

件)

原创

走的那么干脆

于 2019-01-04 21:09:02 发布

9842

收藏 32

版权

分类专栏:

Faster R-CNN

文章标签:

Deep learning

 Faster R-CNN 专栏收录该内容















0 订阅 2 篇文章 订阅专栏

最近要修改Faster R- CNN中实现的GPU版的NMS代码，于是小白的我就看起了CUDA编程，当然也只是浅显地阅读一些教程，快速入门而已，所以具体需要注意的以及一些思想，大家移步此博主的系列教程：

在了解了CUDA编程的核心思想后，我们便可以开始阅读nms_kernel.cu文件了，先直接上 源码（部分简单的已经注释），如下：

```
1 // -----
2 // Faster R-CNN
3 // Copyright (c) 2015 Microsoft
4 // Licensed under The MIT License [see fast-rcnn/LICENSE for details]
5 // Written by Shaoqing Ren
6 // -----
7
8 #include "gpu_nms.hpp"
9 #include <vector>
10 #include <iostream>
11
12 //cudaError_t是cuda中的一个类, 用于记录cuda错误 (所有的cuda函数, 几乎都会返回一个cudaError_t)
13 #define CUDA_CHECK(condition) \
14     /* Code block avoids redefinition of cudaError_t error */ \
15     do { \
16         cudaError_t error = condition; \
17         if (error != cudaSuccess) { \
18             std::cout << cudaGetErrorString(error) << std::endl; \
19         } \
20     } while (0)
21
22 //DIVUP即实现除法的向上取整
23 #define DIVUP(m,n) ((m) / (n) + ((m) % (n) > 0))
24
25 //unsigned Long Long类型是目前C语言中精度最高的数据类型, 为64位精度
26 //threadsPerBlock即自定义的每个Block所含有的线程数目 (每个Block的线程数不宜太多, 也不宜太少)
27 int const threadsPerBlock = sizeof(unsigned long long) * 8; //其实threadsPerBlock = 64
28
29
30 __device__ inline float devIoU(float const * const a, float const * const b) {
31     float left = max(a[0], b[0]), right = min(a[2], b[2]);
32     float top = max(a[1], b[1]), bottom = min(a[3], b[3]);
33     float width = max(right - left + 1, 0.f), height = max(bottom - top + 1, 0.f);
34     float interS = width * height;
35     float Sa = (a[2] - a[0] + 1) * (a[3] - a[1] + 1);
36     float Sb = (b[2] - b[0] + 1) * (b[3] - b[1] + 1);
37     return interS / (Sa + Sb - interS);
38 }
39
40 //nms kernel
41 /*
42 参数n_boxes: 边界框数目
43 参数nms_overlap_thresh: 交并比阈值
```

分类专栏

	数据结构与算法	1篇
	opencv教程	1篇
	FCN	7篇
	Anaconda	1篇
	python	1篇
	PSPNet	2篇
	caffe	3篇
	SSD	6篇
	TensorFlow	2篇
	C++	8篇
	Faster R-CNN	2篇
	PyTorch	2篇
	gcc/g++	1篇
	Cython	1篇

```

44 */
45 __global__ void nms_kernel(const int n_boxes, const float nms_overlap_thresh,
46                          const float *dev_boxes, unsigned long long *dev_mask) {
47     const int row_start = blockIdx.y;
48     const int col_start = blockIdx.x;
49
50     // if (row_start > col_start) return;
51
52     const int row_size =
53         min(n_boxes - row_start * threadsPerBlock, threadsPerBlock);
54     const int col_size =
55         min(n_boxes - col_start * threadsPerBlock, threadsPerBlock);
56
57     __shared__ float block_boxes[threadsPerBlock * 5]; //共享内存
58     if (threadIdx.x < col_size) {
59         block_boxes[threadIdx.x * 5 + 0] =
60             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 0];
61         block_boxes[threadIdx.x * 5 + 1] =
62             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 1];
63         block_boxes[threadIdx.x * 5 + 2] =
64             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 2];
65         block_boxes[threadIdx.x * 5 + 3] =
66             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 3];
67         block_boxes[threadIdx.x * 5 + 4] =
68             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 4];
69     }
70     __syncthreads(); //同步线程
71
72     if (threadIdx.x < row_size) {
73         const int cur_box_idx = threadsPerBlock * row_start + threadIdx.x;
74         const float *cur_box = dev_boxes + cur_box_idx * 5;
75         int i = 0;
76         unsigned long long t = 0;
77         int start = 0;
78         if (row_start == col_start) {
79             start = threadIdx.x + 1;
80         }
81         for (i = start; i < col_size; i++) {
82             if (devIoU(cur_box, block_boxes + i * 5) > nms_overlap_thresh) {
83                 t |= 1ULL << i; //1ULL = unsigned long long型的数字1 (最高位为第64位)
84             }
85         }
86         const int col_blocks = DIVUP(n_boxes, threadsPerBlock);
87         dev_mask[cur_box_idx * col_blocks + col_start] = t;
88     }
89 }
90
91 //设置哪个GPU用于nms
92 void _set_device(int device_id) {
93     int current_device;
94     CUDA_CHECK(cudaGetDevice(&current_device)); //获取当前GPU序号
95     if (current_device == device_id) {
96         return;
97     }
98     // The call to cudaSetDevice must come before any calls to Get, which
99     // may perform initialization using the GPU.
100     CUDA_CHECK(cudaSetDevice(device_id)); //设置device_id号GPU生效
101 }
102
103 void _nms(int* keep_out, int* num_out, const float* boxes_host, int boxes_num,
104          int boxes_dim, float nms_overlap_thresh, int device_id) {

