面系统介绍Spark源码,提供分析源码的实用技巧和合理的阅读顺序,充分了解Spark的设计思想和运行机理) 许鹏 著   ISBN 978-7-121-25420- ...
4. 基于mybatis-generator-core 1.3.5项目的修订版以及源码剖析
   项目简单说明 mybatis-generator,是根据数据库表.字段反向生成实体类等代码文件.我在国庆时候,没事剖析了mybatis-generator-core源码,写了相当详细的中文注释,可以去 ...
5. STL"源码"剖析-重点知识总结
   STL是C++重要的组件之一,大学时看过<STL源码剖析>这本书,这几天复习了一下,总结出以下LZ认为比较重要的知识点,内容有点略多 :) 1.STL概述 STL提供六大组件,彼此可以组合 ...
6. SpringMVC源码剖析（四）- DispatcherServlet请求转发的实现
   SpringMVC完成初始化流程之后,就进入Servlet标准生命周期的第二个阶段,即“service”阶段.在“service”阶段中,每一次Http请求到来,容器都会启动一个请求线程,通过serv ...
7. 自己实现多线程的socket，socketserver源码剖析
   1,IO多路复用 三种多路复用的机制:select.poll.epoll 用的多的两个:select和epoll 简单的说就是:1,select和poll所有平台都支持,epoll只有linux支持2 ...