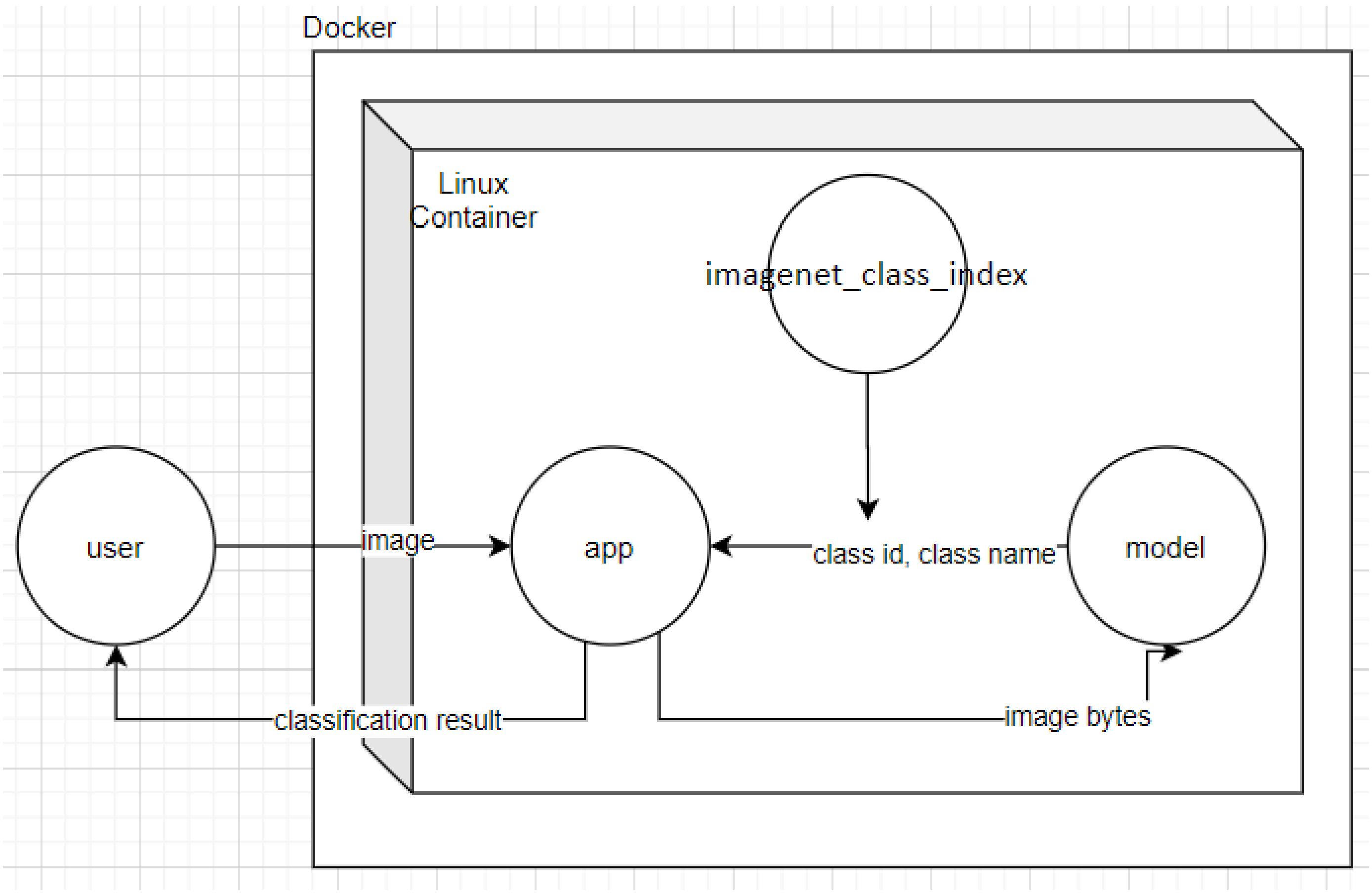
System Design



- *Linux container:* the container with all dependencies installed that runs the flask server
- *user:* The person that sends the inference request
- *app:* App represent a flask server
- *model:* model represent a Pytorch densnet121 model
- imagenet_class_index: The mapping of ImageNet class id to ImageNet class name so that we can have a human readable class name.

