Real time image segmenting red, green, blue, and added yellow

Made it define color boundaries with hsv instead since it can distinct the colors better

```
import cv2
import numpy as np

# Define color boundaries in HSV space
boundaries = {
    "Red": ([0, 120, 70], [10, 255, 255]),          # Red (lower range)
    "Red2": ([170, 120, 70], [180, 255, 255]),      # Red (upper range)
    "Green": ([36, 25, 25], [86, 255, 255]),        # Green
    "Blue": ([94, 80, 2], [126, 255, 255]),         # Blue
    "Yellow": ([15, 150, 150], [35, 255, 255])      # Yellow
}

# Normalize image for display
def normalizeImg(Img):
    Img = np.float64(Img)
    norm_img = (Img - np.min(Img)) / (np.max(Img) - np.min(Img))
    norm_img = np.uint8(norm_img * 255.0)
    return norm_img

# Open webcam
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    raise IOError("Cannot open webcam")

while True:
    ret, frame = cap.read()

    # Convert frame to HSV color space
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    output = []

    # Apply masks for each color
    for color_name, (lower, upper) in boundaries.items():
        lower_bound = np.array(lower, dtype="uint8")
        upper_bound = np.array(upper, dtype="uint8")

        mask = cv2.inRange(hsv_frame, lower_bound, upper_bound)
        segmented = cv2.bitwise_and(frame, frame, mask=mask)
        output.append(normalizeImg(segmented))

    # Handle both red ranges (combine them)