```

```

105     _set_device(device_id);
106
107     float* boxes_dev = NULL;
108     unsigned long long* mask_dev = NULL;
109
110     const int col_blocks = DIVUP(boxes_num, threadsPerBlock);
111
112     CUDA_CHECK(cudaMalloc(&boxes_dev,
113         boxes_num * boxes_dim * sizeof(float)));
114     CUDA_CHECK(cudaMemcpy(boxes_dev,
115         boxes_host,
116         boxes_num * boxes_dim * sizeof(float),
117         cudaMemcpyHostToDevice));
118
119     CUDA_CHECK(cudaMalloc(&mask_dev,
120         boxes_num * col_blocks * sizeof(unsigned long long)));
121
122     dim3 blocks(DIVUP(boxes_num, threadsPerBlock),
123         DIVUP(boxes_num, threadsPerBlock));
124     dim3 threads(threadsPerBlock);
125     nms_kernel<<<blocks, threads>>>(boxes_num,
126         nms_overlap_thresh,
127         boxes_dev,
128         mask_dev);
129
130     std::vector<unsigned long long> mask_host(boxes_num * col_blocks);
131     CUDA_CHECK(cudaMemcpy(&mask_host[0],
132         mask_dev,
133         sizeof(unsigned long long) * boxes_num * col_blocks,
134         cudaMemcpyDeviceToHost));
135
136     std::vector<unsigned long long> remv(col_blocks);
137     memset(&remv[0], 0, sizeof(unsigned long long) * col_blocks);
138
139     int num_to_keep = 0;
140     for (int i = 0; i < boxes_num; i++) {
141         int nblock = i / threadsPerBlock;
142         int inblock = i % threadsPerBlock;
143
144         if (!(remv[nblock] & (1ULL << inblock))) {
145             keep_out[num_to_keep++] = i;
146             unsigned long long *p = &mask_host[0] + i * col_blocks;
147             for (int j = nblock; j < col_blocks; j++) {
148                 remv[j] |= p[j];
149             }
150         }
151     }
152     *num_out = num_to_keep;
153
154     CUDA_CHECK(cudaFree(boxes_dev));
155     CUDA_CHECK(cudaFree(mask_dev));
156 }

```

1.devIoU()函数

```

1 //devIoU计算两个边界框之间的交并比
2 //__device__是CUDA中的限定词，具体含义如下图
3 //float const * const a表示a是常量指针常量，即a是一个指针常量（不可修改的指针），指向一个常量
4 __device__ inline float devIoU(float const * const a, float const * const b) {
5     float left = max(a[0], b[0]), right = min(a[2], b[2]);

```

```
6 float top = max(a[1], b[1]), bottom = min(a[3], b[3]);
7 float width = max(right - left + 1, 0.f), height = max(bottom - top + 1, 0.f);
8 float interS = width * height; //交集
9 float Sa = (a[2] - a[0] + 1) * (a[3] - a[1] + 1); //边界框a的面积
10 float Sb = (b[2] - b[0] + 1) * (b[3] - b[1] + 1); //边界框b的面积
11 return interS / (Sa + Sb - interS);
12 }
```

