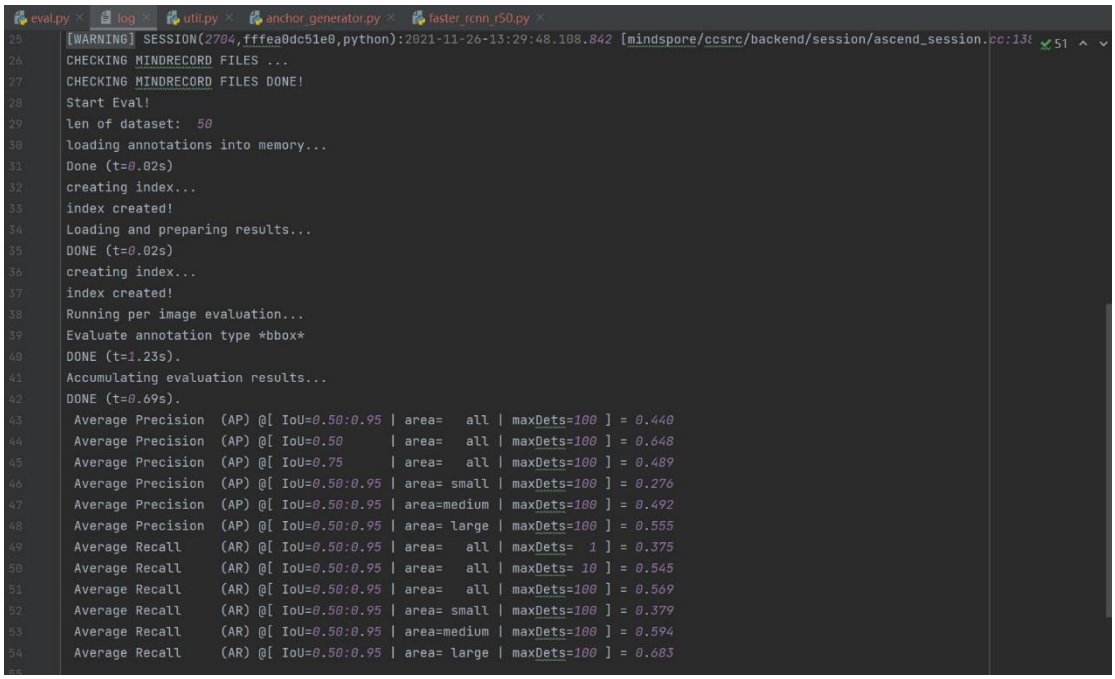
Project 1: Object Detection on COCO

高易远 精 83 2018010650

1 results



2 code implementation

2.1 step1-5

```python
def fasterrcnn_eval(dataset_path, ckpt_path, ann_file):
    """FasterRcnn evaluation."""

    # 1.build dataset (functions may be used: create_fasterrcnn_dataset)
    fasterrcnn_dataset = create_fasterrcnn_dataset(dataset_path, batch_size=config.test_batch_size, is_training=False)
    len_dataset = fasterrcnn_dataset.get_dataset_size()
    print('len of dataset: ', len_dataset)

    # 2.build network
    net = Faster_Rcnn_Resnet50(config=config)

    # 3.load trained checkpoint (functions may be used: load_checkpoint, load_param_into_net)
    ckpt = load_checkpoint(ckpt_path)
    load_param_into_net(net, ckpt)

    # 4.set eval mode
    net.set_train(False)

    # 5.set model as float16 for Ascend inference
    device_type = "Ascend" if context.get_context("device_target") == "Ascend" else "Others"
    if device_type == "Ascend":
        net.to_float(mstype.float16)
```

