



## Senior Project Report

# SEET: Hear Objects, Spot Colors

---

La Min Oo (6540039)

Kaung Khant Lin (6540131)

Rebekah Nianghauching Mark (6610292)

Supervised by Asst. Prof. Dobri Atanassov Batovski

CSX 3010 Senior Project 1 (1/2025)

## Senior Project Approval

Project title: Developing a mobile app using computer vision to help the visually impaired identify road signs and objects in real time

Academic Year: 1/2025

Authors: La Min Oo (650339)

Kaung Khant Lin (6540131)

Rebekah Nianghauching Mark (6610292)

Project Advisor: Asst. Prof. Dobri Atanasov Batovski

---

The Senior Project committee's cooperation between the Department of Computer Science and Information Technology, Vincent Mary School of Science and Technology, Assumption University had approved this Senior Project. The Senior Project in partial fulfilment of the requirement for the degree of Bachelor of Science in Computer Science and Information technology.

Approval Committee:

.....  
(Asst. Prof. Dr. Dobri Atanasov Batovski)  
Project Advisor

.....  
(Dr. Kwankamol Nongpong)  
Committee Member

.....  
(A. Tianai Tang)  
Committee Member

## Abstract

Navigating urban environments poses significant challenges for individuals with visual impairments and color vision deficiencies, particularly in perceiving and interpreting road-related signs and surrounding objects in real time. Existing assistive technologies, including smartphone-based navigation apps and filter-based color aids, offer partial solutions but often lack full integration of features such as object detection, distance estimation, and accessible audio feedback.

This paper presents the development of a real-time, mobile application designed to enhance navigational safety and independence for visually impaired and colorblind users. With the help of computer vision and algorithms, the system detects and interprets road signs and objects, estimates direction (3×3 grid) and numeric distance in meters with confidence-gated smoothing, and delivers prioritized, concise audio feedback. The Color Finder currently supports Protan (red deficiency) analysis. The design builds on insights from current assistive solutions, including applications such as Be My Eyes, Seeing AI, and EnChroma filters, highlighting the limitations of existing systems in terms of adaptability, user-centered design, and multi-sensory interactions. By integrating real-time object and color detection, the proposed application aims to provide a reliable, inclusive, and responsive tool that enhances spatial awareness and promotes independent navigation.

# Table of Contents

Chapter 1: Introduction	4
1.1 Problem Statement	4
1.2 Scope of the Project	4
Chapter 2: Related Work	7
2.1 Challenges	7
2.2 Navigation Systems for Blind and Visually Impaired	7
2.3 User Interface	7
2.4 EnChroma and Filter-based Aids for Color Vision Deficiency	8
2.5 Mobile Applications	9
2.6 Mobile review	10
2.7 Summary	10
Chapter 3: Proposed Methodology	12
3.1 Methodology	12
3.2 Testing	15
Chapter 4: Design of The System	18
4.1 Functional Requirements Specification	18
4.2 System Design	19
Chapter 5: Result	29
Chapter 6: Conclusion	37
References	38
Appendices	39
Appendix A. Live Object Detection	39
Appendix B. Color Finder	44
Appendix C. Metric Evaluation (metrics.py)	46

## Lists of Figures

Figure 1. EnChroma filter range [2]	8
Figure 2. System architecture of the Be My Eyes App for the blind [3]	9
Figure 3. Color finder (capture → analyze)	20
Figure 4. Live detection with speech	21
Figure 5. System context: on device, CV and system TTS	22
Figure 6. System architecture	23
Figure 7. Main camera screen (light/dark theme)	24
Figure 8. Select feature screen (light/dark theme)	25
Figure 9. Camera screen for Color Finder (light/dark theme)	26
Figure 10. Color detection screen (light/dark theme)	27
Figure 11. Feedback screen (light/dark theme)	28
Figure 12. Main camera screen with camera live feed paused	29
Figure 13. Live camera feed with detection on (resume button triggered)	30
Figure 14. Using drawer menu for navigation through the application	31
Figure 15. Color finder detecting color regions (on default protan mode) with captured image & analysis report	32
Figure 16. Feedback screen for users	33
Figure 17. Confusion matrix for the INT8 EfficientNet-B0 model	35

## Lists of Tables

Table 1. Project scope summary	6
Table 2. Cost estimation of the project	14
Table 3. Benefit estimation of the project	15
Table 4. Use cases of our application	19
Table 5. Overall performance of the TFlite model	34
Table 6. Per-class performance on the test dataset	34
Table 7. Top confusion pairs	35

## Lists of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BLE	Bluetooth Low Energy
BitmapIO	Bitmap Input/Output (Decodes image into RGB format)
CPU	Central Processing Unit
CV	Computer Vision
FPS	Frames-per-second
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
IMU	Inertial Measurement Unit
INT8	Integer 8-bit
KPIs	Key Performance Indicators
ML	Machine Learning
ML Kit	Google's SDK
NV21	NV21 (Image format)
OD	Object Detection
pp	Percentage Points
RGB	Red, Green, Blue
RN	React Native
ROI	Return on Investment
SDG	Sustainable Development Goal
SDK	Software Development Kit
TFLite	TensorFlow Lite
TTS	Text-to-Speech
UI	User Interface
UX	User Experience
YOLO	You Only Look Once
YUV	Y-U-V (Color encoding format)

# Chapter 1: Introduction

## 1.1 Problem Statement

Navigating urban environments presents challenges for visually impaired individuals and color vision deficiency, particularly in perceiving and interpreting road related signs and objects in real time. While various assistive technologies exist, many do not completely integrate features such as accurate object detection, distance estimation, and accessible audio feedback. Consequently, there remains a need for solutions that can effectively support enhanced environmental awareness and navigation safety.

This project develops a real-time mobile application that enhances the navigational safety and independence of visually impaired and colorblind individuals by using computer vision to detect and interpret road-related signs and objects, estimate their distance and direction, and also audio outputs.

## 1.2 Scope of the Project

**Objective:** Real-time, device recognition of road-context cues (lights, crosswalks, signs, vehicles) which remain robust under imbalance and visual clutter.

**Pipeline:**

- Collect → Curated images
- Balance & argument per class (caps + targeted negatives)
- Train (two phases: freeze → fine-tune) with label smoothing & class weighting
- Validate & diagnose via per-class metrics and confusion
- Export to TFLite (Float32 + INT8) with representative calibration



## **Data Pipeline**

### **Scope & Classes (10)**

car, crosswalk, cycle, emergency\_exit, stop, traffic green, traffic red, traffic yellow, truck, van.

#### **[TRAIN]**

- car: 3,182
- crosswalk: 1,420
- cycle: 1,080
- emergency\_exit: 1,151
- stop: 1,000
- traffic green: 3,098
- traffic red: 2,195
- traffic yellow: 2,152
- truck: 1,464
- van: 900

**Total train images: 17,642**

#### **[VALID]**

- car: 885
- crosswalk: 239
- cycle: 180
- emergency\_exit: 190
- stop: 120
- traffic green: 226
- traffic red: 109
- traffic yellow: 113
- truck: 136
- van: 75

**Total valid images: 2,273**

### Early Imbalance (Observed)

Head vs tail: car  $\approx$  16,000 images vs stop  $\approx$  400; emergency\_exit had many high-resolution, clean samples. Effect: over-prediction of head classes, with confusions such as crosswalk  $\leftrightarrow$  hazard-like pavements and emergency\_exit  $\leftrightarrow$  traffic green at distance.

### Curation & Balancing actions

Caps on head classes; down-sample near-duplicates from burst captures. Conservative augmentations: horizontal flip, small rotate/translate, mild brightness/contrast; color integrity preserved for traffic lights.

Stratified splits without clip leakage; 224×224 RGB, values preprocessed for EfficientNet.

*Table 1. Project scope summary*

Category	In Scope	Out of Scope/ Limitations
<b>Users</b>	Tested on normal vision users	Blind and visually impaired users
<b>Functionalities</b>	Real-time object/sign detection, distance & direction estimation, audio feedback	Distance measurements
<b>Technology</b>	Smartphone sensors, Computer vision	none
<b>Environment</b>	Pedestrian walkway, outdoors	Complex walkway(crowded/busy), poor lighting

## Chapter 2: Related Work

### 2.1 Challenges

A persistent challenge in assistive design is the lack of user-centered approaches that consider the diversity of visual impairments and individual needs. Systems often fail due to poor adaptation to users' sensory and cognitive capabilities. Effective designs must reduce cognitive load, enhance intuitiveness, and incorporate user feedback loops. The authors stress aligning system outputs with how visually impaired users process non-visual spatial information for better usability and acceptance [\[1\]](#).

### 2.2 Navigation Systems for Blind and Visually Impaired

Smartphones have become key platforms for assistive navigation due to their built-in sensors (GPS, cameras, accelerometers, gyroscopes) and cloud computing capabilities. They support both indoor and outdoor wayfinding through apps like NavCog, Lazzus, BlindSquare, and Seeing AI. These tools leverage AI, GNSS, and BLE beacons to provide real-time environmental descriptions and object detection, fostering independence among the visually impaired [\[1\]](#).

### 2.3 User Interface

The user interface (UI) serves as the communication bridge between the navigation system and the user but remains a major bottleneck. Speech and haptic feedback are the dominant modalities, each with limitations in data throughput and situational awareness. Advanced approaches involve mixed-reality elements, virtual sound sources, and electro-tactile feedback (e.g., BrainPort, vOICe). Future UIs should combine minimal cognitive demand with intuitive spatial cues, allowing users to interpret their environment naturally through multimodal feedback [\[1\]](#).

## 2.4 EnChroma and Filter-based Aids for Color Vision Deficiency

Notch-filter spectacles such as EnChroma have been widely publicized as aids for red–green anomalous trichromacy. Laboratory studies that go beyond anecdote show modest, context-dependent perceptual effects rather than a “cure.” A recent controlled study of EnChroma multi-notch filters in deuteranomalous observers used three experimental tasks (asymmetric color matching, discrimination thresholds, and appearance ratings) and tested performance before and after a week of use; results show significant increases in perceived saturation and modest improvements in appearance along the red-green axis, with discrimination improvements limited (significant primarily for some red stimuli). The authors stress that improvements depend on illumination, stimulus gamut (broadband surface colors vs. display primaries), and observer specifics and explicitly note that filters do not create entirely new color experiences or restore normal trichromacy [2].

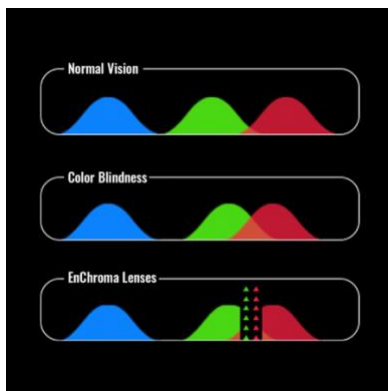


Figure 1. EnChroma filter range [2]

Latane *et al.* (2023) developed an Android-based assistive application called Be My Eyes for Blind People to help visually impaired users navigate independently through real-time object detection and audio feedback. The system utilizes the YOLOv3-tiny algorithm for detecting surrounding objects captured via the mobile camera and converts this information into speech output through a text-to-speech (TTS) engine. The app also supports voice commands for additional tasks such as checking battery level, sending messages, or making calls. Built using Kotlin in Android Studio, the prototype demonstrates how machine learning and computer vision can translate the visual environment into an auditory experience, promoting greater safety and autonomy for users with vision impairments [3].