限定词	执行 (excution)	可调用 (callable)	注意事项 (notes)
__global__	在设备上执行 (GPU)	可由主机 (host) 调用; 可由计算能力为3的设备调用 (callable from the device for devices of compute capability 3)	返回必须为void型 (即不能范围任何其他类型)
__device__	在设备上执行 (GPU)	只能被设备 (device) 调用	
__host__	在主机上执行 (CPU)	只能被主机调用	可以省略

注:

- __device__ : 声明一个函数是设备上执行的, 仅可以从设备调用;
- __global__ : 在设备上执行, 可以从主机调用;
- __host__ : 声明的函数是在主机上执行的, 仅可从主机调用;
- __device__ 和 __global__ 函数不支持递归;
- __device__ 和 __global__ 函数不能声明静态变量在它们内部。

2.nms_kernel()函数

```
1 //nms kernel (CUDA编程中的核函数)
2 /*
3  参数n_boxes:边界框数目
4  参数nms_overlap_thresh:交并比阈值
5  参数dev_boxes:存储边界框信息, 每五位组成一个边界框信息, [left.x,left.y,right.x,right.y,class]
6  参数dev_mask:存储边界框间的交并比是否超过上述阈值的信息, 以ULL类型进行表示, 与哪个框交并比超过阈值, 相应位置1, 否则置0 (输出参数)
7  */
8 __global__ void nms_kernel(const int n_boxes, const float nms_overlap_thresh,
9                           const float *dev_boxes, unsigned long long *dev_mask) {
10     const int row_start = blockIdx.y; //当前调用的block的y坐标 (实际是一个索引)
11     const int col_start = blockIdx.x; //当前调用的block的x坐标
12
13     // if (row_start > col_start) return;
14
15     //min()的目的是防止从dev_boxes中读取数据越界 (原因是n_boxes不一定被threadsPerBlock整除)
16     //实际上只有最后一个block中所需要的线程数目可能小于threadsPerBlock, 其余均等于threadsPerBlock
17     const int row_size =
18         min(n_boxes - row_start * threadsPerBlock, threadsPerBlock);
19     const int col_size =
20         min(n_boxes - col_start * threadsPerBlock, threadsPerBlock);
21
22     //__shared__ 限定词, 即每个block中的所有线程共享内存
23     __shared__ float block_boxes[threadsPerBlock * 5]; //数字5即边界框的5个信息
24     if (threadIdx.x < col_size) {
25         block_boxes[threadIdx.x * 5 + 0] =
26             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 0]; //left.x
27         block_boxes[threadIdx.x * 5 + 1] =
28             dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 1]; //left.y
29         block_boxes[threadIdx.x * 5 + 2] =
```

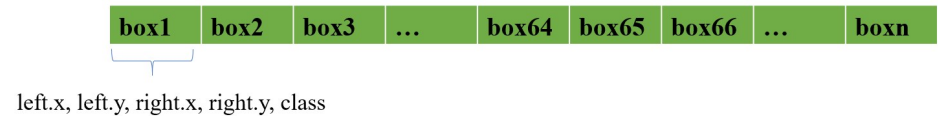
```

30 dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 2]; //right.x
31 block_boxes[threadIdx.x * 5 + 3] =
32     dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 3]; //right.y
33 block_boxes[threadIdx.x * 5 + 4] =
34     dev_boxes[(threadsPerBlock * col_start + threadIdx.x) * 5 + 4]; //class
35 }
36 __syncthreads(); //同步线程(使得当前block中的所有线程均读取到相应边界框信息后再执行后面的代码)
37
38 //以下代码实现某一边界框与其余所有边界框(删去了部分重复)进行交并比的阈值判断
39 if (threadIdx.x < row_size) {
40     const int cur_box_idx = threadsPerBlock * row_start + threadIdx.x; //当前选中的边界框索引
41     const float *cur_box = dev_boxes + cur_box_idx * 5; //当前选中的边界框信息首地址索引
42     int i = 0;
43     unsigned long long t = 0; //用于记录与当前边界框交并比情况, 大于阈值相应位置1
44     int start = 0;
45     if (row_start == col_start) { //如果当前边界框所处的block与要比较的边界框所处的block相同, 则start不从0开始, 减少重复计
46         start = threadIdx.x + 1;
47     }
48     for (i = start; i < col_size; i++) {
49         if (devIoU(cur_box, block_boxes + i * 5) > nms_overlap_thresh) {
50             t |= 1ULL << i; //1ULL = unsigned long long型的数字1 (最高位为第64位); 每一位就代表一个边界框索引, 如果大于阈值,
51         }
52     }
53     const int col_blocks = DIVUP(n_boxes, threadsPerBlock);
54     dev_mask[cur_box_idx * col_blocks + col_start] = t; //存入当前边界框与当前选定的block中的64个边界框的交并比比较情况,
55 }
56 }

