mmdetection2.6标准数据集标准模型的训练与推理

mmdetection的github链接：https://github.com/open-mmlab/mmdetection
mmdetection的官方文档：https://mmdetection.readthedocs.io/en/latest/

主要实现如下的功能：

- 使用现有的模型对给定的图像进行推理
- 在标准数据集上测试已有的模型
- 在标准数据集上训练预定义的模型

1. 使用现有的模型进行推理

对于推理来说就是采用现有的模型来检测图像中的物体，在mmdetection中，模型在config文件中定义，而模型的参数存在于checkpoint文件中。

这里采用faster rcnn的config文件以及checkpoint文件作为基础来进行如下的操作。

1.1 高层次的推理API

mmdetection为照片的推理提供了高层的API，这里是一个简单的样例代码：

```
from mmdet.apis import init_detector, inference_detector
import mmcv

# Specify the path to model config and checkpoint file
config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'

# build the model from a config file and a checkpoint file
model = init_detector(config_file, checkpoint_file, device='cuda:0')

# test a single image and show the results
img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once
result = inference_detector(model, img)
# visualize the results in a new window
# 可视化推理检测的结果
model.show_result(img, result)
# or save the visualization results to image files
# 将推理的结果保存
model.show_result(img, result, out_file='result.jpg')

# test a video and show the results
# 测试视频片段的推理结果
video = mmcv.VideoReader('video.mp4')
for frame in video:
    result = inference_detector(model, frame)
```

```
    model.show_result(frame, result, wait_time=1)
```

在`demo/inference_demo.ipynb`中可以找到这个推理的脚本代码

1.2 python3.7支持的异步接口

对于python3.7来说，mmdetection支持异步接口，通过cuda流的使用，可以在不阻止cpu和gpu绑定的推理代码的情况下，在单线程应用中能够更好的提升cpu与gpu的利用率， 可以在不同的输入数据样本之间或某个推理管道的不同模型之间同时进行推理。

在`test/async_benchmark.py`中可以对比同步与异步接口的速度。

```
import asyncio
import torch
from mmdet.apis import init_detector, async_inference_detector
from mmdet.utils.contextmanagers import concurrent

async def main():
    config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
    checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-
047c8118.pth'
    device = 'cuda:0'
    model = init_detector(config_file, checkpoint=checkpoint_file, device=device)

    # queue is used for concurrent inference of multiple images
    streamqueue = asyncio.Queue()
    # queue size defines concurrency level
    streamqueue_size = 3

    for _ in range(streamqueue_size):
        streamqueue.put_nowait(torch.cuda.Stream(device=device))

    # test a single image and show the results
    img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once

    async with concurrent(streamqueue):
        result = await async_inference_detector(model, img)

    # visualize the results in a new window
    model.show_result(img, result)
    # or save the visualization results to image files
    model.show_result(img, result, out_file='result.jpg')


asyncio.run(main())
```

1.4 Demos

在`demo`文件夹中有两个推理的demo代码。

1.4.1 Image demo

有三个必要的参数，依次为测试图像的路径， config文件的路径， checkpoint文件的路径。还可以指定cpu还是gpu运行， 默认值为gpu， 保留bbox的分数阈值，默认为0.3

```
python demo/image_demo.py \
    ${IMAGE_FILE} \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--device ${GPU_ID}] \
    [--score-thr ${SCORE_THR}]
```

例子:

```
python demo/image_demo.py demo/demo.jpg \
    configs/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --device cpu
```

1.4.2 webcam demo(测试摄像头)

基本的参数同上，必须指定的参数为config文件的路径，checkpoint文件的路径，其他的gpu或者cpu设备， 摄像头id号， 得分阈值可以选择指定

```
python demo/webcam_demo.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--device ${GPU_ID}] \
    [--camera-id ${CAMERA-ID}] \
    [--score-thr ${SCORE_THR}]
```

例子:

```
python demo/webcam_demo.py \
    configs/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth
```