函数传入的三个参数是 dataset 路径，checkpoints 路径，annotations 路径。

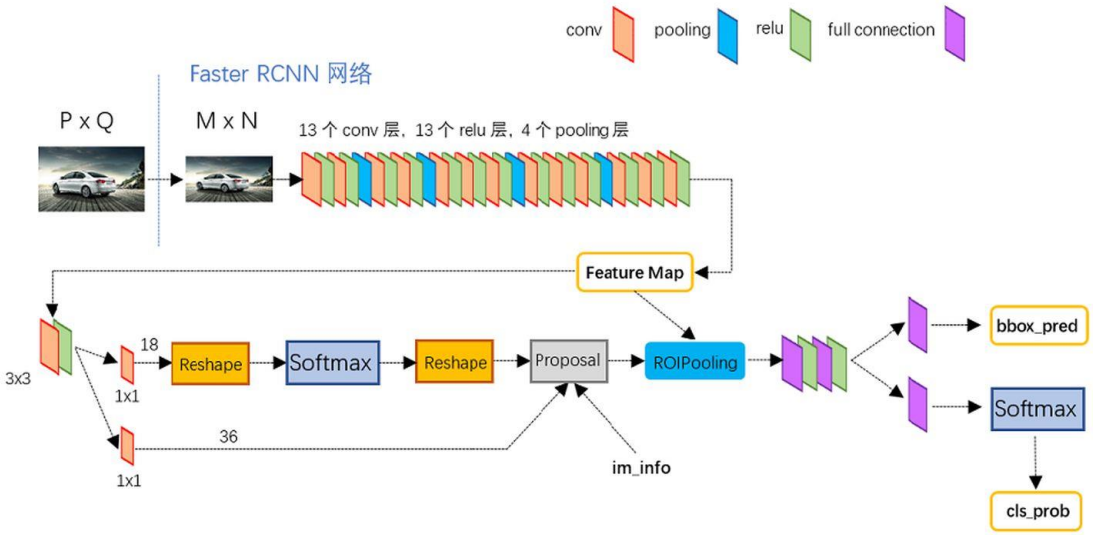
完成 step1-5 时，可以参考同为顶层接口的 train.py，根据提示需要稍作改动就可以。step4 中的 load_checkpoint, load_params_into_net 函数根据 eval.py 文件中开头的引用 from mindspore.train.serialization import load_checkpoint, load_param_into_net，查阅 mindspore 库也可以知道用法。

2.2 step6

```
# 6.inference process (functions may be used: bbox2result_1image)
# Note: the inference process includes both model inference and post-process
outputs = []
max_num = 128  # max num of bboxes reserved for one image

# inference
for data in fasterrcnn_dataset.create_dict_iterator(num_epochs=1):
    img_data = data['image']
    img_metas = data['image_shape']
    gt_bboxes = data['box']
    gt_labels = data['label']
    gt_num = data['valid_num']

    output = net(img_data, img_metas, gt_bboxes, gt_labels, gt_num)
    all_bbox = output[0]
    all_label = output[1]
    all_mask = output[2]
```

```

    # post-process
    for i in range(config.test_batch_size):
        all_bbox_squeezed = np.squeeze(all_bbox.asnumpy()[i, :, :])
        all_label_squeezed = np.squeeze(all_label.asnumpy()[i, :, :])
        all_mask_squeezed = np.squeeze(all_mask.asnumpy()[i, :, :])

        all_bboxes_mask = all_bbox_squeezed[all_mask_squeezed, :]
        all_labels_mask = all_label_squeezed[all_mask_squeezed]

        if all_bboxes_mask.shape[0] > max_num:
            indexes = np.argsort(-all_bboxes_mask[:, -1])
            indexes = indexes[:max_num]
            all_bboxes_mask = all_bboxes_mask[indexes]
            all_labels_mask = all_labels_mask[indexes]

        outputs_tmp = bbox2result_1image(all_bboxes_mask, all_labels_mask, config.num_classes)
        outputs.append(outputs_tmp)
```

inference 过程，首先找到整个网络的定义文件：src/FasterRcnn/faster_rcnn_r50.py，这个文件引用了同一目录下的各个子模块，应用一张网络结构图（结构图中的 backbone 是 VGG-16）：



同一目录下的各个子模块：使用的 backbone 是 ResNet-50，neck 是 FPN，RPN 生成 region proposals，经过 ROI pooling，最后做 classification（对应 rcnn.py）。

Faster_Rcnn_Resnet50 类 construct 函数接收的参数为：

```
def construct(self, img_data, img_metas, gt_bboxes, gt_labels, gt_valids):
```