```

此函数在理解上可能会有一定困难, 以下我以图像的方式稍生动一点来说明该函数在干什么。

A. 函数输入的dev_boxes中存储的内容如下图 (每一个边界框都有5个信息按顺序存储着) :



B. 函数输出的dev_mask中存储的内容如下图 (threadsPerBlock即每个block所含有的线程数) :

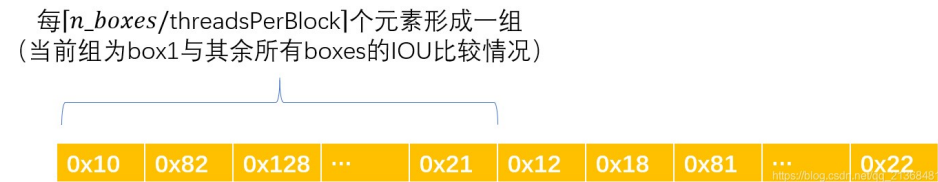
图中是数字, 拿0x11为例说明如下:

0x11 = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0001 (共64位)

其中第5位为1, 表示当前矩形框与当前选中的block中的第5号矩形框的交并比大于设定的阈值。

由于0x10处于box1所在的第1位, 更具体一步表示就是box1与第一个block中的64个边界框中的第5个 (即box5) 的交并比大于设定的阈值。

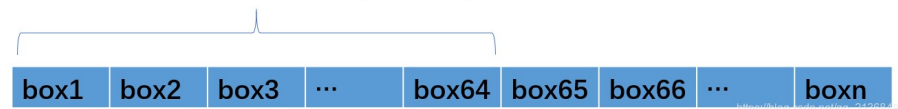
注: 上述所有索引都从1开始 (算法本身是从0开始); 图中的符号 $\lceil \cdot \rceil$ 表示向上取整。



C. 具体在干什么

作者的一大巧妙之处是将二维的block中的两维都表示为dev_boxes，即blockIdx.x和blockIdx.y名义上是block的索引，但实际上表示的是将dev_boxes分块后的块索引，如下图：

每threadsPerBlock（代码中为64）个边界框形成一块
（当前块的索引为blockIdx.x = 0 或blockIdx.y = 0）



代码在干的事就是取当前blockIdx.y块中的第threadIdx.x个边界框与当前blockIdx.x块中的所有边界框进行交并比上的判断，由此为后续nms做准备。

但是为了降低部分重复计算，如(box1, box2)和(box2, box1)这成对的重复计算，采用如下代码：

```
1 if (row_start == col_start) { //如果当前边界框所处的block与要比较的边界框所处的block相同，则start不从0开始，减少重复计算
2     start = threadIdx.x + 1;
3 }
```

但细心的你们一定会发现，其实上述代码只避免了相同块中的重复计算，对于不同块之间仍旧存在重复计算，例如(box1, box65)和(box65, box1)，其中box1属于第blockIdx.y = 0块，box65属于blockIdx.y = 1块。（当然重复计算并不会影响后续的nms，但会消耗时间）

