INITIAL NOTES

Requirements for AI:

- Pip
- Ultralytics (enter in terminal: pip install ultralytics

Requirements for Visualization:

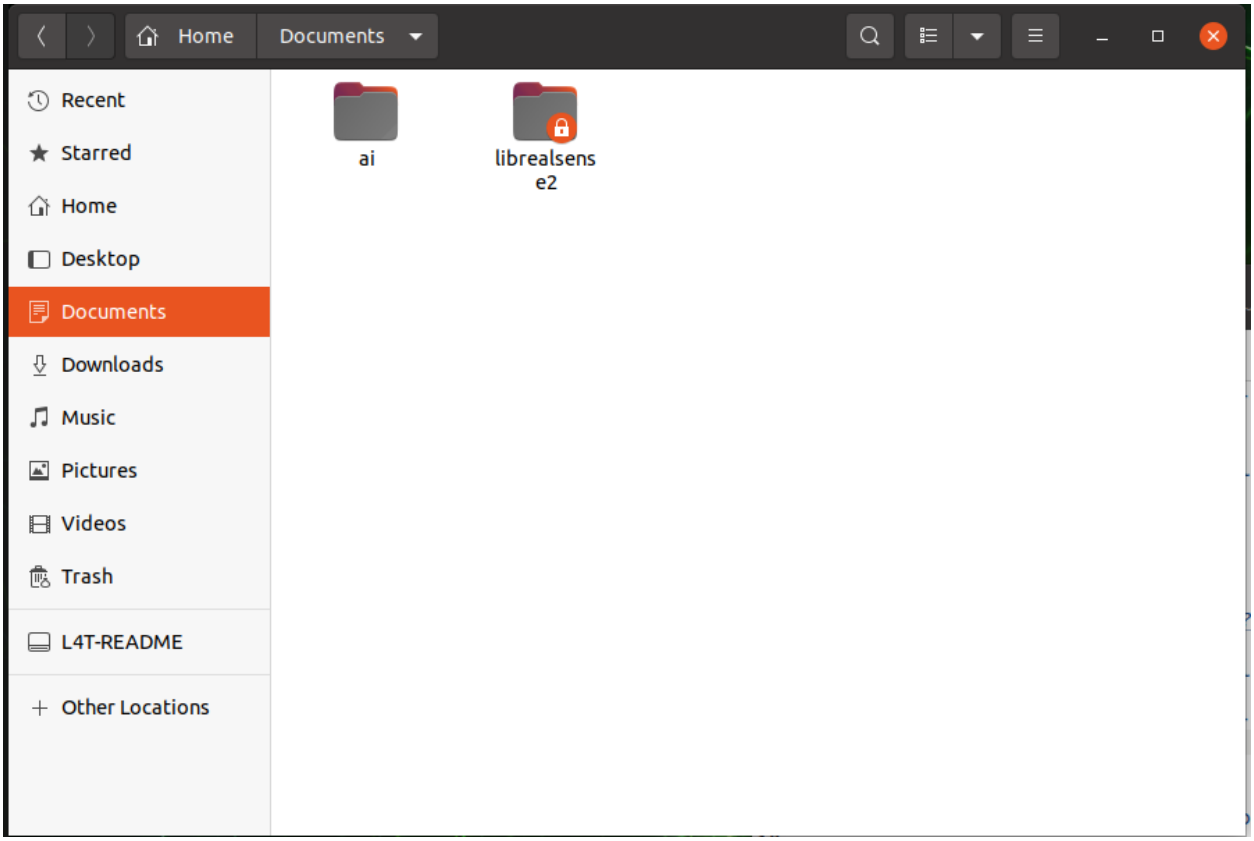
- Opencv

Extra Resources:

- Follow along video
- YOLOv8 documentation
- Test AI repository

SET UP

Create a folder on your **JETSON** computer to store all the AI code, data, info, etc. Make sure you can find and access the folder easily. For example,



(You want to use your jetson since it trains the AI much faster than a regular computer)

Inside the folder you created, you can import code from the github or start from scratch.

Inside the github folder, or the code folder you created, create yet another folder called **data**. Inside the **data** folder, create two more folders, one called **images** and the other called **labels**, and in each of those folders, create a folder called **train**.

You should have two folders called **train** at this point, one in the **images** folder and one in the **labels** folder. It is critical to make sure that the two folder's structures are *identical to each other*. If the folders are not identical, you will get an error message that reads that the program could not find the labels.

In the **images** folder, put all the unannotated pictures you have inside. Make sure all the pictures are in .jpg, .jpeg, or .png format or else OpenCV may have a hard time reading it and throw an error.

In the **labels** folder, upload the annotated data you exported from cvat. I used Ultralytics YOLO Segmentation 1.0. Makes sure that the .txt files have text in them. The text should look something like:

(If the text inside varies in length and content, that's fine. Just make sure there's text inside of it)

Every image in **images** should correspond to a .txt file in **labels**. Make sure that every image has a label, and every label has an image, or else the code will throw an error.

