EN 827102 Pattern Recognition and Object Detection

E002: Familiarize Our Compute Server

Faculty of Engineering, Khon Kaen University

Submission: https://autolab.en.kku.ac.th

==

* Each question or problem is worth 840 points.

* To access the server, you need:

  ** Secure Shell (SSH) tool, e.g., : Putty.

  ** SSH File Transfer (SFTP) tool, e.g., FileZilla.

  ** Note: specify both SSH and SFTP with part 22.

  ** Option: VPN is needed for access from outside KKU.

  ** Account: as delivered in class.

* Server: `mozart.en.kku.ac.th`

==

Q1. Explore server information.

* **Q1.1** Use command `lsb_release -a` to learn about Ubuntu version: what release is the ubuntu running on the server?

* **Q1.2** Use command `lscpu` to learn about the cpus: what is the cpu's model name?

* **Q1.3** Use command `df -H` to learn about the system storage: what is `/dev/sda1` mounted on? And how much available space does it have?

* **Q1.4** Use command `ifconfig` to learn about the server's network connection: what is the IPv4 address of the server's first network card `enp4s0` (hint: look for `inet`)

* **Q1.5** Use command `ps -ef` to learn about the server's running processes: which columns show process id, start time, running time, and command that is running? Answer in order. (The UID column is counted as the first column or column 1.)

Write your answers in the following format. Keep the text in `blue`, but edit the **`red bold`** text for your answer.

```
Q1.1. Release = 0.00
Q1.2. Model name = ????
Q1.3. The /dev/sda1 is mounted on ???? and it has ???? available.
Q1.4. Address = ????
Q1.5. Columns ?, ?, ? and ?
```

Q2. Navigate the file system. Log in the server and do the following.

* **Q2.1** Starting from your home directory (`cd ~`**).** Use command `cd ..` to back out one step and then use command `pwd` to see current location in the file system: what is your current location?

* **Q2.2** Use command `ls -alt` to explore the contents of the current location: what is the attribute of `README`? (hint: the attribute is shown in the first field.)

* **Q2.3** Read the content of `README`: what does it say? (There are various utilities, e.g., `more README`.) Have your answer in only one line. If the content is long, write only the first line.

Write your answers in the following format. Keep the text in `blue`, but edit the **`red bold`** text for your answer.

```
Q2.1. location = ????
Q2.2. attribute = ????
Q2.3. README = ????
```

No-grading exercises:

* Try copy (`cp`) the file into your directory.

* Try edit the copied file (e.g., `pico`).

* Create a directory (`mkdir`) and move (`mv`) the file into the directory.

* Remove the file (`rm`).

* Remove the directory (`rmdir`)

Q3. Run a program

* **Q3.1** At your home directory (`cd ~`), start a python interactive session (`python`) and try running the following code:

```
>>> import os
>>> fh = open(os.path.join(os.getcwd(), “q3p1.out”), “w”)
>>> fh.write(os.getcwd())
>>> fh.close()
```

What is the name of the file it creates? (Answer just the bare filename without its path) And, what is the size of the file (in byte)?

* **Q3.2** Create a file, write the following code, save it as `waitforQ.py`.

```
waitforQ.py
import time
import os

if __name__ == '__main__':

    for n in range(30):
        if os.path.exists("./Q"):
            print("Found Q!")
            break
        print('check:', time.ctime())
        time.sleep(10)

    else:
        print("Q has not come!")
```

Then run it: `python waitforQ.py`. The program will keep checking for a file named “Q”, once it found `Q`, it will terminate and print out: when it found `Q`, what does it print out?

Anyhow, to keep it from running forever, it will terminate within 5 minutes regardless.

Hint: push the running process into background (Press `[Ctrl]` and `[Z]`; then type `bg`) and create the file named Q (`touch  Q`) so that the program can be terminated properly.

Write your answers in the following format. Keep the text in `blue`, but edit the **`red bold`** text for your answer.

```
Q3.1. file name = ???? ; size = 0
Q3.2. print out = ????
```

No-grading exercises:

* Repeat Q3.2, but run the program in the background in the beginning with

`python waitforQ.py &`

While it is still running, check out the process (`ps -ef`): what is the process id?

* Finish the process above by creating Q and re-check the process: is it still running?