3_nms()函数

```
1 //此函数实际上的__host__类型，真正实现nms
2 /*
3 参数keep_out:int型指针，用于存储所有保留下来的边界框索引
4 参数num_out:保留下的边界框数目
5 参数boxes_host:输入参数，存储着边界框信息，来自于主机
6 参数boxes_num:输入的边界框数目
7 参数boxes_dim:边界框维度（一般为5，即左上角、右下角和类别）
8 参数nms_overlap_thresh:交并比值，用于nms
9 参数device_id:GPU设备号
10 */
11 void _nms(int* keep_out, int* num_out, const float* boxes_host, int boxes_num,
12         int boxes_dim, float nms_overlap_thresh, int device_id) {
13     _set_device(device_id); //设置相应设备
14
15     float* boxes_dev = NULL;
16     unsigned long long* mask_dev = NULL;
17
18     const int col_blocks = DIVUP(boxes_num, threadsPerBlock); //向上取整，即当前输入分块后的块数目
19
20     CUDA_CHECK(cudaMalloc(&boxes_dev,
21         boxes_num * boxes_dim * sizeof(float))); //开辟显存
22     CUDA_CHECK(cudaMemcpy(boxes_dev,
23         boxes_host,
24         boxes_num * boxes_dim * sizeof(float),
25         cudaMemcpyHostToDevice)); //将host输入的数据送入到boxes_dev中
26
27     CUDA_CHECK(cudaMalloc(&mask_dev,
28         boxes_num * col_blocks * sizeof(unsigned long long)));
29
30     dim3 blocks(DIVUP(boxes_num, threadsPerBlock),
31         DIVUP(boxes_num, threadsPerBlock)); //所设置的block为二维block，两维的大小相同
32     dim3 threads(threadsPerBlock); //每一个block中的线程为一维，均为threadsPerBlock条线程
33     nms_kernel<<<blocks, threads>>>(boxes_num,
34         nms_overlap_thresh,
```

```

35         boxes_dev,
36         mask_dev); //调用上述定义的核心函数获取交并比情况
37
38     std::vector<unsigned long long> mask_host(boxes_num * col_blocks);
39     CUDA_CHECK(cudaMemcpy(&mask_host[0],
40         mask_dev,
41         sizeof(unsigned long long) * boxes_num * col_blocks,
42         cudaMemcpyDeviceToHost)); //从device中处理好的数据送回mask_host, 进行后续CPU计算
43
44     std::vector<unsigned long long> remv(col_blocks); //存储要移除的边界框索引
45     memset(&remv[0], 0, sizeof(unsigned long long) * col_blocks); //初始化为0
46
47     //以下正式开始进行nms, 思想和CPU版本有所不同, 但本质是一样的
48     //由于输入此函数的boxes_host是按置信度从高到低排序, 所以第一个边界框肯定会存入keep_out中
49     int num_to_keep = 0;
50     for (int i = 0; i < boxes_num; i++) {
51         int nblock = i / threadsPerBlock; //当前边界框输入哪一个block
52         int inblock = i % threadsPerBlock; //当前边界框输入对应block中的第几个
53
54         //当i = 0时, remv[0] = 0 (初始值), 但由于第一个边界框肯定要存入keep_out中, 所以没问题
55         if (!(remv[nblock] & (1ULL << inblock))) { //判断当前边界框与前面保留下来的边界框之间的交并比是否大于阈值
56             keep_out[num_to_keep++] = i; //如果不大于阈值, 则当前边界框应该保留
57             unsigned long long *p = &mask_host[0] + i * col_blocks;
58             for (int j = nblock; j < col_blocks; j++) {
59                 remv[j] |= p[j]; //预存入后续所有边界框是否要被移除的信息 (相应位为1则移除)
60             }
61         }
62     }
63     *num_out = num_to_keep;
64
65     CUDA_CHECK(cudaFree(boxes_dev));
66     CUDA_CHECK(cudaFree(mask_dev));
67 }

```

此函数的nms部分可能较难理解（越是没有几行的代码越是难以理解），我就举个例子引导一下大家的思维：

假如当前的 $i = 0$ ，即取到box1，根据nms的原理可知，box1肯定会保留下来（因为它的置信度最高），即 $!(remv[nblock] \& (1ULL \ll inblock)) = true$ 一定得成立（故remv的所有元素要初始化为0，原因便在于此），由此会进入到if中执行里面的代码。

这时关键就来了，作者通过按位或操作来快速形成要移除的边界框索引，即如下代码：

```

1 unsigned long long *p = &mask_host[0] + i * col_blocks;
2 for (int j = nblock; j < col_blocks; j++) {
3     remv[j] |= p[j]; //预存入后续所有边界框是否要被移除的信息 (相应位为1则移除)
4 }

```

所谓的要移除的边界框索引是指：如果remv[n]中的某一位的值为1，则第n个block中对应的该位所对应的边界框需要被移除，因为该边界框与保留下来的某一边界框的交并比已经超过了所设定的阈值。

