

Yolo cpp_cv_bmcv_bmrt_postprocess Demo 使用说明

1. 说明

本demo可以运行的是darkent框架训练的Yolov3、v4(包括tiny系列)模型，通过BMNNToolchain工具链编译和或量化成fp32或int8的bmodel。可支持任意batch。

2. 修改和自己模型对应的参数

修改yolov3.hpp 203行附近的代码，主要是修改anchor_bias、classes_num_、mask，也有用户的模型anchor_num_也做了变动。

```
.../* anchor */
...float biases_[12] = {10, 14, 23, 27, 37, 58, 81, 82, 135, 169, 344, 319};
...int masks_[6] = {3, 4, 5, 0, 1, 2};
...const int anchor_num_ = 3;
...const size_t classes_num_ = 2;
```

修改参照的是模型对应cfg文件中的参数值。

```
[yolo]
mask = 0,1,2
anchors = 10,14,...23,27,...37,58,...81,82,...135,169,...344,319
classes=82 #80
num=0
jitter=.3
ignore_thresh=.7
truth_thresh=.1
random=1
```

3. 编译

a) 如果硬件是SC5,

```
make -f Makefile.pcie -j4
```

b) 如果硬件是SE5/SM5,

```
make -f Makefile.arm -j4, 将生成的可执行文件yolo_test拷贝到SE5/SM5上执行。
```

4. 运行

```
./yolo_test image imagelist.txt compilation.bmodel
```

其中，imagelist.txt的每一行是图片的路径。

```
image/00000189752.jpg
image/00000177383.jpg
image/00000460147.jpg
image/00000301421.jpg
image/00000561223.jpg
image/00000565597.jpg
image/00000564280.jpg
image/00000120777.jpg
image/00000464089.jpg
image/00000418281.jpg
image/00000243989.jpg
image/00000509824.jpg
~
~
```

代码会根据模型的batch数，凑成batch数的图片后进行检测计算。

```
...if(!is_video){...
...char image_path[1024] = {0};
...vector<cv::Mat> batch_imgs;
...vector<string> batch_names;
...ifstream fp_img_list(image_list);
...while(fp_img_list.getline(image_path, 1024)){...
...cv::Mat img = cv::imread(image_path, cv::IMREAD_COLOR, 0);
...if(img.empty()){...
...cout << "read image error!" << endl;
...exit(1);
...
...fs::path fs_path(image_path);
...string img_name = fs_path.filename().string();
...batch_imgs.push_back(img);
...batch_names.push_back(img_name);
...if (static_cast<int>(batch_imgs.size()) == batch_size) {...
...detect(net, batch_imgs, batch_names, &ts);
...batch_imgs.clear();
...batch_names.clear();
...}
...
...}
```

最终结果图片会在执行目录下的result_imgs文件夹下生成，也有统计的每一步的执行时间（解码、前处理、检测、后处理）。

```
#####
SUMMARY: detect
#####
[ detection overall] loops: 200 avg: 14861 us
[ stage 1: decode] loops: 200 avg: 5265 us
[stage 2: pre-process] loops: 200 avg: 1804 us
[stage 3: detection ] loops: 200 avg: 7384 us
[stage 4:post-process] loops: 200 avg: 396 us
```