Yolov8n:

from ultralytics import YOLO

# Define the model and start training

model = YOLO("yolov8n.pt")

# Train the model

model.train(data=r"C:\Users\data.yaml", epochs=60, imgsz=640)

Yolov8s:

!pip install ultralytics --upgrade # Install YOLOv8

import ultralytics

ultralytics.checks() # Verify installation

import torch

import os

from ultralytics import YOLO

import cv2

import matplotlib.pyplot as plt

device = 'cuda' if torch.cuda.is\_available() else 'cpu'

print(f"Using device: {device}")

model = YOLO("yolov8s.pt")

model.train(data=r"C:\Users\dataset\data.yaml", # Path to your dataset YAML file

epochs=40, # Adjust based on need (increase for better accuracy)

batch=4, # Adjust batch size based on GPU memory

imgsz=640, # Image size (default is 640)

device=device, # Train on GPU if available

workers=4, # Number of CPU workers

project="cotton\_training", # Folder for saving runs

name="yolov8s\_cotton") # Custom name for this training run

Yolov10n:

model = YOLO("yolov10n.pt")

dataset\_path = r"C:\Users\dataset" # Update this path

model.train(

data=r"C:\Users\data.yaml", # Ensure data.yaml is correctly set

epochs=50, # Adjust based on your needs

imgsz=640, # Image size

batch=8, # Adjust based on GPU memory

device=device

)

Yolov10s:

In colab, rest were in jupyter notebook

!pip install -q ultralytics

from ultralytics import YOLO

from google.colab import drive

drive.mount('/content/drive')

model = YOLO("yolov10s.pt")  # Load YOLOv10s model

model.train(data="/content/drive/MyDrive/dataset/data.yaml", epochs=50, imgsz=640, batch=8)

To run the model in the drone(Complete code with cv and gps logging)

import cv2

import time

import os

import random

import string

import threading

import csv

from pymavlink import mavutil

from ultralytics import YOLO

# Load YOLOv10s model

model = YOLO('/home/orin/best.pt')

# Generate video and CSV filenames

def generate\_random\_filename():

random\_str = ''.join(random.choices(string.ascii\_letters + string.digits, k=8))

return f"/home/orin/flight\_video\_{random\_str}.avi"

# Detect camera

for cam\_index in [0, 1]:

cap = cv2.VideoCapture(cam\_index)

if cap.isOpened():

print(f"Camera detected at index {cam\_index}")

break

cap.release()

if not cap.isOpened():

print("No camera detected. Exiting...")

exit(1)

print("Starting video recording...")

time.sleep(2)

video\_filename = generate\_random\_filename()

csv\_filename = video\_filename.replace(".avi", ".csv")

# Get resolution and FPS

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fps = cap.get(cv2.CAP\_PROP\_FPS)

if fps == 0 or fps is None:

fps = 30.0

# Video writer

out = cv2.VideoWriter(video\_filename, cv2.VideoWriter\_fourcc(\*'MJPG'), fps, (frame\_width, frame\_height))

# Font configs

font = cv2.FONT\_HERSHEY\_SIMPLEX

timestamp\_pos = (10, 30)

gps\_pos = (10, 70)

status\_pos = (10, 110)

font\_scale = 0.8

font\_color = (0, 255, 255)

thickness = 2

latest\_gps = None

# GPS thread

def gps\_thread():

global latest\_gps

print("Connecting to MAVLink for GPS data...")

connection = mavutil.mavlink\_connection('udp:127.0.0.1:14550')

while True:

msg = connection.recv\_match(type='GPS\_RAW\_INT', blocking=True)

if msg:

lat = msg.lat / 1e7

lon = msg.lon / 1e7

alt = msg.alt / 1000.0

latest\_gps = (lat, lon, alt)

# Start GPS thread

t = threading.Thread(target=gps\_thread, daemon=True)

t.start()

# CSV Logging

with open(csv\_filename, mode='w', newline='') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(['timestamp\_sec', 'latitude', 'longitude', 'altitude', 'status'])

try:

start\_time\_global = time.time()

last\_logged\_sec = -1

frame\_counter = 0

last\_status = "healthy"

while cap.isOpened():

ret, frame = cap.read()

if not ret:

print("Camera disconnected. Initiating shutdown...")

os.system("shutdown -h now")

break

frame\_counter += 1

draw\_frame = frame.copy() # draw on copy to preserve original

# YOLO every 10th frame (non-blocking recording)

if frame\_counter % 10 == 0:

results = model.predict(source=frame, conf=0.4, device=0, verbose=False)

found\_unhealthy = False

for r in results:

for box in r.boxes:

cls = int(box.cls[0].item())

label = model.names[cls].lower()

conf = float(box.conf[0].item())

x1, y1, x2, y2 = map(int, box.xyxy[0])

color = (0, 0, 255) if "unhealthy" in label else (0, 255, 0)

cv2.rectangle(draw\_frame, (x1, y1), (x2, y2), color, 2)

cv2.putText(draw\_frame, f"{label} {conf:.2f}", (x1, y1 - 10), font, 0.6, color, 2)

if "unhealthy" in label:

found\_unhealthy = True

last\_status = "unhealthy" if found\_unhealthy else "healthy"

# Overlay time, GPS, status

current\_time = time.strftime("%H:%M:%S")

cv2.putText(draw\_frame, f"Time: {current\_time}", timestamp\_pos, font, font\_scale, font\_color, thickness)

if latest\_gps:

gps\_text = f"GPS: Lat={latest\_gps[0]:.6f}, Lon={latest\_gps[1]:.6f}, Alt={latest\_gps[2]:.2f}m"

cv2.putText(draw\_frame, gps\_text, gps\_pos, font, font\_scale, font\_color, thickness)

cv2.putText(draw\_frame, f"Status: {last\_status}", status\_pos, font, font\_scale,

(0, 255, 0) if last\_status == "healthy" else (0, 0, 255), thickness)

# Save the annotated frame

out.write(draw\_frame)

# Log to CSV every second

elapsed\_sec = int(time.time() - start\_time\_global)

if elapsed\_sec != last\_logged\_sec:

last\_logged\_sec = elapsed\_sec

if latest\_gps:

writer.writerow([elapsed\_sec, latest\_gps[0], latest\_gps[1], latest\_gps[2], last\_status])

if frame\_counter > 10000:

frame\_counter = 0 # reset to avoid overflow

except KeyboardInterrupt:

print("Recording interrupted.")

cap.release()

out.release()

cv2.destroyAllWindows()

print(f"Video saved: {video\_filename}")

print(f"CSV log saved: {csv\_filename}")