返回值为：

```
else:
    output = self.get_det_bboxes(rcnn_cls_loss, rcnn_reg_loss, rcnn_masks, bboxes_all, img_metas)

return output
```

```
def get_det_bboxes(self, cls_logits, reg_logits, mask_logits, rois, img_metas):
    """Get the actual detection box."""
```

```
output = self.multiclass_nms(boxes_all_with_batchsize, scores_all, mask_all)

return output
```

```
def multiclass_nms(self, boxes_all, scores_all, mask_all):
    """Multiscale postprocessing."""
```

```
all_bboxes = self.concat(all_bboxes)
all_labels = self.concat(all_labels)
all_masks = self.concat(all_masks)
return all_bboxes, all_labels, all_masks
```

从中可见用于 eval/inference 时，在给出 loss 的基础上还需要进行 get_bbox 和 NMS 后处理。

src/dataset.py 中 create_fasterrcnn_dataset 函数给出了 dataset 中的内容：

```
else:
    ds = ds.map(input_columns=["image", "annotation"],
                output_columns=["image", "image_shape", "box", "label", "valid_num"],
                column_order=["image", "image_shape", "box", "label", "valid_num"],
```

使用 create_dict_iterator 方法从 dataset 中调出 data 放入网络前向传播。

然后进行后处理，对每个 batch 中的 2 张图片（config.test_batch_size=2），通过得到的 masks 来保留每张图片对应的 bboxes 和 labels 的正例；在完成作业的过程中尝试在终端直接运行 eval.py，通过 print 观察相关变量的维度，如下所示：

```
all_bbox_shape: (2, 80000, 5)
all_label_shape: (2, 80000, 1)
all_mask_shape: (2, 80000, 1)
all_mask:
[[[False]
  [False]
  [False]
  ...
  [False]
  [False]
  [False]]

 [[ True]
  [False]
  [False]
  ...
  [False]
  [False]
  [False]]]
------
```

按照网络学堂上的提示，每张图片保留检测结果总数的参考值为 128，但是在调试过程中发现所给出的 50 张图片几乎没有图片检测结果总数超过 128，其中某一张图片的检测结果如下:

```
all_bbox_squ: [[103.8    147.5    430.5    239.8      0.8755]
 [106.5    148.     443.2    248.6      0.7886]
 [134.5    146.4    427.8    237.5      0.714 ]
 ...
 [ -1.      -1.      -1.      -1.       0.    ]
 [ -1.      -1.      -1.      -1.       0.    ]
 [ -1.      -1.      -1.      -1.       0.    ]]
all_label_squ: [ 0  0  0 ... 79 79 79]
all_mask_squ: [ True False False ... False False False]
--------
all_bboxes_mask_num: 16
all_bboxes_mask: [[1.038e+02 1.475e+02 4.305e+02 2.398e+02 8.755e-01]
 [3.660e+02 1.492e+02 4.228e+02 2.116e+02 6.973e-01]
 [2.838e+02 1.460e+02 4.158e+02 2.314e+02 5.762e-01]
 [4.419e+01 3.902e+00 3.698e+02 2.419e+02 1.614e-01]
 [3.535e+02 1.475e+02 4.832e+02 2.296e+02 1.400e-01]
 [2.680e+02 1.024e+01 6.210e+02 2.171e+02 7.953e-02]
 [3.032e+02 1.579e+02 3.848e+02 2.322e+02 9.692e-02]
 [1.512e+02 0.000e+00 6.310e+02 2.266e+02 1.494e-01]
 [0.000e+00 0.000e+00 4.218e+02 2.338e+02 9.491e-02]
 [2.945e+02 1.609e+02 3.782e+02 2.325e+02 1.356e-01]
 [0.000e+00 3.316e+01 6.375e+02 4.082e+02 9.619e-01]
 [3.734e+01 0.000e+00 6.365e+02 2.478e+02 4.260e-01]
 [0.000e+00 1.945e+02 5.270e+02 4.260e+02 2.076e-01]
 [1.722e+02 1.301e+02 6.120e+02 4.205e+02 1.165e-01]
 [0.000e+00 0.000e+00 3.352e+02 2.321e+02 1.164e-01]
 [3.598e+02 2.082e+02 4.222e+02 2.310e+02 7.019e-02]]
all_labels_mask: [ 0  0  0  0  0  0 24 25 25 26 59 59 59 59 59 73]
```

