COMPSCI 532 Project 2 – Design Document

Yen-Sung Chen

1. **Program design**

- About the backend server

  The backend server is developed using Python with Flask. It is a simple HTTP server which has only one function to handle incoming requests. For classifying animals in images, a pre-trained Densenet-121 PyTorch model is used. The classification model is executed in the backend server. The server configuration file, the pre-trained Densenet-121 model, and the json file for mapping predicted index to corresponding class are loaded before the server starts.

- About the Docker image

  The Python:3.7.9-buster is used as the base image. The source code of backend server, the server configuration file, the json file for class mapping, and a python script for downloading pre-trained Densenet-121 model are copied into the base image. After that, commands for installing flask, pytorch and Densenet-121 model are executed. Eventually, a Docker image named “532_project2:v1” would be built out.

2. How it works

The backend server first loads the server configuration code, the pre-trained Densenet-121 model, and the json file for class mapping, then it starts serving HTTP requests. Incoming requests would be handled by function *classify()*, which reads the uploaded image and send it to function *predict()*. The *predict()* function will transform the image into a tensor by image resizing, image cropping, and normalizing etc. Then, the tensor is fed into the model for classification. The Densenet-121 model would output a predicted index, that we can use it to check the corresponding name of predicted animal by using “imagenet_class_index.json”. The result is returned to function *classify()*, and the function would further generate a json response and send it back to clients.

3. Design tradeoffs

- It is fast in response and simple to maintain since the pre-trained Densenet-121 model is executed inside the backend server.
- Loading the model in the backend server consumes the memory space of the server. However, it should be able to cover most scenario of this project.

- If there are millions of HTTP requests to be handled, we might need some dedicated servers for running the model, so that the server can handle more requests by load balancing.

4. Running the program

By executing "run.sh", you can run the whole program. But before executing the script, make sure that below variables in the script are properly set:

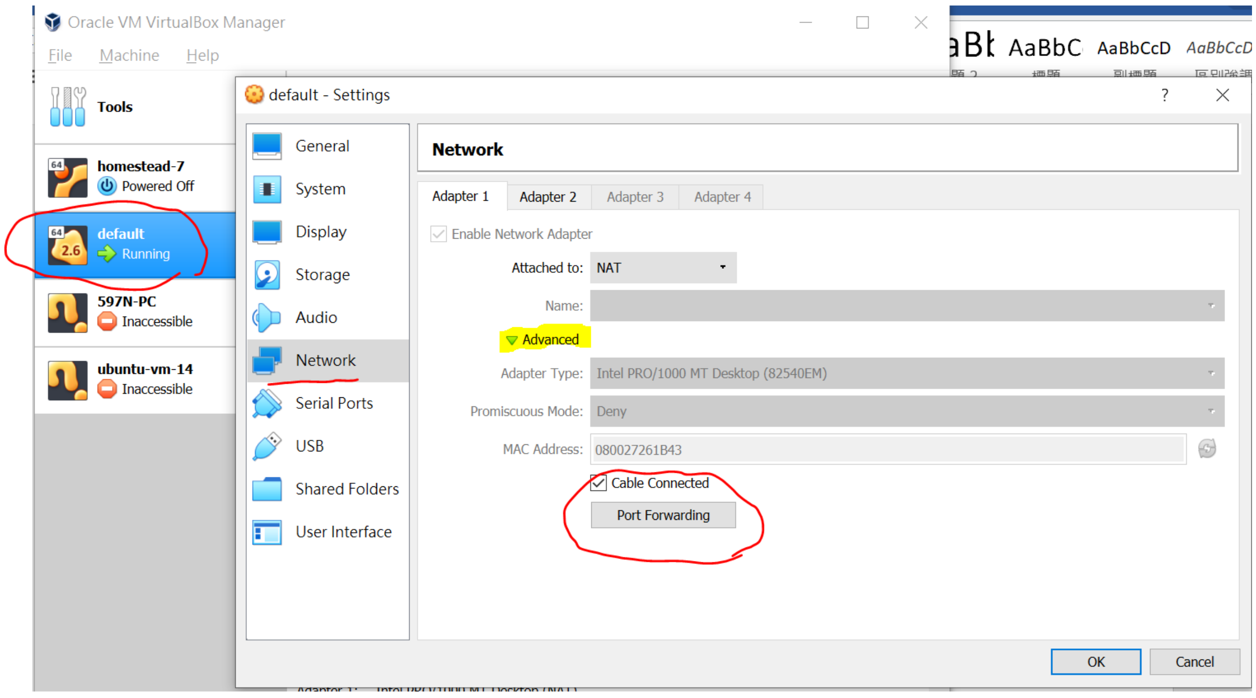
IMG_NAME: the name of the Docker image. By default, it is "cs532_project2:v1".

