Easy-to-Follow Example

This page shows a simple example on how to containerize your python script for this challenge.

Requirements

```
docker
nvidia-docker
```

Settings

Assume that the challenge's name is **StructSeg**. And your working directory is as follows:

```
# In the user directory
./
    Dockerfile
    src/
        Segmentation.py
        ...
```

The input file structure is as follows:

```
# Under the container's running directory
/
    input/
        1/
            data.nii.gz
        2/
            data.nii.gz
        3/
            data.nii.gz
```

The output file structure is as follows:

```
# Under the container's running directory
/
    output/
        1/
            predict.nii.gz
        2/
            predict.nii.gz
        3/
            predict.nii.gz
```

Program example

This code needs a basic Python installation, with numpy, torch, cuda and cudnn packages. We therefore used pytorch/pytorch to inherit.

```
# Segmentation.py
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__()
        '''
        Initialize the net
        '''
    def forward(self, x):
        '''
        Forward
        '''
        return x


def SegmentationFunction(itkCT):
    # Read CT image
    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)))
    # Config device
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print('device is ', device)
    torch.backends.cudnn.deterministic = True
    cudnn.benchmark = True
    # Construct model and predict on the image
    model = Net()
    '''
    Load trained model
    '''
    model = model.to(device)
    model.eval()
    torch_x = torch_x.cuda()
    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):
    # load CT image
    itk_CT = sitk.ReadImage(os.path.join(input_dir, test_id,
'data.nii.gz'))
    # Segment the CT image, return an itk mask
    seg_result = SegmentationFunction(itk_CT)
    #  Write the segmentation mask to output dir
    sitk.WriteImage(
      seg_result, os.path.join(output_dir, test_id, 'predict.nii.gz')
    )


def main():
    input_dir = '/input'
    output_dir = '/output'
    for item in os.listdir(input_dir):
        os.mkdir(os.path.join(output_dir, item))
        predict(input_dir, output_dir, item)


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

Our Python code is saved next to this `Dockerfile` in the directory `src/Segmentation.py`. Our `Dockerfile` looks like this:

```
FROM pytorch/pytorch:0.4.1-cuda9-cudnn7-devel
ADD src /structseg
RUN pip install SimpleITK
```

The command `ADD src /structseg` will recursively copy the `src` in your current directory to the root directory of the docker container `/` and rename it to `structseg`.

Note that the required cuda version is cuda 8, 9 or 10. You can refer to https://docs.docker.com/engine/reference/builder/ for more details about Dockerfile.

Save Your Docker

After fininshing `Segmentation.py` and `Dockerfile`, you can package your docker. Assume the running directory is the directory where Dockerfile resides.

We assume your **username in grand-challenge** is 'lihua', N = 1 | 2 | 3 | 4, indicating task ID from four tasks.

1.Build a Docker container from your Dockerfile

```
docker build -f Dockerfile -t structseg:lihua_taskN .
```

Note: the "." at the end specifies that everything is in the current folder. Hence, you run this build command from the folder that contains the Dockerfile and the source code.

Warning: The docker image should be an operating system(e.g. continuumio/miniconda), not a single python application. One way to check if the image is an operating system is running it through command `docker run -it [image_name]`. If you get a terminal where you can enter commands(such as `ls`), you are successful.

2.Save your docker

```
docker save structseg:lihua_taskN > lihua_taskN.tar
```

All participants should send the organizer this saved file, i.e. lihua_taskN.tar in this context, and then the organizer can run this Docker on our machine to do the evaluation.

How will we test your docker package

Assume your username is lihua, and the task numeber is N.

In our working directory, file structure is as follows:

```
# In the user directory
evaluate.py
lihua_taskN.tar
TestData/
    1/
        data.nii.gz
    2/
        data.nii.gz
    3/
        data.nii.gz
TaskN/
    output/
```

The operation logic of `evaluate.py` is as follows

1. Load docker image from package

```
docker load --input lihua_taskN.tar
```

2. Build a Docker container from image.

Because the segmentation code is run in the docker, we need to map the input folder which contains test data on our machine into the docker. we can run it with the following command:

```
# docker run -dit -v [TEST-DIR]:/input:ro -v /output structseg:[USERNAME]
docker run -dit --net=none -m 32G --runtime nvidia --name
```

```
lihua_taskN_container -v TestData:/input:ro -v /output
structseg:lihua_taskN
```

[TEST-DIR] must be an absolute path. The first -v options map the input folder into the Docker container at /input. The second -v creates an output directory.

3. Run the segmentation script

```
docker exec -it lihua_taskN_container python /structseg/Segmentation.py
```

4. Copy the segmentation result from the Docker container to local machine:

```
# docker cp [CONTAINER-ID]:/output [RESULT-LOCATION]
docker cp lihua_taskN_container:/output [RESULT-LOCATION]
```

5. Kill and delete container

```
docker kill lihua_taskN_container
docker container rm lihua_taskN_container
```

6. Delete image

```
docker image rm structseg:lihua_taskN
```

After running all the above operations, we can get the directory `output` as follows

```
# In the user directory
TaskN/
    output/
        1/
            predict.nii.gz
        2/
            predict.nii.gz
        3/
            predict.nii.gz
```

Then we will run evaluation functions to compare your predicting outcomes with the label files.

Attentions

Before submission, please check the following things:

- The filename is `[username].tar`
- The docker image's name is `digestpath:[username]`
- The `output format` is right

You also need to pay attention to the following：

- Your program is running under `network disconnection`, please check if you used a pre-trained model downloaded from the Internet, or other actions that may connect to the Internet
- Program running cpu memory cannot exceed `32G`
- Program running gpu memory cannot exceed `8G`

All in all, It would be best to run your docker program to check if everything is correct before submission

Here is a video tutorial of docker in DigestPath2019_task2, you can watch it before starting: link-to-video