这里有个疑问，为什么 bbox 是 5 维的？前 4 维应该是 coco 数据集的边框坐标编码 [x_min, y_min, width, height]。

之后根据提示，调用 src/util.py 中的 bbox2result_1image 函数将 results 转 numpy 数组准备后续 eval，并 append 到所有的 output。

2.3 step7-8

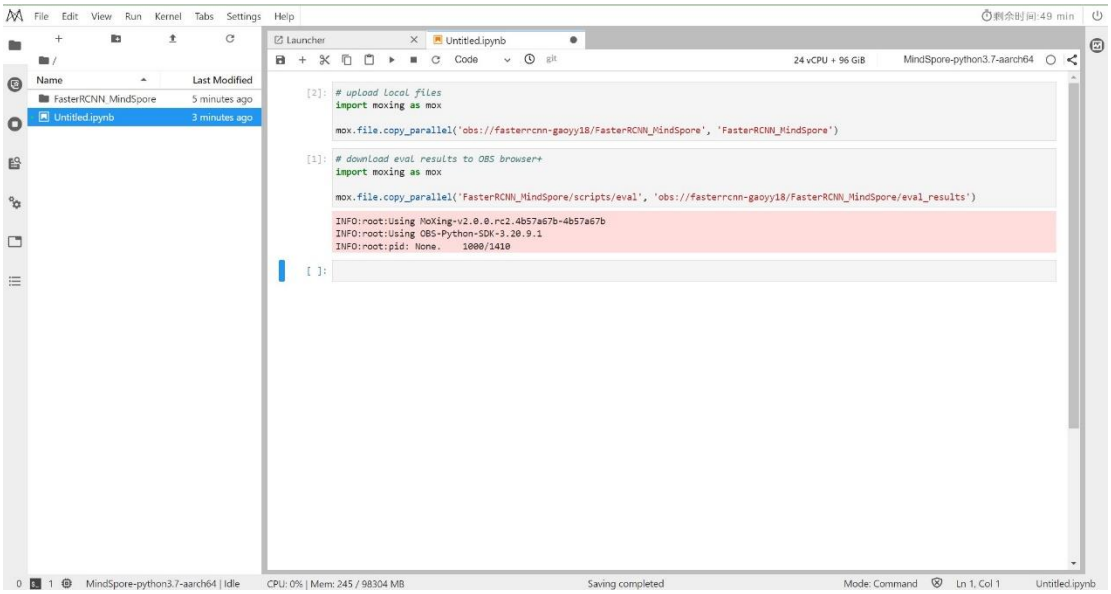
```
    # 7.load ground-truth annotations
    gt = COCO(ann_file)

    # 8.evaluation (functions may be used: results2json, coco_eval)
    result_files = results2json(gt, outputs, "./results.pkl")
    eval_types = ["bbox"]
    coco_eval(result_files, eval_types, gt)
```

根据对应函数的接口进行调用即可。

3 evaluation process illustration

在云服务器上完成 evaluation，使用华为云-ModelArts-Notebook，创建一个 mindspore 环境，规格：Ascend 1*Ascend 910|CPU: 24 核 96GB。

从本地上传相关文件到 OBS Browser+上，在开发环境中的 notebook 中，使用 MoXing 中的 file.copy_parallel 方法完成文件在开发环境和 OBS 中的传输（复制），如下图。



在开发环境的终端中，按照提示完成环境准备与 evaluation，其中 evaluation 的命令为：

```
sh run_eval_ascend.sh ../coco/annotations/instances_demo_val2017.json ../faster_rcnn_trained_model.ckpt
```

最后将 results 的 log 和 json 文件从 OBS 保存到本地。

4 visualization

思路：根据 gt 的 json 文件和得到的 results 的 json 文件（results.pkl.bbox.json）用 cv2 可视化，时间有限没能完成。

5 improvement

简单查阅相关资料:

更好的 backbone：从 VGG 到 ResNet50（本次采用）再到 ResNet101，以及多个 ResNet 等等，速度上更快的 PVANet 等等;

更好的 neck：按照我的理解，Faster R-CNN 原文没有用到 FPN，但是在本次采用的代码实现中使用了 FPN 输出的 4 层不同尺度的特征图;

将 Roi Pooling 替换成 Roi Align（Mask R-CNN）;

后处理部分等。