CODE

I. AI code

In a new python file (you can name this anything that's appropriate, I named it 'main'), you can start writing the AI code. The code should be *outside* the data folder, but *inside* your main AI folder.

Steps:

1) Import YOLO.
2) Replace/Insert model version.
   The model I used for the AI is YOLO v8. If you are using the same, load the model as 'yolov8n.pt'.
3) Train model

Your code should look similar to this:

```
14  #----------------------------------------------------------#
15  from ultralytics import YOLO
16
17  # Load a model
18  model = YOLO("yolov8n.pt") # build model from scratch
19
20  results = model.train(data="config.yaml", epochs=3) #train the model
21
```

The config file should look like this:

```
1   # | ->> [ absolute path ] : path to the data
2   path: [path]
3
4   # | ->> [ relative path ] : path from absolute path to images/training
5   train: images/train # | > training images
6   val: images/train   # | > validation images
7
8   # | ->> [ classes ] : input all object classifications here
9   names:
10    0: [name]
11    1: [name]
12  #etc
```

In [path], enter the path to the data folder. If you followed the instructions above for the images and labels, the next four lines should stay the same. In the names area, enter all object classifications.

To run the code you can type directly 'python3 [name].py'. Make sure to set the epochs to a low number at first to test for bugs. Then, once you know the code can run, you can increase the epochs and experiment with it (usually around 300-400 epochs for a dataset of 50+ images).

After you're done running the code, all the analytics should be kept in a folder called **runs**. The code should tell you the folder path and name (eg, /home/user/runs/detect/train16). The folder contains the trained weights inside the **weights** folder, as well as analytics about your AI's results. You can also run the code directly from your terminal, but it's much easier to just run the python file.

After looking at your analytics, check if the loss graphs show a generally decreasing curve. If so your model should be working.

II. Visualization code – VIDEO FORMAT (MP4 -> AVI)

It shouldn't matter what folder your visualization code is in, but for organization's sake, keep it in the same folder as your AI code. Name it an appropriate title, such as **predict**.

Pick out the video you want the AI to analyze, then create a folder inside your ai folder called **video** and upload it there. Name it something easy to remember, like 'VIDEO.mp4'. Make sure the video is in mp4 format.

Steps:

1) Import libraries
    a. os, YOLO, and cv2

```
import os

from ultralytics import YOLO
import cv2
```

2) Define video_path as the path to your video
    a. You want to use os.path.join since OpenCV can't read the '/' character
    b. Ex. If your path looks like /home/user/ai/video/VIDEO.mp4, your os.path.join should read os.path.join('/', 'home', 'user', 'ai', 'video', 'VIDEO.mp4')

```

video_path = os.path.join([path to video]) # eg: '/', 'home', 'user', 'Documents', 'ai', 'yolo_testai2-main', '
if not os.path.exists(video_path):
    print(f"Error: Video file does not exist at {video_path}")
else:
    print(f"Video file found at {video_path}")

```

(You may throw an exception for easier debugging)

3) Select output format (.avi works best)

```

video_path_out = '{}_out.avi'.format(os.path.splitext(video_path)[0])

```

4) Use cv2 to open the video

```

cap = cv2.VideoCapture(video_path, cv2.CAP_FFMPEG)
if not cap.isOpened():
    print("Error: Could not open video at {video_path}.")
    exit()

```

5) Read the frames

```
ret, frame = cap.read()
if not ret or frame is None:
    print("Error: Could not read the first frame.")
    exit()
```

6) Output video

```
H, W, _ = frame.shape
out = cv2.VideoWriter(video_path_out, cv2.VideoWriter_fourcc(*'MJPG'), int(cap.get(cv2.CAP_PROP_FPS)), (W, H))
```

7) Load model
    a. Use the same path format as before, this time pointing towards the weights
    b. Select best.pt, not last.pt
    c. Set threshold for the prediction confidence (usually use 0.5)

```
model_path = os.path.join([path to weights]) # eg. '/', 'home', 'user', 'runs', 'detect', 'train19', 'weights',

model = YOLO(model_path)

threshold = 0.3
```

8) Draw boxes & text

```
while ret:

    results = model(frame)[0]

    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result

        if score > threshold:
            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)
            cv2.putText(frame, results.names[int(class_id)].upper(), (int(x1), int(y1-10)), cv2.FONT_HERSHEY_SI

    out.write(frame)
    ret, frame = cap.read()

    if ret and frame is not None:
        H, W, _ = frame.shape
    else:
        break
```

9) End code
    a. This is just some clean up

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

When you run the code, it should output a video in the video folder called '[name]_out.avi'. When you watch the video, it should have added boxes and labels to the various objects you trained the AI for.


SHEP