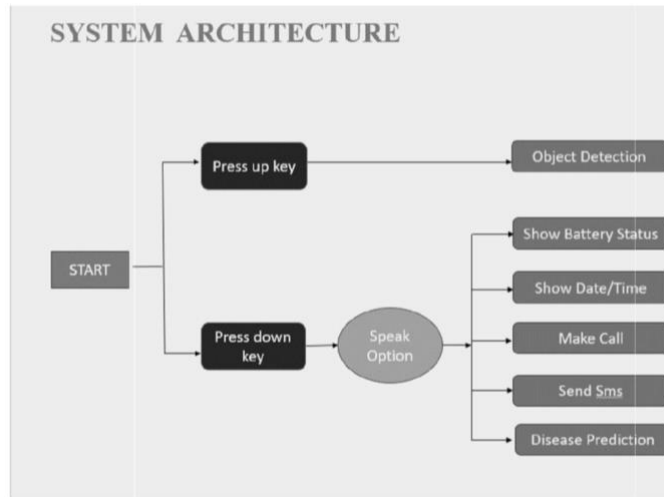


Figure 2. System architecture of the Be My Eyes App for the blind [3]

## 2.5 Mobile Applications

Recent studies emphasize the growing role of smartphone-based assistive technologies in improving accessibility and independence for visually impaired users. Naotunna and Hettige [4] conducted a comprehensive review of more than sixty mobile applications designed for individuals with vision impairment, identifying advancements in artificial intelligence (AI), computer vision, and GPS-based navigation. Applications such as Eye Assistant, SUBE, and Smart Eye demonstrated the integration of YOLO-based object detection and real-time auditory feedback, enabling users to identify objects and navigate their surroundings with greater safety and confidence. Despite these innovations, many systems still face issues related to voice command reliability, recognition accuracy, and the lack of user-centered design frameworks that account for varying degrees of vision loss.

In a similar context, Akkara and Kuriakose [5] analyzed the evolution of smartphone applications for visually impaired persons, categorizing them into accessibility tools, voice assistants, magnifiers, and camera-based object identifiers. Their review included widely adopted apps such as Microsoft Seeing AI, TapTapSee, Lazarillo GPS, and Be My Eyes, highlighting how smartphones have become affordable, multi-functional low vision aids that eliminate the need for costly specialized hardware. The authors further stressed the importance of user training and awareness among both visually impaired individuals and healthcare professionals to maximize the potential of these tools.

Both studies [4], [5] demonstrate that smartphones have evolved from simple communication devices into intelligent assistive platforms that combine AI, audio-haptic feedback, and adaptive interfaces. Future development should focus on multi-modal feedback, context-aware computing, and inclusive interface design to create seamless and responsive experiences that better serve the needs of users with visual impairments.

## 2.6 Mobile review

Abidi *et al.* [6] presented a comprehensive review of modern navigation and assistive systems for visually impaired individuals, highlighting how mobile applications have evolved into powerful accessibility tools. Their review discusses the integration of artificial intelligence (AI), computer vision, and speech synthesis in transforming smartphones into intelligent, context-aware assistive devices. Applications such as Be My Eyes, Seeing AI, and TapTapSee were recognized for enabling users to identify objects, read text, and navigate unfamiliar environments through real-time visual interpretation and auditory feedback. These smartphone-based solutions are noted for being affordable, portable, and inclusive reducing dependence on expensive specialized hardware.

Despite these advantages, the study identifies persistent technical and usability limitations. Most mobile assistive systems still rely on continuous internet connectivity, exhibit high power consumption, and offer limited customization for users with varying degrees of vision loss. Abidi *et al.* [6] emphasize that future development should prioritize offline processing, energy efficiency, multi-sensor integration, and user-centered interface design to enhance reliability and accessibility. Their findings underline the ongoing need for context-aware, AI-driven assistive technologies capable of functioning robustly in real-world conditions.

## 2.7 Summary

Assistive technologies for the visually impaired continue to face design and usability challenges, particularly the lack of adaptive, user-centered approaches. Smartphones have become central to accessibility efforts, integrating sensors, GPS, and AI to deliver real-time object detection, navigation, and scene understanding. Nevertheless, current user interfaces

relying mainly on speech or haptic feedback still fail to provide complete spatial awareness. Emerging multimodal systems that combine audio, tactile, and electro-tactile feedback show promise in improving intuitiveness and reducing cognitive load. While optical aids such as EnChroma filters yield modest perceptual benefits, they do not restore normal vision. Recent studies, including Abidi *et al.* [6], confirm the trend toward on-device, AI-based assistive applications that can operate offline, minimize latency, and adapt to diverse user needs. These developments point to the next step in creating reliable, inclusive mobile systems that enhance autonomy and safety for visually impaired users.

## Chapter 3: Proposed Methodology

### 3.1 Methodology

#### 3.1.1 Research Strategy and Process

##### Search Terms and Keywords

Visual Impairment navigation, types of color blindness, object detection for vision-impaired, speech-to-text TTS accessibility, feedback, crosswalk detection, color finder EnChroma study, mobile assistive apps review, offline on-device inference, BLE beacons wayfinding, user-centered design visually impaired.

#### 3.1.2 Problem Identification

- Many systems have poor user-centered design, not adaptable to different vision loss
- Major UI limitation: speech and haptic feedback are common but do not provide enough detail, feedback needs improvement.
- On the ML side, mobile-friendly detectors (e.g., YOLO variants, lightweight backbones) enable real-time inference but face class imbalance, visual clutter, and power/connectivity constraints.
- Reviews identify recurring technical issues: unreliable voice recognition, limited offline capability, high power consumption, and lack of customization for degrees of vision loss.

#### 3.1.3 Proposed Solution

A smartphone app that performs on-device, real-time road-context and object recognition (crosswalks, traffic lights, vehicles, cycles, etc.) and communicates concise audio feedback in addition it also guides low-vision pedestrians. The app is optimized for offline use via an EfficientNetB0-based model exported to TFLite (Float32 and INT8) for latency/efficiency trade-offs.



## Technical Approach

- **Perception pipeline:** camera capture → preprocess (224×224 RGB) → EfficientNetB0 backbone classifier/detector head → softmax with label smoothing (0.05) plus margin penalty for hard negatives → per-frame predictions aggregated with short temporal smoothing to reduce flicker/false positives.
- **Training & Data handling:** curate dataset of 10 classes (car, crosswalk, cycle, emergency\_exit, stop, traffic green/red/yellow, truck, van). Apply curation actions: cap head-class samples, conservative augmentations (flip, small rotate/translate, mild brightness/contrast preserving traffic-light colors), stratified splits, class weighting and label smoothing, two-phase training (freeze → fine-tune). Validate using per-class metrics and confusion analysis; export representative calibration sets for INT8 quantization.
- **On-device model deployment:** export to TFLite (Float32 & INT8) with quantization-aware steps; run inference in Kotlin on Android using lightweight runtime (ML Kit or TFLite API) with a small batching/windowing policy to balance latency vs stability.
- **Interface/UX:** primary audio guidance via TTS (short, prioritized messages). UI/UX designed with low cognitive load (short utterances, prioritized info, user-configurable verbosity).

3.1.4 ROI Analysis

The financial viability of the project is assessed using **ROI** metric. ROI is a key performance indicator used to evaluate the efficiency of an investment. It is calculated by dividing the net profit (Total Benefits minus Total Costs) by the Total Costs, and the result is expressed as a percentage.

The equation used for calculating the ROI is presented as:

$$ROI = \frac{(Total\ benefits - Total\ Costs)}{Total\ Costs} \times 100. \tag{1}$$

Table 2. Cost estimation of the project

Cost Component	Description	Estimated Cost (USD)
Development	App design, coding, model training (4 developers × \$2,000 × 3 months)	\$24,000
Testing & Optimization	Data collection, labeling, real-world trials	\$6,000
Deployment	Server hosting, Play Store registration, integration tools	\$2,000
Maintenance	Updates, bug fixes, and user support (annual)	\$3,000
Total Cost (C)		\$35,000

*Table 3. Benefit estimation of the project*

Benefit Source	Description	Estimated Annual Benefit (USD)
<b>Accessibility Grants</b>	Government or NGO support for inclusive technology	\$10,000
<b>App Revenue</b>	Premium features, ads, institutional license	\$30,000
<b>Reduced Support Costs</b>	Users rely less on costly assistive hardware	\$5,000
<b>Total Benefit (B)</b>		\$45,000

### ROI Calculation

Using Equation 1 with the estimated figures Table 2 and 3:

$$ROI = \frac{(45,000 - 34,000)}{35,000} \times 100 = \frac{10,000}{35,000} \times 100 \approx 28.57\%.$$

Thus, the project yields an estimated ROI of 28.57% in the first year. This indicates a profitable and sustainable investment for assistive technology.

## 3.2 Testing

### 3.2.1 Unit Testing

- Software module tested independently to verify all functions perform as expected
- **Scope:** Model loading, camera input capture, text-to-speech output
- **Tools:** Android Studio Unit Test Framework (JUnit), Espresso for UI component testing
- **Expected Outcome:** Each function should return correct outputs under valid inputs and handle exceptions gracefully

### 3.2.2 Integration Testing

- Integration testing will confirm that different modules (object detection model, TTS engine, and user interface) work cohesively
- **Scenario:** Test whether detected objects are correctly converted into speech messages and delivered with minor delay
- **Expected Outcome:** Smooth data flow between components with no crashes, speech delay  $\leq 0.7$  seconds

### 3.2.3 System Testing

- Comprehensive end-to-end evaluation of the entire system on real devices.
- **Parameters Tested:**
  - Accuracy of objects and crosswalk detection
  - Latency of model inference and audio response
  - Power consumption during continuous use
- Offline functionality (TFLite INT8 model)
- **Environment:** Tested on multiple Android devices with varying specifications under different lighting and outdoor conditions

### 3.2.4 Performance Testing

Focuses on evaluating speed, scalability, and resource efficiency. (Tested with Samsung S25 Ultra, Xiaomi 14)

Metrics:

- **Inference speed:**  $\leq 300$  ms per frame
- **FPS rate:**  $\geq 14$  frames/sec for smooth feedback
- **Battery drain:**  $< 30\%$  after 1.5 hours of continuous use
- **Model size:**  $\leq 5$  MB (INT8 quantized version)
- **Tools:** Android Profiler, TensorFlow Lite Benchmark Tool

### 3.2.5 Validation Testing

This ensures that the final system meets the intended functional requirements defined in [Chapter 4.1](#).

## SEET: Hear Objects, Spot Colors

- **Crosswalk Detection:**  $\geq 90\%$  precision and recall
- **Object Detection:** Correctly identify and announce cars, cycles, and obstacles within 3 meters
- **Color Finder:**  $\geq 75\%$  accuracy in identifying basic color categories under daylight illumination

## Chapter 4: Design of The System

### 4.1 Functional Requirements Specification

#### 4.1.1 Stakeholders

##### Primary Users

- **Visually impaired users:** benefit from crosswalk detection and obstacle/moving vehicle alerts
- **Colorblind users:** benefit from the color finder filter

##### Crosswalk Detection

- **Actor:** Visually impaired user
- **Goal:** Detect presence and location of crosswalks to enhance safety
- **Main flow:** App captures camera frames → model detects crosswalk → TTS announces, “crosswalk ahead, 6 meters”

##### Object Detection (Obstacles / Moving Vehicles)

- **Actor:** Visually impaired user
- **Goal:** Identify obstacles and moving vehicles to avoid hazards
- **Main flow:** Continuous inference on camera frames → prioritized hazard messages (e.g., “2 Cars on the left”) → temporary alert suppression to avoid overload

##### Color Finder (For Color-blind Users)

- **Actor:** Color-deficient user
- **Goal:** Identify and distinguish colors when requested
- **Main flow:** User invokes color-finder feature → app samples pixel regions → app reports detected color regions and color names

*Table 4. Use cases of our application*

Use Case	Actor	Goal	Predictions	Main Flow	Outcomes
<b>Crosswalk Detection</b>	Visually impaired users	Safety detect crosswalk	Mobile App (SEET), use camera	App captures and model detect crosswalk	Alert through speech “Crosswalk ahead, 6 meters”
<b>Object Detection</b>	Visually impaired users	Identify obstacles, moving vehicle	Detect cars, automobiles, road signs	Continuous inference on camera frames	Speech: “Car, truck, motorcycle, etc.”
<b>Color Finder</b>	Color Blind users	Identify and distinguish colors	Depicts exact color in picture	App captures or import pictures and samples pixel regions	Displays the exact color of the image.