    red_combined = cv2.add(output[0], output[1])  # Red + Red2
    green_img = output[2]
```

```
    blue_img = output[3]
    yellow_img = output[4]

    # Concatenate images for display
    catImg = cv2.hconcat([frame, red_combined, green_img, blue_img,
yellow_img])
    cv2.imshow("Original | Red | Green | Blue | Yellow", catImg)

    # Exit condition
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

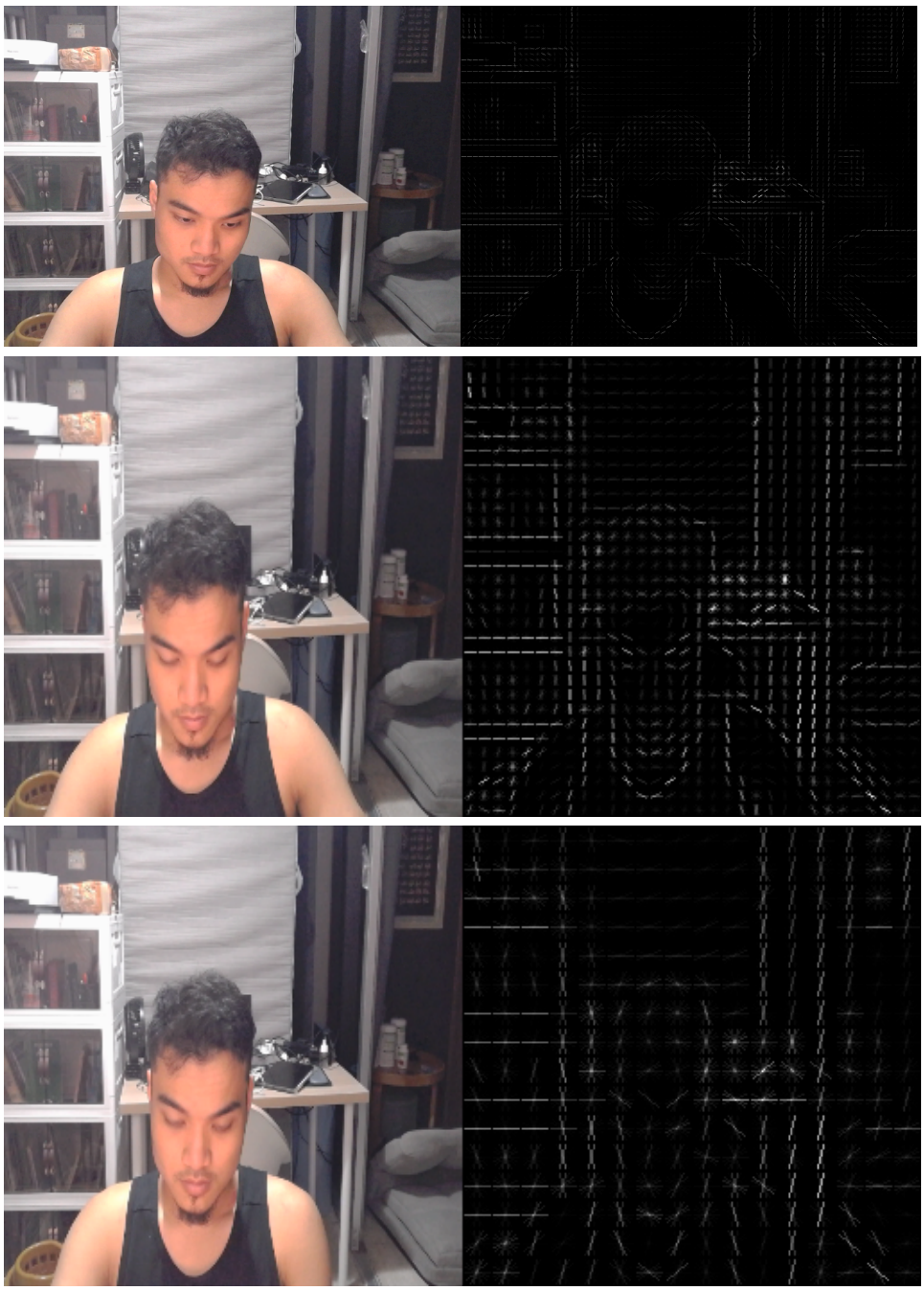
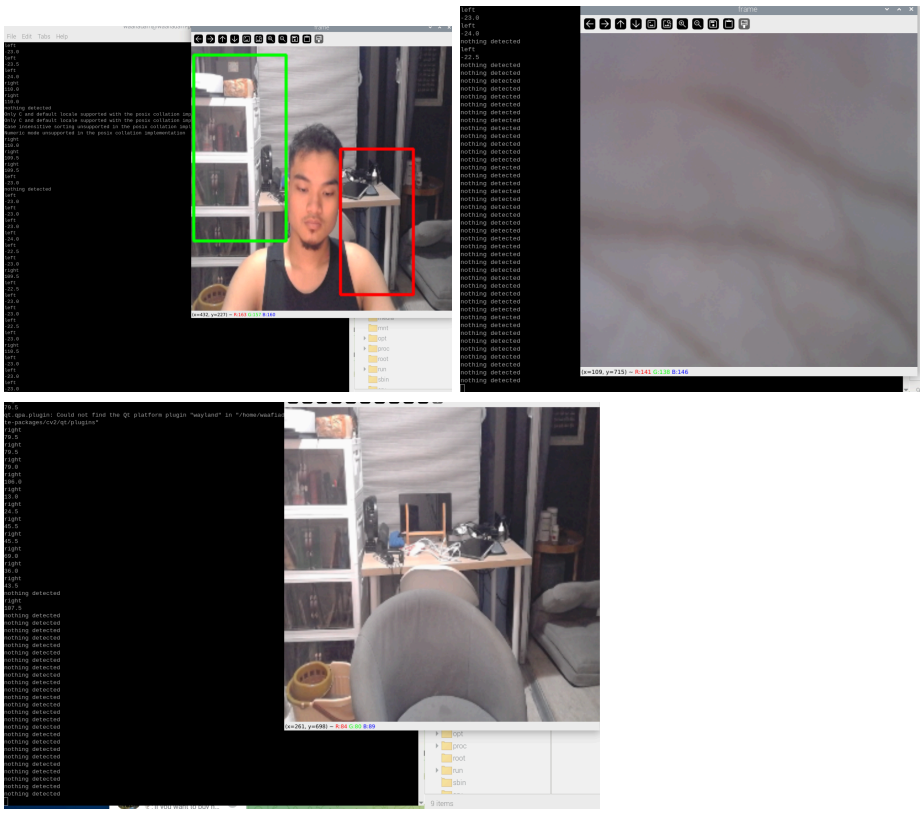
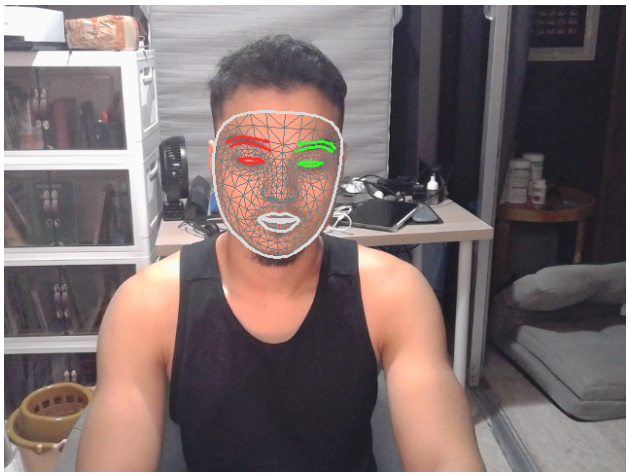
