**Project Documentation: Microservice for Object Detection**

**Overview**

This project implements a microservice architecture with two main components:

UI Backend Service: Allows users to upload images.

AI Backend Service: Processes the uploaded images using a lightweight YOLOv4 object detection model and outputs the results as bounding boxes in both image and JSON formats.

**Project Structure**

project-folder/

│

├── app.py # Main backend service script (UI + AI logic)

├── requirements.txt # List of Python dependencies

├── yolov4/ # YOLOv4 folder containing model files

│ ├── yolov4.cfg # YOLO configuration file

│ ├── yolov4.weights # Pretrained YOLO weights

│ └── coco.names # Object classes names

│

├── templates/ # Folder containing HTML templates for the UI

│ └── index.html # Upload interface for the user

│

├── Dockerfile # Dockerfile for containerizing the application

├── readme.txt # High-level project description and setup steps

└── output/ # Generated bounding box images and JSON files

**Steps to Set Up and Run the Project**

**Step 1: Prerequisites**

Ensure the following software is installed:

1. **Python 3.8 or above**: For running the backend services.
2. **Docker**: For containerizing and running the application.
3. **Git**: For cloning the project if hosted on GitHub.

**Step 2: Setting Up the Environment**

**2.1 Clone the Repository**

Clone the repository or download the zipped project folder:

bash

Copy code

git clone <repository-link>

cd project-folder

**2.2 Install Python Dependencies**

Create a virtual environment (optional) and install the required libraries:

bash

Copy code

pip install -r requirements.txt

**Contents of requirements.txt:**

txt

Copy code

Flask==2.1.1

opencv-python==4.5.5.64

numpy==1.22.3

**2.3 Prepare the YOLO Model**

Ensure the yolov4/ folder contains the following files:

* yolov4.cfg: Model configuration file.
* yolov4.weights: Pretrained YOLOv4 weights.
* coco.names: Class labels.

If these files are missing, download them from the [YOLOv4 repository](https://github.com/AlexeyAB/darknet).

**Step 3: Running the Application**

**3.1 Run Without Docker**

If you are running locally without Docker:

bash

Copy code

python app.py

This starts the Flask server on http://127.0.0.1:5000.

**3.2 Run Using Docker**

To containerize the application:

1. Build the Docker image:

bash

Copy code

docker build -t yolo-detection .

1. Run the container:

bash

Copy code

docker run -p 5000:5000 yolo-detection

The service will now be accessible at http://localhost:5000.

**Step 4: Using the Application**

1. Open the application in a browser at http://localhost:5000.
2. Upload an image via the UI.
3. The application:
   * Processes the image using YOLOv4.
   * Displays the output image with bounding boxes.
   * Saves the result as:
     + Processed image: Saved in the output/ folder.
     + JSON file: Contains detected object details (classes, confidence, bounding box coordinates).

**Step 5: Output Format**

**Example JSON Output:**

json

Copy code

[

{

"object": "person",

"confidence": 0.98,

"bounding\_box": [100, 150, 200, 300]

},

{

"object": "car",

"confidence": 0.95,

"bounding\_box": [50, 50, 100, 100]

}

]

**Processed Image Output:**

The image with bounding boxes is saved in the output/ folder.

**Dockerfile Breakdown**

**Dockerfile**:

Dockerfile

Copy code

# Use a lightweight Python base image

FROM python:3.8-slim

# Set the working directory

WORKDIR /app

# Copy application files

COPY . /app

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Expose the port

EXPOSE 5000

# Start the Flask app

CMD ["python", "app.py"]

**Building and Running the Docker Image**

1. Build the image:

bash

Copy code

docker build -t yolo-detection .

1. Run the container:

bash

Copy code

docker run -p 5000:5000 yolo-detection

**References**

1. **YOLOv4 Documentation**: [YOLOv4 GitHub](https://github.com/AlexeyAB/darknet).
2. **Flask Framework**: Flask Documentation.
3. **OpenCV**: [OpenCV Documentation](https://opencv.org/).