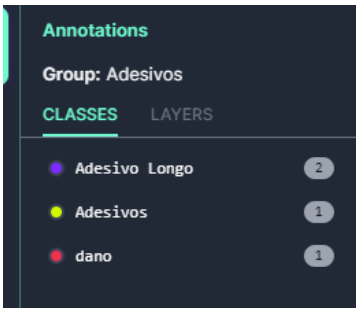
Documentação Projeto RCD

Como treinar o modelo com dataset personalizado.

1- Primeiro é a coleta de dados. É necessário tirar fotos do objeto que deseja que a IA reconheça. Em seguida, utilizamos uma ferramenta que auxilia na construção do dataset personalizado (https://roboflow.com), onde mapeamos manualmente algumas fotos (1166). Usamos um sistema de classes e é gerada uma label com coordenadas das classes "Adesivo longo", "Adesivo“ e “dano". Acredito que, com outros objetos, devemos usar o dataset de adesivos junto com o dataset do outro objeto, pois quanto maior a variedade da classe "dano", melhor a IA reconhecerá o que é um dano. Tivemos uma alta confiança na classe adesivo, então não precisamos mais treinar a IA com o objeto "adesivo". Atualmente, ela identifica qualquer foto que possua "adesivo Mills" e "Mills 0800".

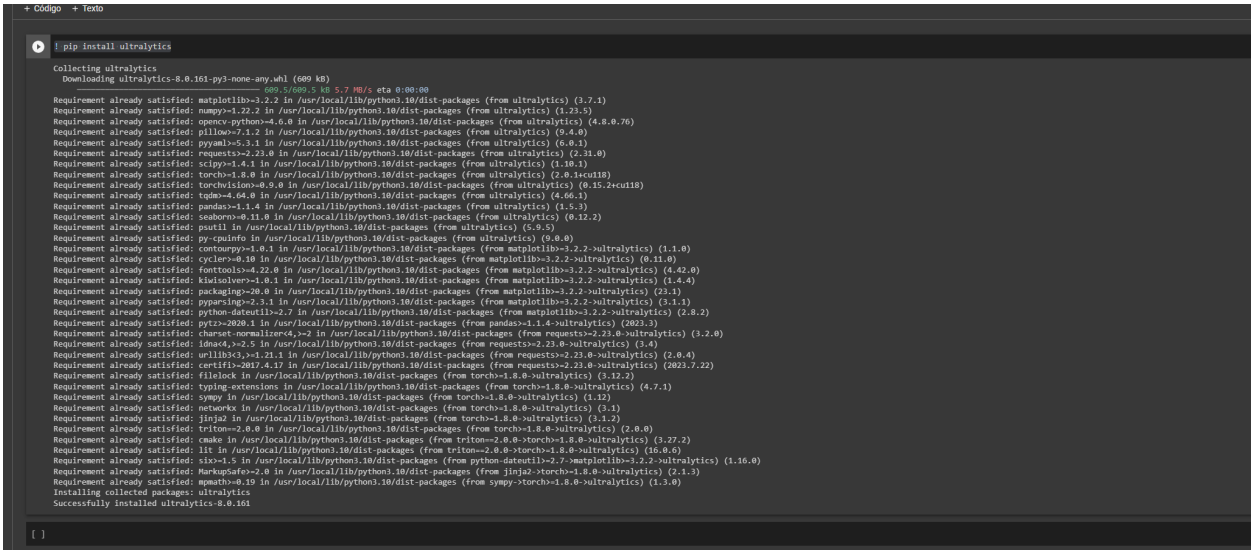


Para replicar, basta fazer a marcação do objeto e rotular em todas as fotos. (Obs: Temos mais de 1k fotos de adesivo com alguns danos. A IA, em muitos casos, tem mais de 90% de confiança. A ideia principal, ao trabalhar com outros objetos, seria usar o dataset de adesivos e adicionar o novo objeto, incluindo a classe desse objeto no mesmo dataset). Após isso, partimos para o passo 2.

