**NOTE**

**10-Day Project Plan: Waste Monitoring using Object Detection (YOLOv5 + Spring Boot Integration)**

**Overview: This 10-day project focuses on building an intelligent waste monitoring system using real-time object detection. It integrates YOLOv5 for AI-based waste classification and localization with a Spring Boot backend to offer a scalable and deployable solution suitable for smart city initiatives. The objective is to detect and classify different types of waste (plastic, glass, paper, cardboard, metal, and trash) from images and videos, enabling improved municipal waste management.**

**Day 1: Project Setup and Dataset Preparation**

**Goals:**

* **Set up the development environment**
* **Collect and annotate datasets**
* **Plan classes and folder structure**

**Tasks Completed:**

* **Created project structure including folders for dataset (waste\_data), YOLOv5, and Spring Boot backend.**
* **Collected a custom dataset of images containing six classes of waste: cardboard, glass, metal, paper, plastic, and trash.**
* **Annotated images using Roboflow and exported them in YOLOv5 format.**
* **Split dataset into training, validation, and test sets.**
* **Created a data.yaml file to define classes and paths for YOLOv5 training.**

**Day 2: Training YOLOv5 Model**

**Goals:**

* **Train YOLOv5 on the custom dataset**
* **Verify model performance and refine as needed**

**Tasks Completed:**

* **Configured the training command using YOLOv5s with batch size 16, 640x640 resolution, and 50 epochs.**
* **Successfully trained the model using the custom waste dataset.**
* **Evaluated model using key metrics: Precision, Recall, mAP@0.5, and mAP@0.5:0.95.**
* **Final model showed promising results, with high mAP scores for most classes (e.g., cardboard, paper, and trash), though plastic detection was slightly weaker.**
* **Saved the best model at: runs/train/waste\_yolo\_improved/weights/best.pt.**

**Day 3: Inference and Testing**

**Goals:**

* **Test model on unseen images**
* **Visualize detection results**

**Tasks Completed:**

* **Used YOLOv5 detect.py script to test the model on new images in the test\_images/ folder.**
* **Verified that bounding boxes and labels were displayed correctly.**
* **Noticed strong accuracy in most classes, with trash and cardboard being the most consistently detected.**
* **Performance for plastic was noted as an area for future improvement.**
* **Cleaned up the runs/train folder to retain only the final model.**

**Day 4: Backend Development Begins (Spring Boot)**

**Goals:**

* **Initialize Spring Boot application**
* **Define basic REST endpoints**

**Tasks Completed:**

* **Initialized a Maven-based Spring Boot project in backend/ folder.**
* **Created core packages: controller, service, model, and utils.**
* **Set up basic REST endpoints for file upload and health check.**
* **Integrated Swagger for API documentation.**
* **Connected backend with a temporary static folder for storing uploaded files.**

**Day 5: Python Script Integration Setup**

**Goals:**

* **Enable communication between Spring Boot and YOLOv5 inference**

**Tasks Completed:**

* **Created a Python script run\_detection.py to load YOLOv5 model and return results.**
* **Modified Spring Boot backend to execute the script using ProcessBuilder.**
* **Set up API endpoint /detect that accepts image uploads, saves them locally, invokes the Python model, and returns detection results as JSON.**
* **Verified backend detects objects and returns proper class labels and confidence scores.**

**Day 6: Refactoring and Response Mapping**

**Goals:**

* **Improve communication between Java and Python**
* **Format detection results properly**

**Tasks Completed:**

* **Enhanced run\_detection.py to return results in a structured JSON format.**
* **Created DetectionResult Java model class to map JSON output.**
* **Parsed results in Spring Boot and returned response to frontend/API client.**
* **Added error handling for missing files, failed predictions, and timeouts.**
* **Verified the full flow: Image upload -> Python script -> Response returned.**

**Day 7: Real-Time API Testing and Improvements**

**Goals:**

* **Test real-world usage and improve performance**

**Tasks Completed:**

* **Used Postman and a React-based frontend to test API functionality.**
* **Measured response time; optimized image I/O and file handling.**
* **Deployed model on GPU to reduce inference time.**
* **Optimized Python code to load model only once using Flask+Gunicorn (optional).**
* **Backend working in near-real-time conditions with uploaded images.**

**Day 8: Frontend UI (Optional)**

**Goals:**

* **Design a minimal UI for users to upload and view results**

**Tasks Completed (Optional):**

* **Built a simple frontend using React.**
* **UI had an upload form, preview, and detection result display.**
* **Integrated frontend with Spring Boot using REST APIs.**
* **Users could upload waste images and get detection results instantly.**

