VisionAI Object Detector

Overview

Team:

Animesh Jain (MCA120),

Rohit Kadam (MCA119)

- **Internal guide:** Prof.R.Salunkhe
- **Project type:** AI/ML desktop application (computer vision)

Problem and objectives

Detecting and identifying multiple objects in real time is essential for safety, automation, and analytics. Traditional methods struggle with accuracy and speed, especially on resource-constrained systems. This project delivers a fast, accurate, and portable object detection pipeline that runs on standard hardware.

- **Primary objectives:**
  - **Real time:** Detect multiple objects from a live webcam feed with low latency.
  - **Accuracy:** Use a proven model (YOLOv3) trained on the COCO dataset.
  - **Portability:** Run on CPU via OpenCV DNN; keep setup lightweight.
  - **Usability:** Provide a clear visual overlay and simple controls.

Approach and technologies

- **Model:** YOLOv3 (You Only Look Once), pre-trained on COCO (80 classes)
- **Inference engine:** OpenCV DNN (CPU target by default; optional CUDA later)
- **Language and libs:** Python, OpenCV, NumPy
- **Interface:** OpenCV window (optional Tkinter/Gradio UI extension)
- **Environment:** VS Code / Jupyter Notebook
- **How it works:**
  - **Preprocess:** Resize to 416×416, normalize, create blob.
  - **Infer:** Forward pass through YOLOv3 with configured layers.
  - **Postprocess:** Confidence filtering and Non-Maximum Suppression.
  - **Render:** Draw bounding boxes and class labels in real time.

Features and architecture

- **Key features:**
  - **Multi-class detection:** 80 COCO classes with labels and confidence scores.
  - **Live webcam feed:** 640×480 by default; adjustable resolution.
  - **Configurable thresholds:** Confidence and NMS for easy tuning.
  - **Modular code:** Clean separation of loading, inference, and visualization.
- **Structure:**
  - **Core:** object_detector.py — model load, inference, NMS, drawing
  - **Runner:** webcam_stream.py — capture loop, key handling, display
  - **Assets:** yolo/yolov3.cfg, yolo/yolov3.weights, yolo/coco.names
  - **Optional:** gui.py — simple UI to toggle input source or thresholds
  - **Tests:** test_image.py, test_video.py — run on files instead of webcam
- **Setup essentials:**
  - **Dependencies:** opencv-python, numpy
  - **Assets folder:** yolo/ with yolov3.cfg, yolov3.weights, coco.names
  - **Paths:** Use absolute paths or a single base_path variable for portability

---

Dataset, evaluation, and deployment

- **Dataset:** COCO (Common Objects in Context) — pre-trained model used for inference
- **Metrics for validation on test clips:**
  - **Throughput:** Frames per second (FPS) on CPU
  - **Detection quality:** Precision/recall on selected labeled frames
  - **Latency:** End-to-end frame processing time
- **Deployment:**
  - **Desktop run:** Python script or Jupyter notebook
  - **Packaging (optional):** requirements.txt; PyInstaller/virtual env for handover
  - **Extensibility:** Swap model to YOLOv4/YOLOv5 later; enable CUDA if available
- **Future scope:**
  - **Video analytics:** Counting, region-of-interest alerts, heatmaps
  - **Edge deployment:** NVIDIA Jetson/Raspberry Pi optimizations
  - **Tracking:** SORT/DeepSORT for persistent IDs across frames
  - **Custom training:** Fine-tune on domain-specific objects

---

Timeline and deliverables

- **Week 1:** Environment setup, asset collection (cfg/weights/names), baseline run
- **Week 2:** Implement detection pipeline (pre/postprocess, thresholds, NMS)
- **Week 3:** Integrate webcam, stabilize FPS, add logging and simple configs
- **Week 4:** Optional UI layer; file-based tests (image/video) and path management
- **Week 5:** Evaluation on sample clips; parameter tuning; prepare demos
- **Week 6:** Documentation, packaging, final testing, and presentation
- **Deliverables:**
    - **Executable demo:** Live webcam detection with overlays
    - **Source code:** Modular, documented Python scripts and notebooks
    - **Assets:** YOLOv3 cfg/weights/names organized under yolo/
    - **Docs:** Setup guide, usage instructions, results summary, and future work

If you want, I can turn this into a neatly formatted one-pager PDF or a slide deck, and include a diagram of the detection pipeline.