## 4.2 System Design

### 4.2.1 Workflow

#### A. Color Analysis Flow

The flow of color region detections on the image (see Figure 3):

- **User interaction** - User taps Capture button on the RN color finder
- **Capturing Image** - RN color finder screen tells VisionCamera (photo) to take photos
- **Detecting colors** - App triggers detectConfusableColors with image URI and setting
- **Image decoding** - BitmapIO decodes the image into RGB format
- **Pixel level analysis** - Each pixel is converted to Lab space, simulated for protan, and color differences are computed.
- **Applying Rules** - Pixels are grouped into buckets based on color confusion
- **Region Extraction** - Confusable color regions and stats are identified
- **Displaying Results** - Modal opens showing highlighted regions; analysis complete

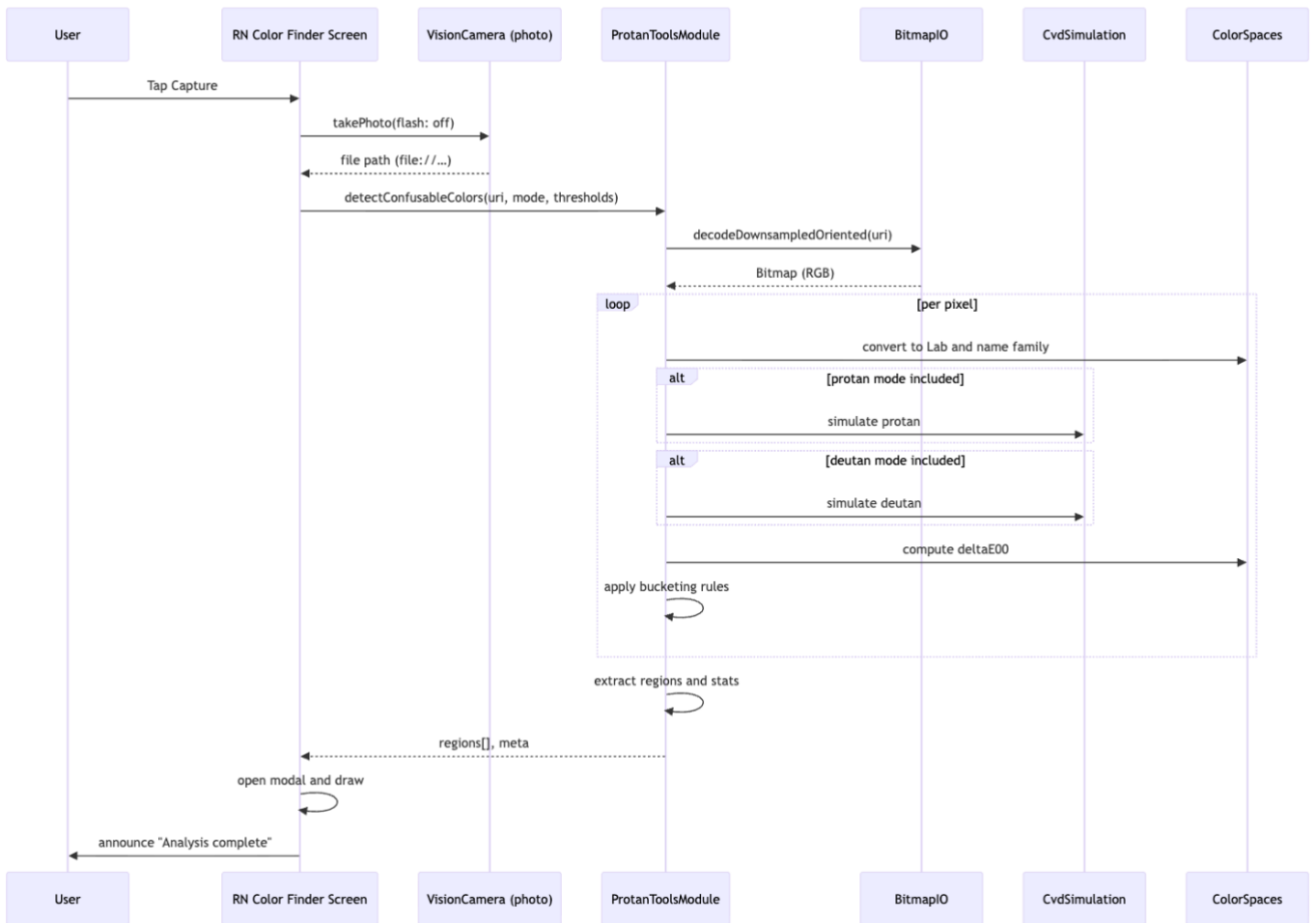


Figure 3. Color finder (capture → analyze)

## B. Sequence Diagram, Real-time Object Detection and Speech

The flow of real-time object detection and speech (see Figure 4):

- **Frame Capture** - VisionCamera captures frames (YUV)
- **Frame Processing** - Frame Processor triggers `mlKitObjectDetect(frame)`
- **Image Conversion** - Kotlin Plugin converts YUV to NV21 and orients image
- **Object Detection** - ML Kit Detector processes input image and returns detected objects
- **Pitch Reading** - SensorManager provides pitch data
- **ROI Handling** - Detected regions are cropped, resized, and preprocessed
- **Classification** - TFLite model classifies each ROI and returns logits/tables
- **Post Processing** - Fuse distances (bounding box + pitch), smooth results, and format JSON output



- **Thresholding** - Apply thresholds and majority voting to refine detections
- **State Update** - Kotlin Plugin updates state with detected objects and directions
- **UI Update** - RN UI displays final message and overlay data
- **Speech Request** - RN UI triggers speech output request
- **Speech Output** - SpeechSupervisor and System TTS synthesize and speak the result

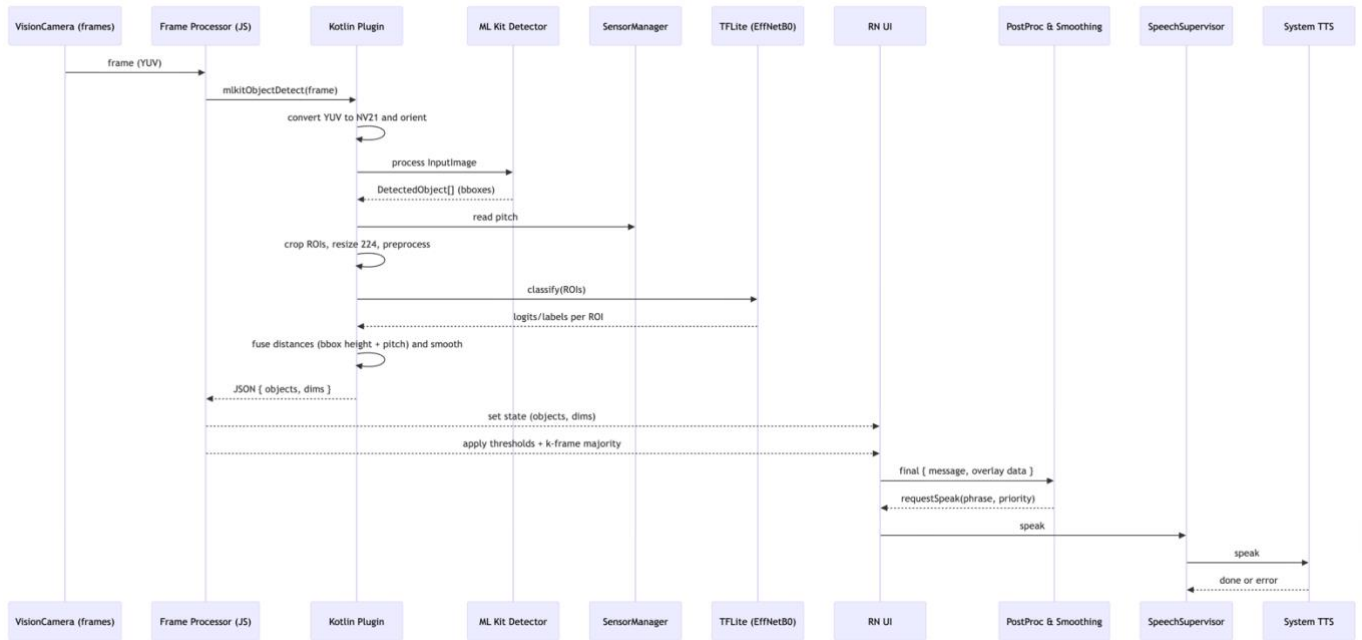


Figure 4. Live detection with speech

#### 4.2.2 System Architecture

The SEEiT app runs entirely on-device. The RN layer hosts the UI, user settings, and overlay rendering. An Android native layer (Kotlin) handles performance-critical work camera/IMU access, CV routines, and bridges to the operating system. The System TTS service delivers spoken feedback. Data flows one way from sensors → CV → app logic → TTS, with events and commands crossing the RN–native bridge. This design keeps latency low, preserves privacy (no cloud dependency), and remains robust offline (see Figure 5).

The pipeline is modular (see Figure 6):

- **Acquisition:** Camera stream and motion sensors are initialized in the native layer

- **Perception:** CV modules (e.g., color detection and scene classifiers) process frames on the native side to avoid JS-thread bottlenecks
- **Decision & Mapping:** Results are normalized, thresholds applied, and priorities resolved (e.g., “crosswalk ahead” over background cues)
- **App Integration:** Structured events are emitted to React Native for overlay drawing, haptics, and user controls; configuration changes flow back to native via the bridge
- **Speech:** A lightweight “speech supervisor” coalesces messages and calls System TTS to speak concise prompts

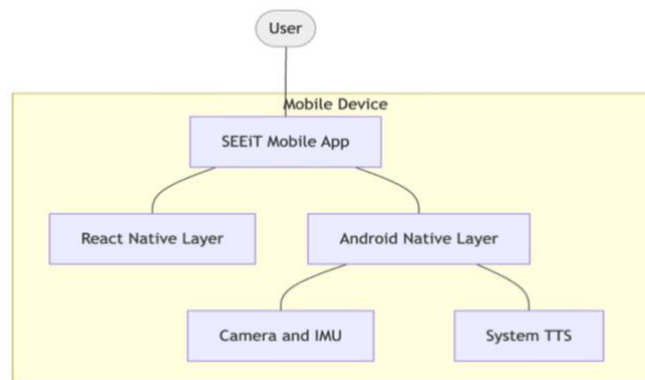


Figure 5. System context: on device, CV and system TTS

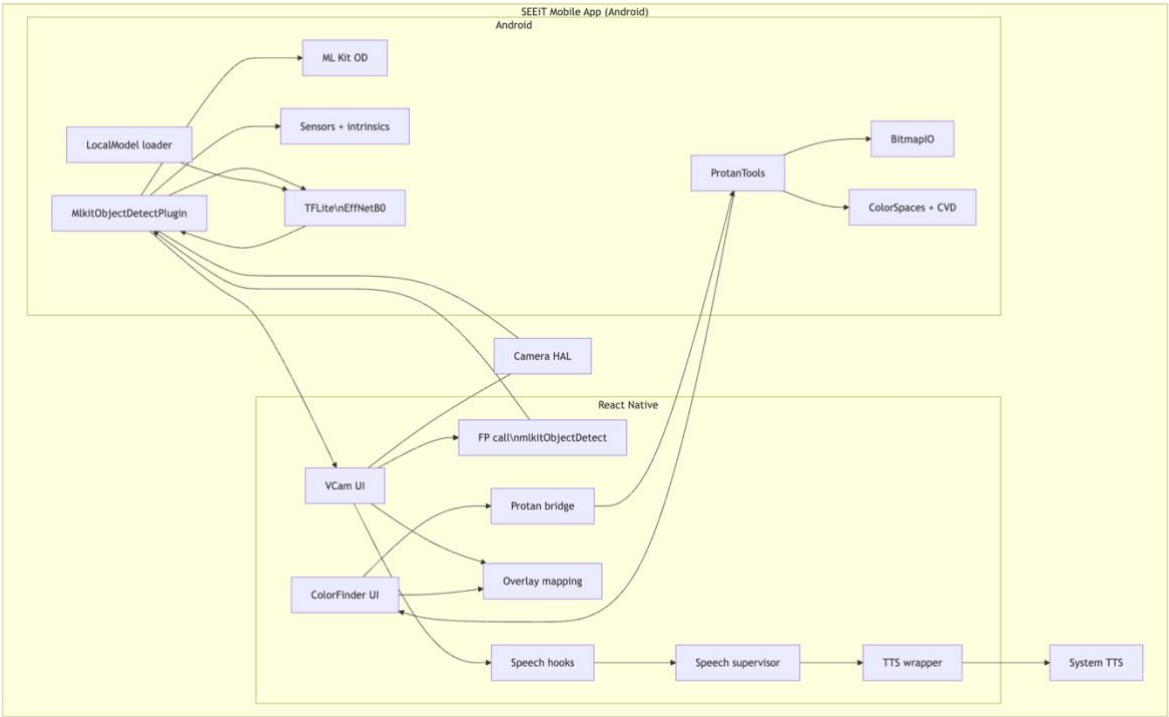


Figure 6. System architecture

### 4.3 Tech Stack

- GitHub
- EfficientNet
- Expo
- ML kit
- RN

### 4.4 UI/UX Design

The primary capture UI: live preview with quick chips for overlay/detection, a bottom action row buttons (mute/unmute speech, torch, resume/pause, language/TTS), and a menu for settings (see Figure 7).

