Face Detection Service

What is this

Flask based microservice for face detection based on facenet. The service only provide detected face boxes however face verification is also possible with small changes from here.

(The repository contains the facenet as a submodule)

Development environment

This repository is developed with followings.

AWS Instance: p2.xlarge
AMI: Deep Learning Base AMI
Trained model: 20180402-114759.zip (VGGFace2)

How to run the service

1. (Optional) Prepare your AWS environment
2. Set the virtualenv and activate
3. `git clone` this repository
4. Don't forget after cloning `git submodule update --init`
5. `pip3 install -r requirements.txt` for this repository.
6. Make a symbolic link from `myCompare.py` with
   `ln -s ~/FaceDetectionService/myCompare.py ~/FaceDetectionService/facenet/src`
7. Run the server with `python3 myServer.py`
8. (Optional) Try with a simple health check with `curl localhost:5000/ping`
9. Prepare an image in `./photo`
10. From another terminal, POST a photo with `python3 myClient.py your_file.jpg -f` . You may try it from same host or setup SSH port forwarding.
    `ssh -L localhost:5000:localhost:5000 ...`
11. You will receive image size and detected face box with JSON, for example
    `{'message': 'image received. size=760x882'}`
    `{'py/object': 'numpy.ndarray', 'values': [208, 130, 522, 547], 'dtype': 'int32'}`

How to run the service by Docker

1. (Optional) Prepare your AWS environment

2. Pull the image with `docker pull tensorflow/tensorflow:latest-gpu`
3. Build an image `docker build -t face/detect:1.0 .`
4. Run the image `nvidia-docker run --rm -p 5000:5000/tcp face/detect:1.0`
5. Prepare an image in `./photo`
6. From another terminal, POST a photo with `python3 myClient.py your_file.jpg -f` . You may try it from same host or setup SSH port forwarding.
   `ssh -L localhost:5000:localhost:5000 ...`
7. You will receive image size and detected face box with JSON, for example
   `{'message': 'image received. size=760x882'}`
   `{'py/object': 'numpy.ndarray', 'values': [208, 130, 522, 547], 'dtype': 'int32'}`

Primitive performance test

1. Latency

1.1. Network vs Server side GPU processing

Task	Time
Size of image	0.9729[sec]
Face detection	36.2582[sec]

`client_10.py` tests two services at 10 times. Request went though SSH tunnelling thus it was more realistic environment compared to just localhost only.
Simple processing return size of image. Face detection with DNN is compared with the simple task to see network latency. Regarding the test result, DNN processing is main bottleneck since network latency can be 0.1 sec at maximum.

1.2. GPU processing in server

Task	Time
Create NN	2.9040[sec]
np.asarray	0.0000[sec]
Detect	0.2521[sec]
np.squeeze	0.0000[sec]

Measured all process for face detection. Bottleneck is network creation. Second biggest one was detection.

1.3. Conclusion (Improving plan)

1. Although Flask is state less server, DNN should be initialise with microservice server since loading network parameter to GPU is the biggest time consuming part. To be honest, I knew that but couldn't take time to integrate server and facenet enough.

2. Then main part to tackle for lower latency is face detection. According to implementation, minimum size of face and factor for scaling pyramid is configurable parameters for performance improvement. However recognition rate might be go down with tweaking.

3. It is good to review the implementation with the original paper. There might be chance to rase performance.

4. Higher spec instance might be help to improve performance but it is depends on the budget and requirements.

2. Throughput

This is not tested but multiple instances with load balancer raise throughput generally.

3. Concurrency

Most probably network level concurrency improve performance which is mentioned in previous section. More lower level implementation may not have chance to work for higher performance since most of the libraries (CUDA and Tensorflow) are well implemented and optimised. However it is worth to review how the face detection is implemented with original paper.

CLI client

The client send an image file to GPU server.
`python3 myClient.py your_file.jpg -f`
-f is face detection option. If you execute without -f option then the server just reply size of an image.

An idea for deployment

Jenkins based solution might one of the best for automation. In the solution Kubernetes and Ansible will play important role. Following link seems worth to read. https://medium.com/containerum/configuring-ci-cd-on-kubernetes-with-jenkins-89eab7234270

