Design Document - Image Classification

Disha Singh | Pranjali Ajay Parse

---

1. Workflow of the Project

The steps in the workflow are as follows:

- Train the machine learning model on a local system
- Wrap the inference logic into an application using Flask
- Containerize the flask application using docker

2. Project Overview

a) Train the machine learning model on a local system

We use the densenet model in pytorch to train our machine learning model for object detection. Dense Convolutional Network (DenseNet) is a feed-forward network that connects each layer to every other layer. The network has L(L+1)/2 direct connections, whereas standard convolutional networks with L layers have L connections - one between each layer and its subsequent layer. DenseNets have their own set of benefits: they solve the vanishing-gradient problem, improve feature propagation, promote feature reuse, and reduce the number of parameters significantly.

After training, the model file is saved as a pickle file, which is a serialized format for storing objects (The file is called **'densenet_model.pkl'** in the repository). The inference call (.predict()) predicts and generates scores for every class, and retrieves the top most class with highest score.

b) Wrap the inference logic into an application using Flask

We can query the model to get a class label for a test sample now that we have the trained model file. The inference is as simple as calling the predict() function on the trained model by using the test data. We use Flask to build the inference as a web-service. Flask is a powerful Python microwebserver framework that lets us quickly create REST API-based web services with minimal configuration.

The Script **app.py** includes:

1) A function load_model() to load the trained model file.
2) Then, we instantiate a Flask object called 'app'.
3) A function preprocess_image() that preprocesses the input image and creates a mini-batch as expected by the model.
4) We define a home endpoint that, when hit, returns the message *"Predict Dogs!"*
5) We now define a *'predict'* endpoint. The endpoint accepts a 'POST' request, which contains the test data for which we want the prediction.
6) Declare main function

c) *Containerize the flask application using docker*

Since our code itself should be independent of the underlying machine/OS that runs it, we should containerize it, allowing developers to run the code on any platform given they have docker on their system, without running into environment related issues. For the purpose of this project, we use the most popular container "Docker" to achieve this. We have exposed our Flask application at the port 5000 of the docker server.

Docker Documentation:

- First, we installed Docker Desktop and make sure it's running.
- Built the docker image using the following command:
  docker build -t 532-project .
- Either go to docker desktop to run your image or use the following command:
  docker run -p 5000:5000 532-project .
- Once verified that the docker server is running smoothly, check the container name from docker desktop pull the image using:
  docker save -o <container_name>:latest -o 532_project 532-project

3. Design TradeOffs:

- We have used Python to create our project since it is one of the most trusted technologies for ML inference to-date. This is because it is has a large number of frameworks which makes coding easier and helps in saving development time.
- We use Flask to build the inference as a web-service since Flask is a powerful Python microwebserver framework that lets us quickly create REST API-based web services with minimal configuration.
- Additionally, we use Docker to containerize the application since it can get more applications running on the same hardware than other technologies.

4. Systems Specifications:

- Windows - 64 bit system
- Python 3.7
- Docker Desktop installed (to run docker image on local)

5. How to run the project and tests:

To run project **with** docker:

- Download the docker image of the project named "532_project" here**.**
- After downloading, run this command to make a local docker image:
  docker load -i 532_project.image
- Run this local docker image using the command(do not forget the dot in the end):
  docker run -p 5000:5000 532_project .
- Use the command (replace <images/dog.jpg> with path of the image you want to predict relative to the current directory of command prompt):
  curl -F "file=@images/dog.jpg" http:127.0.0.1:5000/predict
- **Alternatively,** you can change the image path in the **run_me.bat** file in our project directory and double click on it to run the inference for your own image. (The path of the image should be relative to the current directory of command prompt)

To run project **without** docker:

- Clone the repository and in the project directory run: **python app.py**
- Then for getting a dummy prediction about an image in images/dog.jpg simply double click on the file: **run_me.bat**
- At this point, your result should be “Samoyed”
- **Alternatively**, use the command (replace <images/dog.jpg> with path of the image you want to predict relative to the current directory of command prompt):
  curl -F “file=@images/dog.jpg” http:127.0.0.1:5000/predict

6. Results:

When we run the python script **app.py**, we get:

```
(env) PS C:\Users\sowmy\Desktop\Spring21\532\ml-inference-p2_mlinference-team-12> python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Output of the code when the *input image* is given as follows:

Input Image

```
C:\Users\sowmy\Desktop\Spring21\532\ml-inference-p2_mlinference-team-12>curl -F "file=@images/dog.jpg" http://127.0.0.1:
5000/predict
"Samoyed"
```

Output: **Samoyed**

***