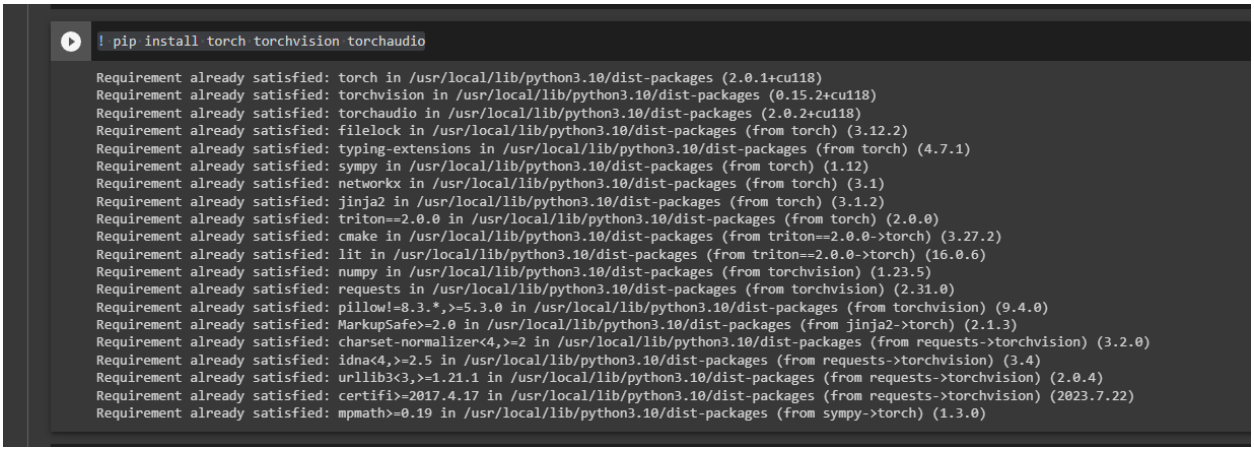
2- Neste segundo passo, ocorre o treinamento efetivo da IA. Precisamos de muito poder computacional. Utilizamos uma plataforma do Google voltada para a comunidade de machine learning. Eles disponibilizam uma GPU para realizar esses processos. Como há limites diários, recomendo comprar alguns créditos para não perder a GPU durante o treinamento.

Usamos a biblioteca ultralytics (YOLOv8. https://github.com/ultralytics/ultralytics)

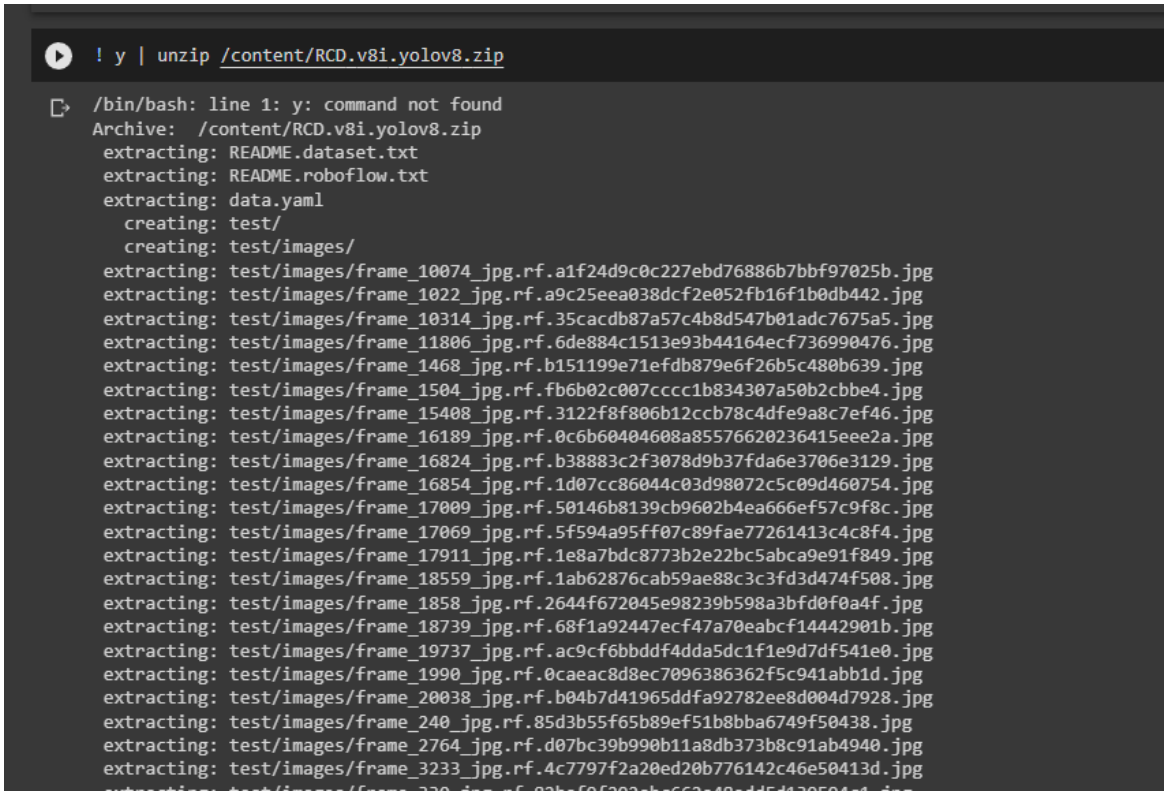
A primeira célula é para a instalação da biblioteca (!pip install ultralytics).