TODO

- Toward production
  - Unit test
  - WSGI server

- Error handling
- Logging
- Security test

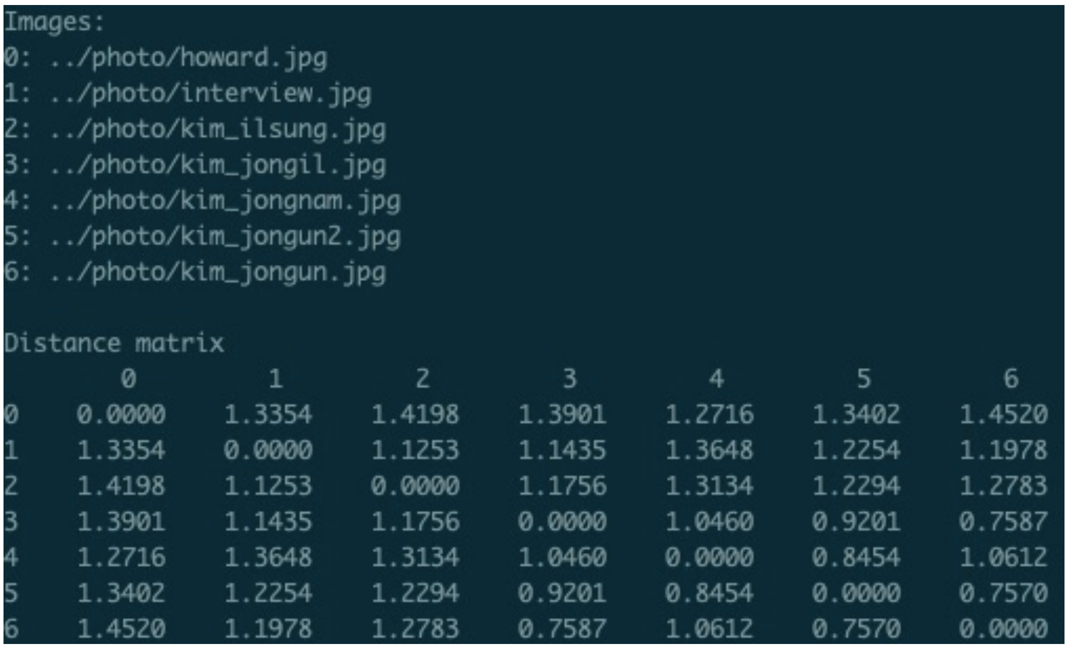
Good to know

facenet

facenet repository contains a lot of interesting scripts. You can compare similar face photos or family members with following command.

`python3 src/compare.py 20180402-114759 ../photo/* --gpu_memory_fraction 0.9`

Then you will get L2 distance matrix.



```
Images:
0: ../photo/howard.jpg
1: ../photo/interview.jpg
2: ../photo/kim_ilsung.jpg
3: ../photo/kim_jongil.jpg
4: ../photo/kim_jongnam.jpg
5: ../photo/kim_jongun2.jpg
6: ../photo/kim_jongun.jpg

Distance matrix
        0         1         2         3         4         5         6
0    0.0000    1.3354    1.4198    1.3901    1.2716    1.3402    1.4520
1    1.3354    0.0000    1.1253    1.1435    1.3648    1.2254    1.1978
2    1.4198    1.1253    0.0000    1.1756    1.3134    1.2294    1.2783
3    1.3901    1.1435    1.1756    0.0000    1.0460    0.9201    0.7587
4    1.2716    1.3648    1.3134    1.0460    0.0000    0.8454    1.0612
5    1.3402    1.2254    1.2294    0.9201    0.8454    0.0000    0.7570
6    1.4520    1.1978    1.2783    0.7587    1.0612    0.7570    0.0000
```

Useful tool (tmux)

When you use AWS, tmux enable multiple windows with a SSH session. It is good to see how do you consume GPU memory with `nvidia-sml -l` meanwhile checking main machine status with `top` or `htop` with fancy colour bars.

For splitting the window, `ctr+b %` or `"` is shortcut then you can move `ctr+b arrow-key`. For close a window, `ctr+b x`. Please note that you can't copy lines with mouse easily if it goes two rows. For more information, please refer following link => https://lukaszwrobel.pl/blog/tmux-tutorial-split-terminal-windows-easily/

Traps and Pitfalls?

Wasted time with those two.

Nameko

When you Google `python microservice` then `Nameko` might come up. It seems similar to `Flask` since both of them are based on Werkzeug. Regarding GitHub information, Flask have bigger community therefore you can get better support and find more materials easily.

Anaconda

There are some AMI with Anaconda but it might be mainly for research or study purpose. Some of Python packages are not supported by Anaconda then you need use `pip` for that. It is not impossible but it is safe to avoid such mixture. Please also pay attention when you work with OpenCV especially when you manually compile it. Anaconda may harm your installation. Finally Anaconda consume disk space around 25GB more. Therefore if your goal is not quick access to Jupyter Notebook, then basic AMI might be a wise choice.

References

Flask sample code (tranfser image)

https://gist.github.com/kylehounslow/767fb72fde2ebdd010a0bf4242371594

How to setup AWS Deep learning AMI

https://aws.amazon.com/blogs/machine-learning/get-started-with-deep-learning-using-the-aws-deep-learning-ami/

GPU supported Docker image from Tensorflow

https://www.tensorflow.org/install/docker#gpu_support