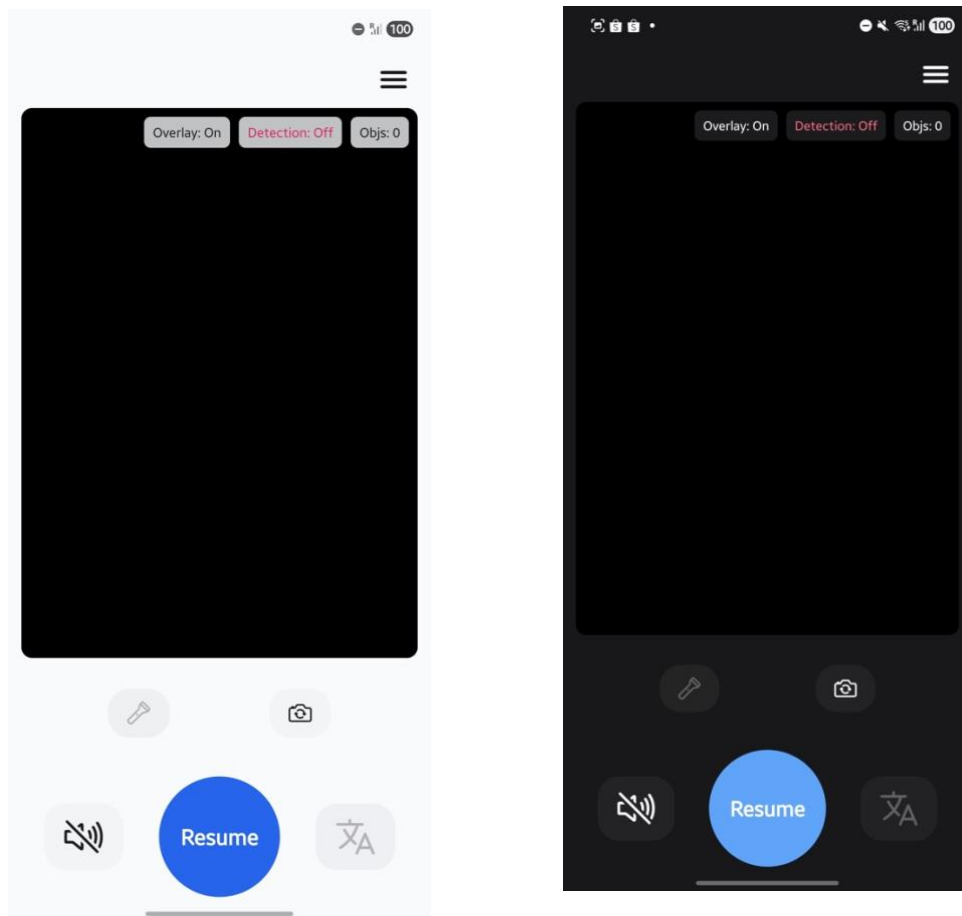
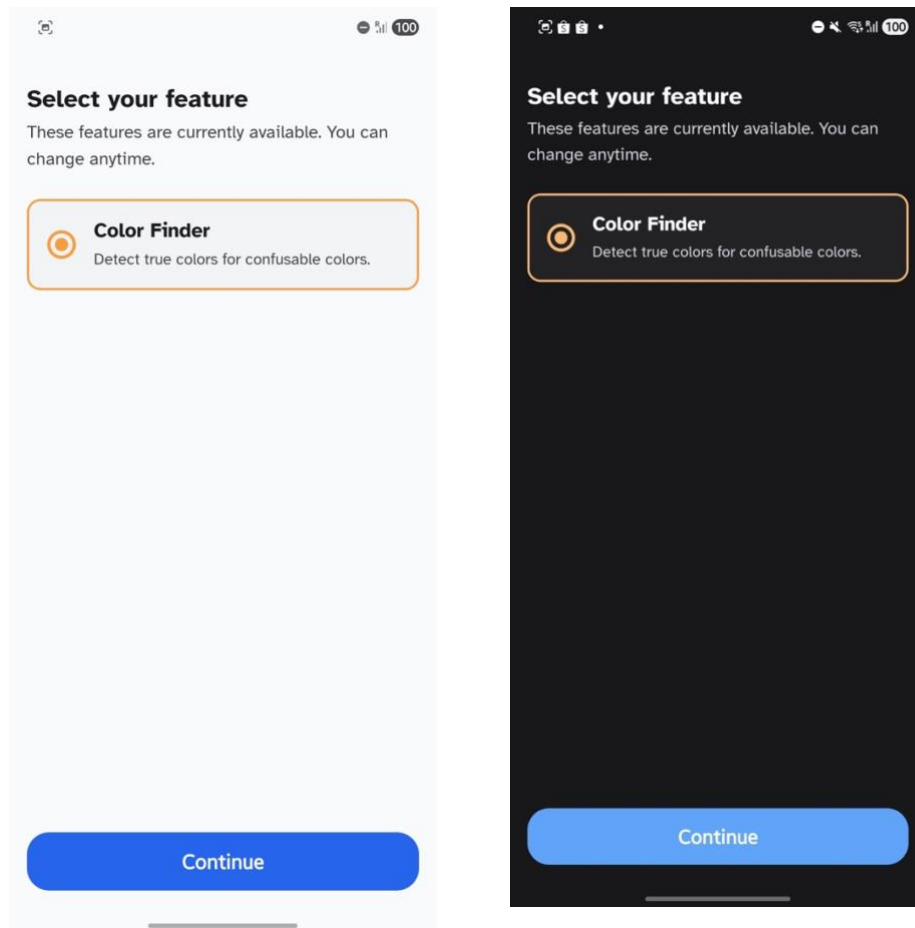


Figure 7. Main camera screen (light/dark theme)

Feature selector screen that highlights “Color Finder” as a card/tile with a brief description and tap-to-launch action – “Continue” button. Consistent icons and spacing make scanning fast (see Figure 8).



*Figure 8. Select feature screen (light/dark theme)*

An additional camera screen is dedicated to capturing or importing images, while reusing the primary controls for user familiarity (see Figure 9).

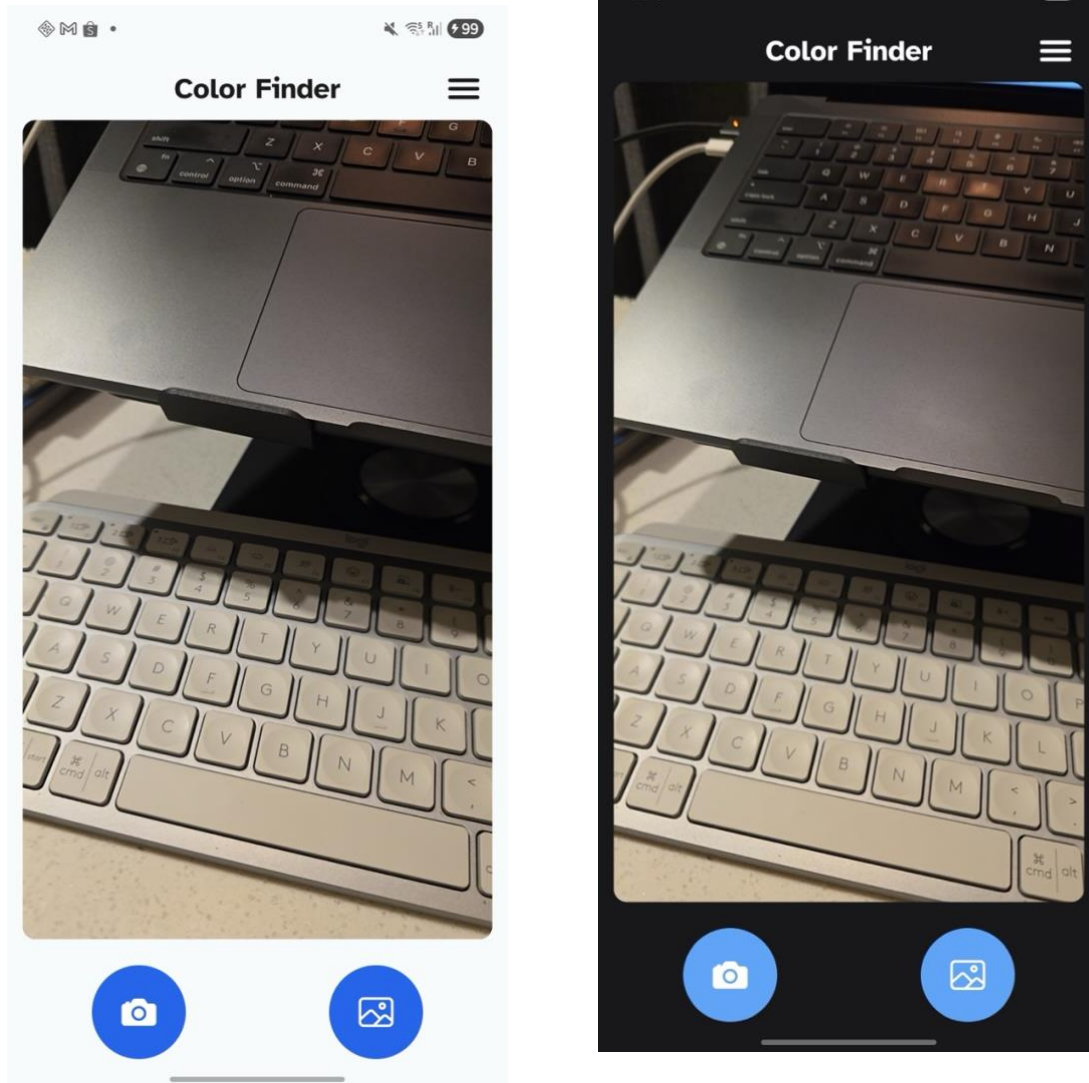


Figure 9. Camera screen for Color Finder (light/dark theme)

This screen displays real-time visual feedback as the system identifies, and labels detected colors. The detected color name and sample swatch appear on-screen, ensuring users can visually confirm results (see Figure 10).