How it Works

Docker image

I define a docker file and build images using command: `docker build -t myimage .` Dockerfile contains the following important component:

Install Dependency

- Install all Pytorch related decencies like Cuda, Nvdia Driver, Pytorch, torchvision, …
- Install all Python-related decencies like Flask by defining request.txt and invoke command like: RUN pip install -r requirements.txt

Copy files to work directory

- Copy all the source code and imagenet_class_index to the work directory so that we can use them

Default command

- Let container to execute the server when started. I used command likes: `CMD [ "python3", "server.py" ]`

Container

We link the server's port in the container to our local host port. By default, flask listen to 5000, so we link port 5000 in the container to our local host port 5000. I used command likes:

```
docker run -d -p 5000:5000 myimage
```

ML Inference Server

Request stage:

- User will send a http request to the server with image files (Either through GET or POST method)

Inference Stage:

- The server will iterate over each image sent by use. For each image, the server transforms the image bytes into tensor and pass the tensor into the Pytorch model.
- The model output the class id and lookup imagenet_class_index to get human-readable class name for the image
- The server record the classification as following JSON format:

  {'image_name': {'class_id': class_id, 'class_name': class_name}}

Response Stage:

- The server finished processing all images sent by use, return the classification result to the user

Tradeoffs

Flask Server Framework v.s. Default Server Framework

I finally decide using flask framework as the server implementation since Flask framework provides powerful server and Restful API infrastructure and allows me to focus on high-level design.

Pros: Allow user to focus on high-level API logic. Save effort implementing server infrastructure.

Cons: Need to understand the complicated flask framework and API documentation. At the same time it loses some flexibility on implementation.

Linux Container v.s. Windows/Mac Container

I finally decide using a Linux Container since it provides better support to PyTorch-GPU speedup (Ex. CUDA) for PyTorch and GPU speedup is critical for ML application performance.

Pros: Better support for PyTorch-GPU speedup

Cons: Mac/Windows users need to re-study how to install all Pytoch/Python decencies using Linux command. Also, they will need to install some Linux kerel simulation program for the Docker to run Linux container.

JSON Response Format of Json v.s. Self-Defined Response Format

I decide to use JSON as response format to user since JSON is a widely-used format, allowing user to easily understand the format.

Pros: Widely used format. Easy to read.

Cons: May not be the most efficient data structure with regards of performance and memory

Using latest version of libraries v.s Using stable version of libraries

I finally decide to use older/stable version for Pytorch/Flask and other Python dependencies since I want to avoid any potential unfounded bugs and want my system to be stable.

Pros: Older version of library is usually much stable and have less problem

Cons: May lose some new feature or performance improvement made in newer version

Validation

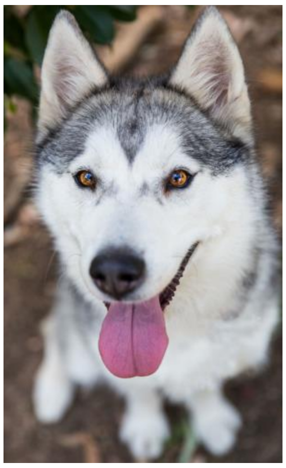
I used 5 images for testing and each of one is correctly classified by the inference model. Testing command as follows:

Run docker image: `docker run -d -p 5000:5000 myimage`

Request inference: `curl -X POST -F file=@dog1.jpg -F file=@dog2.jpg -F file=@cat1.jpg -F file=@cat2.jpg -F file=@squirrel1.jpg http://localhost:5000/predict`

Dog1.jpg

Dog2.jpg



Cat1.jpg

Cat2.jpg

Squirrel1.jpg

Classification Result:

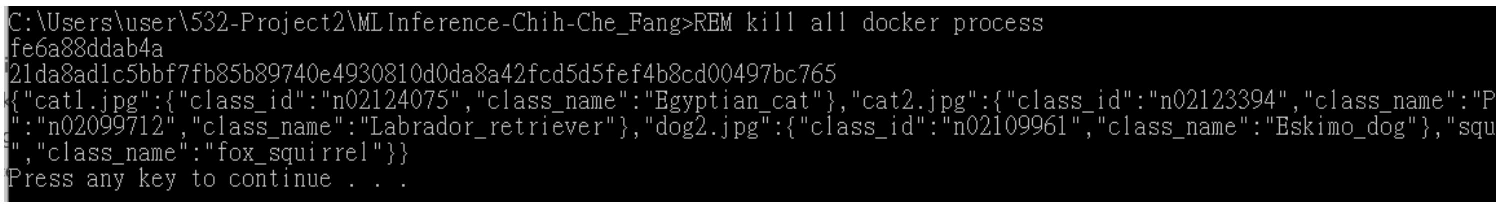
{"cat1.jpg":{"class_id":"n02124075","class_name":"Egyptian_cat"},

