You:

import requests import cv2 import numpy as np import imutils # Load YOLO yolo_net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg") yolo_layer_names = yolo_net.getUnconnectedOutLayersNames() # Load Haar Cascade Classifier for face detection face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml') # Replace the below URL with your own. Make sure to add "/shot.jpg" at last. url = "http://192.168.0.104:8080/shot.jpg" # List of object classes for YOLO yolo_classes = [ "Unknown", "dog", "cat", "bird", "rabbit", "fish", "snake", "turtle", "lion", "monkey", "human", ] # List of objects corresponding to detected faces feeder_objects = [ "Human", "Dog", "Cat", "Bird", "Rabbit", "Fish", "Snake", "Turtle", "Hamster", "Guinea Pig", "Lizard", "Ferret", "Parrot", "Horse", "Pig", "Cow", "Sheep", "Goat", "Elephant", "Monkey" ] # Flags for different modes surveillance_mode = False feeder_mode = False # Create a window with minimize and maximize flags cv2.namedWindow("Pet_Feeder_CAM_V4.0_Stable (Dual mode) by alan_cyril for IoT Mini Project [Team Pet Feeder]", cv2.WINDOW_NORMAL) # While loop to continuously fetch data from the URL while True: img_resp = requests.get(url) img_arr = np.array(bytearray(img_resp.content), dtype=np.uint8) img = cv2.imdecode(img_arr, -1) img = imutils.resize(img, width=640, height=480) # Press 's' key to toggle surveillance mode key = cv2.waitKey(1) if key == 27: # Press Esc key to exit break elif key == ord('s'): # Press 's' key to toggle surveillance mode surveillance_mode = not surveillance_mode feeder_mode = False # Turn off feeder mode when surveillance mode is activated print(f"Surveillance Mode {'On' if surveillance_mode else 'Off'}") # Press 'f' key to toggle feeder mode elif key == ord('f'): feeder_mode = not feeder_mode surveillance_mode = False # Turn off surveillance mode when feeder mode is activated print(f"Feeder Mode {'On' if feeder_mode else 'Off'}") # Process detections only if surveillance mode or feeder mode is on if surveillance_mode or feeder_mode: # Get the blob from the image blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416, 416), swapRB=True, crop=False) if surveillance_mode: # Set the input to the YOLO neural network for surveillance mode yolo_net.setInput(blob) yolo_detections = yolo_net.forward(yolo_layer_names) # Process the YOLO detections for surveillance mode for detection in yolo_detections: for obj in detection: scores = obj[5:] class_id = np.argmax(scores) confidence = scores[class_id] if confidence > 0.5: # You can adjust the confidence threshold # Get the coordinates of the bounding box box = obj[0:4] * np.array([img.shape[1], img.shape[0], img.shape[1], img.shape[0]]) (x, y, w, h) = box.astype("int") # Draw the bounding box and label cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2) # Ensure that the class_id is within the range of the classes list if class_id < len(yolo_classes): label = f"{yolo_classes[class_id]}: {confidence:.2f}" else: label = f"Unknown: {confidence:.2f}" cv2.putText(img, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2) elif feeder_mode: # Convert the image to grayscale for face detection gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Perform face detection for feeder mode faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)) # Draw rectangles around the detected faces and display objects for i, (x, y, w, h) in enumerate(faces): cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2) # Display object label on the face obj_label = feeder_objects[i] if i < len(feeder_objects) else "Unknown Object" cv2.putText(img, obj_label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2) # Display the mode status on the frame mode_status = "Surveillance Mode: ON" if surveillance_mode else "Feeder Mode: ON" if feeder_mode else "Both Modes: OFF" cv2.putText(img, mode_status, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2) # Display the frame cv2.imshow("Pet_Feeder_CAM_V4.0_Stable (Dual mode) by alan_cyril for IoT Mini Project [Team Pet Feeder]", img) # Cleanup cv2.destroyAllWindows() explain in detail

---

ChatGPT:

Sure, let me explain the code in detail:

1. **Import Libraries:**
    - `requests`: Used to make HTTP requests to get the image from the specified URL.
    - `cv2` (OpenCV): Main library for computer vision tasks.
    - `numpy`: Used for numerical operations on arrays.
    - `imutils`: A set of convenience functions to make working with OpenCV easier.
2. **Load YOLO (You Only Look Once):**
    - `cv2.dnn.readNet`: Loads the YOLO model with its pre-trained weights and configuration files.
    - `yolo_layer_names`: Retrieves the output layer names from the YOLO network.
3. **Load Haar Cascade Classifier for Face Detection:**
    - Uses a pre-trained Haar Cascade Classifier for face detection.
4. **Set Up URL for Video Stream:**
    - Specifies the URL for the video stream. This is typically from an IP camera or webcam. The stream is fetched continuously.
5. **Define Classes for YOLO:**
    - `yolo_classes`: List of classes that the YOLO model can detect.
6. **Define Objects for Feeder Mode:**
    - `feeder_objects`: List of objects corresponding to detected faces in the feeder mode.
7. **Flags for Different Modes:**
    - `surveillance_mode`: Flag to indicate whether surveillance mode is active.
    - `feeder_mode`: Flag to indicate whether feeder mode is active.
8. **Create OpenCV Window:**
    - `cv2.namedWindow`: Creates a window to display the video feed with a specific name.
9. **Main Loop:**
    - A continuous loop to fetch frames from the video stream.
10. **Processing User Input:**

- Press 's' to toggle surveillance mode.
- Press 'f' to toggle feeder mode.
- Press 'Esc' to exit the program.

11. **Processing Modes:**

- If either surveillance mode or feeder mode is active, it processes the frames accordingly.

12. **Surveillance Mode:**

- Uses YOLO to detect objects in the frame.
- Draws bounding boxes and labels around detected objects.

13. **Feeder Mode:**

- Converts the frame to grayscale.
- Uses Haar Cascade Classifier to detect faces.
- Draws rectangles around detected faces.
- Displays labels for detected faces based on the `feeder_objects` list.

14. **Display Mode Status:**

- Displays the current mode status on the frame.

15. **Display Frame:**

- Displays the processed frame with all the annotations and labels.

16. **Cleanup:**

- Closes all OpenCV windows when the program is terminated.

This code essentially creates a simple system that can operate in two modes: surveillance mode for general object detection using YOLO and feeder mode for detecting and labeling faces using a Haar Cascade Classifier. It's designed to work with a live video stream from an IP camera or webcam.

---