A segunda célula é para a instalação do torch (!pip install torch torchvision torchaudio).



A terceira célula seria para descompactar o dataset (caso não saiba como fazer, há uma pasta ao lado onde se fazem os uploads dos arquivos para o servidor do Colab) (!y | unzip /content/RCD.v8i.yolov8.zip).



Na próxima célula, fazemos a importação do torch, e o treinamento.

! yolo segment train data=data.yaml model=yolov8l-seg.yaml epochs=150 imgsz=640 batch=16

Chamamos o Yolo (yolo) na task de treinamento de segmentação (segment train) com o nosso dataset o data.yaml esta nos arquivos do dataset (data=data.yaml) chamamos o modelo yolob8”L” segundo melhor em precisão (esta na documentação esses modelos) com a tarefa de segmentação (seg.yaml) em 150 epocas (seria a passagem completa por todo o conjunto) imgsz é o tamanho da imagem (640) e colocamos um batch de 16 (tamanho do lote seria o numero de exemplos usados em uma interação), para fazer os pesos recomendo saber sobre o basico de machine learning.

```
[ ] import torch

! yolo segment train data=data.yaml model=yolov8l-seg.yaml epochs=150 imgsz=640 batch=16
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.28it/s]
                     all        105        196      0.855      0.836      0.861      0.727      0.909      0.781      0.832       0.65

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    143/150      11.7G     0.5794     0.7488     0.4174     0.9148          3        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.10it/s]
                     all        105        196      0.928      0.802      0.866      0.725      0.929      0.786      0.839      0.654

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    144/150      11.7G     0.5972     0.7687     0.4236     0.9252         10        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.29it/s]
                     all        105        196      0.911      0.811       0.86      0.727      0.889      0.795      0.839      0.657

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    145/150      11.7G     0.5798     0.7585     0.4123     0.9153          5        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.05it/s]
                     all        105        196       0.93      0.797      0.859       0.73        0.9      0.782      0.825      0.654

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    146/150      11.7G     0.5805     0.7536     0.4106     0.9198          4        640: 100% 64/64 [00:59<00:00,  1.07it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.26it/s]
                     all        105        196      0.903      0.819      0.867      0.728      0.886      0.797      0.833      0.652

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    147/150      11.8G     0.5839     0.7718     0.4144     0.9135          3        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.14it/s]
                     all        105        196      0.911      0.815      0.869      0.735      0.929      0.789      0.835      0.651

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    148/150      11.8G     0.5671     0.7451     0.4106     0.9042          4        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.28it/s]
                     all        105        196      0.914      0.807      0.869      0.731      0.913      0.796      0.843      0.654

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    149/150      11.8G     0.5774     0.7494     0.4145     0.9157         11        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:03<00:00,  1.26it/s]
                     all        105        196       0.92      0.812      0.868      0.735      0.889      0.792      0.836      0.656

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
    150/150      11.7G      0.556     0.7564     0.4068     0.9107          4        640: 100% 64/64 [00:59<00:00,  1.08it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:08<00:00,  2.01s/it]
                     all        105        196      0.921      0.808      0.868       0.74      0.899      0.792      0.834      0.658

150 epochs completed in 2.908 hours.
Optimizer stripped from runs/segment/train3/weights/last.pt, 92.3MB
Optimizer stripped from runs/segment/train3/weights/best.pt, 92.3MB

Validating runs/segment/train3/weights/best.pt...
Ultralytics YOLOv8.0.153 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-seg summary (fused): 295 layers, 45914201 parameters, 0 gradients
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 4/4 [00:07<00:00,  1.95s/it]
                     all        105        196      0.921      0.808      0.868      0.739      0.899      0.792      0.834      0.658
          Adesivo Longo        105         71      0.911      0.958      0.947      0.886      0.913      0.958      0.947      0.834
               Adesivos        105         59      0.932      0.949       0.96       0.93      0.932      0.949       0.96      0.876
                   dano        105         66      0.919      0.516      0.697      0.402      0.851       0.47      0.596      0.263
Speed: 6.5ms preprocess, 19.7ms inference, 0.0ms loss, 2.4ms postprocess per image
```

