CS532: Project 2: ML Inference Design Document

Introduction

The goal of this project is to implement a basic version of a pre-trained model based ML Inference Server. We have used Docker and Pytorch documentation as the basis to build this server. Server accepts image file paths as HTTP ‘POST’ requests and provides the classification tag for that image. We have provided a folder with images to test the server and also have provided an image of the server we used to test the pre-trained model.

Description (Program Design)

Model Information :

As per project directions, the model is a pre-trained neural network based machine learning model. The architecture of the model is known as Densenet-121 and it is a convolutional network based model. PyTorch Library ships a pretrained version of this model along with a JSON file to match the model tags to real world classes.

Application Side:

IO Argument Handling

Front-end:
The system accepts HTTP ‘POST’ requests. We have used the curl command available in Windows and Linux OS to interact with the server.

The curl command contains the filepath of the image and the server location as it’s arguments. The server location contains the server address on the location system, the port and the function it wishes to invoke at the server level.

Back-end:
At the backend, we handle http requests using a python library known as flask. The library has an inbuilt routing methodology which allows us to assign invocation paths to various functions when they are called externally. Based on the path provided in the HTTP request, if the path exists and points to a function in the application running on the server side, the function is invoked and the output of the function is returned at the Command Line Interface. (CLI)

Server Side

Program and Execution

The program to find a classification for input image has the following steps :

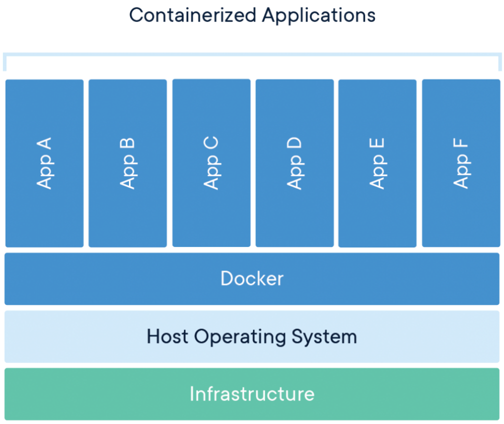
1. Request handling and function Invocation : The appropriate function is invoked by the HTTP request. This is accomplished using the python library flask. flask helps handle the program code as an application and has functionality to allow routing of request to appropriate functions based on invoked function name and acceptable HTTP methods.
2. Preprocessing : With the help of PIL library and PyTorch preprocessing tools, the input image is converted into a 3d matrix with 3 channels for the RGB color values and then normalized.
3. Prediction and Mapping : The pre-trained model is set to evaluation mode and the preprocessed matrix is passed onto the trained model for a label. The model label is matched with a predetermined mapping to produce the final label.
4. Output : The final label is output to the Application Side CLI.

The program was built with the following as reference :
https://pytorch.org/tutorials/intermediate/flask_rest_api_tutorial.html

Portability

As per project requirements and for ease of reproducibility of the server, the server is packaged into a container using Docker. Docker provides a ‘ship your environment’ functionality, allowing us to package/containerize our application and its prerequisites. This provides us the functionality to use and host our server cross platforms through the docker software.

Docker essentially works similar to a hypervisor layer above the Host OS but without the need of running a completely new OS image. It runs with usability of a hypervisor but the portability of a JVM.



To create a containerized image of our server application a Dockerfile is required.

Dockerfile:
A Dockerfile is a text file that contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format.

A Dockerfile is built into an image using the docker build command. This built image is then initialized with a docker run command.

The docker run command initializes the docker container and runs the application specified in the Dockerfile.

Design Trade Offs:

1. The server only accepts http ‘POST’ requests. They are usually considered a little safer than ‘GET’ request which while necessary in our project is good practice. ‘POST’ requests also allow the user to upload files to the server while ‘GET’ requires actionable URLs being attached.
2. The current application is unable to accept direct URLs to images on the internet and can only accept images uploaded to the server through the ‘POST’ comment.
3. The image preprocessing and transformation is hardcoded in the system independent of the image type. So if complex 3d images are sent to the server, the system may just access a part of it or be unable to reply with a classification.
4. The system can only accept images less than 20GB.

Execution:

To build the Docker Image please run the following command:
(The image will be created from the Dockerfile mentioned)

```
docker build -t <Docker Image Name>:<Version>
```

To create a Docker Container, run the following:

```
docker run -p 5000:5000 <Docker Image Name>:<Version>
```

The command above will create a Docker Container at localhost port 5000.
To send an image to the flask application in the container, use the following command (preferably :

```
curl -F "file=@<file name.jpg>" http://127.0.0.1:5000/upload
```

To download and use image directly (This image has been made using Linux Containers) :

Download Link:
https://drive.google.com/file/d/1XrGhPwX044Ir8v2IpIu33eRFO-0uWm5i/view?usp=sharing

```
docker load < test_image.img
```

To initialize the container at localhost and pinging the server, the 'docker run' and 'curl' commands from above can be used.