iiit srishti task2

task word-to-word:

(a) you are required to write code to perform segmentation on a "list of images" from your local folder in your computer. i think the relevant part of code snippet can be searched, retrieved (from ultralytics documentation pages) and inserted in your code. once you have done this, i.e., for achieving segmented output for multiple images, the code needs to be extended for a piece of video.

(b) you should retrieve a video snippet from youtube or other social media forum. (please see that video selected doesn't hurt the feelings/sentiments of others in this group).

(c) the video should be split into images at some regular intervals (try using ffmpeg for this, other tools are also available).

(d) then after segmentation of all images extracted from this video, all the images should be joined and converted to a new video (use ffmpeg if other simpler tools aren't available).

let the second part of fun begin !! one whole week to complete. take your time, go through tutorials/documentations on web, and solve. cheers...

---

my exploration and understanding of each part

(a) segmentation on a list of local images

i first had to figure out how yolov8 handles segmentation, not just detection, especially in video. segmentation means identifying which pixels belong to which object, which is much more detailed than simply drawing boxes.

i wrote a script that loads all images from a local folder, applies segmentation, and saves the results in a separate folder. the key method was `model.predict()` and passing a folder path instead of a single image. i had to make sure all images were in the same format and placed in a folder before running the script.

(b) retrieving a video

i tried using `yt-dlp.exe` to download a youtube video, but the command `./yt-dlp.exe` didn't work in git bash. later i realized that syntax probably does not work on git cmd. since i was already juggling between environments and didn't want to waste time, i just used a browser-based youtube downloader, y2mate to save an mp4 and placed it in the `task2` folder.

(c) splitting the video using ffmpeg

what exactly is `ffmpeg?` ffmpeg is a powerful open-source tool used to process videos and audio. it can convert formats, extract frames, stitch images into a video, and much more. it works via command line, and once installed and added to the system path, i could run it from anywhere.

to split the video into frames, i used:

```
ffmpeg -i downloaded.mp4 -vf fps=1 frames/frame_%04d.jpg
```

this extracts one frame per second and names them sequentially. i created a folder called `frames` beforehand to keep things neat.

(d) segmenting the frames and converting back to video

after extracting the frames, i reused my segmentation script but pointed it to the `frames/` folder. yolov8 segmented each frame and saved the output into `runs/segment/predict/`. to create a video back from the segmented images, i used another ffmpeg command:

```
ffmpeg -framerate 1 -i runs/segment/predict/image%04d.jpg -c:v libx264 -
pix_fmt yuv420p output_video.mp4
```

this stitched the segmented images into a new video.

implementation part:

the photos:

i downloaded some images from pinterest

segmentation on local images

```
from ultralytics import YOLO
import os
import cv2

model = YOLO("yolov8x-seg.pt")
input_folder = "local_images"
output_folder = "segmented_images"
os.makedirs(output_folder, exist_ok=True)

count = 1
for img in sorted(os.listdir(input_folder)):
    if img.endswith((".jpg", ".jpeg", ".png")):
        path = os.path.join(input_folder, img)
        results = model.predict(source=path, save=False, stream=True)
        for r in results:
            seg_img = r.plot()
            cv2.imwrite(os.path.join(output_folder, f"{count:04d}.jpg"), seg_im
g)
            print(f"Segmented {img}")
            count += 1
```

extract frames from video using ffmpeg

first, i downloaded ffmpeg from the official site, and then, i added it to path. i then downloaded a yt video- the trailer of “air” from amazon mgm studios.

```
mkdir frames
ffmpeg -i downloaded.mp4 -vf fps=1 frames/frame_%04d.jpg
```

segmentation on extracted video frames

```
from ultralytics import YOLO
import os
import cv2

model = YOLO("yolov8x-seg.pt")
input_folder = "frames"
output_folder = "segmented_frames"
os.makedirs(output_folder, exist_ok=True)

count = 1
for img in sorted(os.listdir(input_folder)):
    if img.endswith(".jpg"):
        path = os.path.join(input_folder, img)
        results = model.predict(source=path, save=False, stream=True)
        for r in results:
            seg_img = r.plot()
            out_path = os.path.join(output_folder, f"{count:04d}.jpg")
            cv2.imwrite(out_path, seg_img)
            print(f"Processed {img} → {out_path}")
            count += 1
```

but then i realised this was quite choppy- for it only takes one frame per second. i decided to do something further and keep in mind that i have to take care of this while i am exploring further.

