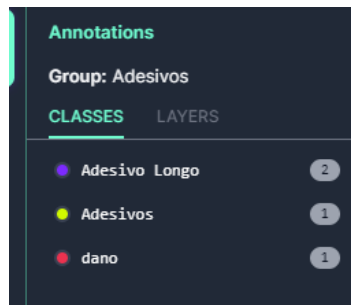


Documentação Projeto RCD

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).

```
+ Código + Texto

! pip install ultralytics

Collecting ultralytics
  Downloading ultralytics-8.0.101-py3-none-any.whl (600 kB)
    600/5/600.5 KB 5.7 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>3.2.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.23.5)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.0.76)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.1)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.10.4)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.0.1+cu118)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.15.2+cu118)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.1)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.5.3)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: puzill in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.8.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (4.42.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (23.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2023.7.22)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.12.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (4.7.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (3.27.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (16.0.6)
Requirement already satisfied: sio>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.2.2->ultralytics) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch>=1.8.0->ultralytics) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.8.0->ultralytics) (1.3.0)
Installing collected packages: ultralytics
Successfully installed ultralytics-8.0.101
```

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

```
! pip install torch torchvision torchaudio

Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.0.1+cu118)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.15.2+cu118)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.0.2+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.12.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.7.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.1)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (3.27.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (16.0.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow>=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2023.7.22)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

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).

```
! y | unzip /content/RCD.v8i.yolov8.zip

/bin/bash: line 1: y: command not found
Archive: /content/RCD.v8i.yolov8.zip
extracting: README.dataset.txt
extracting: README.roboflow.txt
extracting: data.yaml
creating: test/
creating: test/images/
extracting: test/images/frame_10074_jpg.rf.a1f24d9c0c227ebd76886b7bbf97025b.jpg
extracting: test/images/frame_1022_jpg.rf.a9c25eea038dcf2e052fb16f1b0db442.jpg
extracting: test/images/frame_10314_jpg.rf.35cacdb87a57c4b8d547b01adc7675a5.jpg
extracting: test/images/frame_11806_jpg.rf.6de884c1513e93b44164ecf736990476.jpg
extracting: test/images/frame_1468_jpg.rf.b151199e71efdb879e6f26b5c480b639.jpg
extracting: test/images/frame_1504_jpg.rf.fb6b02c007cccc1b834307a50b2cbb4e.jpg
extracting: test/images/frame_15408_jpg.rf.3122f8f806b12ccb78c4dfe9a8c7ef46.jpg
extracting: test/images/frame_16189_jpg.rf.0c6b60404608a85576620236415eee2a.jpg
extracting: test/images/frame_16824_jpg.rf.b38883c2f3078d9b37fda6e3706e3129.jpg
extracting: test/images/frame_16854_jpg.rf.1d07cc86044c03d98072c5c09d460754.jpg
extracting: test/images/frame_17009_jpg.rf.50146b8139cb9602b4ea666ef57c9f8c.jpg
extracting: test/images/frame_17069_jpg.rf.5f594a95ff07c89fae77261413c4c8f4.jpg
extracting: test/images/frame_17911_jpg.rf.1e8a7bdc8773b2e22bc5abca9e91f849.jpg
extracting: test/images/frame_18559_jpg.rf.1ab62876cab59ae88c3c3fd3d474f508.jpg
extracting: test/images/frame_1858_jpg.rf.2644f672045e98239b598a3bfd0f0a4f.jpg
extracting: test/images/frame_18739_jpg.rf.68f1a92447ecf47a70eabcf14442901b.jpg
extracting: test/images/frame_19737_jpg.rf.ac9cf6bbddf4dda5dc1f1e9d7df541e0.jpg
extracting: test/images/frame_1990_jpg.rf.0caeac8d8ec7096386362f5c941abb1d.jpg
extracting: test/images/frame_20038_jpg.rf.b04b7d41965ddfa92782ee8d004d7928.jpg
extracting: test/images/frame_240_jpg.rf.85d3b55f65b89ef51b8bba6749f50438.jpg
extracting: test/images/frame_2764_jpg.rf.d07bc39b990b11a8db373b8c91ab4940.jpg
extracting: test/images/frame_3233_jpg.rf.4c7797f2a20ed20b776142c46e50413d.jpg
```

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 yolov8l" 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  0.882  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 YOLOv8.0.152 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 [00:00:00, ?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      0.932  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 [00:00:00, ?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      0.932  0.949  0.96  0.93  0.932  0.949  0.96  0.876
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 recém treinado e escolhemos onde salvar