nessa celula usamos metodos de validação onde é gerado as metricas do modelo

! yolo segment val model=runs/segment/train5/weights/best.pt data=data.yaml

vai ser gerado esse arquivo best.pt na pasta wights

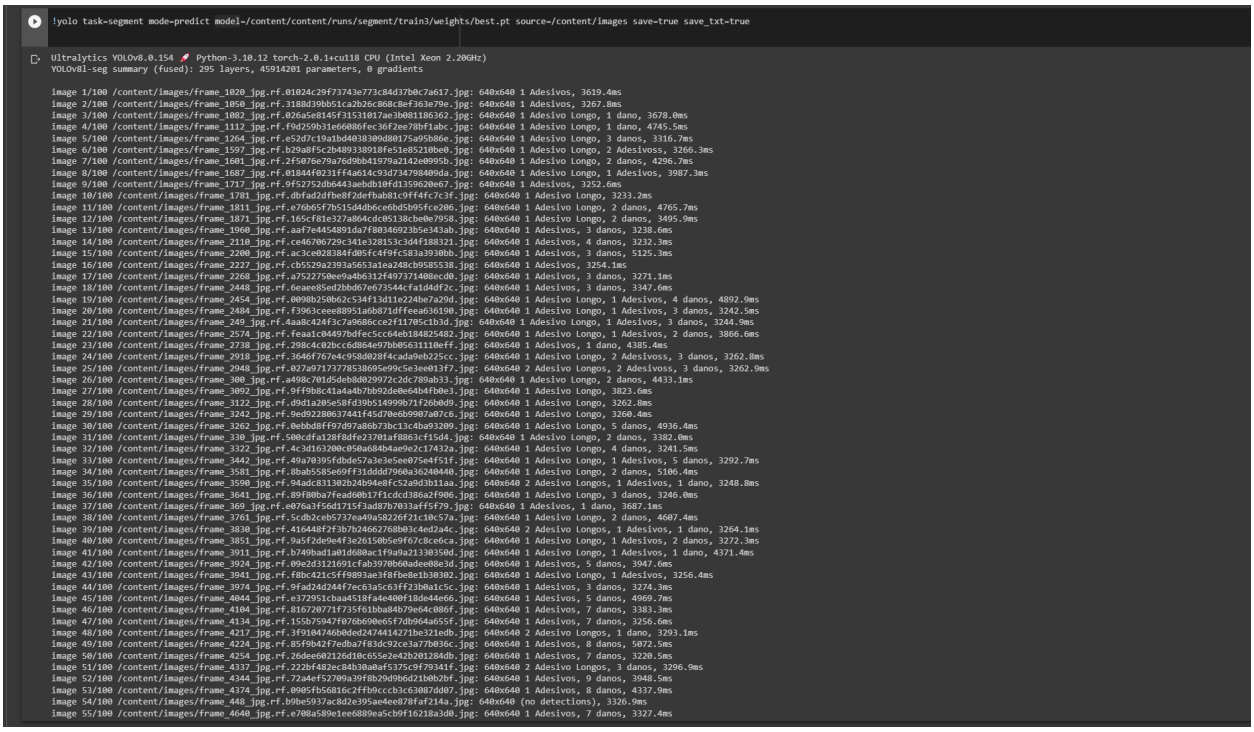
```
! yolo segment val model=runs/segment/train5/weights/best.pt data=data.yaml

Ultralytics YOLOv8.0.152 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-seg summary (fused): 295 layers, 45914201 parameters, 0 gradients
val: Scanning /content/valid/labels.cache... 105 images, 1 backgrounds, 0 corrupt: 100% 105/105 [00:00<?, ?it/s]
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100% 7/7 [00:10<00:00,  1.47s/it]
                     all        105        196      0.902      0.797      0.837      0.721      0.908      0.795      0.826      0.649
          Adesivo Longo        105         71      0.924      0.958      0.956      0.898      0.927      0.958      0.956       0.83
               Adesivos        105         59      0.905      0.949      0.958      0.922       0.91      0.949      0.958      0.865
                   dano        105         66      0.876      0.485      0.598      0.344      0.887      0.478      0.564      0.253
Speed: 7.2ms preprocess, 42.4ms inference, 0.0ms loss, 2.9ms postprocess per image
Results saved to runs/segment/val
```

