Technical Assessment: Object Detection Microservices

Overview

This project consists of two main microservices:

1. **UI Backend Service**: Handles user image inputs and communicates with the AI Backend.

2. **AI Backend Service**: Uses an object detection model to process images and return results (bounding boxes and object labels) in a structured JSON format.

The services work together seamlessly, with the UI backend accepting image inputs and the AI backend performing object detection using a pre-trained deep learning model. The result is returned as a JSON response containing the bounding box coordinates and labels for the detected objects.

Prerequisites

Before replicating this solution, ensure you have the following installed:

1. **Docker**: For containerizing both backend services.

o Docker documentation: https://docs.docker.com/get-docker/

2. **Python** (Flask and FastAPI): For building both backend services. Flask is used in this case, but FastAPI can also be an alternative.

o Install Flask: pip install flask

o Install FastAPI: pip install fastapi uvicorn

3. **Lightweight Object Detection Model**: Any open-source model suitable for object detection.

o **YOLOv3** is used for this project for detecting objects in images.

▪ YOLOv3 repository: https://github.com/ultralytics/yolov3

For systems without a GPU, use the CPU version of the model to complete the task.

4. **Additional Dependencies**:

o OpenCV for image handling: pip install opencv-python

o Numpy for handling arrays: pip install numpy

o Flask for API development: pip install flask

o Pillow for image processing: pip install pillow

Solution Overview

1. Architecture Overview

The architecture involves two main components running as Dockerized microservices:

1. **UI Backend Service**:

o This service accepts an image uploaded by the user, forwards it to the AI backend for object

detection, and sends back the processed image with bounding boxes drawn around detected objects.2. **AI Backend Service**:

o The AI backend is responsible for loading the pre-trained object detection model, processing the

image received from the UI backend, performing object detection, and returning the results in a

structured JSON format.

o **Communication between services** is handled through HTTP requests, and Docker Compose is used to manage both

services in a single environment.

2. Dockerized Microservices Setup

The project includes two Docker containers:

• **ai-backend**: The AI service running the detection model.

• **ui-backend**: The UI service that handles image input and outputs results to the user.

Steps to Implement the Solution

Step 1: Prepare the AI Backend

1. **Clone YOLOv3 Repository**:

o Clone the YOLOv3 repository from GitHub to get access to the object detection model and related

files.

git clone https://github.com/ultralytics/yolov3.git

2. **Install Dependencies**:

o Install required dependencies for running YOLOv3.

pip install -r requirements.txt

3. **Load Pre-Trained Model**:

o The YOLOv3 model weights are available in the repository. Download the weights file:wget

https://github.com/ultralytics/yolov3/releases/download/v7.0/yolov3.weights

4. **Create the AI Backend Flask App**:

o Develop a Flask application (ai_backend.py) that loads the YOLOv3 model and processes the

incoming image to detect objects.

o The code takes the image, processes it using the YOLO model, and returns a JSON containing

bounding box coordinates and labels.

Step 2: Prepare the UI Backend

1. **Create UI Backend Flask App**:

o The UI backend accepts images from users, forwards them to the AI backend for object detection,

and receives the response with the bounding box results.

2. **Communication with AI Backend**:o The UI backend makes a POST request to the AI backend, passing the image and receiving the

detection results as a JSON.

Step 3: Dockerize the Application

1. **Create Dockerfiles**:

o **AI Backend Dockerfile**:

o **UI Backend Dockerfile**:

2. **Create a Docker Compose File**:

o This file orchestrates both services.

3. **Run the Services with Docker Compose**:

o Start both services using Docker Compose:

docker-compose up --build

Output Files and Results Reference

1. Output Images:

The AI backend will return processed images with bounding boxes drawn around the detected objects. These images should be stored and shared as part of the deliverables.

2. JSON Results:

For each processed image, the AI backend returns a JSON file containing details of the detected objects. Example format:

Example:

json

```
{
"objects": [
{
"label": "person",
"confidence": 0.98,
"bounding_box": [100, 150, 400, 450]
}, {
"label": "dog",
"confidence": 0.95,
"bounding_box": [500, 200, 600, 350]
} ]
```

Reference

1. **YOLOv3 Object Detection**:
    - Ultralytics YOLOv3 repository: https://github.com/ultralytics/yolov3

- YOLOv3 is an efficient and real-time object detection algorithm that was used in this project to detect objects in images.

2. **Flask Documentation**:
   - Flask is the web framework used to build the backend services. For more details on Flask and its features, refer to the official documentation: https://flask.palletsprojects.com/
3. **Docker Documentation**:
   - Docker is used to containerize the application. To understand how Docker works and how to set it up, refer to the official documentation: https://docs.docker.com/
4. **OpenCV Documentation**:
   - OpenCV is used for image processing tasks. For more information on image processing and related functionalities, refer to the OpenCV documentation: https://docs.opencv.org/
5. **Numpy Documentation**:
   - Numpy is used for numerical operations, particularly in manipulating arrays. For further reading on Numpy, visit: https://numpy.org/doc/
6. **Pillow Documentation**:
   - Pillow is used for image handling in this project. To learn more about image processing with Pillow, refer to the official documentation: https://pillow.readthedocs.io/en/stable/
7. **ChatGpt**
8. **https://youtu.be/hTacGMfL8lc?si=vHOE_S25Q3umVKCF**