```
!yolo task=segment mode=predict model=/content/content/runs/segment/train3/weights/best.pt source=/content/images save=true save_txt=true
YOLOv11-seg summary (fused): 295 layers, 45914201 parameters, 0 gradients

image 1/100 /content/images/frame_1020.jpg.rf.01824c29f71743e773c84d17b0c7a617.jpg: 640x640 1 Adesivos, 3619.4ms
image 2/100 /content/images/frame_1050.jpg.rf.3188b19b051ca2b26c8d8c8ef163e79e.jpg: 640x640 1 Adesivos, 3267.1ms
image 3/100 /content/images/frame_1081.jpg.rf.026a6e0145f313107a2e3b081186162.jpg: 640x640 1 Adesivo Longo, 1 dano, 3678.0ms
image 4/100 /content/images/frame_1112.jpg.rf.f9d259b31e66086fec36f2ee78bf1abc.jpg: 640x640 1 Adesivo Longo, 1 dano, 4745.5ms
image 5/100 /content/images/frame_1264.jpg.rf.e52d7c19a1b4d83839d80175a95b86e.jpg: 640x640 1 Adesivo Longo, 3 danos, 3316.7ms
image 6/100 /content/images/frame_1597.jpg.rf.02a04f5c2b49339818f5e3d53210b0e.jpg: 640x640 1 Adesivo Longo, 2 Adesivos, 3260.3ms
image 7/100 /content/images/frame_1691.jpg.rf.2f5876c79a7690b41792a1242e6905b.jpg: 640x640 1 Adesivo Longo, 2 danos, 4295.7ms
image 8/100 /content/images/frame_1687.jpg.rf.01844f0231ff4a614c93d73479849dda.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 3987.3ms
image 9/100 /content/images/frame_1717.jpg.rf.9f52752d6e441aebd18fd135926b67.jpg: 640x640 1 Adesivos, 3252.6ms
image 10/100 /content/images/frame_1741.jpg.rf.dbfad21bf887d6eb1abb81c0ff4fcfc31.jpg: 640x640 1 Adesivo Longo, 3233.2ms
image 11/100 /content/images/frame_1811.jpg.rf.e7db65f7b515d4db6ce6bd5b95fce206.jpg: 640x640 1 Adesivo Longo, 2 danos, 4765.7ms
image 12/100 /content/images/frame_1871.jpg.rf.165cf81e127a864cd851381cbebe795b.jpg: 640x640 1 Adesivo Longo, 2 danos, 3495.9ms
image 13/100 /content/images/frame_1960.jpg.rf.aaf44549491d7f00346923b5c31b3b.jpg: 640x640 1 Adesivos, 3 danos, 3238.6ms
image 14/100 /content/images/frame_2110.jpg.rf.c46796729c341e328153c3d8f188321.jpg: 640x640 1 Adesivos, 4 danos, 3232.3ms
image 15/100 /content/images/frame_2200.jpg.rf.ac3ce028184f4d85fc4f9fc583a3910bb.jpg: 640x640 1 Adesivos, 3 danos, 5125.3ms
image 16/100 /content/images/frame_2227.jpg.rf.c3529a2393a5653a1ea248c1958553b.jpg: 640x640 1 Adesivos, 3254.1ms
image 17/100 /content/images/frame_2240.jpg.rf.0752272b0a94b6312f407271480bcdb.jpg: 640x640 1 Adesivos, 3 danos, 3271.1ms
image 18/100 /content/images/frame_2440.jpg.rf.6eaee85ed2bbd67e673544cafa1d4dfc.jpg: 640x640 1 Adesivos, 3 danos, 3347.6ms
image 19/100 /content/images/frame_2454.jpg.rf.0098b258b62c534f13d11e224be7a29d.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 4 danos, 4892.9ms
image 20/100 /content/images/frame_2481.jpg.rf.f3983c4ee08951a6b871d7feaa36190.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 3 danos, 3242.5ms
image 21/100 /content/images/frame_249.jpg.rf.4a8a424f3c74989ccce111701c13d4.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 3 danos, 3244.9ms
image 22/100 /content/images/frame_2574.jpg.rf.feal1c8497bdfcc5cc64eb184825482.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 2 danos, 3866.6ms
image 23/100 /content/images/frame_2738.jpg.rf.798c4c02bc6d884e97bb05631110eff.jpg: 640x640 1 Adesivos, 1 dano, 4385.4ms
image 24/100 /content/images/frame_2910.jpg.rf.2648f7744c95d803ff4c4dab0d25c5c.jpg: 640x640 1 Adesivo Longo, 2 Adesivos, 3 danos, 3262.9ms
image 25/100 /content/images/frame_2940.jpg.rf.027a071737783895e99c5e3ee013f7.jpg: 640x640 2 Adesivo Longos, 2 Adesivos, 3 danos, 3262.9ms
image 26/100 /content/images/frame_300.jpg.rf.a498c701d5debbd829972cd789ab33.jpg: 640x640 1 Adesivo Longo, 2 danos, 4433.1ms
image 27/100 /content/images/frame_3092.jpg.rf.9ff8b041414ab7b092d8e6d4b4f0e1.jpg: 640x640 1 Adesivo Longo, 3823.6ms