Aqui fazemos o teste do modelo

!yolo task=segment mode=predict model=/content/content/runs/segment/train3/weights/best.pt source=/content/images save=true save_txt=true

chamamos a task de segmentação no modo predict e colocamos o path do nosso modelo recem treinado e escolhemos onde salvar



-Customizações do algoritimo

Aqui temos um algoritimo de classificação de danos, area do dano em relação ao adesvio.

```
import cv2
import os
import numpy as np
from ultralytics import YOLO
import torch

def is_inside(box1, box2):
    """Verifica se o centro de box1 está dentro de box2."""
    cx1, cy1 = (box1[0] + box1[2]) // 2, (box1[1] + box1[3]) // 2
    return box2[0] <= cx1 <= box2[2] and box2[1] <= cy1 <= box2[3]

def calculate_area(box):
    x1, y1, x2, y2 = box[:4]
    return (x2 - x1) * (y2 - y1)

def calculate_damage_severity(adesivo_box, danos_inside_boxes):
    adesivo_area = calculate_area(adesivo_box)
    total_damage_area = sum([calculate_area(dano) for dano in danos_inside_boxes])

    damage_ratio = total_damage_area / adesivo_area

    if damage_ratio < 0.05:
        return "Leve"
    elif damage_ratio < 0.1:
        return "Medio"
    else:
        return "Severo"

def process_frame(frame):
    results = model(frame)
    result = results[0]
    detected_boxes = result.boxes.data.cpu().numpy()

    detected_masks = None if result.masks is None else result.masks.data.cpu().numpy()

    adesivo_counter = 0
```

```
    adesivo_longo_counter = 0
    total_danos = 0

    danos_severidade_info = []

    for i, box in enumerate(detected_boxes):
        x1, y1, x2, y2, score, class_id = map(int, box)
        label = class_labels[class_id]

        if class_id == 0 or class_id == 1:
            danos_inside_boxes = [dano_box for dano_box in detected_boxes if int(dano_box[5]) == 2 and is_inside(dano_box[:4], box[:4])]
            severity = calculate_damage_severity(box, danos_inside_boxes)
            danos_inside = len(danos_inside_boxes)

            if class_id == 0:
                label += f"-{adesivo_longo_counter} tem {danos_inside} danos com severidade {severity}"
                adesivo_longo_counter += 1
            else:
                label += f"-{adesivo_counter} tem {danos_inside} danos com severidade {severity}"
                adesivo_counter += 1

            total_danos += danos_inside
            danos_severidade_info.append((danos_inside, severity))

        if detected_masks is not None:
            mask = detected_masks[i]
            mask_resized = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
            overlay = frame.copy()
            overlay[mask_resized > 0.5] = segmentation_color[:3]
            alpha = 0.5
            frame = cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0)

        cv2.rectangle(frame, (x1, y1), (x2, y2), box_color, box_thickness)
        cv2.putText(frame, label, (x1, y1 - 10), text_font, text_size, text_color, text_thickness)

    return frame, total_danos, danos_severidade_info

torch.set_default_tensor_type('torch.FloatTensor')
weights_path = r'E:\Mills\ProjectMills\IARCD\fotoscolab\train\content\runs\segment\train5\weights\best.pt'
model = YOLO(weights_path)