---

folder structure i used

```
C:\Users\madha\iiitsrishti\task2\
│
├── downloaded.mp4
├── yt-dlp.exe
├── frames\
│   ├── frame_0001.jpg
│   └── ...
├── images\
├── runs\
│   └── segment\
│       └── predict\
│           ├── image0001.jpg
│           └── ...
├── task2_image_segmentation.py
└── task2_video_segmentation.py
```

further work- static video other-worldly object detection

"well the other people heavily funding vision research are the military and they've never done anything horrible like killing lots of people with new technology oh wait.....!

i have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it, like counting the number of zebras in a national park [13], or tracking their cat as it wanders around their house [19]. but computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it. we owe the world that much."

according to one of the older papers i was going through on yolov3: an incremental improvement, the creator wanted us to find that.

sooo, as a responsible student, i've decided to explore just a little bit into this. although i did not want to count zebras, i wanted to do something more thrilling — so i came up with this idea where i thought, if we can detect burglars at night time and raise an automatic alarm, it could be a useful real-world application of computer vision.

the basic concept was to process video footage, detect humans as anomalies, and trigger an alarm when one appears in the frame.

initially, i wrote a batch processing pipeline to extract frames from a video, run detection on each frame, save the annotated frames, and then recombine them into a processed video. i used ffmpeg for splitting the video into images and later for recombining them. here is the core part of the code i used for this approach:

```
import cv2
import os
import json
from ultralytics import YOLO
from datetime import timedelta
import pygame

VIDEO_PATH = "robbery_clip.mp4"
FRAME_DIR = "frames2"
OUTPUT_VIDEO = "output/annotated_video.mp4"
ALARM_FRAMES_DIR = "output/alarm_frames2"
ANOMALY_LOG = "output/anomalies.json"
CONFIDENCE_THRESHOLD = 0.4
SKIP_FRAMES = 3

model = YOLO("yolov8n.pt")



os.makedirs(FRAME_DIR, exist_ok=True)
os.makedirs(ALARM_FRAMES_DIR, exist_ok=True)



pygame.init()
pygame.mixer.init()
pygame.mixer.music.load("alarm.mp3")



cap = cv2.VideoCapture(VIDEO_PATH)
fps = cap.get(cv2.CAP_PROP_FPS)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(OUTPUT_VIDEO, fourcc, fps, (frame_width, frame_height))

frame_count = 0
anomalies = []
```

```
print("[INFO] Starting anomaly detection...")

while True:
    ret, frame = cap.read()
    if not ret:
        break


    if frame_count % SKIP_FRAMES != 0:
        frame_count += 1
        continue

    results = model(frame)[0]
    detected_person = False

    for result in results.boxes:
        cls = int(result.cls[0])
        conf = float(result.conf[0])
        label = model.names[cls]

        if conf >= CONFIDENCE_THRESHOLD and label == "person":
            detected_person = True
            x1, y1, x2, y2 = map(int, result.xyxy[0])
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    if detected_person:
        timestamp = str(timedelta(seconds=frame_count / fps))
        anomalies.append({
            "frame": frame_count,
            "timestamp": timestamp,
            "object": "person"
        })

        cv2.imwrite(f"{ALARM_FRAMES_DIR}/frame_{frame_count}.jpg", fram
e)
```

```
            if not pygame.mixer.music.get_busy():
                pygame.mixer.music.play()

        out.write(frame)
        frame_count += 1

    cap.release()
    out.release()


    with open(ANOMALY_LOG, "w") as f:
        json.dump({"anomalies": anomalies}, f, indent=2)

    print(f"[DONE] {len(anomalies)} anomalies detected.")
    print(f"Annotated video saved to: {OUTPUT_VIDEO}")
```

some explanation about what each part of code does:

```
import cv2
import os
import json
from ultralytics import YOLO
from datetime import timedelta
import pygame
```

here, i'm importing all the libraries i need.

- `cv2` is for working with videos and images using opencv.
- `os` helps with file and folder operations.
- `json` is used to save the detected anomalies as a json file.
- `YOLO` is the detection model from ultralytics.
- `timedelta` helps convert frame number into readable time.
- `pygame` is used for playing the alarm sound. i wanted to actually use "playsound from playsound library" but that ended up not working- because it was blocking (waits for sound to finish), it silently failed.

```
VIDEO_PATH = "robbery_clip.mp4"
FRAME_DIR = "frames2"
OUTPUT_VIDEO = "output/annotated_video.mp4"
ALARM_FRAMES_DIR = "output/alarm_frames2"
ANOMALY_LOG = "output/anomalies.json"
CONFIDENCE_THRESHOLD = 0.4
SKIP_FRAMES = 3
```

here i'm setting all the important parameters.

