# Relatório

# Tech Challenge

# Fase V

Aluno: Claidson Yuldi Kanno

Matrícula: RM355901

14/02/2025

### **Introdução**

Este relatório detalha o fluxo de trabalho empregado no desenvolvimento de uma solução para identificar objetos cortantes utilizando o YOLOv8, alimentado por um conjunto de dados de mais de 1700 imagens marcadas. O YOLOv8 oferece um desempenho de ponta em termos de precisão e velocidade para tarefas de detecção de objetos.

### **Definição do Problema e Objetivos**

O objetivo principal deste projeto é desenvolver um modelo de detecção de objetos capaz de identificar objetos cortantes em imagens e vídeos com alta precisão. A solução visa atender a aplicações como segurança, controle de acesso e monitoramento em ambientes onde a presença de objetos cortantes representa um risco.

### **Conjunto de Dados**

* **Coleta e Marcação:** O conjunto de dados consistiu em mais de 1700 imagens marcadas. As imagens foram coletadas com as identificações e delineações dos objetos cortantes já marcadas.
* **Formato:** Os dados coletados estavam organizados para compatibilidade com o YOLOv8. As imagens foram armazenadas em diretórios separados para treinamento, validação e teste. Arquivos de anotação, contendo as coordenadas das caixas delimitadoras e as classes dos objetos, estavam criados no formato exigido pelo YOLOv8.
* **Estrutura:** Os diretórios de treinamento, validação e teste continham subdiretórios para imagens e labels.

### **Escolha do Modelo: YOLOv8**

O YOLOv8 foi selecionado como a arquitetura base para este projeto devido às suas características e vantagens:

* **Arquitetura Avançada:** YOLOv8 utiliza arquiteturas de backbone (espinha dorsal) e "pescoço" (neck) de última geração, resultando em melhor extração de características.
* **Detecção Sem Âncoras:** Adota uma cabeça de detecção "sem âncoras" (anchor-free), o que contribui para uma melhor precisão e um processo de detecção mais eficiente.
* **Otimização de Precisão e Velocidade:** YOLOv8 busca um equilíbrio ideal entre precisão e velocidade, tornando-o adequado para tarefas de detecção de objetos em tempo real.
* **Variedade de modelos pré-treinados:**YOLOv8 oferece uma gama de modelos pré-treinados para atender a várias tarefas e requisitos de desempenho, tornando mais fácil encontrar o modelo certo para o seu caso de utilização específico.

**Implementação**

* **Endereço do projeto no Git:**
  + <https://github.com/cykanno/postech_iadev.git>
* **Configuração do Ambiente:**
  + Instalação das bibliotecas necessárias, incluindo ultralytics[2](https://docs.ultralytics.com/pt/models/yolov8/)[1](https://visionplatform.ai/pt/yolov8-deteccao-de-objetos-de-ultima-geracao-em-visao-computacional-computer-vision/):  
    pip install -U ultralytics
* **Preparação dos Dados:**
  + Criação de um arquivo de configuração YAML (data\_yolov8.yaml) para especificar os caminhos para os diretórios de treinamento, validação e teste, bem como o número de classes e os nomes das classes.  
    O arquivo de configuração está no Git.
* **Treinamento do Modelo:**
  + Utilização da classe YOLO da biblioteca ultralytics para carregar um modelo pré-treinado (yolov8n.pt).
  + Execução do treinamento especificando o arquivo de configuração YAML, o número de épocas e o tamanho das imagens.
  + O treino foi realizado com as informações abaixo:  
    yolo detect train data=data\_yolov8.yaml model=yolov8n.pt epochs=100 imgsz=640& f:/Projeto/python/postech/venv/Scripts/python.exe f:/Projeto/python/postech/Fase\_V/YOLOv8/identificarObjetosYOLOv8.py
* **Avaliação do Modelo:**
  + Após o treinamento, o modelo foi avaliado utilizando o conjunto de dados de validação para medir a precisão e outras métricas de desempenho.  
    As imagens estão em anexo no relatório.

**Implantação**

O modelo treinado foi salvo e utilizado na aplicação de detecção de objetos cortantes.

**Conclusão**

Este relatório detalhou o processo de desenvolvimento de uma solução de identificação de objetos cortantes utilizando o YOLOv8, que atende parcialmente à proposta do desafio. Mesmo utilizando um conjunto de dados de alta qualidade, uma arquitetura de modelo avançada e técnicas de treinamento e otimização, não foi possível criar uma solução capaz de atender integralmente às necessidades de aplicações da empresa FIAP VisionGuard, empresa de monitoramento de câmeras de segurança.