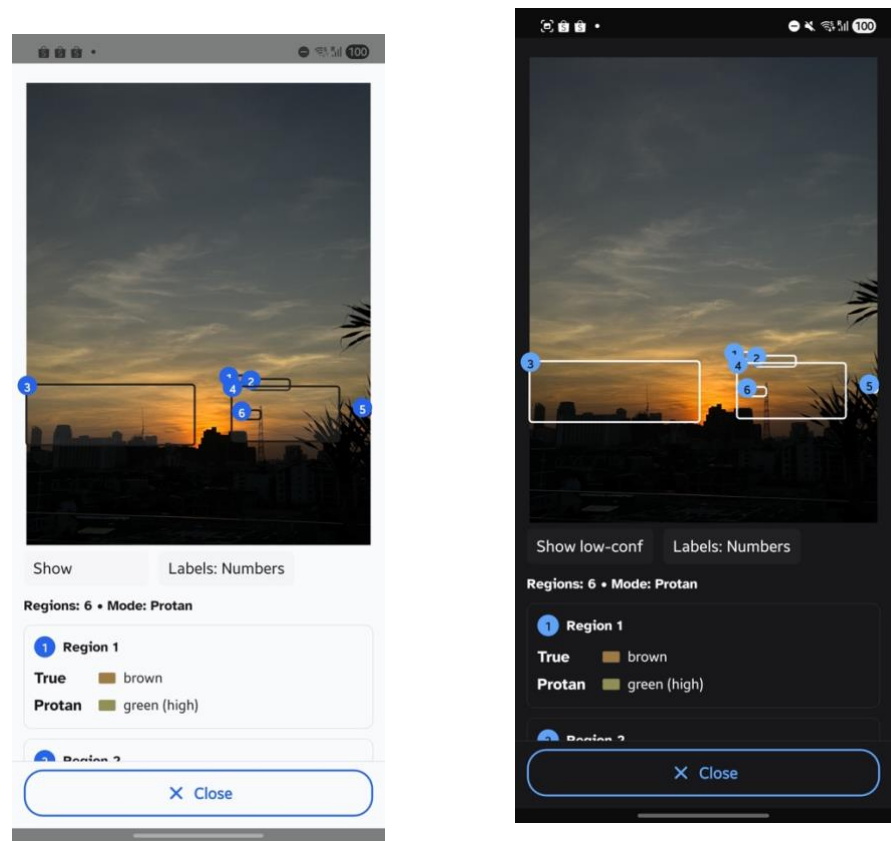
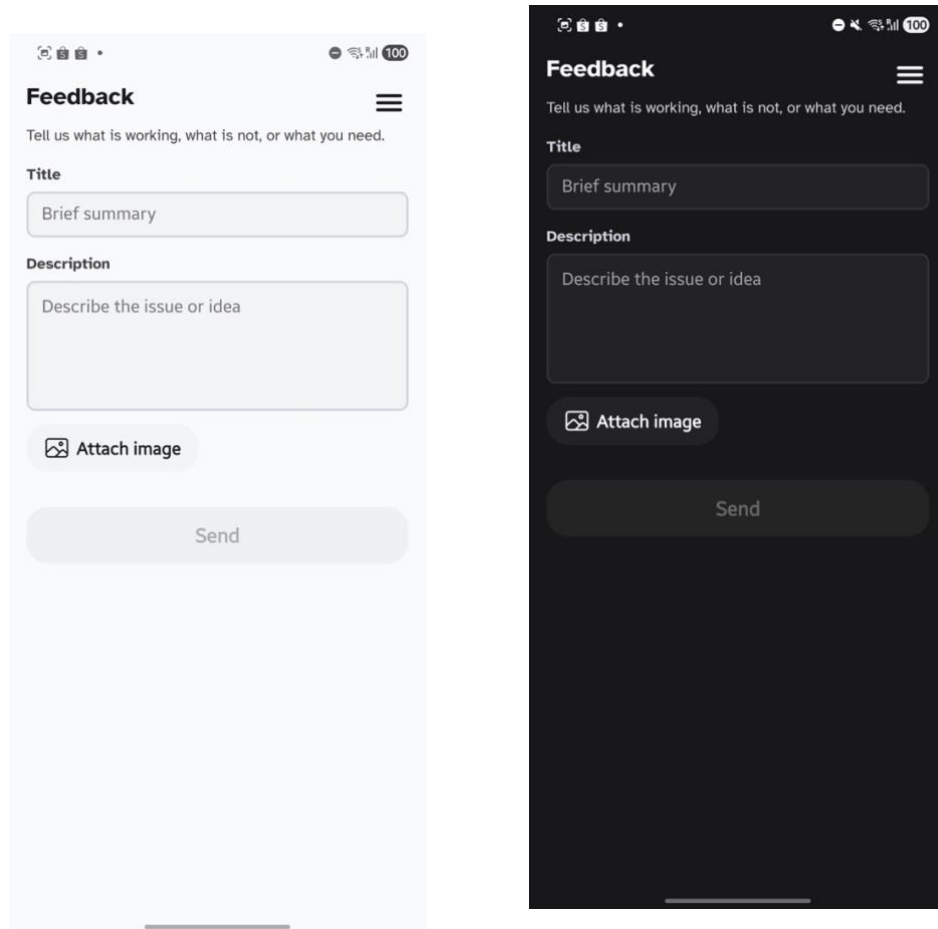


Figure 10. Color detection screen (light/dark theme)

This screen presents a simple interface for user evaluation of detection accuracy. Users can provide comments, or report issues directly by attaching screenshot images (see Figure 11).



The figure displays two versions of a mobile application's 'Feedback' screen, side-by-side, illustrating a light theme (left) and a dark theme (right). Both screens feature a status bar at the top with icons for signal, Wi-Fi, and battery (100%). The title 'Feedback' is prominently displayed at the top of each screen, accompanied by a hamburger menu icon on the right. Below the title, a subtitle reads 'Tell us what is working, what is not, or what you need.' The form consists of two main sections: 'Title' and 'Description'. The 'Title' section has a text input field with the placeholder 'Brief summary'. The 'Description' section has a larger text input field with the placeholder 'Describe the issue or idea'. Below the description field, there is an 'Attach image' button with a camera icon. At the bottom of the form is a 'Send' button. The light theme version has a white background with gray text and buttons, while the dark theme version has a black background with white text and buttons.

*Figure 11. Feedback screen (light/dark theme)*



## Chapter 5: Result

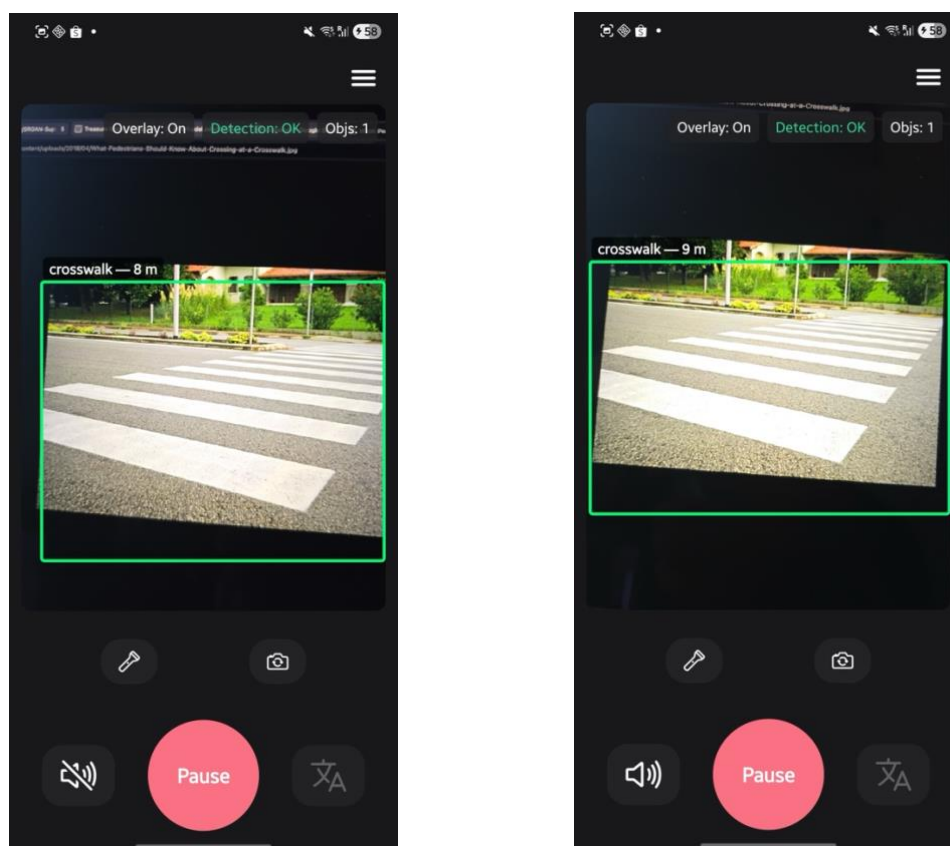
### 5.1 Demonstration of the Application

Figure 12 shows the main camera screen while the live feed is paused. The control row at the bottom includes: (1) Speech toggle (speaker icon) for the app’s own TTS summaries, (2) the central Resume button to start the live view, (3) Torch and Camera-switch controls, and (4) language change button which is not live now. At the top, status chips display Overlay (On/Off), Detection status, and the current object count. Accessibility announcements such as “Camera permission is required” are sent through Android’s accessibility channel and are only voiced if “TalkBack” is enabled; they are not the same as the in-app Speech button. When the user taps Resume, the live camera feed starts, and the detector activates (see Figure 12).



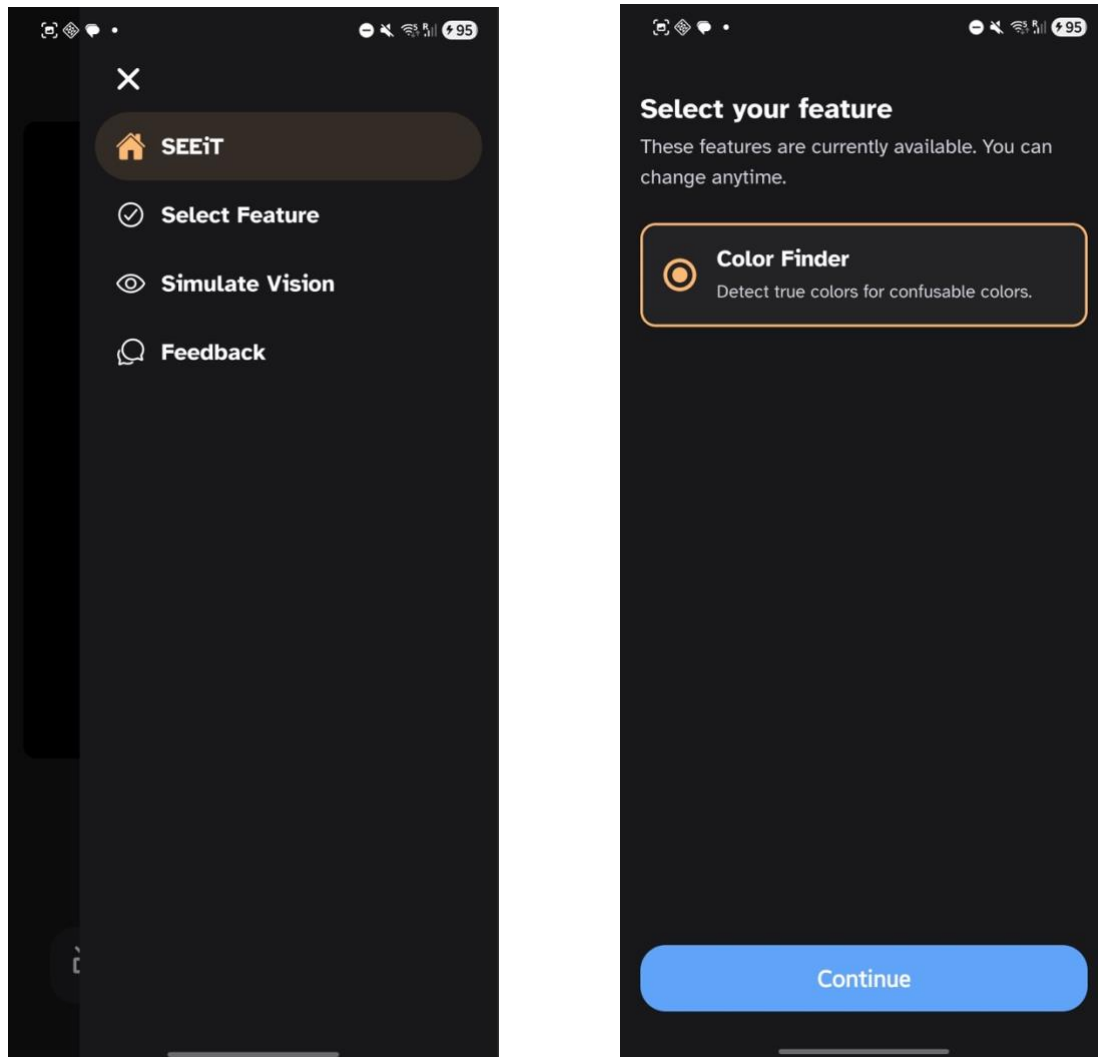
*Figure 12. Main camera screen with camera live feed paused*

Figure 13 illustrates the live camera feed after the user taps Resume (the central button becomes Pause). Detected objects are outlined and, when policy allows, labeled with smoothed distance in meters (see Appendix A.2, A.4). Distance is displayed and spoken only when confidence is medium or high and after smoothing to reduce jitter (see Appendix A.1). The app supports multiple simultaneous detections; to reduce cognitive load, in-app speech summarizes up to three label groups per utterance. Important distinction: the in-app Speech button controls our own TTS announcements (see Appendix A.3, A.5) (works even if TalkBack is off), while short system-level announcements (for example, “Live view resumed”) are routed via Android accessibility and require TalkBack to be on to be voiced (see Figure 13).