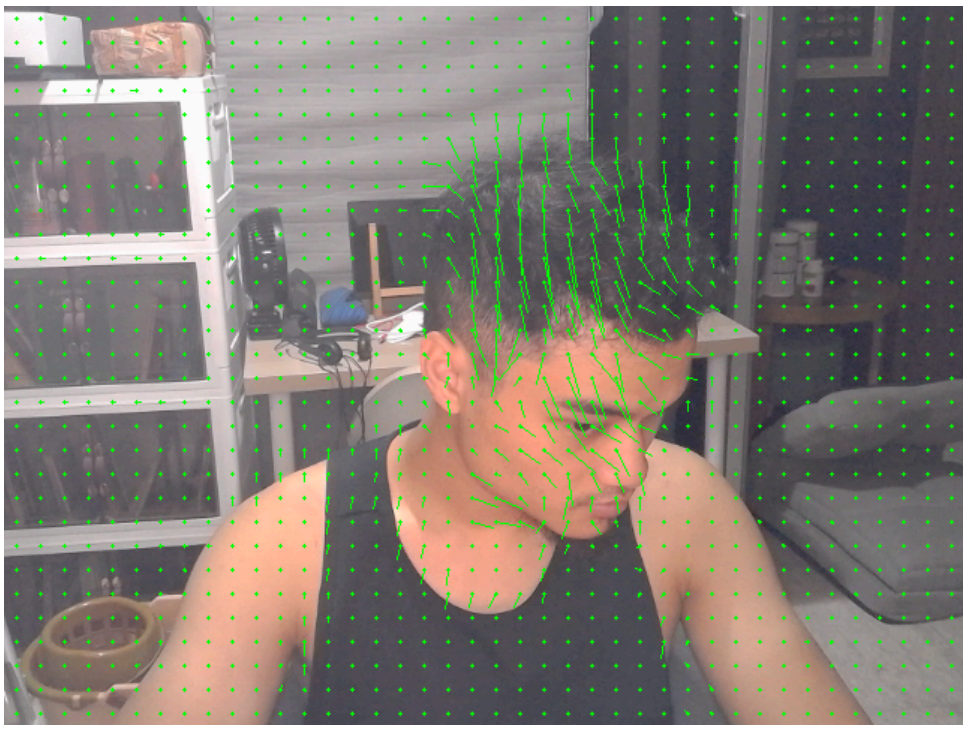
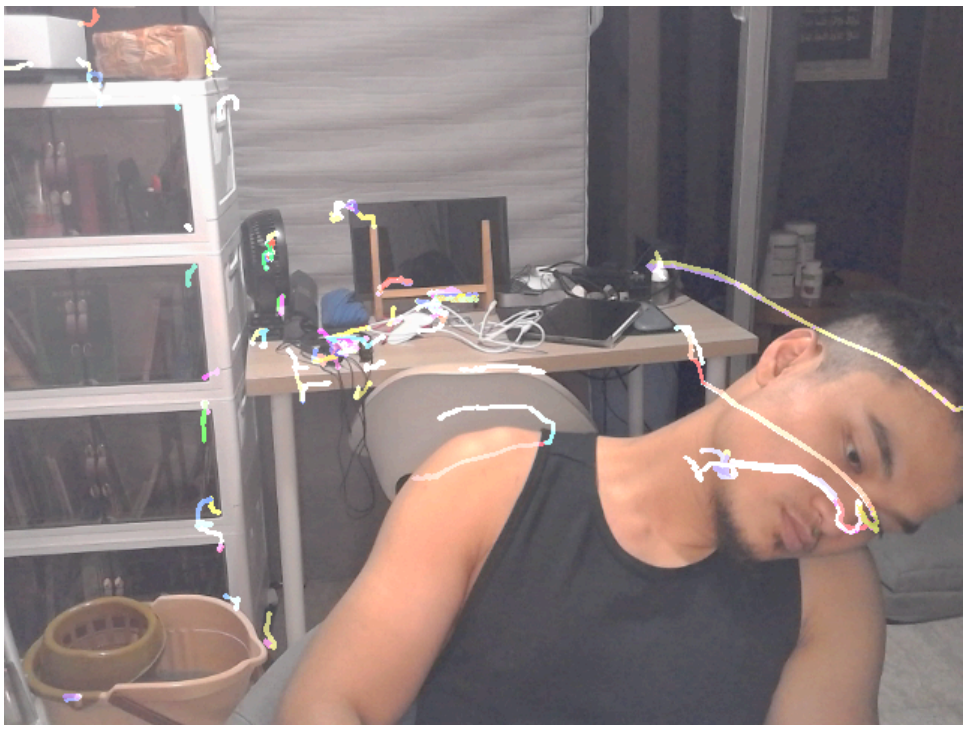
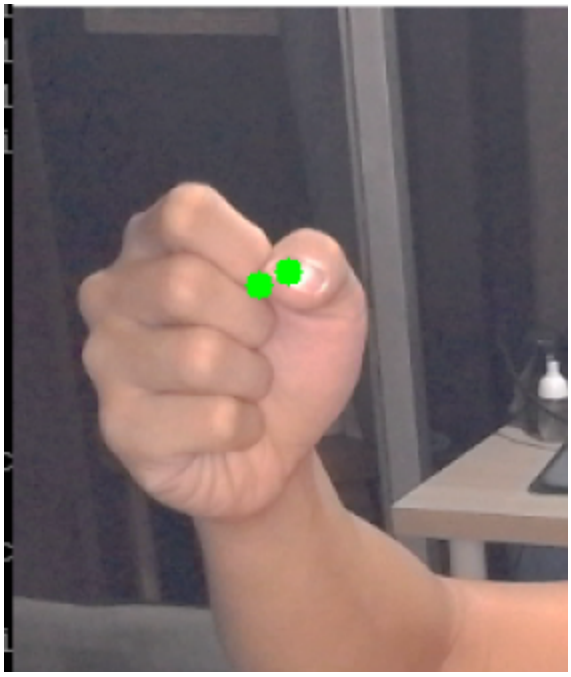
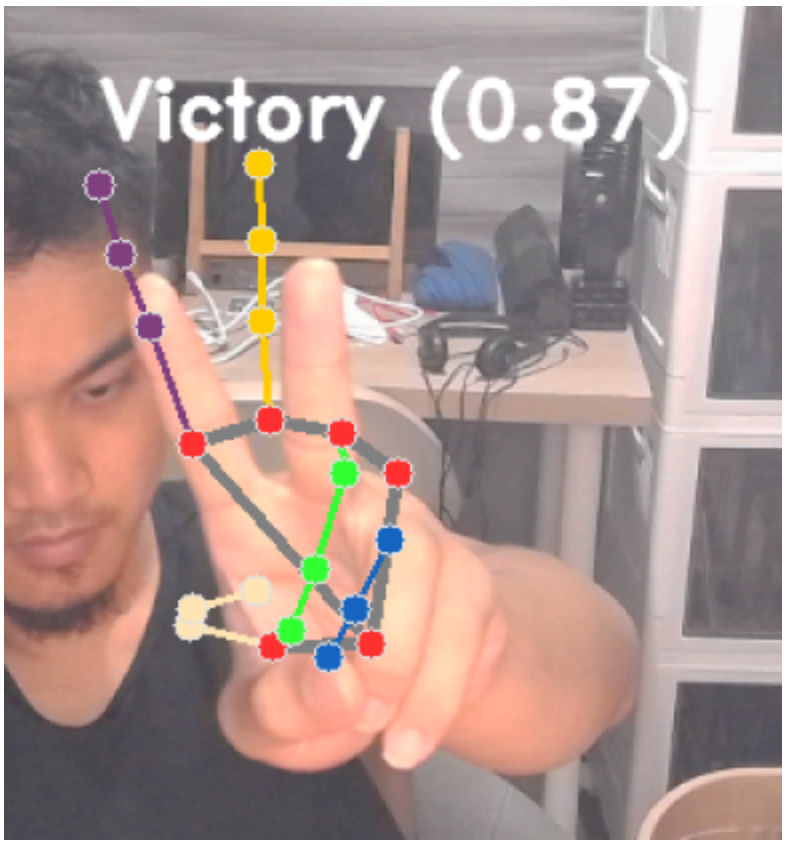
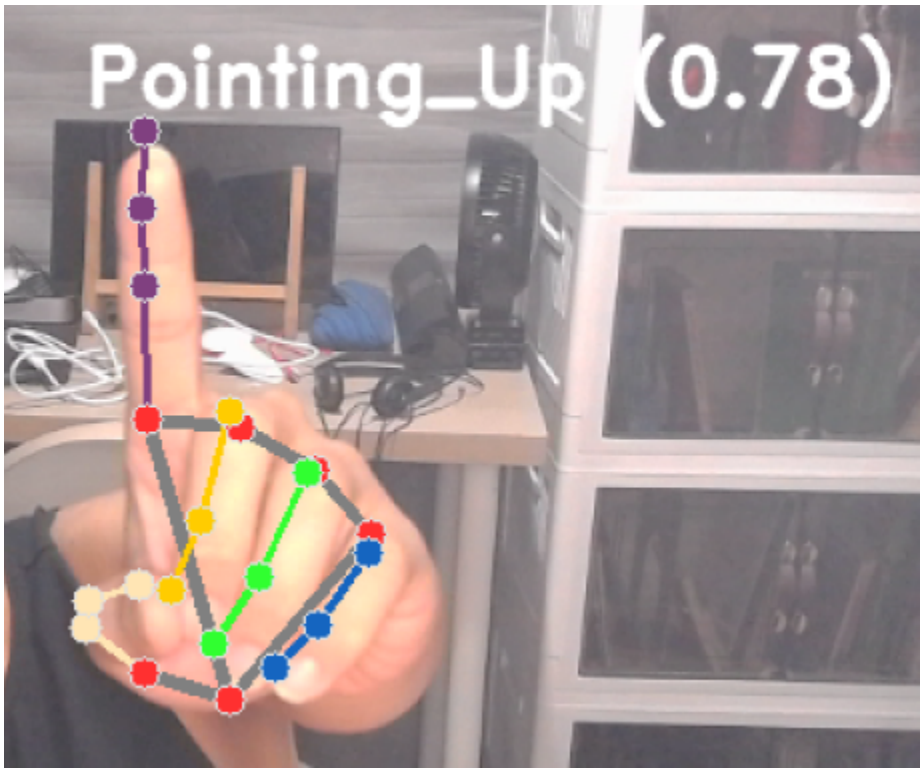
Images are in these order: HOG before HOG after resizing image for speed HOG after increasing patch size making the result coarser but faster processing	

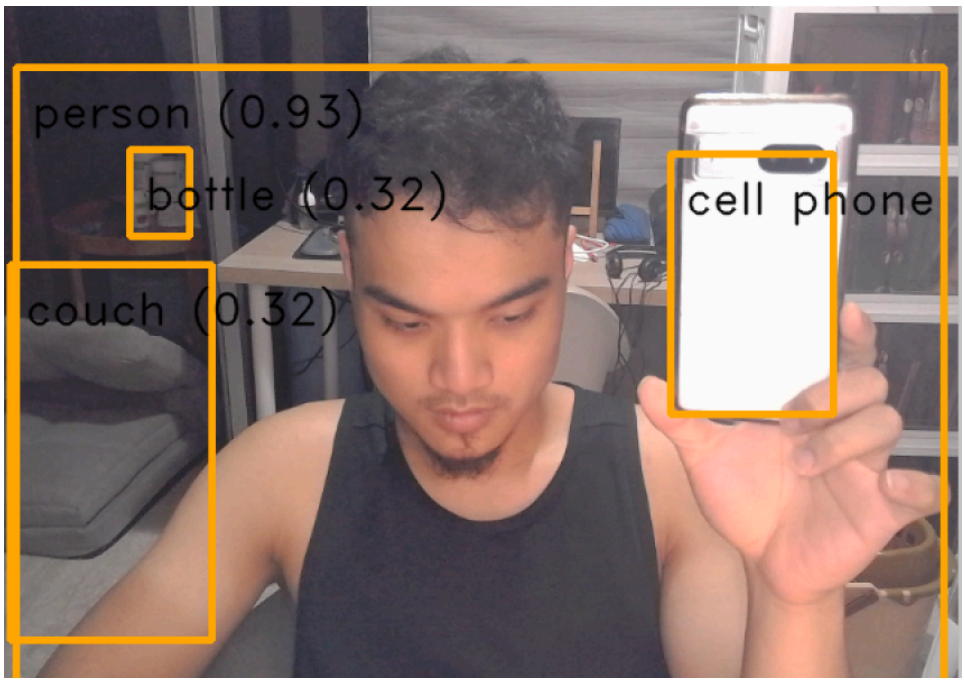
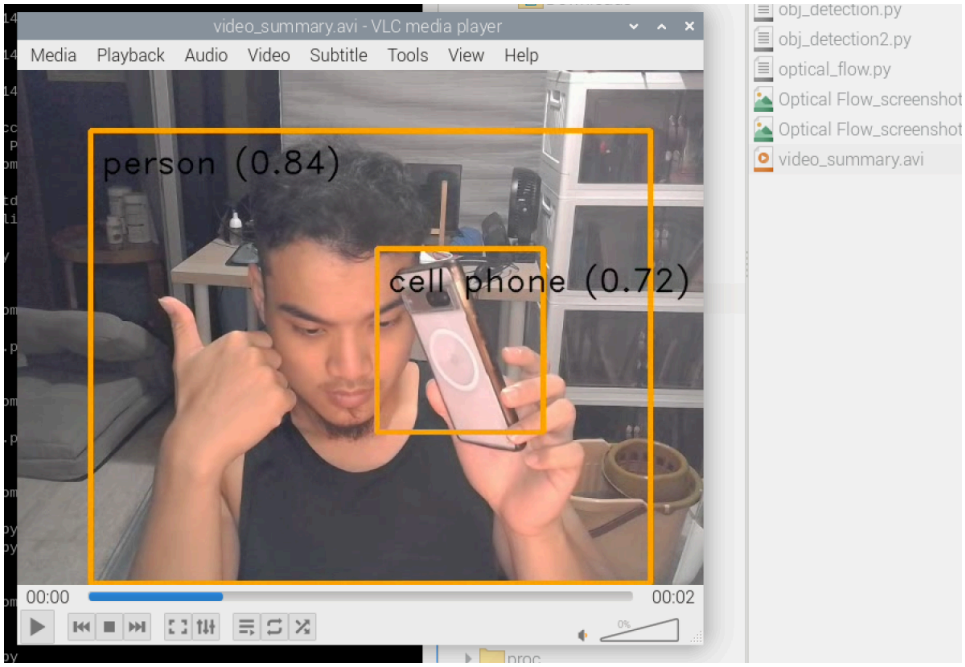
Image in this order: With human in camera Camera covered Without human in camera	
Face mesh result	

video

Flow franeback Lucas kanade optical flow	 

Predict thumbs up	 

Hand gesture	 

<table>
<tr>
<td></td>
<td> 
</td>
</tr>
<tr>
<td>Records and save as video_summary.avi when it detect a cell phone</td>
<td> 

# Reference: https://github.com/googlesamples/mediapipe
import cv2
import mediapipe as mp
import time
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

# Parameters
maxResults = 5</td>
</tr>
</table>