**Tentativas de Aprimoramento e Resultados Não Esperados**

Durante o desenvolvimento da solução de identificação de objetos cortantes, diversas tentativas foram realizadas na busca por aprimorar o desempenho do modelo. Algumas dessas iniciativas foram promissoras, outras não resultaram nos ganhos esperados.

**Treinamento realizados:**

**Descrição:** Um conjunto de dados foi baixado, contendo mais de 9000 imagens rotuladas, na esperança de que a maior quantidade de dados melhorasse a precisão do modelo.

**Resultados:** Após o treinamento com este conjunto ampliado, observou-se uma queda no desempenho em comparação ao modelo treinado com 1700 imagens. O reconhecimento de objetos cortante nos mesmos vídeos foi inferior, indicando que a adição de mais dados não contribuiu para a melhoria da detecção.

**Análise:** Talvez o novo conjunto de dados incluía imagens com menor qualidade e maior variação nas condições de captura, o que pode ter dificultado o aprendizado do modelo.

**Processamento:** Uma das características importantes do treinamento para reconhecimento de objetos foi o tempo de processamento. No computador local, o treinamento que apresentou o melhor desempenho levou mais de 20 horas para ser concluído. Em contraste, o treinamento realizado no Google Colab, utilizando uma GPU T4 e um conjunto de dados com mais de 9000 imagens, foi finalizado em pouco mais de 5 horas, com um total de 100 épocas. Cada época consumiu aproximadamente 3 minutos no Colab, enquanto no computador local esse mesmo treinamento levou cerca de 1 hora por época, o que se tornou inviável o treinamento na máquina local.

Diversos treinamentos adicionais foram realizados utilizando conjuntos de dados com menor quantidade de imagens e, em alguns casos, com um número maior de rótulos, até que se identificou o treinamento que apresentou o melhor desempenho. A base de dados foi aprimorada para incluir imagens mais específicas de objetos cortantes e para apoiar esse processo, foram desenvolvidos scripts auxiliares, os quais estão disponíveis no Git.

**Conclusão:**  O tempo de treinamento teve um impacto significativo nas tentativas de realizar novos processamentos com alterações nos parâmetros, visando melhorar os resultados obtidos.

**Informações Complementares**

**Scripts utilizados no projeto**

**identificarObjetosYOLOv8.py** – script principal de identificação de objetos cortantes

**Scripts de apoio utilizados:**Devido a junção de outras imagens marcadas e a exclusão de imagens que não representavam imagens cortantes, os scripts foram desenvolvidos para auxiliar o processo.

**alterarClasseEmLabels.py** – desenvolvido para alterar o identificador de classificação das imagens

**apagarArquivos.py** – desenvolvido para apagar imagens selecionadas que não eram de objetos cortantes

**igualarLabelsImagens.py** – desenvolvido para apagar os arquivos de labels que não tinham imagem relacionada a eles

**Estrutura de Diretórios para Execução**

Os scripts para execução devem estar no diretório:

* /Fase\_V/Hack

O arquivo com as configurações do treino (best.pt) deve estar no diretório:

* /Fase\_V/Hack/treino/

Os arquivos e imagens resultados do treino estão no mesmo diretório acima:

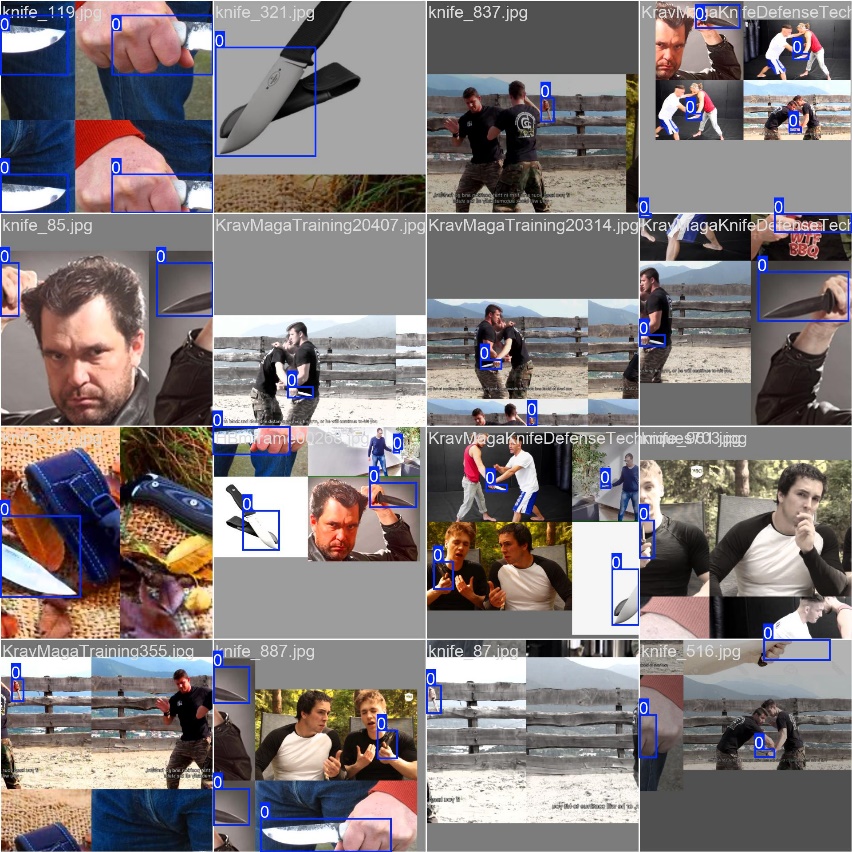
* /Fase\_V/Hack/treino/

**Informações para envio do E-mail**

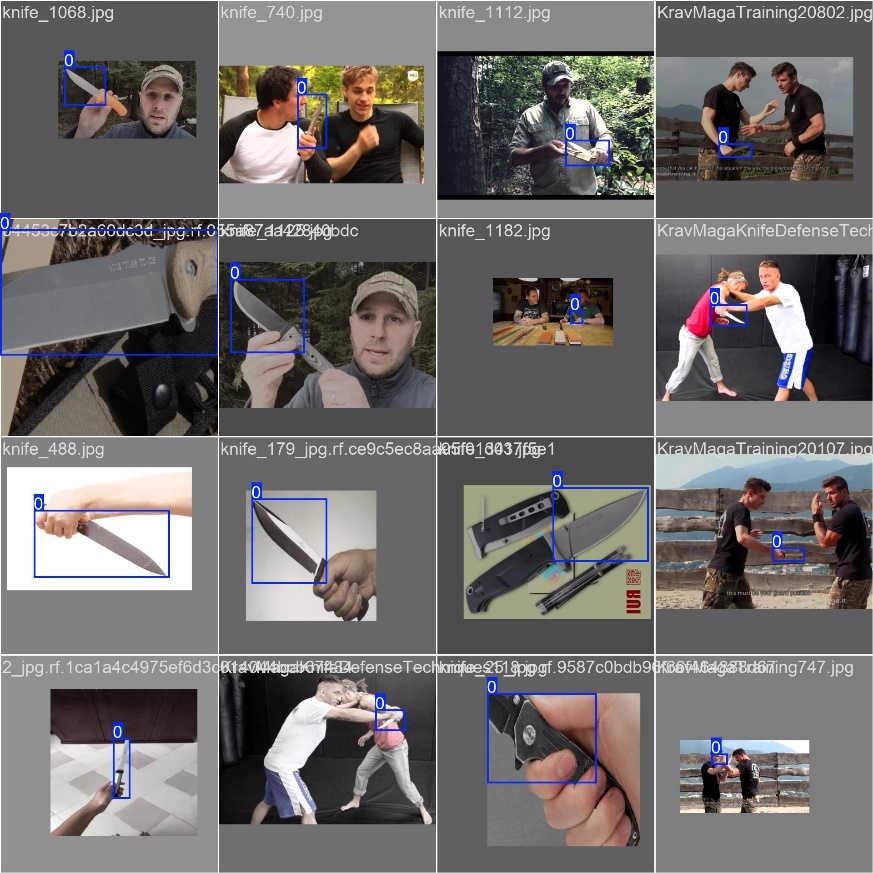
Foi utilizado um e-mail do Gmail com senha de aplicativo que não foram enviados por se tratar de informação particular.

Utilizar um e-mail do Gmail para a execução do projeto.

Variáveis a serem preenchidas:  
remetente = "remente@gmail.com"   
senha = "senha"  
Obs.: Senha do GMail (ou senha de aplicativo se a autenticação de dois fatores estiver ativada)  
destinatario = "destinatario@gmail.com"

**Imagens do resultado do treino**

****

****

**Imagens não reconhecidas no vídeo**

Padrão do plano de fundo

O conteúdo gerado por IA pode estar incorreto.

