International Journal of Enhanced Research in Management & Computer Applications,
ISSN: 2319-7471, Volume 14 Issue 4, April 2025

Implementation of Real-Time Object Detection

Ajit Kumar Chansauriya[1], Mr. Manoj Kumar[2]

[1]M.Tech, Computer Science, Shri Venkateshwara University, Gajraula
[2]Asst. Prof., Computer Science Department, Shri Venkateshwara University, Gajraula

ABSTRACT

This project focuses on the development of an Android-based real-time object detection and pose estimation application using YOLOv8. The model is trained using Google Colab and converted to the ONNX format for seamless integration with Android. The application leverages CameraX for efficient camera access and live image analysis, while the detection pipeline utilizes YOLOv8 models for identifying objects and estimating human poses. Implemented in Kotlin and built with Jetpack libraries, the app demonstrates real-time performance and responsiveness. This project highlights the deployment of advanced deep learning models on mobile platforms with optimal performance using lightweight model conversion techniques and modern Android APIs.

Keywords: YOLOv8, Object Detection, Pose Estimation, Android, CameraX, ONNX, Kotlin, Jetpack Libraries, Google Colab, Real-time Detection.

INTRODUCTION

In recent years, real-time object detection and human pose estimation have gained immense popularity due to their vast applications in areas such as augmented reality, fitness tracking, surveillance, and interactive gaming. The You Only Look Once (YOLO) algorithm has stood out for its speed and accuracy in object detection tasks. YOLOv8, the latest iteration, offers enhanced performance and supports both object detection and pose estimation in a unified framework.

This project leverages the capabilities of YOLOv8 to develop a mobile Android application that performs real-time object detection and human pose estimation using the device's camera. Built with Kotlin and Android's Jetpack libraries, the application integrates CameraX for camera management and renders live inference through ONNX models optimized for mobile devices. The YOLOv8 model is trained and converted using Google Colab to ensure compatibility and performance on mobile platforms.

By combining deep learning with mobile development, this project demonstrates how cutting-edge AI models can be deployed efficiently on Android devices, providing a practical solution that is both fast and accurate for real-time use cases.

Process Overview
The development of the Object Detection Android application involved a systematic process, divided into several key stages:

a. Problem Definition & Objective
The primary objective was to create a mobile application capable of detecting multiple objects in real-time using the device's camera.

Additional features like pose estimation were included to enhance the utility for use cases like surveillance and fitness monitoring.

b. Research and Dataset Collection
Studied popular object detection algorithms like YOLO, SSD, and Faster R-CNN.

Chose pre-trained models trained on large-scale datasets like COCO (Common Objects in Context) for better accuracy and performance.

c. Model Selection and Optimization
Selected YOLOv5 (or YOLOv4-tiny) due to its balance of speed and accuracy, optimized for mobile environments.

The model was converted to TensorFlow Lite (TFLite) format to ensure smooth operation on Android devices.

d. Android App Development
The app was developed using Android Studio with Java/Kotlin.

Integrated TensorFlow Lite Interpreter for on-device model inference.
Implemented real-time camera stream processing using CameraX API for efficient frame capturing.

e. Testing and Evaluation
The app was tested on multiple Android devices to evaluate performance, speed, and accuracy.

Ensured real-time detection with minimal latency and high accuracy for commonly detected objects and human poses.

f. Deployment and Documentation
Final build was deployed on Android devices.

Complete documentation including project report, user guide, and future scope was prepared.

METHODOLOGY

This section describes the overall process of the real-time object detection application, the system architecture, and the technologies used at each step to ensure efficient and accurate object detection on Android devices.

1 System Overview
The proposed system is a mobile application capable of detecting and identifying multiple objects in real-time using the device's camera. The application is built using Android Studio with Kotlin/Java as the base language and leverages CameraX API for real-time video input. For object detection, a pre-trained YOLOv5/YOLOv4-tiny model, converted to TensorFlow Lite, is integrated for on-device inference.

2 Working Mechanism
2.1 Initialization
Upon launching the application, the Android CameraX API initializes the camera and starts capturing live video frames. These frames are displayed in real time on the mobile screen. The system ensures permission handling and device compatibility for accessing camera resources.

2.2 Frame Acquisition
Each video frame is continuously passed from the CameraX preview stream to a background thread for processing. The frames are resized and preprocessed to match the input size expected by the YOLO model (e.g., 416×416 pixels).

2.3 Model Inference
The resized frame is input into the TFLite model, which performs:
Feature extraction
Bounding box prediction
Class label assignment
The model returns a list of bounding boxes, confidence scores, and class indices for each detected object.

2.4 Post-Processing
The output of the model is parsed using a Non-Maximum Suppression (NMS) algorithm to remove duplicate or overlapping boxes and retain the most confident predictions. The system then maps the indices to their respective class names (e.g., person, car, bicycle) using a predefined label map.

2.5 Displaying Results
The bounding boxes and corresponding labels are drawn on the preview stream using Android's Canvas API or OverlayView. The real-time results are updated frame-by-frame, providing an interactive and intuitive detection experience.

Code :

```
import cv2
thres = 0.5 # Threshold to detect object

cap = cv2.VideoCapture(0)
cap.set(3,648)
cap.set(4,448)
cap.set(10,70)

classNames = []
classFile = 'coco.names'
with open(classFile,'rt') as f:
   classNames = f.read().rstrip('\n').split('\n')

configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
weightsPath = 'frozen_inference_graph.pb'
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)

while True:
   success,img = cap.read()
   #cv2.imshow(img)
   classIds, confs, bbox = net.detect(img,confThreshold=thres)
   print(classIds,bbox)

   if len(classIds) != 0:
      for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
         cv2.rectangle(img,box,color=(0,255,0),thickness=2)
         cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),
                  cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
         cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),
                  cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

   cv2.imshow("Output",img)
   #cv2.waitKey(1)
   if cv2.waitKey(1) & 0xFF == ord('q'):
      break
```

Impact

The implementation of this project demonstrates the feasibility and effectiveness of running deep learning models on resource-constrained mobile devices. The impact is outlined as follows:

Real-Time Performance: Achieves object detection in milliseconds per frame, allowing seamless user interaction.
On-Device Inference: Eliminates the need for a cloud backend, reducing latency and preserving user privacy.
Educational Utility: Serves as a practical demonstration of deploying machine learning models on edge devices.
Scalability: The modular design enables easy integration with other applications such as smart surveillance, AR, and accessibility tools.

BIBLIOGRAPHY

[1]. **GeeksforGeeks**. (n.d.). A computer science portal for geeks. Retrieved April 10, 2025, from https://www.geeksforgeeks.org/

[2]. **Kaggle**. (n.d.). Your machine learning and data science community. Retrieved April 10, 2025, from https://www.kaggle.com/
[3]. **OpenAI**. (n.d.). OpenAI. Retrieved April 10, 2025, from https://openai.com/
[4]. **Stack Overflow**. (n.d.). Where developers learn, share, & build careers. Retrieved April 10, 2025, fromhttps://stackoverflow.com/
[5]. **Twitter**. (n.d.). Twitter. Retrieved April 10, 2025, from https://twitter.com/
[6]. **YouTube**. (n.d.). YouTube. Retrieved April 10, 2025, from https://www.youtube.com/
[7]. **Analytics Vidhya**. (n.d.). Data science community & knowledge portal. Retrieved April 10, 2025, from https://www.analyticsvidhya.com/