Q4. Run a program through a docker container.

* **Q4.1** Check versions of base `python`, `numpy`, and `torch` in the host (`mozart`). Also, check if library `apex` has been installed in the host. I.e., run `python Q4.py` on the host.

```
Q4.py
import numpy as np
import torch
import sys


if __name__ == '__main__':

    print("Python version:", sys.version.split()[0])
    print ('Numpy version:', np.__version__)
    print('PyTorch version:', torch.__version__)

    try:
        import apex
        print('Apex is installed.')
    except Exception as e:
        #print('import apex: error =', e)
        print('Apex is not installed.')
```

* **Q4.2** Check versions of base `python`, `numpy`, and `torch` in the docker image `nvidia_ssd`. Also, check if library `apex` has been installed in the image `nvidia_ssd`.

We will run the same program `Q4.py`, but through container `nvidia_ssd`.

4.2.1. Start docker in an interactive mode with mounting point, e.g., suppose `Q4.py` is in `~/work`, we will mount `~/work` to the container mounting point, e.g., `/host`.

I.e., run

```
docker run --rm -it --ipc=host -v ~/work:/host nvidia_ssd
```

Or, if GPU is available (and we need it), we can run

```
docker run --rm -it --gpus=all --ipc=host -v ~/work:/host nvidia_ssd
```

Note: check if GPU is available using either `nvidia-smi` or `nvtop`.

4.2.2. In a container, run `python /host/Q4.py`.

Example

```
Host
$ ls ~/work
Q4.py
```

```
$ docker run --rm -it --ipc=host -v ~/work:/host nvidia_ssd
                          Container (nvidia_ssd)
root@e3775d2461f6:/workspace/ssd# python /host/Q4.py
Python version:                   <blind>
Numpy version:                    <blind>
PyTorch version:                  <blind>
Apex                              <blind>
root@e3775d2461f6:/workspace/ssd# exit
exit
```

Write your answers in the following format. Keep the text in `blue`, but edit the **`red bold`** text for your answer.

```
Q4.1. mozart: python = 0.0.0; numpy = 0.0.0; torch = 0.0.0??+????; apex = no/yes
Q4.2. nvidia_ssd: python = 0.0.0; numpy = 0.0.0; torch = 0.0.0??+????; apex =
no/yes
```

Further study:

* Study a Single-Shot Detector model (SSD) by Liu et al., ECCV 2016, along with implementation by Nvidia:

`https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch`

under folder `detection`.

* If GPU is available, you can try train Nvidia's implementation of SSD: see example 4A or 4B.

Example 4A (train in an interactive mode): user `sandee` has directory `box` in the home directory, run docker in an interactive mode (`-it`) and remove the container after use (`--rm`) with gpu (`--gpus=all`) and shared memory (`--ipc=host`) and mounting host directory `~/box` to container's `/host/box`. Here, since this is just to take a glimpse into how things go, we train the model for only 2 epochs (~30min per epoch), but the default is 65 epochs (expect ~33 hours). The trained weights will be saved to directory `box` on the host.

```
                                  Host
sandee@bach:~$ ls
box  examples.desktop
sandee@bach:~$ docker run --rm -it --gpus=all --ipc=host -v ~/box:/host/box
nvidia_ssd
                          Container (nvidia_ssd)
root@b48a2f010667:/workspace/ssd# ls /host
box
root@b48a2f010667:/workspace/ssd# bash ./examples/SSD300_FP16_1GPU.sh .
./COCO --save /host/box --epochs 2
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth"
to /root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
100%|██████████████████████████████████████████████████████████████|
97.8M/97.8M [00:01<00:00, 64.6MB/s]
DLL 2022-07-28 10:24:00.420611 - PARAMETER dataset path : ./COCO  epochs :
2  batch size : 64  eval batch size : 32  no cuda : False  seed : None
```

```
checkpoint path : None  mode : training  eval on epochs : [21, 31, 37, 42,
48, 53, 59, 64]  lr decay epochs : [43, 54]  learning rate : 0.0026
momentum : 0.9  weight decay : 0.0005  lr warmup : 300  backbone :
resnet50  backbone path : None  num workers : 4  AMP : True  precision :
amp
Using seed = 4374
loading annotations into memory...
Done (t=0.55s)
creating index...
...
<omitted for brevity>
...
saving model...
DLL 2022-07-28 11:24:07.387370 - (1, 3696) model path :
/host/box/epoch_1.pt
DLL 2022-07-28 11:24:07.387495 - () total time : 3562.8619406223297
DLL 2022-07-28 11:24:07.387535 - ()
root@b48a2f010667:/workspace/ssd# ls /host/box
epoch_0.pt  epoch_1.pt
root@b48a2f010667:/workspace/ssd# exit
exit
sandee@bach:~$ ls box
epoch_0.pt  epoch_1.pt
```