folder_path = r'C:\Users\samuel.grillo\OneDrive - MILLS LOCAÇÃO, SERVIÇOS E LOGÍSTICA S.A\Documents\IA-RCD\datacolab-ruim\train\images'
save_folder = r'C:\Users\samuel.grillo\OneDrive - MILLS LOCAÇÃO, SERVIÇOS E LOGÍSTICA S.A\Documents\IA-RCD\salvafotos'
os.makedirs(save_folder, exist_ok=True)

class_labels = {
    0: "Adesivo Longo",
    1: "Adesivos",
    2: "dano"
}

box_color =  (2,64,195)
text_color = (255,255,255)
segmentation_color = (33,112,243)


box_thickness = 2
text_font = cv2.FONT_HERSHEY_PLAIN
text_size = 1.2
#text_size = 3
text_thickness = 2


for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)
    save_path = os.path.join(save_folder, filename)

    if filename.endswith('.jpg') or filename.endswith('.png'):
        image = cv2.imread(file_path)
        processed_image, total_danos, danos_severidade_info = process_frame(image)
        cv2.imwrite(save_path, processed_image)
        print(f"Imagem processada salva em {save_path}")
        print(f"Total de danos na imagem: {total_danos}")
        for i, (qtd, severity) in enumerate(danos_severidade_info):
            print(f"Adesivo {i+1}: Quantidade de danos: {qtd}, Severidade: {severity}")

    elif filename.endswith('.MOV') or filename.endswith('.mp4'):
        cap = cv2.VideoCapture(file_path)
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
```

```
        out = cv2.VideoWriter(save_path, fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))

        while(cap.isOpened()):
            ret, frame = cap.read()
            if ret:
                processed_frame, _, _ = process_frame(frame)
                out.write(processed_frame)
            else:
                break

        cap.release()
        out.release()
        print(f"Vídeo processado salvo em {save_path}")
```

1. `is_inside(box1, box2)`

Objetivo: Verifica se o centro de `box1` está localizado dentro do `box2`

Entradas:

- `box1` e `box2`: Array que representam bounding boxes no formato [x1, y1, x2, y2], onde (x1,y1) é o canto superior esquerdo e (x2,y2) é o canto inferior direito.

Retorno:

- Retorna `True` se o centro de `box1` estiver dentro de `box2`. Caso contrário, retorna `False`.

---

2. `calculate_area(box)`

Objetivo: Calcula a área de um bounding box.

Entrada:

- `box`: Representando um bounding box no formato [x1, y1, x2, y2].

Retorno:

- A área do bounding box.

---

3. `calculate_damage_severity(adesivo_box, danos_inside_boxes)`

Objetivo: Determinar a severidade do dano com base na relação entre a área do dano e a área do adesivo.

Entradas:

- `adesivo_box`: Bounding box do adesivo.
- `danos_inside_boxes`: Uma lista de bounding boxes de danos dentro do adesivo.

Retorno:

- Uma string indicando a severidade do dano: "Leve", "Medio" ou "Severo".

---

4. `process_frame(frame)`

Objetivo: Processar cada frame de uma imagem ou vídeo, realizar a detecção de objetos usando o modelo YOLO, classificar e anotar os adesivos com danos e, se disponíveis, sobrepor as máscaras de segmentação na imagem.

Entrada:

- `frame`: Uma imagem ou um frame de vídeo.

Retorno:

- `frame`: O frame processado com anotações.
- `total_danos`: O número total de danos detectados no frame.

- `danos_severidade_info`: Uma lista de tuplas, onde cada tupla contém a quantidade de danos e a severidade associada a cada adesivo.