2. 在标准数据集上测试已经存在的模型

为了测试模型的准确率，往往需要在标准数据集上测试模型，mmdetection支持coco，voc，cityscapes等一些其他的标准数据集，在目标检测中最常用的数据集格式就是coco数据集，我的习惯都会将自己的数据集转换为这种标准格式，进行后续的训练验证测试。

2.1 测试现有的模型

mmdetection提供了测试现有模型的测试脚本。

- 单GPU
- 多GPU的单节点
- 多节点

对于不同的环境选择合适的测试脚本。

```
# single-gpu testing
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}] \
    [--show]

# multi-gpu testing
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}]
```

`tools/dist_test.sh`支持多节点的测试。

其中除了必要的cofig文件与checkpoint文件，还有其他可选择的参数：

- `RESULT_FILE`：输出结果的文件名采用pickle格式。如果未指定，结果将不会保存到文件中。
- `EVAL_METRICS`：要根据结果评估的项目。允许的值取决于数据集，例如`proposal_fast`，`proposal`，`bbox`，`segm`可用于COCO，`mAP`，`recall`为PASCAL VOC。可以通过`cityscapes`所有COCO度量标准来评估城市景观。
- `--show`：如果指定，检测结果将绘制在图像上并显示在新窗口中。它仅适用于单个GPU测试，并用于调试和可视化。请确保您的环境中有GUI。否则，您可能会遇到类似的错误。`cannot connect to X server`
- `--show-dir`：如果指定，检测结果将绘制在图像上并保存到指定目录。它仅适用于单个GPU测试，并用于调试和可视化。您不需要环境中的GUI即可使用此选项。
- `--show-score-thr`：如果指定，则分数低于此阈值的检测将被删除。

2.2 一些例子

假设已经下载checkpoint文件到`checkpoint/`文件夹中。

1. 测试faster rcnn并可视化结果，按任意建来查看下一张图像

```
python tools/test.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --show
```

2. 测试Faster R-CNN，并保存绘制的图像以供将来可视化。

```
python tools/test.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --show-dir faster_rcnn_r50_fpn_1x_results
```

3. 在PASCAL VOC上测试Faster R-CNN（不保存测试结果）并评估mAP。、

```
python tools/test.py \
    configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc.py \
    checkpoints/faster_rcnn_r50_fpn_1x_voc0712_20200624-c9895d40.pth \
    --eval mAP
```

4. 使用8个GPU测试Mask R-CNN，并评估bbox和mask AP。

```
./tools/dist_test.sh \
    configs/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    --out results.pkl \
    --eval bbox segm
```

5. 使用8个GPU测试Mask R-CNN，并评估**分类**bbox和mask AP。

```
./tools/dist_test.sh \
    configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    --out results.pkl \
    --eval bbox segm \
    --options "classwise=True"
```

6. 在具有8个GPU的COCO test-dev上测试Mask R-CNN，并生成JSON文件以提交给官方评估服务器。

```
./tools/dist_test.sh \
    configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    -format-only \
    --options "jsonfile_prefix=./mask_rcnn_test-dev_results"
```

此命令生成两个JSON文件`mask_rcnn_test-dev_results.bbox.json`和`mask_rcnn_test-dev_results.segm.json`。

7. 在Cityscapes上使用8个GPU测试 Mask R-CNN，并生成txt和png文件提交给官方评估服务器。

```
./tools/dist_test.sh \
    configs/cityscapes/mask_rcnn_r50_fpn_1x_cityscapes.py \
    checkpoints/mask_rcnn_r50_fpn_1x_cityscapes_20200227-afe51d5a.pth \
    8 \
    --format-only \
    --options "txtfile_prefix=./mask_rcnn_cityscapes_test_results"
```

生成的png和txt将在`./mask_rcnn_cityscapes_test_results`目录下。

3. 在标准数据集上训练预训练模型

mmdetection也提供了开禧那个即用的工具来训练检测模型，这部分来显示如何训练预定的模型。

注意：默认的学习率是8gpu，每张gpu上两幅图像，也就是batch size = 16的学习率，对于不同的训练环境可以按照线性的策略来修改学习率。

3.1 单cpu训练

在`tools/train.py`中提供了在单gpu上的基本训练策略，基本使用方式是：

```
python tools/train.py \
    ${CONFIG_FILE} \
    [optional arguments]
```