*Figure 13. Live camera feed with detection on (resume button triggered)*

Figure 14 shows navigation via the drawer menu (top-left menu icon). From here, the user opens the Select Feature screen and chooses among available modules. After confirming, the app routes to the chosen feature Color Finder in this demonstration. This navigation pattern keeps the critical live detection path one tap away while making secondary features discoverable without cluttering the camera UI (see Figure 14).



*Figure 14. Using drawer menu for navigation through the application*

Figure 15 demonstrates Color Finder in the default Protan mode (red deficiency). The interface presents two equal primary actions Capture (camera icon) and Import (image icon). After acquiring an image, the module analyzes it and highlights regions likely to be confusable under Protan vision (see Appendix B.1). On the analysis view: (a) overlaid boxes mark regions, (b) an optional label pill shows a stabilized coarse family shift, and (c) a legend lists each region's "True" (original) color family and "Protan" (simulated) family with a confidence tag (see Appendix B.2). By default, only high-confidence regions are shown; a "More/Fewer regions" toggle exposes additional findings. Short completion cues such as "Analysis complete" are accessibility announcements and require TalkBack to be enabled to be spoken.

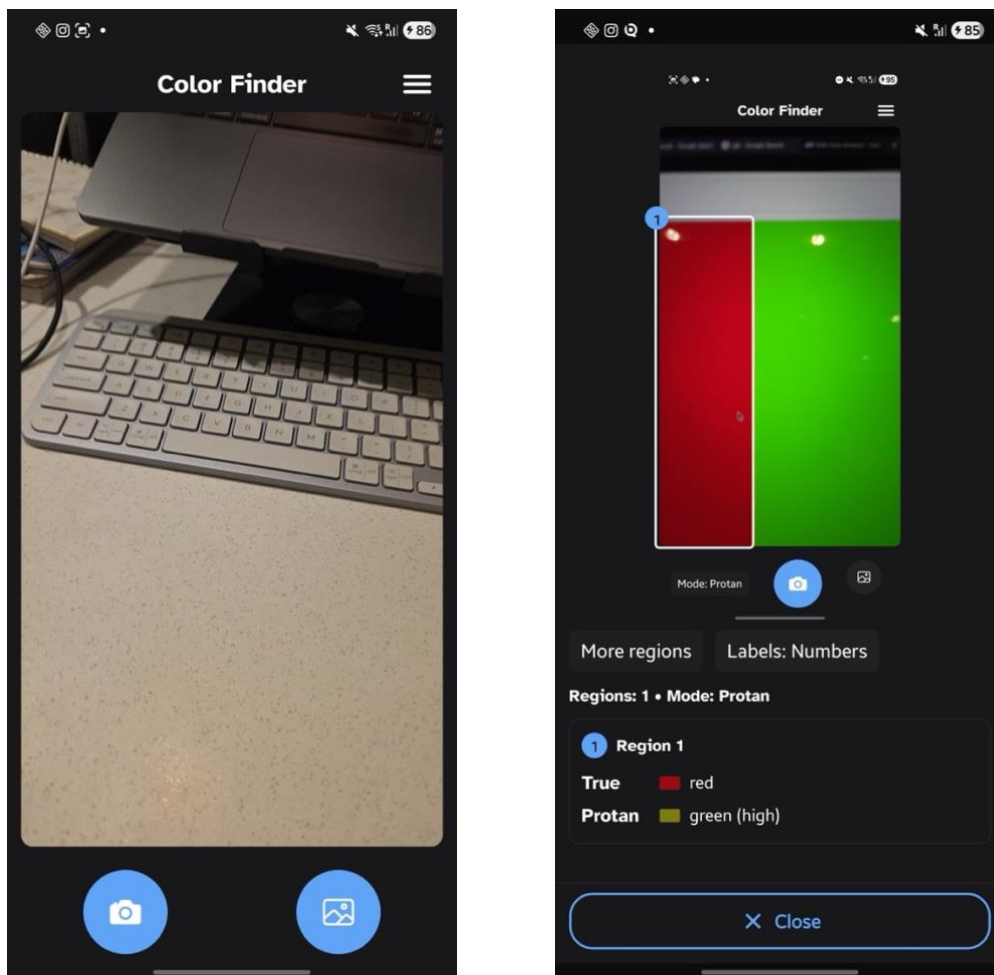
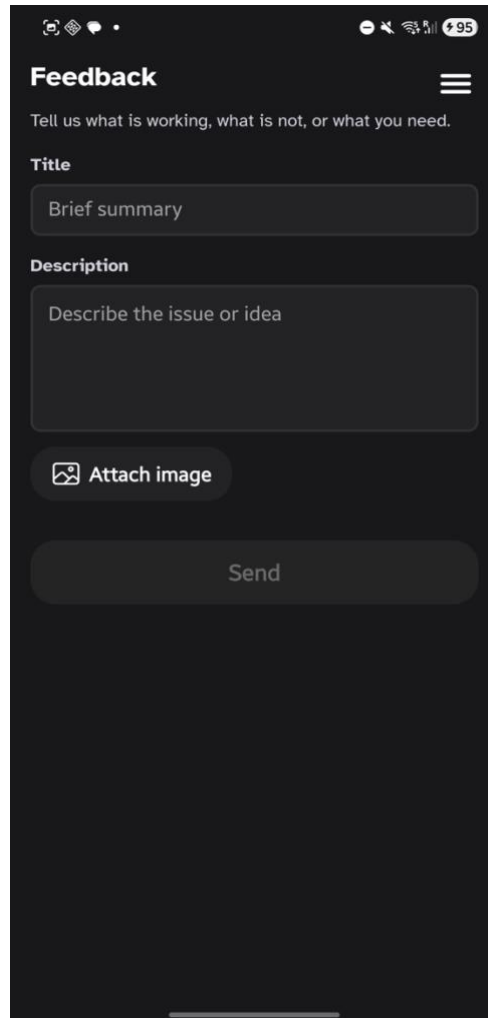


Figure 15. Color finder detecting color regions (on default protan mode) with captured image & analysis report

Figure 16 shows the optional Feedback screen. Users can enter a title and description and attach an image from the gallery, then tap Send button. The drawer menu provides consistent entry/exit without interfering with the live camera workflow.



*Figure 16. Feedback screen for users*

## 5.2 Quantitative Evaluation

The EfficientNet-B0 INT8 TFLite model was evaluated using on the test set of 769 images across 10 classes. The evaluation script generated a confusion matrix, per-class metrics, and overall summary values saved in the Evaluation folder (see Appendix C).

Overall performance of the model shown on Table 5 (see Appendix C.6).

*Table 5. Overall performance of the TFlite model*

Metric	Value (%)
Accuracy	90.51
Precision (macro)	88.88
Recall (macro)	95.27
F1-score (macro)	91.21

Table 6 describes class-wise performance on the held-out test set. As seen in Table 6, results are balanced across most categories. “car” and “crosswalk” achieve very high precision with strong F1, while the traffic-light classes (“traffic red/green/yellow”) show the expected pattern of high recall but comparatively lower precision—consistent with hue shifts and background reflections in bright scenes (see Appendix C.3).

*Table 6. Per-class performance on the test dataset*

Class	Precision	Recall	F1-Score	Support
car	1.00	0.76	0.86	268
crosswalk	0.87	1.00	0.93	131
cycle	0.95	1.00	0.98	21
emergency exit	0.97	0.97	0.97	63
stop	0.94	1.00	0.97	44
traffic green	0.68	0.98	0.80	52
traffic red	0.65	1.00	0.79	30
traffic yellow	0.88	0.85	0.86	26
truck	0.98	1.00	0.99	95
van	0.97	0.97	0.97	39

Figure 17 provides the confusion matrix for the INT8 EfficientNet-B0 model. As highlighted in Figure 17, the matrix exhibits strong diagonal dominance with a small set of

systematic confusions—primarily “car → traffic green” and within the traffic-light triad under high illumination—indicating that most errors are attributable to color/scene context rather than gross misrecognition (see Appendix C.4).

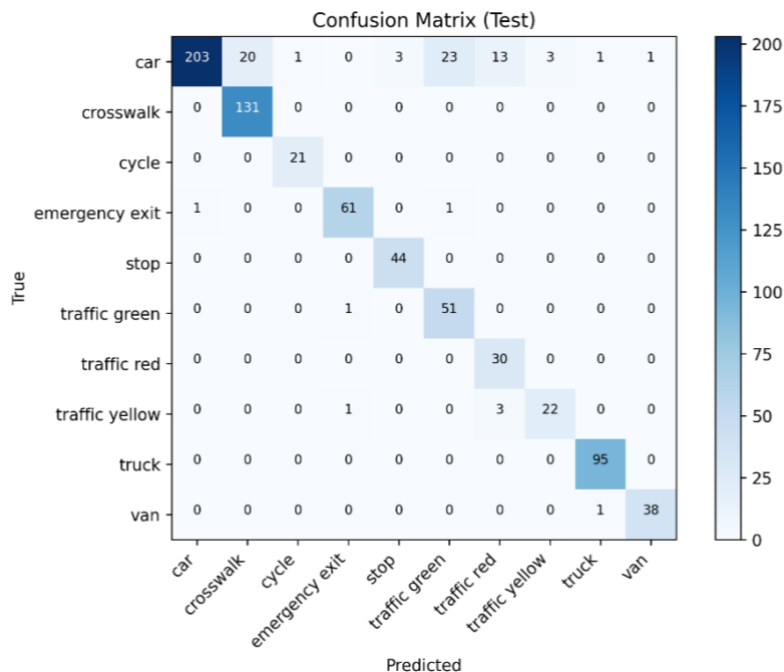


Figure 17. Confusion matrix for the INT8 EfficientNet-B0 model

Table 7 lists the most frequent confusion pairs. As Table 7 indicates, the majority involve contextual color influence (e.g., green casts on vehicle surfaces) or structured backgrounds (e.g., crosswalk patterns near vehicles). This aligns with live app behavior, where traffic-light colors are generally discriminated reliably once exposure stabilizes (see Appendix C.5).