Uma imagem contendo livro, bolo, mesa, cortando

O conteúdo gerado por IA pode estar incorreto.

Uma imagem contendo mesa, de madeira, velho, quarto

O conteúdo gerado por IA pode estar incorreto.

**Imagens com objetos cortantes reconhecidos no vídeo**

Uma imagem contendo comida, mesa, cortando, faca

O conteúdo gerado por IA pode estar incorreto.

Rosto de pessoa

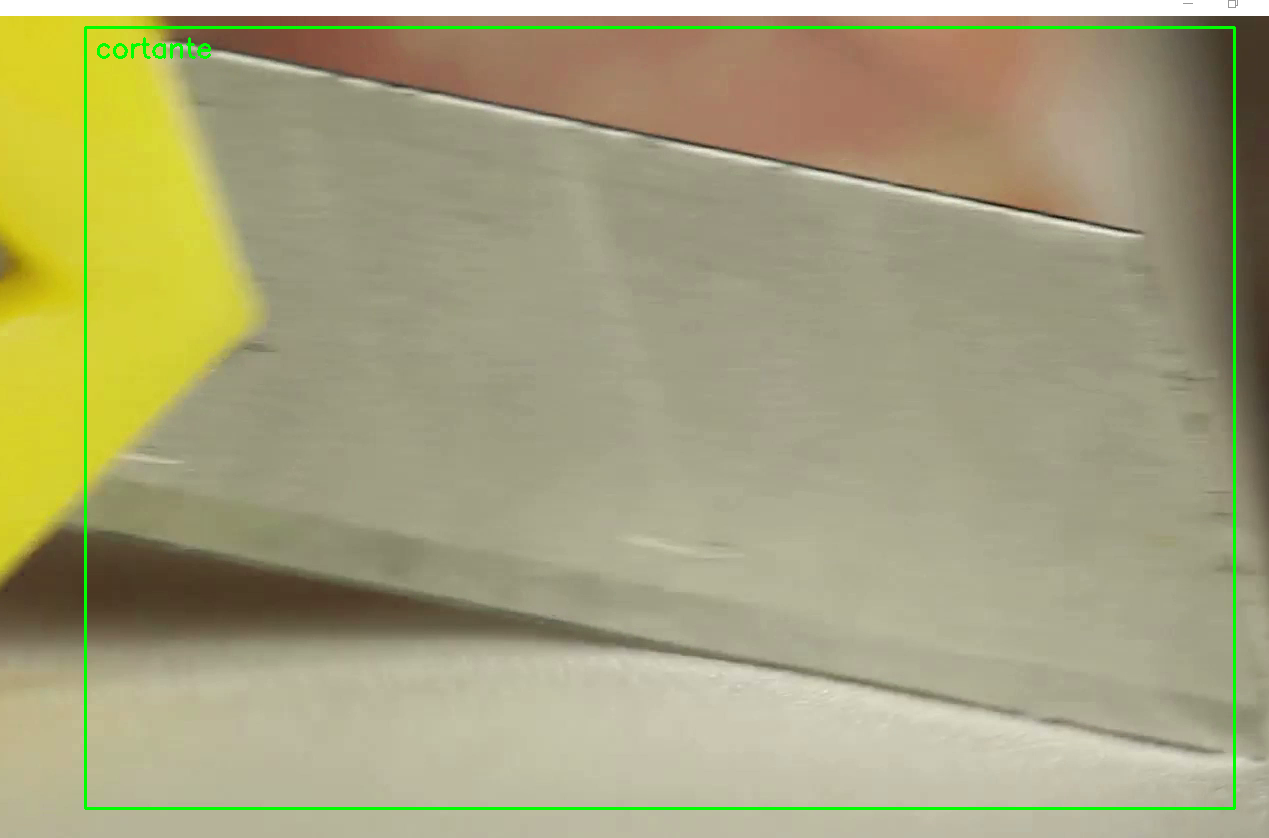
O conteúdo gerado por IA pode estar incorreto.

Padrão do plano de fundo

O conteúdo gerado por IA pode estar incorreto.

Mão segurando embalagem de plástico

O conteúdo gerado por IA pode estar incorreto.



Diagrama

O conteúdo gerado por IA pode estar incorreto.

**E-mail de alerta enviado**

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

**Log do Treinamento Realizado**

Treinamento com 100 épocas e mais de 1700 imagens que foi utilizado no projeto.  
Foram recortados os logs de processamento das épocas, deixei somente a primeira época.

