Docker使用

Docker简介

docker是一个轻量型的虚拟机，可以不像传统虚拟机一样占用很多系统资源导致系统变慢，在linux下运行docker跟终端基本一样。

Docker安装

普通docker的安装可以参考https://docs.docker.com/install/linux/docker-ce/ubuntu/
再执行https://docs.docker.com/install/linux/linux-postinstall/操作，以后运行docker就不需要加sudo关键词。

普通docker一般只能使用CPU的应用，如果想要使用GPU，需要在安装了普通docker基础上再安装nvidia-docker。安装nvidia-docker可以参照以下两个网址：

- https://github.com/nvidia/nvidia-docker/wiki/Installation-(version-2.0)
- https://medium.com/@sh.tsang/docker-tutorial-5-nvidia-docker-2-0-installation-in-ubuntu-18-04-cb80f17cac65

当然，如果想要用nvidia-docker，前提是本机要装好nvidia驱动。

作为参照，这里也给出ubuntu docker安装过程样例：

1. 安装docker-ce

```
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo apt-key add -
```

```
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo docker run hello-world
```

如果运行成功，结果截图应该是这样的：

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:41a65640635299bab090f783209c1e3a3f11934cf7756b09cb2f1e02147c6ed8
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2. 以非根用户的身份管理Docker

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

操作结束后记得重启或注销，再次开启系统后尝试运行命令`docker run hello-world`，看看是否成功。

3. 安装nvidia-docker

```
docker volume ls -q -f driver=nvidia-docker | \
  xargs -r -I{} -n1 docker ps -q -a -f volume={} | \
  xargs -r docker rm -f
sudo apt-get purge -y nvidia-docker
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
  sudo apt-key add -
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-doc
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update
sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd
docker run --runtime=nvidia --rm nvidia/cuda:9.0-base nvidia-smi
```

如果觉得下载非常慢，可以尝试切换ubutnu的源，详见
https://blog.csdn.net/qq_35451572/article/details/79516563

Docker使用方法

一般的docker镜像都是继承自别人修改好的镜像，这里给出的样例继承自`pytorch/pytorch:0.4.1-cuda9-cudnn7-devel`
镜像的基本操作:

```
docker search xxx # 从docker仓库中搜索镜像
docker pull xxx   # 从docker仓库中拉取镜像
docker push xxx   # 将本地镜像push到仓库中
docker image ls   # 显示本地所有镜像
docker commit     # 从容器创建镜像
docker image rm xxx          # 删除本地某个镜像
docker save xxx > xxx.tar    # 将镜像保存成tar包
docker load --input xxx.tar # 从tar包读取镜像
```

容器的基本操作

```
docker create   # 创建容器但不启动它
docker start    # 启动一个容器
docker stop     # 停止容器运行
docker kill     # 杀死容器
docker run      # 创建并启动一个容器
docker rm       # 删除一个容器
docker cp containerID:/xxx/xxx  /yyy/yyy
                # 从容器的xxx路径向宿主机yyy复制文件，反之亦然
docker export   # 将容器整个文件导出为tar包
docker impor    # 从tar包导入容器
```

获取容器信息

```
docker ps     # 显示正在运行的容器
docker ps -a  # 显示所有容器
docker container ls -a  # 显示所有容器
docker logs   # 查看容器日志
docker prot   # 显示容器端口映射
docker top    # 显示容器进程信息
docker diff   # 显示容器文件系统前后变化
```

参赛docker容器标准

假设我们的比赛名字叫OARSeg Challenge
样例的文件结构如下

```
root/
  Dockerfile
  src/
    Segmentation.py
```

输入数据文件结构如下

```
input/
  01/
    data.nii.gz
  02/
    data.nii.gz
  03/
    data.nii.gz
```

输出数据文件结构如下

```
output/
  01/
    predict.nii.gz
  02/
    predict.nii.gz
  03/
    predict.nii.gz
```

不使用GPU的样例

这里展示了一个将python脚本docker容器化的实例。
下面的名为Segmentation.py的python脚本示例使用阈值方法进行分割，参赛队伍将Segmentation_Function替换为自己的分割算法。

Segmentation.py如下

```python
import os
import SimpleITK as sitk

def Segmentation_Function(itkCT):
  seg_result = sitk.BinaryThreshold(
      itkCT, lowerThreshold=300, upperThreshold=1000
  )
  return seg_result
```

```
def predict(input_dir, output_dir, test_id):
    # 读取CT图像
    itk_CT = sitk.ReadImage(os.path.join(input_dir, test_id, 'data.nii.gz'))
    # 运行预测函数
    seg_result = SegmentationFunction(itk_CT)
    # 存储结果
    sitk.WriteImage(
      seg_result, os.path.join(output_dir, test_id,'predict.nii.gz')
    )

def main():
    input_dir = '/input'
    output_dir = '/output'
    os.mkdir(os.path.join(output_dir, '01'))
    os.mkdir(os.path.join(output_dir, '02'))
    os.mkdir(os.path.join(output_dir, '03'))
    predict(input_dir, output_dir, '01')
    predict(input_dir, output_dir, '02')
    predict(input_dir, output_dir, '03')

if __name__ == '__main__':
    main()
```