在训练的过程中，日至文件与checkpoint文件会被保存在工作目录，这个工作目录可以由config文件中的work_dir 或者通过命令行--work-dir指定。

默认情况下，每个epoch都会验证一次模型的准确率， 在config文件中可以调整模型验证步骤的间隔。

```
# evaluate the model every 12 epoch.
evaluation = dict(interval=12)
```

下边是其他的一些可选择的参数：

- `--no-validate`（**不建议**）：禁用在训练期间进行评估。
- `--work-dir ${WORK_DIR}`：覆盖工作目录。
- `--resume-from ${CHECKPOINT_FILE}`：从先前的checkpoint文件继续训练。
- `--options 'Key=value'`：覆盖使用的配置中的其他设置。

注意： 在config文件中存在两个参数，`load_from`与`resume_from`,这两个参数是不同的。

resume_from不仅加载模型的权重，而且加载优化器的状态，epoch从指定的checkpoint文件中得到，这个参数通常应用于在训练过程中偶然断开的恢复继续训练。而load_from仅仅加载模型的权重，并且训练过程从0开始，通常在finetune微调时使用。

3.2 多GPU训练

在`tools/dist_train.sh`中提供了多gpu训练的脚本，基本的使用方式是：

```
./tools/dist_train.sh \
    ${CONFIG_FILE} \
    ${GPU_NUM} \
    [optional arguments]
```

其他可选的参数与单gpu训练时一致。

3.2.1 同时启动多个jobs

如果要在一台计算机上启动多个jobs，例如，在具有8个GPU的计算机上执行2个4-GPU训练jobs，则需要为每个job指定不同的端口（默认为29500），以避免通信冲突。

如果`dist_train.sh`用于启动训练jobs，则可以在命令中设置端口。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 ./tools/dist_train.sh ${CONFIG_FILE} 4
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 ./tools/dist_train.sh ${CONFIG_FILE} 4
```

3.2.2 在多个节点上训练

MMDetection依靠`torch.distributed`软件包进行分布式训练。

3.2.3 使用Slurm管理jobs

Slurm是用于计算集群的良好作业调度系统。在Slurm管理的群集上，您可以`slurm_train.sh`用来生成训练作业。它支持单节点和多节点训练。

基本用法如下：

```
[GPUS=${GPUS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE}
${WORK_DIR}
```

下面是一个使用16 GPU在一个名为dev的slurm分区来训练Mask rcnn的例子，并且设置work_dir为某个共享文件系统。

```
GPUS=16 ./tools/slurm_train.sh dev mask_r50_1x
configs/mask_rcnn_r50_fpn_1x_coco.py /nfs/xxxx/mask_rcnn_r50_fpn_1x
```

您可以检查源代码以查看完整的参数和环境变量。

使用Slurm时，需要通过以下方式之一设置端口选项：

1. 通过设置端口`--options`。推荐这样做，因为它不会更改原始配置。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 ./tools/slurm_train.sh ${PARTITION}
${JOB_NAME} config1.py ${WORK_DIR} --options 'dist_params.port=29500'
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 ./tools/slurm_train.sh ${PARTITION}
${JOB_NAME} config2.py ${WORK_DIR} --options 'dist_params.port=29501'
```

2. 修改配置文件以设置不同的通信端口。

   在中`config1.py`，设置

```
dist_params = dict(backend='nccl', port=29500)
```

   在中`config2.py`，设置

```
dist_params = dict(backend='nccl', port=29501)
```

   然后，您可以使用`config1.py`和启动两个作业`config2.py`。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 ./tools/slurm_train.sh ${PARTITION}
${JOB_NAME} config1.py ${WORK_DIR}
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 ./tools/slurm_train.sh ${PARTITION}
${JOB_NAME} config2.py ${WORK_DIR}
```

以上基本上就是官方英文文档的翻译。