- `VIDEO_PATH` is the video file i want to process.
- `FRAME_DIR` is where the raw frames will be saved (although in this version i don't actually use it).
- `OUTPUT_VIDEO` is where i'll save the final annotated video.
- `ALARM_FRAMES_DIR` is where i'll save the frames that contain a person.
- `ANOMALY_LOG` is the json file where i log each anomaly.
- `CONFIDENCE_THRESHOLD` means i'll only consider detections if the confidence is more than 0.4.
- `SKIP_FRAMES` means i'll process every 3rd frame to save time.

```
model = YOLO("yolov8n.pt")
```

i'm loading the pretrained yolov8n model. this is a lightweight object detection model that already knows how to detect people.

```
os.makedirs(FRAME_DIR, exist_ok=True)
os.makedirs(ALARM_FRAMES_DIR, exist_ok=True)
```

i make sure the required folders exist. if they don't, this creates them so the rest of the code doesn't crash while saving files.

```
pygame.init()
pygame.mixer.init()
pygame.mixer.music.load("alarm.mp3")
```

i initialize pygame and load an alarm sound. this alarm will play if a person is detected in any frame.

```
cap = cv2.VideoCapture(VIDEO_PATH)
fps = cap.get(cv2.CAP_PROP_FPS)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(OUTPUT_VIDEO, fourcc, fps, (frame_width, frame_he
ight))
```

here, i open the video using opencv, get the video’s frame rate and dimensions, and set up a video writer object that will write the output with annotations to a new video file.

```
frame_count = 0
anomalies = []
```

i’m initializing a frame counter and an empty list to store anomaly data (i.e., when a person is detected).

```
print("[INFO] Starting anomaly detection...")
```

this is just a debugging message

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

i start reading the video frame by frame. if there no more frame, i break out of the loop.

```
if frame_count % SKIP_FRAMES != 0:
    frame_count += 1
    continue
```

i skip every few frames based on the `SKIP_FRAMES` value. this helps speed up the detection process, because i found that analysing too many frames naturally take too long

---

```
results = model(frame)[0]
detected_person = False
```

i run object detection on the current frame.

`results` contains the detection info.

i also start with assuming no person is detected yet.

```
for result in results.boxes:
    cls = int(result.cls[0])
    conf = float(result.conf[0])
    label = model.names[cls]

    if conf >= CONFIDENCE_THRESHOLD and label == "person":
        detected_person = True
        x1, y1, x2, y2 = map(int, result.xyxy[0])
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
        cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
```

i go through all the boxes detected by yolov8.

if the label is “person” and the confidence is above the threshold, i mark that a person is detected.

then i draw a red rectangle around the person and add the label text (with confidence score) on the frame.

```
    if detected_person:
        timestamp = str(timedelta(seconds=frame_count / fps))
        anomalies.append({
            "frame": frame_count,
            "timestamp": timestamp,
            "object": "person"
        })

        cv2.imwrite(f"{ALARM_FRAMES_DIR}/frame_{frame_count}.jpg", fram
e)

        if not pygame.mixer.music.get_busy():
            pygame.mixer.music.play()
```

if a person was detected in the frame:

- i convert the frame number into a time format
- i log the detection in the `anomalies` list
- i save the frame as an image for reference
- if the alarm isn't already playing, i play it

```
    out.write(frame)
    frame_count += 1
```

i write the processed frame (with bounding boxes, if any) to the output video.

then i increase the frame counter.

```
cap.release()
out.release()
```

i release both the video input and output resources once i'm done processing.

```
with open(ANOMALY_LOG, "w") as f:
    json.dump({"anomalies": anomalies}, f, indent=2)
```

i write all the anomalies i collected into a json file, nicely formatted with indentation.

```
print(f"[DONE] {len(anomalies)} anomalies detected.")
print(f"Annotated video saved to: {OUTPUT_VIDEO}")
```

i display a final message telling me how many anomalies were found and where the annotated video is saved.

---

and that's it — this whole script processes a video, detects people, raises alarms, saves frames with detections, and outputs a full log + video.

this code worked well as a prototype, and it included key features such as confidence thresholding, skipping frames to improve speed, drawing bounding boxes, saving anomaly frames, and raising an alarm sound.

burglary detection system

however, as i tested it, i realized that batch processing video frames like this—reading from disk, writing to disk, and then combining frames back into a video—was inefficient and not suitable for real-time applications. i needed to detect intrusions as they happen, for example from a live webcam feed or real-time CCTV footage, not after the fact.