好了，回到当前的box1，因为所有的边界框都被分配到了相应的块（block）中，所以remv数组的大小为col_blocks，而通过循环按位或后，remv中存储的是box1与其余边界框的交并比比较情况，也即要移除的边界框索引。

当 $i = 1$ 时，如果remv[0]的第2位（从1开始）为1，则不进入if，即直接移除不保留；如果为0，则进入if，保留box2的索引，以及更新remv。更新过程就是将box1的dev_mask中的内容（也即当前的remv）与box2的dev_mask中的内容进行按位或，意思就是如果box3与更新后的remv中的对应为吻合，则我们不需要管是和box1还是box2的交并比超过了阈值，直接将其移除即可。

后面的过程依此类推。



走的那么干脆

码龄8年  暂无认证

35
原创

83万+
周排名

136万+
总排名

16万+
访问


等级

1871
积分

76
粉丝

110
获赞

98
评论

354
收藏



私信

关注

搜博主文章

Q

热门文章

解析caffe生成的caffemodel文件

23976

Anaconda2和Anaconda3反复安装出现的
问题的解决方法

21076

Ubuntu16.04下Cython编译出现command
'gcc' failed with exit status 1

14625

SSD网络解析之PriorBox层

14468

SSD网络解析之Permute层

9907

最新评论

目标检测中NMS的GPU实现 (来自于Fas...
weixin_47755558: 没有, 不会C语言

目标检测中NMS的GPU实现 (来自于Fas...
Elaine✂: 请问你成功实现了吗? 我正好在做实验, 但是不知道怎么把soft-nms变成...

目标检测中NMS的GPU实现 (来自于Fas...
Elaine✂: 我也想问这个问题

bash: */anaconda3/bin/conda: bad inter...
Cora_LY: 重启终端之后也不可以啊, 百度的方法也是几乎都试过了, 还是没有结果...