Table 7. Top confusion pairs

True Label	Predicted Label	Count
car	traffic green	23
car	crosswalk	20
car	traffic red	13
traffic yellow	traffic red	4
car	stop	3

### 5.3 On-Device Performance (Latency / FPS)

Devices: Samsung Galaxy S25 Ultra, Xiaomi 14; EfficientNet-B0 INT8 (TFLite); confidence threshold = 0.65.

- Live detection and speech are generally responsive in steady scenes
- However, TTS may keep speaking previously detected objects (queued/duplicate prompts), so feedback can lag on fast-changing scenes
- After ~5 minutes of continuous use, device warming was noticeable on both phones, with no thermal warnings
- Frame rate is configured at 30 FPS max and 14 FPS min; actual FPS adapts with load and lighting. (See **Appendix A.6**)
- Android Logcat during testing showed ~28–30 FPS in steady scenes. (See **Appendix A.6**)

### 5.4 Error Analysis

Residual errors mainly occur when:

- strong sunlight or reflective glare shifts traffic-light hue (green ↔ yellow),
- multiple objects overlap heavily,
- vehicles partially occlude crosswalk markings.

Figure 18 and Table 6 confirm that these account for a small fraction of detections.

### 5.5 Summary

The INT8 EfficientNet-B0 classifier achieved 90% overall accuracy and 91% macro F1, indicating that quantization preserved high precision while meeting mobile efficiency. Evaluation files (classification\_report.txt, per\_class.csv, top\_confusions.csv, confusion\_matrix.png) are archived in the project repository.



## Chapter 6: Conclusion

This project is set out to develop a mobile application capable of assisting vision impaired and color-blind individuals through real-time object detection, color analysis, and audio feedback. The proposed method successfully addressed the initial problem by integrating computer vision and text-to-speech technologies into a single, accessible system. The app demonstrates the ability to accurately detect common road objects, identify crosswalks and highlight color regions that are challenging for individuals with red-green color vision deficiencies.

Through testing, the system achieves a certain level of reliable detection performance and responsive audio output, making it suitable for real-time use in outdoor pedestrian environments. Although the app has yet been tested with real target users, the results indicate potential to improve navigation safety and spatial awareness for those with vision impairments.

### Future work

To further enhance usability and inclusivity, these are the following proposed improvements:

- **Multilingual Support:** Implement speech output in multiple languages to support a wider range of users.
- **Expanded Color Blindness Coverage:** Extend color detection to include multiple types of color vision deficiencies beyond protan.

In conclusion, the proposed solution provides a foundation for inclusive assistive technology. With additional development and real-world testing, it can become a practical tool for independent and safe navigation.

## References

- [1] S. Real and A. Araujo, “Navigation Systems for the Blind and Visually Impaired: Past Work, Challenges, and Open Problems,” *Sensors*, vol. 19, no. 15, p. 3404, Aug. 2019.
- [2] L. P. Somers, A. Franklin, and J. M. Bosten, “Empirical tests of the effectiveness of EnChroma multi-notch filters for enhancing color vision in deuteranomaly,” *Vision Research*, vol. 218, p. 108390, Mar. 2024.
- [3] P. C. Latane, A. Bankar, S. Ahire, N. Wadekar, and M. Ghodkhnde, “Be My Eyes App for Blind People,” *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, vol. 2, no. 11, Nov. 2023.
- [4] S. Naotunna and B. Hettige, “Mobile Applications for Visually Impaired: A Review,” in *Proceedings of the International Conference on Emerging Trends in Engineering and Technology*, Jan. 2024.
- [5] J. D. Akkara and A. Kuriakose, “Smartphone apps for visually impaired persons,” *Kerala Journal of Ophthalmology*, vol. 31, no. 3, pp. 242–248, Sep.–Dec. 2019.
- [6] M. H. Abidi, A. N. Siddiquee, H. Alkhalefah, and V. Srivastava, “A comprehensive review of navigation systems for visually impaired individuals,” *Heliyon*, vol. 10, no. 11, e31825, May 2024, doi: 10.1016/j.heliyon.2024.e31825.

## Appendices

### Appendix A. Live Object Detection

#### A.1. Labels with smoothed distance in meters

To display and speech only when confidence  $\geq$  med:

```
// DetectionOverlay.tsx - UI label text with smoothed meters (display)
const { display } = smoothMeters(o.id, raw /* distance_m */, conf /*
'high'|'med'|'low' */, now)
const distText = typeof display === 'number' ? `${display} m` : ''

let text = lbl ? `${lbl.name}` : '-'
if (distText) text += ` - ${distText}` // meters appended only when smoothing
allows
```

Gate meters to confidence: allow when high, or med for critical/common traffic objects. This ensures we only speak distance when it's at least medium confidence.

```
// useLiveDetectionsSpeech.ts - spoken phrasing (speech channel)
const allowMeters =
  o.dm !== null &&
  !!o.dconf &&
  (o.dconf === 'high' || (o.dconf === 'med' && (CRITICAL_LABELS.has(nat) ||
MED_OK_LABELS.has(nat))))
const numTail = allowMeters ? `, about ${o.dm! < 10 ? o.dm!.toFixed(1) :
Math.round(o.dm!)} meters` : ''
```

Native gating + smoothing (data source for meters). Fuses ground-plane and pinhole estimates, then smooths by trackingId.

```
// MlkitObjectDetectPlugin.kt
if (doDistanceUpdate && bestNameNorm !== null &&
  bestConf >= DIST_MIN_LABEL_CONF &&
  DIST_CLASS_WHITELIST.contains(bestNameNorm))
) {
  val zPos = distanceGroundPlane(bottomY, uprightH)
  val zPin = distancePinhole(nh, uprightH, bestNameNorm!!)
  var fused = fuseDistances(zPos, zPin, bestNameNorm!!)
  if (fused.z > 0.0) {
    val smoothed = smoothDistance(tid, fused.z, nowMs) // native EMA cache
    distanceM = smoothed
    distanceSrc = fused.src
    distanceConf = fused.conf // 'high'|'med'|'low'
  }
}
```

## A.2. Distance calculation on detected objects (Kotlin)

Ground-plane and pinhole estimates, fusion with confidence, per-ID smoothing, serialize into objects:

```
// MlkitObjectDetectPlugin.kt- distance calculation
private data class DistEntry(var valueM: Double, var lastMs: Long)
private val distanceCache = mutableMapOf<Int, DistEntry>()

// Ground-plane using camera pitch and intrinsics (fyPx, cyPx), units: meters
private fun distanceGroundPlane(bottomYNorm: Double, imageH: Int): Double {
    if (fyPx <= 0.0 || imageH <= 0) return -1.0
    val vb = bottomYNorm * imageH; val yh = cyPx + fyPx * tan(pitchRad); val denom =
vb - yh
    if (denom <= 2.0) return -1.0
    val z = camHeightM * fyPx / denom
    return if (z.isFinite() && z in 0.3..40.0) z else -1.0
}

// Pinhole using canonical class height
private fun distancePinhole(boxHNorm: Double, imageH: Int, nat: String): Double {
    val prior = CLASS_HEIGHTS_M[nat] ?: return -1.0
    if (fyPx <= 0.0 || imageH <= 0) return -1.0
    val hpx = boxHNorm * imageH; if (hpx < 12) return -1.0
    val z = fyPx * prior / hpx
    return if (z.isFinite() && z in 0.3..50.0) z else -1.0
}

private data class Fused(val z: Double, val src: String, val conf: String)
private fun fuseDistances(zPos: Double, zPin: Double, nat: String): Fused {
    val pos = zPos > 0; val pin = zPin > 0
    if (pos && pin) {
        val r = kotlin.math.abs(zPos - zPin) / max(zPos, zPin)
        if (r <= 0.20) return Fused((zPos + zPin) * 0.5, "fused", "high")
        val preferPin = CLASS_HEIGHTS_M.containsKey(nat)
        return if (r <= 0.40) Fused(if (preferPin) zPin else zPos, if (preferPin)
"pinhole" else "ground", "med")
        else Fused(if (preferPin) zPin else zPos, if (preferPin) "pinhole" else
"ground", "low")
    }
    if (pos) return Fused(zPos, "ground", "med")
    if (pin) return Fused(zPin, "pinhole", if (CLASS_HEIGHTS_M.containsKey(nat))
"med" else "low")
    return Fused(-1.0, "none", "low")
}

private fun smoothDistance(tid: Int, raw: Double, nowMs: Long): Double {
    if (tid < 0 || raw <= 0.0) return raw
    // purge
    distanceCache.entries.removeIf { nowMs - it.value.lastMs > DIST_CACHE_TTL_MS }
    val prev = distanceCache[tid]
    val z = if (prev == null) raw else DIST_SMOOTH_ALPHA * raw + (1 -
DIST_SMOOTH_ALPHA) * prev.valueM
    distanceCache[tid] = DistEntry(z, nowMs); return z
}
```

```
// In serialize(): compute and attach distance fields (distance_m, distance_src,
distance_conf)
val doDistanceUpdate = (detSeq % 2L == 0L)
if (doDistanceUpdate && bestNameNorm != null && bestConf >= DIST_MIN_LABEL_CONF &&
DIST_CLASS_WHITELIST.contains(bestNameNorm)) {
  val bottomY = (ny + nh).coerceIn(0.0, 1.0)
  val zPos = distanceGroundPlane(bottomY, uprightH)
  val zPin = distancePinhole(nh, uprightH, bestNameNorm!!)
  var fused = fuseDistances(zPos, zPin, bestNameNorm!!)
  if (fused.z > 0.0) {
    val smoothed = smoothDistance(tid, fused.z, System.currentTimeMillis())
    out["distance_m"] = smoothed; out["distance_src"] = fused.src;
  }
  out["distance_conf"] = fused.conf
}
```

### A.3. Display and Speech function for Calculated Distance

Distance is displayed and spoken only when confidence is medium or high (after smoothing):

```
// useDistanceMetersSmoothing.ts – Distance displayed with smoothing
type Conf = 'high'|'med'|'low'|undefined
type Entry = { ema: number; shown?: number; conf?: Conf; lastSeen: number }
const ALPHA = 0.35, TTL_MS = 9000, LOW_CONF_HIDE = true
const MIN_DELTA_NEAR = 0.5, MIN_DELTA_FAR = 1.0
const quantize = (v:number) => v < 10 ? Math.round(v*2)/2 : Math.round(v)
const thrFor = (v:number) => v < 10 ? MIN_DELTA_NEAR : MIN_DELTA_FAR

export function useDistanceMetersSmoothing() {
  const mapRef = React.useRef<Map<number, Entry>>(new Map())
  const smoothMeters = React.useCallback((id:number, raw?:number, conf?:Conf,
now=Date.now()) => {
    if (id==null || id<0 || typeof raw!=='number' || raw<=0) return { display:
undefined, conf: undefined }
    if (LOW_CONF_HIDE && conf==='low') {
      mapRef.current.get(id)?.let(e=>e.lastSeen=now); return { display: undefined, conf
} }
    let e = mapRef.current.get(id)
    if (!e) { const q = quantize(raw); e = { ema: raw, shown: q, conf, lastSeen:
now }; mapRef.current.set(id, e); return { display:q, conf } }
    e.lastSeen = now; e.ema = ALPHA*raw + (1-ALPHA)*e.ema; e.conf = conf
    const desired = quantize(e.ema), prev = e.shown
    if (prev==null || Math.abs(desired - prev) >= thrFor(Math.min(prev, desired)))
e.shown = desired
    return { display: e.shown, conf: e.conf }
  }, [])
  const purgeDistCache = React.useCallback((now=Date.now()) => {
    for (const [id, e] of mapRef.current) if (now - e.lastSeen > TTL_MS)
mapRef.current.delete(id)
  }, [])
  return { smoothMeters, purgeDistCache, reset: () => mapRef.current.clear() }
}
```