PS F:\Projeto\python\postech\fase\_v\yolov8> yolo detect train data=data\_yolov8.yaml model=yolov8n.pt epochs=100 imgsz=640

New https://pypi.org/project/ultralytics/8.3.73 available 😃 Update with 'pip install -U ultralytics'

Ultralytics 8.3.67 🚀 Python-3.12.2 torch-2.5.1+cpu CPU (Intel Core(TM) i5-8400 2.80GHz)

engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=data\_yolov8.yaml, epochs=100, time=None, patience=100, batch=16, imgsz=640, save=True, save\_period=-1, cache=False, device=None, workers=8, project=None, name=train13, exist\_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single\_cls=False, rect=False, cos\_lr=False, close\_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multi\_scale=False, overlap\_mask=True, mask\_ratio=4, dropout=0.0, val=True, split=val, save\_json=False, save\_hybrid=False, conf=None, iou=0.7, max\_det=300, half=False, dnn=False, plots=True, source=None, vid\_stride=1, stream\_buffer=False, visualize=False, augment=False, agnostic\_nms=False, classes=None, retina\_masks=False, embed=None, show=False, save\_frames=False, save\_txt=False, save\_conf=False, save\_crop=False, show\_labels=True, show\_conf=True, show\_boxes=True, line\_width=None, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=True, opset=None, workspace=None, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight\_decay=0.0005, warmup\_epochs=3.0, warmup\_momentum=0.8, warmup\_bias\_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, nbs=64, hsv\_h=0.015, hsv\_s=0.7, hsv\_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy\_paste=0.0, copy\_paste\_mode=flip, auto\_augment=randaugment, erasing=0.4, crop\_fraction=1.0, cfg=None, tracker=botsort.yaml, save\_dir=runs\detect\train13

Overriding model.yaml nc=80 with nc=1

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]

1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]

2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]

3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]

4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]

5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]

6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]

7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]

8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]

9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]

10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]

12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]

13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]

15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]

16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]

17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]

20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]

21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]

22 [15, 18, 21] 1 751507 ultralytics.nn.modules.head.Detect [1, [64, 128, 256]]

Model summary: 225 layers, 3,011,043 parameters, 3,011,027 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

TensorBoard: Start with 'tensorboard --logdir runs\detect\train13', view at http://localhost:6006/

Freezing layer 'model.22.dfl.conv.weight'

train: Scanning F:\Projeto\python\postech\Fase\_V\imagens\train\labels.cache... 1714 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1714/1714 [00:00<?, ?it/s]

train: WARNING ⚠️ F:\Projeto\python\postech\Fase\_V\imagens\train\images\knife\_71.jpg: corrupt JPEG restored and saved

WARNING ⚠️ Box and segment counts should be equal, but got len(segments) = 449, len(boxes) = 1815. To resolve this only boxes will be used and all segments will be removed. To avoid

this please supply either a detect or segment dataset, not a detect-segment mixed dataset.

val: Scanning F:\Projeto\python\postech\Fase\_V\imagens\valid\labels.cache... 286 images, 0 backgrounds, 0 corrupt: 100%|██████████| 286/286 [00:00<?, ?it/s]

WARNING ⚠️ Box and segment counts should be equal, but got len(segments) = 108, len(boxes) = 301. To resolve this only boxes will be used and all segments will be removed. To avoid t

his please supply either a detect or segment dataset, not a detect-segment mixed dataset.

Plotting labels to runs\detect\train13\labels.jpg...

optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...

optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)

TensorBoard: model graph visualization added ✅

Image sizes 640 train, 640 val

Using 0 dataloader workers

Logging results to runs\detect\train13

Starting training for 100 epochs...

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

1/100 0G 1.467 2.483 1.567 5 640: 100%|██████████| 108/108 [11:35<00:00, 6.44s/it]

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 9/9 [00:34<00:00, 3.89s/it]

all 286 301 0.436 0.39 0.354 0.164

100 epochs completed in 19.238 hours.

Optimizer stripped from runs\detect\train13\weights\last.pt, 6.3MB

Optimizer stripped from runs\detect\train13\weights\best.pt, 6.3MB

Validating runs\detect\train13\weights\best.pt...

Ultralytics 8.3.67 🚀 Python-3.12.2 torch-2.5.1+cpu CPU (Intel Core(TM) i5-8400 2.80GHz)

Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 9/9 [00:25<00:00, 2.80s/it]

all 286 301 0.923 0.835 0.914 0.629

Speed: 2.3ms preprocess, 72.2ms inference, 0.0ms loss, 0.3ms postprocess per image

Results saved to runs\detect\train13