Cell phone detection code explanation

Certainly! Let's break down each line of the code:

```
import cv2
```

- This line imports the OpenCV library, which is a popular open-source computer vision library in Python. OpenCV provides functions for image and video processing tasks, including reading, writing, and manipulating images and video streams.

```
from ultralytics import YOLO
```

- This line imports the YOLO class from the `ultralytics` module. YOLO (You Only Look Once) is a state-of-the-art real-time object detection algorithm. The `YOLO` class provides an interface to use pre-trained YOLO models for object detection tasks.

```
import pygame
```

- This line imports the pygame library, which is a set of Python modules designed for writing video games. In this context, pygame is used for playing sound alerts when certain conditions are met.

```
import math
```

- This line imports the math module, which provides mathematical functions and constants. While not used explicitly in the code snippet provided, the `math` module may

be used elsewhere in the program for mathematical calculations.

`pygame.init()` is used to initialize the pygame library

Certainly! Let's break down the following code:

```
# Start webcam
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)
```

1. `cv2.VideoCapture(0)`: This line initializes a video capture object `cap` to capture video from a webcam. The argument `0` specifies the index of the webcam to use. In this case, `0` typically represents the default webcam connected to the computer. If you have multiple webcams connected, you can specify a different index to choose a different webcam.
2. `cap.set(3, 640)`: This line sets the width of the captured video frames to `640` pixels. The `set()` function is used to adjust various properties of the video capture object, such as width, height, frame rate, etc. Here, `3` represents the property ID for width.
3. `cap.set(4, 480)`: This line sets the height of the captured video frames to `480` pixels. Similarly, `4` represents the property ID for height.

Certainly! Let's break down the provided code:

```
# Load YOLO model
model = YOLO("yolo-Weights/yolov8n.pt")
```

This line of code loads the YOLO (You Only Look Once) object detection model. The `YOLO` class is imported from the `ultralytics` library, which provides a simple interface for using pre-trained YOLO models. The argument `"yolo-Weights/yolov8n.pt"` specifies the path to the YOLO model weights file. This file contains the learned parameters of the neural network, which are used to make predictions about objects in images.

```
while True:
    success, img = cap.read()
    results = model(img, stream=True)
```

In this block of code, a loop is started to continuously read frames from some video input source (`cap`). Inside the loop:

- `cap.read()` reads the next frame from the video input source (`cap`). It returns two values: `success`, a boolean indicating whether the frame was successfully read, and `img`, the actual image frame.
- `model(img, stream=True)` performs object detection on the current frame (`img`) using the YOLO model (`model`). The `stream=True` argument tells the model to return results in streaming mode, which is more efficient for real-time applications. The result is stored in the variable `results`.

```
    # Flag to check if a cell phone is detected
    is_cellphone_detected = False
```

This line initializes a boolean variable `is_cellphone_detected` to `False`. This variable will be used to track whether a cell phone is detected in the current frame.

The provided code snippet is part of a larger program that continuously reads frames from a video source, performs object detection using the YOLO model, and sets a flag to indicate whether a cell phone is detected in each frame. Depending on the

rest of the program, this flag can be used to trigger actions or alerts when a cell phone is detected in the video stream.

Certainly! Let's break down the code snippet:

```
# Coordinates
for r in results:
    boxes = r.boxes
    for box in boxes:
        # Class name
        cls = int(box.cls[0])
        class_name = model.names[cls]

        # If a cell phone is detected, set the flag to True
        if class_name == "cell phone":
            is_cellphone_detected = True

        # Bounding box
        x1, y1, x2, y2 = box.xyxy[0]
        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
# Convert to int values

        # Draw bounding box
        cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255),
3)
```

1. **Looping through results**: The code iterates over the `results` object, which contains the output of object detection for a single frame of the video. Each `r` in `results` corresponds to a detected object or class.
2. **Extracting bounding boxes**: For each detected object, the code extracts its bounding box coordinates from `r.boxes`. The `r.boxes` attribute contains the bounding box coordinates for all detected objects in the frame.

3. **Extracting class name**: The class of the detected object is determined by `box.cls`. The class index is converted to an integer (`int(box.cls[0])`) and then mapped to the corresponding class name using `model.names[cls]`. This gives us the name of the detected object class.
4. **Checking for cell phone detection**: If the detected object is a "cell phone", the `is_cellphone_detected` flag is set to `True`. This flag can be used later in the program to take specific actions based on the detection of a cell phone.
5. **Bounding box coordinates**: The bounding box coordinates (`x1`, `y1`, `x2`, `y2`) are extracted from `box.xyxy[0]`. These coordinates represent the top-left and bottom-right corners of the bounding box around the detected object. The coordinates are converted to integers to ensure compatibility with OpenCV functions.
6. **Drawing bounding box**: Finally, a bounding box is drawn around the detected object on the input image (`img`) using the `cv2.rectangle()` function. The bounding box is drawn with a specified color (`(255, 0, 255)`) and thickness (`3`). This visualizes the detection result on the image.

Overall, this code snippet processes the output of object detection for a single frame, extracts the bounding box coordinates and class name for each detected object, and draws bounding boxes around the detected objects on the input image.

Sure, let's break down the provided code snippet:

```
# If a cell phone is detected, play the sound
if is_cellphone_detected:
    pygame.mixer.music.play()

cv2.imshow('Webcam', img)
if cv2.waitKey(1) == ord('q'):
```

```
    break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

1. `if is_cellphone_detected:`: This line checks if the variable `is_cellphone_detected` is `True`. This variable is typically set to `True` when a cell phone is detected in the video feed by your object detection mechanism.
2. `pygame.mixer.music.play()`: If a cell phone is detected (i.e., `is_cellphone_detected` is `True`), this line plays the sound loaded using the `pygame.mixer.music.load()` function. The sound is played using the `play()` method of the `pygame.mixer.music` module.
3. `cv2.imshow('Webcam', img)`: This line displays the current frame (`img`) from the webcam feed in a window with the title `'Webcam'`. This is achieved using the `cv2.imshow()` function from the OpenCV library.
4. `if cv2.waitKey(1) == ord('q'):`: This line waits for a key press event for 1 millisecond. If the pressed key is `'q'`, it breaks out of the loop. This is commonly used to allow the user to exit the program by pressing the 'q' key.
5. `cap.release()`: This line releases the video capture object `cap`, freeing up any resources associated with it. It's important to release resources properly after use to avoid memory leaks.
6. `cv2.destroyAllWindows()`: This line closes all OpenCV windows that were created. It's used to clean up any windows that were opened during the execution of the program.

Overall, this code snippet is responsible for playing a sound when a cell phone is detected in the video feed, displaying the video feed in a window, and handling user input to exit the

program. Finally, it releases resources associated with the webcam feed and closes any OpenCV windows.