image 28/100 /content/images/frame_3122.jpg.rf.d91d3d8e48f19b0c3499971120b0b.jpg: 640x640 1 Adesivo Longo, 3262.8ms
image 29/100 /content/images/frame_3242.jpg.rf.9ed92280617441f45d70eb9907a07c6.jpg: 640x640 1 Adesivo Longo, 3260.4ms
image 30/100 /content/images/frame_3262.jpg.rf.bebdbff9f097a8b731c11cd4b91289.jpg: 640x640 1 Adesivo Longo, 5 danos, 4936.4ms
image 31/100 /content/images/frame_330.jpg.rf.58b0cfaf12818f6c2701aaf081cf16d4.jpg: 640x640 1 Adesivo Longo, 2 danos, 3302.0ms
image 32/100 /content/images/frame_3322.jpg.rf.4c3d161206c058a0d84b4ae9e2c17432a.jpg: 640x640 1 Adesivo Longo, 4 danos, 3241.5ms
image 33/100 /content/images/frame_3442.jpg.rf.49a70395fdbde57a3e3ee075e4f51f.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 5 danos, 3292.7ms
image 34/100 /content/images/frame_3581.jpg.rf.8ba5585e49ff31d4dd7968a3d24b44d.jpg: 640x640 1 Adesivo Longo, 2 danos, 5106.4ms
image 35/100 /content/images/frame_3590.jpg.rf.94a4c6313b01c4b9a4ef5c2a9d311aa.jpg: 640x640 2 Adesivo Longos, 1 Adesivos, 1 dano, 3243.8ms
image 36/100 /content/images/frame_3641.jpg.rf.89f80ba7ead68b17f1cdd386a2f906.jpg: 640x640 1 Adesivo Longo, 3 danos, 3246.0ms
image 37/100 /content/images/frame_369.jpg.rf.e078a3f6d1715f3ad87b7033aff5f79.jpg: 640x640 1 Adesivos, 1 dano, 3687.1ms
image 38/100 /content/images/frame_3761.jpg.rf.5c0d2c059737e4945e222f211c0557a.jpg: 640x640 1 Adesivo Longo, 2 danos, 4607.4ms
image 39/100 /content/images/frame_3830.jpg.rf.416448f2f3b7b246c2768b03c4ed2a4c.jpg: 640x640 2 Adesivo Longos, 1 Adesivos, 1 dano, 3264.1ms
image 40/100 /content/images/frame_3851.jpg.rf.9a5f2de9e4f3e26150b5e9f67c8ce6ca.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 2 danos, 3272.3ms
image 41/100 /content/images/frame_3911.jpg.rf.b749d1a1d1d688ac1f9a9a213835d0.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 1 dano, 4371.4ms
image 42/100 /content/images/frame_3924.jpg.rf.09a25121691cfab397b0bdaee0eb3d.jpg: 640x640 1 Adesivos, 5 danos, 3947.6ms
image 43/100 /content/images/frame_3941.jpg.rf.f8bc421c5ff9891ae3f8fb8e1b10302.jpg: 640x640 1 Adesivo Longo, 1 Adesivos, 3256.4ms
image 44/100 /content/images/frame_3974.jpg.rf.9fad24244f7ec93a5c63ff23b0a1c5c.jpg: 640x640 1 Adesivos, 3 danos, 3274.3ms
image 45/100 /content/images/frame_4044.jpg.rf.e72951cb4a418f44e400f11b04b6e6.jpg: 640x640 1 Adesivos, 5 danos, 4069.7ms
image 46/100 /content/images/frame_4104.jpg.rf.816720771f735f61bba4b79a64c806.jpg: 640x640 1 Adesivos, 7 danos, 3183.1ms
image 47/100 /content/images/frame_4134.jpg.rf.155b75947f076b09b0e5f7db964a655f.jpg: 640x640 1 Adesivos, 7 danos, 3256.6ms
image 48/100 /content/images/frame_4217.jpg.rf.3f9104746b0dd247444217b321e0b.jpg: 640x640 2 Adesivo Longos, 1 dano, 1293.1ms
image 49/100 /content/images/frame_4224.jpg.rf.85f004717edbf78163c2c3a77b039c.jpg: 640x640 1 Adesivos, 8 danos, 5072.5ms
image 50/100 /content/images/frame_4254.jpg.rf.26dee082126d18c65e2e42b201284db.jpg: 640x640 1 Adesivos, 7 danos, 3220.5ms
image 51/100 /content/images/frame_4337.jpg.rf.222bf482ec84b38a8af5375c9f79341f.jpg: 640x640 2 Adesivo Longos, 3 danos, 3296.9ms
image 52/100 /content/images/frame_4344.jpg.rf.72a4ef52709a39f8f29a0b6d1b0b20f.jpg: 640x640 1 Adesivos, 9 danos, 3948.5ms
image 53/100 /content/images/frame_4374.jpg.rf.908f7f6e016c2ffbf80cc3c3c3807d407.jpg: 640x640 1 Adesivos, 8 danos, 4337.9ms
image 54/100 /content/images/frame_448.jpg.rf.b9be5937ac8d2e395aa4ee878faf214a.jpg: 640x640 (no detections), 3326.9ms
image 55/100 /content/images/frame_4640.jpg.rf.e708a580ee6889eac5b9f162181a3d0.jpg: 640x640 1 Adesivos, 7 danos, 3327.4ms
```

-Customizações do algoritmo

Aqui temos um algoritmo 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"Video 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.