# Project 2: ML Inference - Design Doc

**By:**
**Parin Rajesh Jhaveri 32601659**
**Tejas Chheda - 32673487**
**Sharmistha Gupta - 32698753**

## Introduction:

The objective of this project is to implement an image classification server and containerize it using Docker. We will be using PyTorch as the machine learning framework to use the pre-trained Densenet-121 model for inference. The server will be built using the Flask framework. Both of these components will be containerized in a Docker container.

## Design Description:

**HTTP SERVER:**

- As mentioned in the introduction above we have used the Flask framework to build the HTTP Server which is capable of handling POST as well as GET requests.
- The Server will be running on the port number 5000.
- We have constructed an API endpoint called */inference* which takes the image filename as the parameter and then invokes a method called *infer* in the *ml_service.py* file, which then loads the image from the */images* directory and the Densenet-121 model for classifying the image.
- The class that was predicted to the image is then returned as part of the response body for the */inference* API.
- We have also included a default path (“/”), which can be used to detect whether the Flask Server is running or not.

**MODEL INFERENCE:**

- As mentioned in the introduction we have used the machine learning framework PyTorch to load the pre-trained Densenet-121 model for inference.
- Below are the steps for model inference:
    1. Load the Densenet-121 model in eval mode
    2. Load image and perform transformation and preprocessing steps
    3. Use softmax to get prediction probabilities from the Densenet-121 model
    4. Map the highest probability to corresponding imagenet class and return the result

**DOCKER CONTAINERIZATION:**

- We are containerizing this model inference server in a Docker container.
- To be able to build this Docker container we first designed a Dockerfile that would build an image when executed. This Docker image is then leveraged to deploy the Docker Container when run.

- We used *python:3.7-slim* as the base image as it is lightweight as compared to *Buster* and more compatible than the *alpine* version.
- We have made */app* our working directory and copied our project there as well as the images directory
- We then install Flask and the CPU version of PyTorch.
- We expose the 5000 port on Docker as the HTTP Server runs on that port and to be able to accept requests from outside the container we need to expose that particular port
- The command that will be triggered when the docker container is *python3 main.py*. This will start the HTTP Server on port 5000.
- The docker shell script *run_docker.sh* contains the commands required to build the Dockerfile and then run the container with the image that is built - *flask-docker-base-app.*

**ERROR HANDLING**

- If the image file name that is passed as a parameter in the curl command doesn't have that image in the images folder then an exception will be thrown and caught by our implementation. We will then return the statement that the image should be downloaded into the */images* directory.

## Design Tradeoff:

**PLACING IMAGE DIRECTORY IN DOCKER CONTAINER:**
We placed the images directory in the docker container which if it contains a lot of images can make the docker image size very large and bulky. Another option would be to in a real world scenario would be to create a mount for these images.

**USING PYTHON3.7-SLIM AS BASE IMAGE:**
We chose to use Python3.7-Slim as base image over the *Buster* and *Alpine* counterparts because it is lightweight as compared to *Buster* which has more dependencies installed, and the *Alpine* image has a few compatibility issues.

## How to Execute:

Steps to create and run docker containers (ensure that the docker daemon is running):

1. Run the shell script `run_docker.sh` using the command `sh run_docker.sh`
2. Check to see if server is up and running at `0.0.0.0:5000/` or `localhost:5000/`

Steps to check inference on images (once docker is running and previous step is executed):
1. Download the image to run inference on (eg. dog.jpg) and store it in the */images* directory
2. Run the command `curl http://localhost:5000/inference/dog.jpg` to get the prediction (class of object in the image)