#### A.4. Real-life object detections (Kotlin)

ML Kit Object Detection + classification model (local):

```
// MlkitObjectDetectPlugin.kt – detector setup (local custom model + streaming
mode)
private val detector: ObjectDetector by lazy {
    val localModel = com.google.mlkit.common.model.LocalModel.Builder()
        .setAssetFilePath("effb0_meta.tflite") // bundled classifier
        .build()
    val opts = CustomObjectDetectorOptions.Builder(localModel)
        .setDetectorMode(CustomObjectDetectorOptions.STREAM_MODE)
        .enableMultipleObjects()
        .enableClassification()
        .setClassificationConfidenceThreshold(CONF_THRESHOLD) // e.g., 0.65
        .setMaxPerObjectLabelCount(MAX_LABELS) // e.g., 5
        .build()
    ObjectDetection.getClient(opts)
}
```

Frame -> Detection -> Normalized result:

```
// MlkitObjectDetectPlugin.kt
val input = InputImage.fromByteArray(nv21, rawW, rawH, rot,
InputImage.IMAGE_FORMAT_NV21)
detector.process(input)
    .addOnSuccessListener { objs ->
        // For each DetectedObject:
        // 1) normalize box to [0..1], 2) normalize labels, 3) compute/attach smoothed
meters if eligible.
        val serialized = serialize(objs, rawW, rawH, rot, durMs, seq, timestamp)
        lastResult.set(serialized)
    }
```

#### A.5. In-app Speech button controls our own TTS announcements

This controls our TTS pipeline (useSpeechChannel + useLiveDetectionsSpeech).

```
// index.tsx – Speech toggle (in-app channel)
const [speechOn, setSpeechOn] = React.useState(DEFAULT_SPEECH_ON)

const { requestSpeak } = useSpeechChannel({ active: speechOn && isActive &&
detectionStatus === 'ok', ... })

useLiveDetectionsSpeech({
    enabled: speechOn && detectionStatus === 'ok', // drive speaking based on
toggle + detector state
    objects,
    requestSpeak,
})

const toggleSpeech = () => {
    setSpeechOn(prev => !prev)
}
```

```

    AccessibilityInfo.announceForAccessibility(speechOn ? 'Speech off' : 'Speech
on') // system ally cue; requires TalkBack to be heard
}

```

```

// tts.ts – minimal speaking primitive (interruptible; tracks done/error)
export function ttsSpeak(text: string, opts: SpeakOpts = {}) {
  const utteranceId = opts.utteranceId || 'utt_' +
Math.random().toString(36).slice(2, 10)
  Tts.stop().catch(() => {}) // optional: interrupt current
  Tts.speak(text, { utteranceId, rate: opts.rate, pitch: opts.pitch } as any)
  // resolve on 'tts-finish' or 'tts-error' events
}

```

## A.6. Runtime Diagnostics: FPS & Camera Format Logging

```

// index.tsx
const frameTimesRef = React.useRef<number[]>([]);

const setPluginResultOnJS = Worklets.useRunOnJS((raw: any) => {
  if (!raw || typeof raw.detSeq !== 'number') return;

  // track last detection timestamp for existing logic
  if (raw.detSeq > 0) setLastDetTs(Date.now());
  setFpError(null);

  // compute measured FPS from arrival times
  const now = Date.now();
  const times = frameTimesRef.current;
  times.push(now);
  // keep last N samples (e.g. 30)
  if (times.length > 30) times.shift();
  frameTimesRef.current = times;
  if (times.length >= 2) {
    const durationSec = (times[times.length - 1] - times[0]) / 1000;
    const measuredFps = durationSec > 0 ? (times.length - 1) / durationSec :
0;
    console.log('Measured FPS:', measuredFps.toFixed(1));
  }

  const objs = Array.isArray(raw.objs) ? raw.objs : [];
  setObjects(objs);
  setFrameDims({
    width: typeof raw.width === 'number' ? raw.width : 0,
    height: typeof raw.height === 'number' ? raw.height : 0,
  });
}, []);

```

## Appendix B. Color Finder

### B.1. Color Finder — detect color regions (Kotlin)

Steps: decode+orient → per-pixel simulation+mask → dilate → extract components →  
prune+stabilize+merge → emit boxes:

```
// ProtanToolsModule.kt – detectConfusableColors() core (high-level)
val bmp = decodeDownscaled(inputUri, if (maxSide > 0) maxSide else 480) ?: return
reject(...)
val (w, h, total) = Triple(bmp.width, bmp.height, bmp.width * bmp.height)

val minSatGen = max(minSat.toFloat(), MIN_SAT_CLAMP)
val maskProtan = BooleanArray(total)

// 1) Per-pixel Protan simulation + rule-based “hit”
for (i in 0 until total) {
    val x = i % w; val y = i / w; val p = bmp.get(x, y)
    val rs = Color.red(p)/255f; val gs = Color.green(p)/255f; val bs =
Color.blue(p)/255f
    val hsv0 = rgbToHsv(rs, gs, bs)
    val fam0 = hueFamilyRGB(rs, gs, bs)
    val sp = CvdSimulation.simulate(floatArrayOf(rs, gs, bs), "protanopia")
    val famP = hueFamilyRGB(sp[0], sp[1], sp[2])
    val dE00p = ColorSpaces.deltaE00(ColorSpaces.srgbToLab(rs, gs, bs),
ColorSpaces.srgbToLab(sp[0], sp[1], sp[2]))
    val passSV = (hsv0.s >= minSatGen) && (hsv0.v.toDouble() >= minVal)

    var hit = false
    // red/magenta/purple → big ΔE or dark; warm/dark flips; purple/magenta → blue
when saturated
    if ((fam0 == "red" || fam0 == "magenta" || fam0 == "purple") && (hsv0.v <= 0.30f
|| dE00p >= DE_HIGH)) hit = true
    else if (hsv0.v < 0.40f && (fam0 == "brown" || fam0 == "green" || fam0 ==
"orange") && famP != fam0 && dE00p >= DE_MED) hit = true
    else if ((fam0 in arrayOf("red", "green", "brown")) && (famP in
arrayOf("red", "green", "brown")) && famP != fam0 && dE00p >= DE_HIGH) hit = true
    if (passSV && (fam0 == "purple" || fam0 == "magenta") && famP == "blue") hit =
true

    if (hit) maskProtan[i] = true
}

// 2) Dilate + 3) extract connected components with stats → 4) prune+stabilize
(red↔brown, blue↔purple) → 5) merge similar boxes
// ... (see full file for BFS extraction and merging)

// 6) Emit normalized boxes + mean colors + families + confidence
map.putDouble("x", minX.toDouble() / w); map.putDouble("y", minY.toDouble() / h)
map.putDouble("w", (maxX - minX + 1).toDouble() / w); map.putDouble("h", (maxY -
minY + 1).toDouble() / h)
map.putString("trueFamily", familyMean)
map.putString("simFamilyProtan", hueFamilyRGB(meanPR/255f, meanPG/255f,
meanPB/255f))
map.putString("confProtan", confProtan) // 'high'|'med'|'low'
```



## B.2. Analysis view for Color Finder

Analysis view: legend lists each region's True vs Protan with confidence tag

```
// colorBlindCameraScreen.tsx
type ConfLevel = 'low'|'med'|'high'
type ConfRegion = { meanR:number; meanG:number; meanB:number; meanProtanR:number;
meanProtanG:number; meanProtanB:number; trueFamily:string; dominantFamily?:string;
simFamilyProtan:string; confProtan:ConfLevel; x:number; y:number; w:number;
h:number; areaFrac:number }

const coarseFamily = (name: string) => {
  const n = (name||'').toLowerCase()
  if (['red','brown','pink','orange','magenta'].includes(n)) return 'warm'
  if (['blue','purple'].includes(n)) return 'blueish'
  if (['black','white','grey','gray'].includes(n)) return 'neutral'
  if (['yellow','green','cyan'].includes(n)) return n
  return n || '-'
}

const confTag = (lvl?: ConfLevel) => (lvl ? ` (${lvl})` : '')

const buildPillLabel = (cr: ConfRegion) => {
  const domC = coarseFamily(cr.dominantFamily || cr.trueFamily || '')
  const simC = coarseFamily(cr.simFamilyProtan || '')
  if (!cr.simFamilyProtan || cr.confProtan === 'low') return domC ||
(cr.dominantFamily || cr.trueFamily || '-')
  return `${domC} → ${simC}${cr.confProtan === 'high' ? ' ↑' : ''}`
}

// In legend card:
<Line label="True" swatch={`rgb(${cr.meanR},${cr.meanG},${cr.meanB})`}
text={`(cr.dominantFamily || cr.trueFamily || '-')`} />
<Line label="Protan"
swatch={`rgb(${cr.meanProtanR},${cr.meanProtanG},${cr.meanProtanB})`}
text={`${cr.simFamilyProtan || '-'}${confTag(cr.confProtan)}`} />
```

## Appendix C. Metric Evaluation (metrics.py)

### C.1. Model Loading and Pre-processing Evaluation Loop and Metric Computation

```
# Load INT8 EfficientNet-B0 TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path=MODEL_TFLITE)
interpreter.allocate_tensors()
in_det = interpreter.get_input_details()[0]
out_det = interpreter.get_output_details()[0]
# Pre-process image → uint8 tensor (0-255) expected by INT8 model.
def preprocess(path):
    im = Image.open(path).convert("RGB").resize((IMG_SIZE, IMG_SIZE))
    x = np.expand_dims(np.array(im, dtype=np.uint8), 0)
    return x
# Run inference on one image and return logits.
def predict(path):
    x = preprocess(path)
    interpreter.set_tensor(in_det["index"], x)
    interpreter.invoke()
    return interpreter.get_tensor(out_det["index"])[0]
```

### C.2. Evaluation Loop and Metric Computation

```
# Iterate over all test images and collect predictions.
y_true = [LABELS.index(lbl) for lbl in df["label"]]
y_pred = [int(np.argmax(predict(p))) for p in df["filepath"]]

# Compute accuracy, precision, recall, and F1 (macro average).
acc = accuracy_score(y_true, y_pred)
p, r, f1, _ = precision_recall_fscore_support(
    y_true, y_pred, average="macro", zero_division=0
)
```

### C.3. Per-Class Metrics

```
# Build structured per-class table (precision/recall/F1/support).
rep_dict = classification_report(
    y_true, y_pred, target_names=LABELS, output_dict=True, zero_division=0
)
rows = []
for name in LABELS:
    d = rep_dict.get(name, {})
    rows.append({
        "class": name.replace("_", " "),
        "precision": d.get("precision", 0.0),
        "recall": d.get("recall", 0.0),
        "f1": d.get("f1-score", 0.0),
        "support": d.get("support", 0)
    })
pd.DataFrame(rows).to_csv(out_dir / "per_class.csv", index=False)
```

## C.4. Confusion Matrix

```
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap="Blues")
plt.title("Confusion Matrix (Test)")
plt.colorbar()
plt.xticks(range(NUM_CLASSES), [c.replace("_", " ") for c in LABELS], rotation=45)
plt.yticks(range(NUM_CLASSES), [c.replace("_", " ") for c in LABELS])

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j],
             ha="center",
             color="white" if cm[i, j] > cm.max()/2 else "black",
             fontsize=8)

plt.xlabel("Predicted"); plt.ylabel("True")
plt.tight_layout()
plt.savefig(out_dir / "confusion_matrix.png", dpi=250)
plt.close()
```

## C.5. Top Confusion Pairs

```
top = []
for i in range(NUM_CLASSES):
    for j in range(NUM_CLASSES):
        if i != j and cm[i, j] > 0:
            top.append((LABELS[i].replace("_", " "),
                       LABELS[j].replace("_", " "),
                       int(cm[i, j])))

pd.DataFrame(sorted(top, key=lambda x: x[2], reverse=True),
             columns=["true", "predicted", "count"]
).to_csv(out_dir / "top_confusions.csv", index=False)
```

## C.6. Summary Export

```
result = {
    "accuracy": round(acc * 100, 2),
    "precision": round(p * 100, 2),
    "recall": round(r * 100, 2),
    "f1": round(f1 * 100, 2)
}
with open(out_dir / "summary.json", "w") as f:
    json.dump(result, f, indent=2)
```