CMD: the path of Docker command. By default, it is "docker.exe" since my OS environment is Windows. You might need to change it to "docker" under Linux.

SERVER_ADDR: the URL of the backend server. By default, it is "http://localhost:5000". Make sure that this address is the same as the values in "config.json". Also, make sure that the address and port are available.

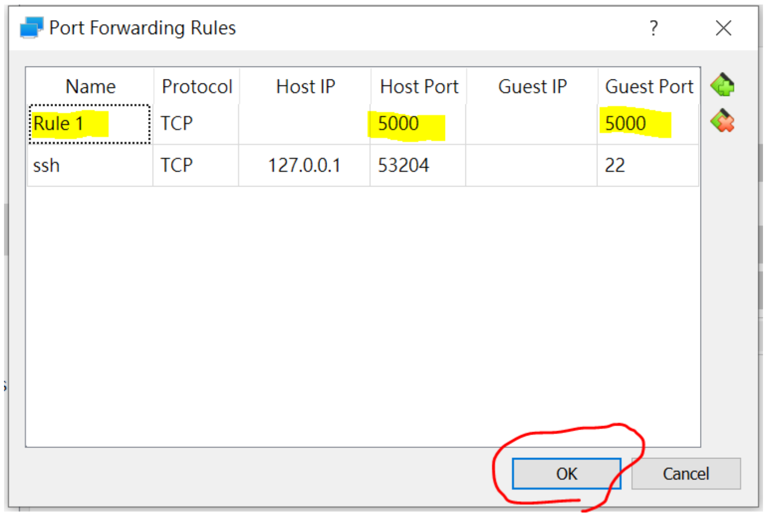
It is important to mention that my OS doesn't have native Docker support. It runs Docker by using "Docker Toolbox for Windows", which initiates a virtual machine for running Docker daemon. Docker containers are also run on the virtual machine. Therefore, you might need to enable port forwarding so that the backend server in the container can receive HTTP requests from local host machine. Please follow below instructions if you have a similar environment:

Open the Docker VM settings in VirtualBox → Select Network tab → Click "Port Forwarding":



Enable Network Adapter
Attached to: NAT
Name:
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Deny
MAC Address: 080027261B43
Cable Connected

Set a port forwarding rule. Make sure that the port value here is the same as the port value in config.json:



Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1	TCP		5000		5000
ssh	TCP	127.0.0.1	53204		22

Click "OK" button after setup. Your container should be able to receive requests from your host machine.

The script contains 3 parts:

a. Docker image building

   The script will build a Docker image on your machine first according to the Dockerfile under the same folder.

b. Docker container initializing

   After the Docker image is built, the script will create a container based on the previously built image. As the container is brought up, it will start the backend server. By default, the container binds to your localhost on your machine. You should be able to connect to the backend server in the container via localhost.

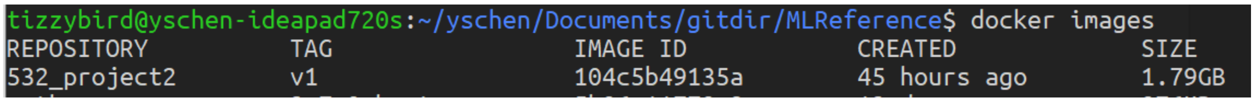
c. Testing

   The final step is testing the whole inference serving system. Test images should be put under "data/" folder. The script will loop over the folder and send those images to the backend server by using "curl" command to generate POST requests. If success, a json response with class_id and class_name should be returned for each image.

After making sure above steps are set correctly, you can run the program by executing "run.sh". Comment out corresponding lines if you want to skip some steps.