Anaconda2和Anaconda3反复安装出现...
zxzxwcdw: 感谢, 解决了大问题了

您愿意向朋友推荐“博客详情页”吗？







强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

NMS-gpu 和 Cython 非极大值抑制在 windows 上的 编译及使用 目标检测NMS-GPU 和 Cython （非极大值抑制）在 window 下的 编译文件 ，包括 soft_NMS实现 。小批量情况下 Cython 速度高于 GPU	04-19
win10, python35, cuda90实现NMS 原帖 https://www.cnblogs.com/king-lps/p/9031568.html , Linux平台下。方法1、2没有大问题，只需要注意模块名的一致性 方法3: setup2.py 需要添加nu...	yuezhilanyi 的博客 1056
评论 15 您还未登录，请先 登录 后发表或查看评论	
NMS算法的GPU实现(使用CUDA加速计算)_不会写代码的完结... #include"gpu_nms.hpp" #include<vector> #include<iostream> //cudaError_t是cuda中的一个类,用于记录cuda错误(所有的cuda函数,几乎都会返回一个cu...	3-29
No module named 'nms.gpu_nms' or No module named... 跑开源项目时报错 ModuleNotFoundError: No module named 'utils.nms.cpu_nms',查了很多方法大部分都是让在lib/utlis/nms目录下创建cpu_nms.py 这种...	6-2
ssd pytorch转 libtorch c++实现 (nms采用cuda并行实现) 背景: 因为所里面大多数同学在研究论文时用的比较多的还是pytorch,所以考虑在pc端的部署采用libtorch ,当然 libtorch比较新, 所以还是要谨慎地采用,因...	qq_33671888 的博客 2205
cuda 怎么读_一、faster-rcnn源码阅读: nms的CUDA编程 打算写一系列faster-rcnn的阅读笔记,侧重于程序实现的细节问题。包括安装, 版本选择, 编译, 数据读取, 事无巨细的——细说。没有规划, 没有顺序...	weixin_39963287 的博客 135
【NMS总结】: 一文打尽目标检测NMS——精度提升 总体概要: 非极大值抑制NMS是目标检测常用的后处理算法, 用于剔除冗余检测框, 本文将对可以提升精度的各种NMS方法及其变体进行阶段性总结。对...	m0_59962306 的博客 352
Tensorflow踩坑记 (三) : ImportError: No module named gpu_nms 和 ImportError: No module na... 我在跑faster rcnn的时候运行python3 tool/demo.py就会出现错误提示: ImportError: No module named gpu_nms。这是因为没有在tf-faster-rcnn/lib路径...	热门推荐 骑着蜗牛向前跑 的博客 1万+
【模型加速】自定义TensorRT NMS3D插件(1) 需求是这样的, 在做PointPillars模型的加速的时候我注意到网络的检测头部分小型操作很多, 加速效果不明显。此外, 3D检测模型的NMS部分通常是作为...	580
CSP运行出现No module named gpu_nms解决办法 CSP源码: https://github.com/liuwei16/CSP CSP论文: https://arxiv.org/abs/1904.02948 CSP算法思想: https://www.jiqizhixin.com/articles/2019-04-13 ...	xunan003 的博客 5317
ModuleNotFoundError: No module named 'nms.gpu_nms' 参考: https://github.com/rbgirshick/py-faster-rcnn/issues/8 参考: https://blog.csdn.net/qq_42647047/article/details/105341478?spm=1001.2101.3001....	GXQx 的博客 799
win10 vs2015 编译nms和gunms rcnn r-fcn tiny-face 需要用, win10 vs2015 编译nms和gunms, 亲测成功, 推荐下载	02-07
setup_cuda.py 编译gpu_nms setup_cuda.py setup_cuda 1.修改 include_dirs = [numpy_include, r'C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\include'] 2.报错: LIN...	jacke121 的专栏 6686
NMS技术总结 (NMS原理、多类别NMS、NMS的缺陷、NMS的改进思路、各种NMS方法) 前言本文介绍了NMS的应用场合、基本原理、多类别NMS方法和实践代码、NMS的缺陷和改进思路、介绍了改进NMS的几种常用方法、提供了其它不常用...	最新发布 CV技术指南 (微信公众号) 852
非极大值抑制NMS代码实现 (Python) 文章目录前言NMS代码实现1.导入必要的库2.人为生成一组位置坐标, 模拟候选框3.定义NMS (1) 获取位置坐标, 本代码用对角坐标表示位置 (2) 计算...	weixin_41761357 的博客 252
一文打尽目标检测NMS——效率提升篇 在笔者上一篇文章《一文打尽目标检测NMS——精度提升篇》中, 总结了近几年出现的一些可以提升NMS精度的方法。可以看到, NMS由于顺序处理的原...	weixin_43900320 的博客 1975
nms python代码_faster rcnn two stage(分步训练方式)代码解读 人在美国, 刚下飞机, 在飞机上阅读了下faster rcnn 分步训练的源码, 感觉网上关于end2end方式的代码解读不少, 却鲜有alt opt方式的代码解读, 写此...	weixin_39777488 的博客 164
cuda第一次计算耗时_nms的cuda实现解析 代码 https://github.com/rbgirshick/py-faster-rcnn/blob/master/lib/nms/nms_kernel.cugithub.com nms是不太好在cuda上实现的, 因为nms的计算过程是有...	weixin_33700809 的博客 707
【模型加速】CUDA-Pointpillars项目解读(3) 后处理 接上篇【模型加速】CUDA-Pointpillars项目解读(2), 就PointPillars而言神经网络部分的耗时相对较少, 时间消耗主要在后处理部分。PointPillars检...	1051
MXNet的高效率CUDA NMS解析 非极大抑制 (NMS: Non Maximum Suppression) 起到边框 (水平框或倾斜框) 去重叠的作用, 广泛应用于通用目标检测、人脸检测与OCR检测等算法的...	AI Flash 1366
python gpu_nms编译错误 解决方案: 将出错的函数变量__pyx_t_5numpy_int32_t*, 改成int*(见红色字体)。接着, 为了保证gpu_nms.cpp不再由gpu_nms.pyx自动生成, 需要将setu...	fanhenghui 的专栏 2996



interpreter: No such file or directory

堆排序

Ubuntu16.04下Cython编译出现command
'gcc' failed with exit status 1

2020年 1篇

2019年 10篇

2018年 24篇

“相关推荐”对你有帮助？



非常没帮助



没帮助



一般



有帮助



非常有帮助

©2022 CSDN 皮肤主题：大白 设计师：CSDN官方博客 返回首页

[关于我们](#) [招贤纳士](#) [商务合作](#) [寻求报道](#) [☎ 400-660-0108](#) [✉ kefu@csdn.net](#) [💬 在线客服](#) 工作时间 8:30-22:00



走的那么干脆

关注



6



15



32



专栏目录



举报