**Day 9: Cloud Deployment (Optional)**

**Goals:**

* **Make the system accessible online**

**Tasks Completed (Optional):**

* **Containerized application using Docker.**
* **Pushed image to Docker Hub and deployed on AWS EC2.**
* **Enabled backend to serve public API endpoint.**
* **Verified functionality on remote server with GPU support.**

**Day 10: Final Review, Testing, and Documentation**

**Goals:**

* **Final testing**
* **Create project report and README**

**Tasks Completed:**

* **Conducted end-to-end testing on different image samples.**
* **Verified class-wise detection consistency.**
* **Documented the project with a detailed README.md, API docs, and model performance.**
* **Captured screenshots of training graphs, detection results, and API usage.**
* **Submitted project with summary:**

**"Waste Monitoring using Object Detection" is an AI-powered system combining YOLOv5 and Spring Boot to classify municipal waste types in real-time. With 83% mAP@0.5 and scalable backend integration, it offers a smart-city-ready solution for waste segregation and management.**

**🔚 Final Outcome**

**By the end of the 10-day plan, you will have:**

* **A fully trained YOLOv5 model on 6 waste classes**
* **An inference-ready backend built using Spring Boot**
* **Integration between Java and Python for AI-powered detection**
* **(Optional) Frontend UI for user testing**
* **(Optional) Cloud-deployable solution**

**This system is ready to be extended for edge devices or integrated into municipal solid waste management software.**

**--------------------------------------------------------------------------------------------------------------------------------------**

**Perfect—yes, your goal is to build an app that detects waste in real-time using a camera. Based on everything you've shared (including YOLOv5 training, model improvement, and the project description), here's how we align the 10-day plan to a real-time app, especially with camera integration:**

**✅ What You've Done So Far (Days 1–2):**

* **Trained a YOLOv5 model on custom waste data with 6 classes (cardboard, glass, metal, paper, plastic, trash).**
* **Improved performance metrics (mAP@0.5 = 82.8%, etc.).**
* **Final trained model: waste\_yolo\_improved/best.pt.**

**📱 What You Need Now (Day 3+): Real-Time Waste Detection App**

**We will build the app in React Native (frontend) + Flask or Spring Boot (backend API) and integrate:**

1. **Camera feed for real-time detection**
2. **Use YOLOv5 model for inference**
3. **Show detection results live in the app (bounding boxes + labels)**

**🔟 Updated 10-Day Plan (App + Real-Time Detection)**

| **Day** | **Tasks** |
| --- | --- |
| **Day 1** | **Setup project, install YOLOv5, understand object detection** |
| **Day 2** | **Collect/label data, train YOLOv5, evaluate metrics (done ✅)** |
| **Day 3** | **Build a Flask API to load best.pt and run inference on uploaded or live images** |
| **Day 4** | **Test inference using camera with OpenCV (real-time detection via webcam or phone camera)** |
| **Day 5** | **Build React Native frontend (Expo) with camera module** |
| **Day 6** | **Connect frontend to backend via REST API, send frames or images** |
| **Day 7** | **Display real-time predictions in app (bounding boxes + labels overlay)** |
| **Day 8** | **Optimize performance: reduce model size (if needed), compress images** |
| **Day 9** | **Integrate additional features: logs, auto-upload, offline caching** |
| **Day 10** | **Testing, UI polishing, deployment (mobile & backend) – ready for demo 🚀** |

**📸 What You’ll See in the App**

* **App opens to camera view**
* **You point it at trash — and bounding boxes + labels appear live (e.g., “plastic”, “metal”)**
* **Button to capture + store detections or send to backend for logs**

**🛠 What’s Needed Now**

1. **✅ Final YOLOv5 model: You already have best.pt from waste\_yolo\_improved/**
2. **🧠 Flask Backend (Next Task):**
   * **Load YOLOv5 model**
   * **Accept image frames from frontend**
   * **Run inference and return JSON response with bounding boxes + labels**
3. **📱 React Native Frontend:**
   * **Use expo-camera**
   * **Send camera frames to backend**
   * **Show results live on screen**

**⚡ Real-Time: Is It Possible?**

**Yes! It’s very doable using:**

* **cv2.VideoCapture(0) on PC for webcam**
* **expo-camera on mobile**
* **Flask can infer 1 frame at a time — not full video, but enough to simulate “real-time”**

**If you want edge detection without server, we can convert YOLOv5 → TFLite or ONNX → Run inside Android/iOS (advanced, optional).**

**✅ Next Step (Day 3 Today): Flask Backend Setup**

**Would you like me to:**

* **Help you write the Flask backend inference code?**
* **Or build the full React Native front camera + UI setup first?**

**Let me know and we’ll continue step by step.**

**Ask ChatGPT**

**Top of Form**

**Tools**

**Bottom of Form**

**ChatGPT can make mistakes. Check important info. See Cookie Preferences.**