These (e.g., `epoch_0.pt`) are trained weights saved at the specified epochs.

Example 4B (train in a script mode): to keep it short, here we also train for only 2 epoch, (Nvidia's default is set to 65). We prepare directory `~/box2` for the trained weights. Note that shell script `SSD300_FP16_1GPU.sh` along with other codes and data is already in the container, c.f., command we use in example 4A.

```
Host
sandee@bach:~$ mkdir ~/box2
sandee@bach:~$ nohup docker run --rm --gpus=all --ipc=host -v ~/box2:/host/box2
nvidia_ssd bash ./examples/SSD300_FP16_1GPU.sh . ./COCO --save /host/box2 --epochs 2 &
 [4] 109404
sandee@bach:~$ nohup: ignoring input and appending output to '/home1/sandee/nohup.out'
sandee@bach:~/box/ssd$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Jul13 ?        00:00:28 /sbin/init
root           2       0  0 Jul13 ?        00:00:00 [kthreadd]
root           3       2  0 Jul13 ?        00:00:00 [rcu_gp]
...
<omitted for brevity>
...
sandee    109404  105022  0 21:02 pts/1    00:00:00 docker run --rm --gpus=all --
ipc=host -v ~/box2:/h
...
<omitted for brevity>
...
sandee    109751  105022  0 21:05 pts/1    00:00:00 ps -ef
```

Tips: (1) recall that the trained weights will be saved in `~/box2`; (2) it is a good idea to memorize the process id, e.g., ***109404***, of the running program; (3) the print out will be re-directed to, e.g., */home1/sandee/nohup.out* as specified and we can check the progress, e.g, `tail /home1/sandee/nohup.out`.

Note you don't have to do both interactive training (4A) and batch training (4B). You can choose either way.

Once it is done, the trained weights are ready to use. You can load the weights and test or use the model as shown in Example 4C.

Example 4C (load trained weights and test or use the model): suppose the inference code (`nvidiaSSD_inference.py`) in `~/work/code`, trained weight (e.g., `epoch_64.pt`) in `~/work/weight`, input images in `~/work/input` on the host, these folder along with `~/work/output` prepared for saving the detection output will be mounted to the container. We will call the inference code with the interactive session, then reap the detection results from `~/work/output` once the inference is done.

```
Host
sandee@bach:~$ cd work
sandee@bach:~/work$ ls
code  input  output  weight
sandee@bach:~/work$ ls code
nvidiaSSD_inference.py
sandee@bach:~/work$ ls weight
epoch_64.pt
sandee@bach:~/work$ ls input
000000000139.jpg  000000000632.jpg
sandee@bach:~/work$ docker run --rm -it --gpus=all --ipc=host -v ~/work:/host
nvidia_ssd
     Container (nvidia_ssd)
     root@1a3539a6ea14:/workspace/ssd# cp /host/code/nvidiaSSD_inference.py .
     root@1a3539a6ea14:/workspace/ssd# python ./nvidiaSSD_inference.py >
     /host/output/log
     Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth"
     to /root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
     100%|██████████████████████████████████████| 97.8M/97.8M [00:01<00:00,
     64.9MB/s]

     root@1a3539a6ea14:/workspace/ssd# ls /host/output
     log                     pred000000000139.out  pred000000000632.out
     root@1a3539a6ea14:/workspace/ssd# exit
     exit
sandee@bach:~/work$ ls output
log                     pred000000000139.out  pred000000000632.out
```

Example 4D (visually inspect the detection results):

Run `ShowDetectionResults_colab.ipynb` on colab or locally.

(See the attached hand-out.)