"cat2.jpg":{"class_id":"n02123394","class_name":"Persian_cat"},

"dog1.jpg":{"class_id":"n02099712","class_name":"Labrador_retriever"},

"dog2.jpg":{"class_id":"n02109961","class_name":"Eskimo_dog"},

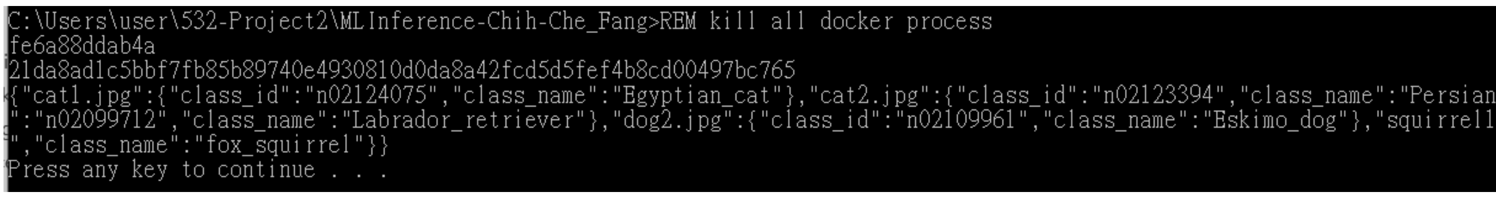
"squirrel1.jpg":{"class_id":"n02356798","class_name":"fox_squirrel"}}



```
C:\Users\user\532-Project2\MLInference-Chih-Che_Fang>REM kill all docker process
fe6a88ddab4a
21da8ad1c5bbf7fb85b89740e4930810d0da8a42fcd5d5fef4b8cd00497bc765
{"cat1.jpg":{"class_id":"n02124075","class_name":"Egyptian_cat"},"cat2.jpg":{"class_id":"n02123394","class_name":"P
":"n02099712","class_name":"Labrador_retriever"},"dog2.jpg":{"class_id":"n02109961","class_name":"Eskimo_dog"},"squ
","class_name":"fox_squirrel"}}
Press any key to continue . . .
```

How to Run the Program

1. Change to the root directory of this project
2. Execute run_test.bat, it will automatically run the docker image and perform a http request to the server with image files. Explanation for the command executed:
   - Run docker image: `docker run -d -p 5000:5000 myimage`
   - Request inference: `curl -X POST -F file=@dog1.jpg -F file=@dog2.jpg -F file=@cat1.jpg -F file=@cat2.jpg -F file=@squirrel1.jpg http://localhost:5000/predict`
3. See the classification results of all images on consle (This server support multiple images in one single http request). Here is one example:



```
C:\Users\user\532-Project2\MLInference-Chih-Che_Fang>REM kill all docker process
fe6a88ddab4a
21da8ad1c5bbf7fb85b89740e4930810d0da8a42fcd5d5fef4b8cd00497bc765
{"cat1.jpg":{"class_id":"n02124075","class_name":"Egyptian_cat"},"cat2.jpg":{"class_id":"n02123394","class_name":"Persian
":"n02099712","class_name":"Labrador_retriever"},"dog2.jpg":{"class_id":"n02109961","class_name":"Eskimo_dog"},"squirrel1
","class_name":"fox_squirrel"}}
Press any key to continue . . .
```

Project Directory

- *Src:* all the source code of this project
- *Docs:* contains all the documentation
- *Run_test.bat:* the testing script
- *Dockerfile:* docker file used to build the docker image
- *requirements.txt:* record all python dependencies for the docker image. It is used by docker file to install all the Python dependencies.
- *local_debug.bat:* testing script for locally debugging purpose
- *imagenet_class_index.json:* The mapping of ImageNet class id to ImageNet class name (Used by inference model to interpret classification result).
- **.jpg:* testing jpeg