thus, i moved towards implementing a real-time detection pipeline where frames are captured directly from the camera, processed immediately, and anomalies trigger immediate alarms and visual feedback. this means i would not be able to use ffmpeg anymore- for it was only for breaking "videos" into frames. so,

here's the real-time detection code:

```
import cv2
from ultralytics import YOLO
from datetime import timedelta
import pygame

CONFIDENCE_THRESHOLD = 0.4
SKIP_FRAMES = 3
```

```
model = YOLO("yolov8n.pt")
pygame.init()
pygame.mixer.init()
pygame.mixer.music.load("alarm.mp3")

cap = cv2.VideoCapture(0)
fps = cap.get(cv2.CAP_PROP_FPS) or 30

frame_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break

    if frame_count % SKIP_FRAMES == 0:
        results = model(frame)[0]
        detected_person = False

        for result in results.boxes:
            cls = int(result.cls[0])
            conf = float(result.conf[0])
            label = model.names[cls]

            if conf >= CONFIDENCE_THRESHOLD and label == "person":
                detected_person = True
                x1, y1, x2, y2 = map(int, result.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
                cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        if detected_person and not pygame.mixer.music.get_busy():
            pygame.mixer.music.play()

    cv2.imshow("Real-time Anomaly Detection", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    frame_count += 1

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

detailed explanation of each part:

im setting two important constants here.

```
cap = cv2.VideoCapture(0)  # open default webcam
fps = cap.get(cv2.CAP_PROP_FPS) or 30  # default to 30 if 0
```

im opening the default webcam with `cv2.VideoCapture(0)`.

`fps` gets the frames per second from the webcam, and if it returns 0, im just setting it to 30 by default so calculations are stable.

```
frame_count = 0
```

im initializing a counter to keep track of how many frames have been processed.

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

im starting a loop that keeps running as long as the webcam is feeding frames.

if no frame is returned (maybe the webcam was unplugged), then the loop will break.

```
    if frame_count % SKIP_FRAMES == 0:
        results = model(frame)[0]
        detected_person = False
```

im processing every 3rd frame (based on `SKIP_FRAMES`).

then im passing the frame to the yolov8 model and getting the first result.

`detected_person` is a flag that starts as false for each frame.

```
        for result in results.boxes:
            cls = int(result.cls[0])
            conf = float(result.conf[0])
            label = model.names[cls]
```

im looping over each detected object in the frame.

- `cls` is the class id (like 0 for person).
- `conf` is the confidence score for that detection.
- `label` gets the actual name of the detected object from the model’s label list.

```
            if conf >= CONFIDENCE_THRESHOLD and label == "person":
                detected_person = True
                x1, y1, x2, y2 = map(int, result.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
                cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
```

if the confidence is high enough and the object is a person, im setting `detected_person` to true.

then im drawing a red rectangle around the person and adding the label with the confidence score on top of the box using `cv2.putText`.

```
        if detected_person and not pygame.mixer.music.get_busy():
            pygame.mixer.music.play()
```

if a person was detected and the alarm is not already playing, im playing the alarm sound.

```
    cv2.imshow("Real-time Anomaly Detection", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

im showing the video frame with any annotations in a window titled "real-time anomaly detection".

if i press the 'q' key, the loop will break and the program will stop.

```
    frame_count += 1
```

im increasing the frame count after every loop, so skipping works properly.

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

once the loop is done, im releasing the webcam and closing all open opencv windows to clean up.

---

this worked beautifully — it enabled me to open my webcam, detect humans in real-time, draw bounding boxes, and play an alarm sound instantly when a person is detected. this code lets me use my webcam to do real-time detection of people using yolov8, and if someone shows up in the frame, it plays an alarm automatically.

scope for future work

next, i considered expanding this system to detect other dangerous objects like knives and guns, which would make the system more robust for security applications.

however, i quickly realized that:

- the base yolov8n model does not include classes for knives or guns,
- reliable detection of such weapons would require custom training on specialized datasets or finding a pre-trained model with these classes,
- and training such a model or even searching and integrating one would take more time and effort than i could afford before my exams.

thus, i decided to postpone the weapon detection feature for now, focusing on perfecting person detection and real-time alarms first.

in summary:

- i began with batch video processing using ffmpeg and yolov8, which worked but was inefficient for real-time needs,
- i then implemented a real-time webcam anomaly detection pipeline with live bounding box overlays and alarm sounds,
- finally, i scoped the possibility of detecting knives and guns but deferred it due to dataset limitations and time constraints.

this project taught me a lot about the practical trade-offs between batch and real-time processing, integrating audio alarms, and the realities of deploying computer vision models beyond off-the-shelf datasets.

overall, it was an exciting journey from theory to a functioning prototype that can detect burglars and alert users automatically — a small but meaningful application of computer vision technology i'm proud to have built.