Design Documentation

Team 24 : Jiye Choi, Sora Ryu

Overall Program Design

This program shows model serving with PyTorch and Docker, and performs inference efficiently. This program is consisted of a container which holds the framework that is PyTorch and a Python web framework Flask application which implements an image classification server. For a classification model, we use a pre-trained Densenet-121 model for PyTorch.

The server is included in a docker container and it is exposed as REST API which will be accessed over HTTP requests. The python Flask is used to expose the API through web interface. Simple HTTP resource called ‘upload’ is defined in the code to accept the image request from the client side. Also, since the host’s port 5000 is mapped into the container’s port 5000 while running the container, the port number 5000 needs to be specified when running the flask application in the container.

How It Works

On the client side, a client will see ‘Server is Running’ by using the command line ‘curl localhost:5000’. To send a request for classification, the user should use the command line shown below.

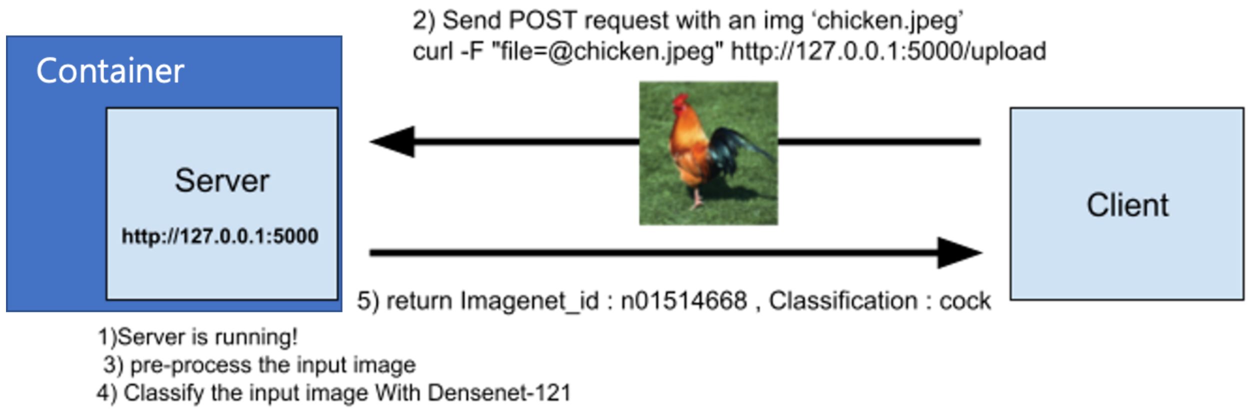
```
curl -F "file=@[your test image file]" http://127.0.0.1:5000/upload
```

On the server side, when the server gets a request from the client, the server finds the input image with the input path, and pre-processes on it. After all, the classification model, Densenet-121, predicts which category the image belongs to in a category list. The category list is obtained by json.load() with json file ‘imagenet_class_index.json’ in the project folder. Then, the server returns the output including imagenet_id and a classification label for the image.

The overall process of this program is as follows

1. Running the Server application
2. Getting a request from a client with an image
3. Pre-processing the input image
4. Predicting which category the image belongs to
5. Return the output

An example process is shown below.


2) Send POST request with an img 'chicken.jpeg'
curl -F "file=@chicken.jpeg" http://127.0.0.1:5000/upload
Container
Server
http://127.0.0.1:5000
Client
5) return Imagenet_id : n01514668 , Classification : cock
1)Server is running!
3) pre-process the input image
4) Classify the input image With Densenet-121

Design Tradeoffs

This program is a simple classification server. To keep this simple, we did not work on any front-end side. It only works with a curl command. In addition, in order to predict a category for an input image, this program uses the category list from 'imagenet_class_index.json', which has 999 distinct categories. Thus, if the input image category does not exist in the list, it will not be able to predict correctly.

There was some tradeoffs while creating a docker container. The information, such as which base image is going to be used and which dependencies are required to be installed, depends the docker image and container size. At first, when we built our first docker image, its size was too big. So, we tried to minimize the size by removing unnecessary dependencies and changing the base image into Alpine which has a pragmatic balance with small size. However, Alpine didn't work to find and install right version of torch. Thus, we decided to use the base image as a python and install only must-have dependencies such as Flask, torch and torchvision.

How to run the program

The list of step is clearly described on the docker container documentation and README file. Please refer to those documentations.