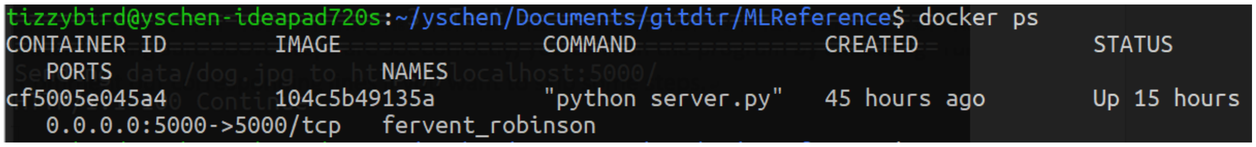
5. Test output

Docker image:



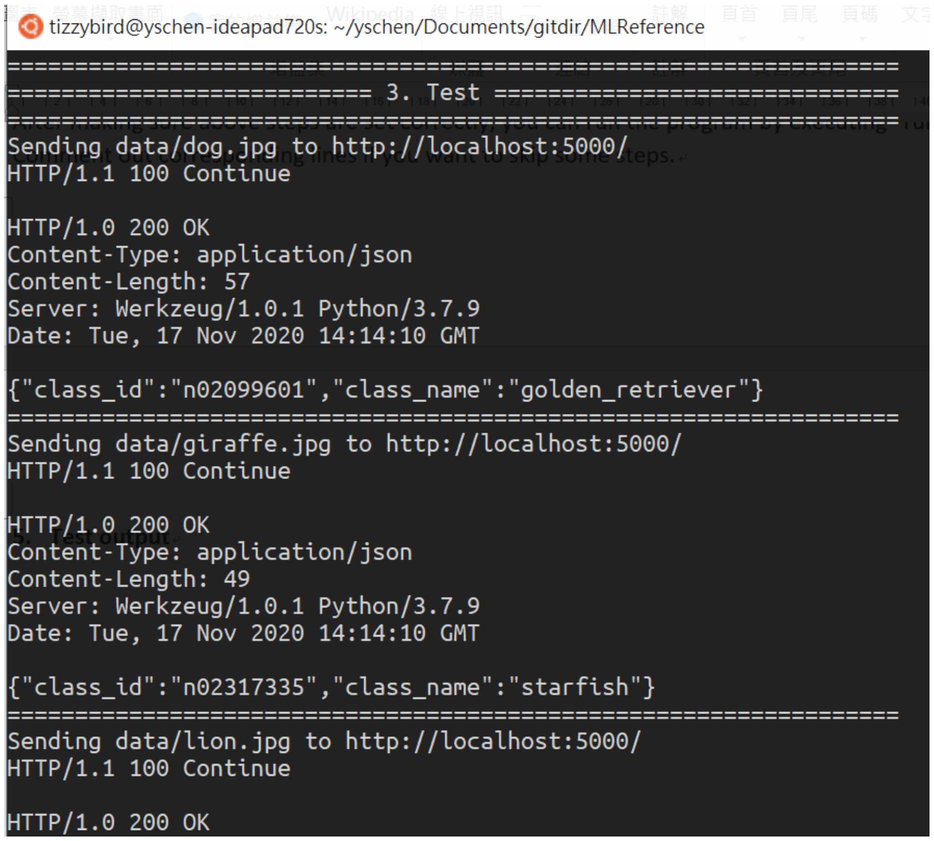
```
tizzybird@yschen-ideapad720s:~/yschen/Documents/gitdir/MLReference$ docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
532_project2         v1                  104c5b49135a        45 hours ago        1.79GB
```

Container of backend server:



```
tizzybird@yschen-ideapad720s:~/yschen/Documents/gitdir/MLReference$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS                    NAMES
cf5005e045a4        104c5b49135a        "python server.py"   45 hours ago        Up 15 hours
   0.0.0.0:5000->5000/tcp   fervent_robinson
```

Test outputs:



tizzybird@yschen-ideapad720s: ~/yschen/Documents/gitdir/MLReference

```
==================================================================
========================== 3. Test ===============================
==================================================================
Sending data/dog.jpg to http://localhost:5000/
HTTP/1.1 100 Continue

HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 57
Server: Werkzeug/1.0.1 Python/3.7.9
Date: Tue, 17 Nov 2020 14:14:10 GMT

{"class_id":"n02099601","class_name":"golden_retriever"}
==================================================================
Sending data/giraffe.jpg to http://localhost:5000/
HTTP/1.1 100 Continue

HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 49
Server: Werkzeug/1.0.1 Python/3.7.9
Date: Tue, 17 Nov 2020 14:14:10 GMT

{"class_id":"n02317335","class_name":"starfish"}
==================================================================
Sending data/lion.jpg to http://localhost:5000/
HTTP/1.1 100 Continue

HTTP/1.0 200 OK
```

6. Project link

Github: https://github.com/tizzybird/CS532-Project2

Download built image: docker pull tizzybird/cs532_project2:v1