```
scoreThreshold = 0.25
frameWidth = 640
frameHeight = 480
model = 'efficientdet.tflite'
target_object = 'cell phone'  # Object to detect and summarize

# Visualization parameters
MARGIN = 10
ROW_SIZE = 30
FONT_SIZE = 1
FONT_THICKNESS = 1
TEXT_COLOR = (0, 0, 0)  # black

# Initialize lists to store detection results and frames
detection_result_list = []
summary_frames = []  # Store frames with the target object

# Callback function to save detection results
def save_result(result: vision.ObjectDetectorResult,
unused_output_image: mp.Image, timestamp_ms: int):
    detection_result_list.append(result)

# Create object detection model
base_options = python.BaseOptions(model_asset_path=model)
options = vision.ObjectDetectorOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.LIVE_STREAM,
    max_results=maxResults,
    score_threshold=scoreThreshold,
    result_callback=save_result
)
detector = vision.ObjectDetector.create_from_options(options)

# OpenCV Video Capture
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)

if not cap.isOpened():
    raise IOError("Cannot open webcam")

# Video writer for summarized output
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('video_summary.avi', fourcc, 20.0, (frameWidth,
frameHeight))

while True:
    try:
        ret, frame = cap.read()
        frame = cv2.flip(frame, 1)
```

```
        # Convert BGR to RGB
        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        mp_image = mp.Image(image_format=mp.ImageFormat.SRGB,
data=rgb_image)

        # Run object detection
        detector.detect_async(mp_image, time.time_ns() // 1_000_000)

        # Draw detection results
        if detection_result_list:
            for detection in detection_result_list[0].detections:
                bbox = detection.bounding_box
                start_point = (bbox.origin_x, bbox.origin_y)
                end_point = (bbox.origin_x + bbox.width, bbox.origin_y +
bbox.height)

                category = detection.categories[0]
                category_name = category.category_name
                probability = round(category.score, 2)
                result_text = f'{category_name} ({probability})'

                # Draw bounding box and label
                cv2.rectangle(frame, start_point, end_point, (0, 165, 255), 3)
                text_location = (MARGIN + bbox.origin_x, MARGIN +
ROW_SIZE + bbox.origin_y)
                cv2.putText(frame, result_text, text_location,
cv2.FONT_HERSHEY_DUPLEX,
                            FONT_SIZE, TEXT_COLOR, FONT_THICKNESS,
cv2.LINE_AA)

                # Save frames containing the target object
                if category_name.lower() == target_object:
                    summary_frames.append(frame.copy())
                    out.write(frame)  # Save directly to the summarized video

            detection_result_list.clear()

        # Display the current frame
        cv2.imshow('Object Detection', frame)

        # Press 'q' to quit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    except KeyboardInterrupt:
        break

# Release resources
cap.release()
```

	out.release()
	cv2.destroyAllWindows()
	
	# Save the summarized video (optional if already written during detection)
	if summary_frames:
	  print(f"Summary video saved as 'video_summary.avi'")
	else:
	  print("No frames with the target object were detected.")