Dockerfile 如下

- 当前要创建的docker将继承自continuumio/miniconda
- 将src目录及子文件添加到根目录下，并重命名为OARSeg
- 用pip安装numpy和SimpleITK，配置环境

```
FROM continuumio/miniconda
ADD src /OARSeg
RUN pip install numpy SimpleITK
```

补充：关于Dockerfile的使用，详情可以参照https://docs.docker.com/engine/reference/builder/

使用GPU的样例

Segmentation.py如下

```
import os
import torch
import torch.nn as nn
import torch.backends.cudnn as cudnn
import numpy as np
```

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        '''
        网络初始化
        '''
    def forward(self, x):
        '''
        前向传递过程
        '''
        return x

def SegmentationFunction(itkCT):
    # 读取CT图像
    img_arr = sitk.GetArrayFromImage(itkCT).astype(np.float32)
    ori_shape = img_arr.shape
    torch_x = torch.from_numpy(img_arr.reshape((1, 1, *img_arr.shape)))
    # 设定device
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print('device is ', device)
    torch.backends.cudnn.deterministic = True
    cudnn.benchmark = True
    # 构造模型运行前向传递，并返回结果
    model = Net()
    '''
    加载训练好的模型
    '''
    model = model.to(device)
    model.eval()
    with torch.no_grad():
        out = model(torch_x)
    out = out.cpu().data.numpy()
    out = np.asarray(out).astype(np.int16)
    return sitk.GetImageFromArray(out.reshape(ori_shape))

def predict(input_dir, output_dir, test_id):
    # 读取CT图像
    itk_CT = sitk.ReadImage(os.path.join(input_dir, test_id, 'data.nii.gz'))
    # 运行预测函数
    seg_result = SegmentationFunction(itk_CT)
    # 存储结果
    sitk.WriteImage(
      seg_result, os.path.join(output_dir,  test_id,'predict.nii.gz')
    )

def main():
    input_dir = '/input'
    output_dir = '/output'
    os.mkdir(os.path.join(output_dir, '01'))
```

```
    os.mkdir(os.path.join(output_dir, '02'))
    os.mkdir(os.path.join(output_dir, '03'))
    predict(input_dir, output_dir, '01')
    predict(input_dir, output_dir, '02')
    predict(input_dir, output_dir, '03')

if __name__ == '__main__':
    main()
```

Dockerfile 如下

- 当前要创建的docker将继承自pytorch/pytorch:0.4.1-cuda9-cudnn7-devel
- 将src目录及子文件添加到根目录下，并重命名为OARSeg
- 用pip安装cython，配置环境

```
FROM pytorch/pytorch:0.4.1-cuda9-cudnn7-devel
ADD src /OARSeg
RUN pip install cython
```

打包过程

编写好Segmentation.py和Dockerfile后就可以进行打包了，假设运行目录就在Dockerfile目录下，执行以下操作：

- 通过Dockerfile创建出image镜像
  - -f Dockerfile，使用当前的Dockerfile
  - -t oarseg:lihua，镜像名，oarseg为比赛名，lihua是队伍名
  - . 表示打包当前目录
- 将镜像打包成tar包
  - 打包成lihua.tar
- 删除镜像

```
docker build -f Dockerfile -t oarseg:lihua .
docker save oarseg:lihua > lihua.tar
docker image rm oarseg:lihua
```

得到的<队伍名>.tar就是要提交的打包docker文件

我们将如何测试

假设参赛队伍名字叫lihua
测试目录的文件结构如下

```
evaluate.py
lihua.tar
TestData/
  01/
    data.nii.gz
    label.nii.gz
  02/
    data.nii.gz
    label.nii.gz
  03/
    data.nii.gz
    label.nii.gz
Result/
```

测试程序为evaluate.py它将从lihua.tar中加载镜像并运行，预测TestData中的data.nii.gz并将结果输出到Result目录下
evaluate.py的运行逻辑如下：

1. 从lihua.tar中加载镜像

```
docker load --input lihua.tar
```

2. 创建container

- --runtime nvidia表示要用到GPU
- --name lihua_container 将新建的container命名为lihua_container
- -v TestData:/input:ro 表示将TestData以只读方式映射到docker运行环境中的input目录

```
docker run -dit --runtime nvidia --name lihua_container \
  -v TestData:/input:ro -v /output oarseg:lihua
```

3. 运行container，执行Segmentation.py文件

```
docker exec -it lihua_container python /OARSeg/Segmentation.py
```

4. 将结果从container中拷贝到Result

```
docker cp lihua_container:/output Result
mv Result/output/* Result/
```

5. 杀死并删除container

```
docker kill lihua_container
docker container rm lihua_container
```

6. 删除image

```
docker image rm oarseg:lihua
```

运行完以上操作之后，得到的Result的文件结构如下，从而可以调用评估函数
将 `Result/<test_id>/predict.nii.gz` 与 `TestData/<test_id>/label.nii.gz` 两个结果进行比对。

```
Result/
  01/
    predict.nii.gz
  02/
    predict.nii.gz
  03/
    predict.nii.gz
```