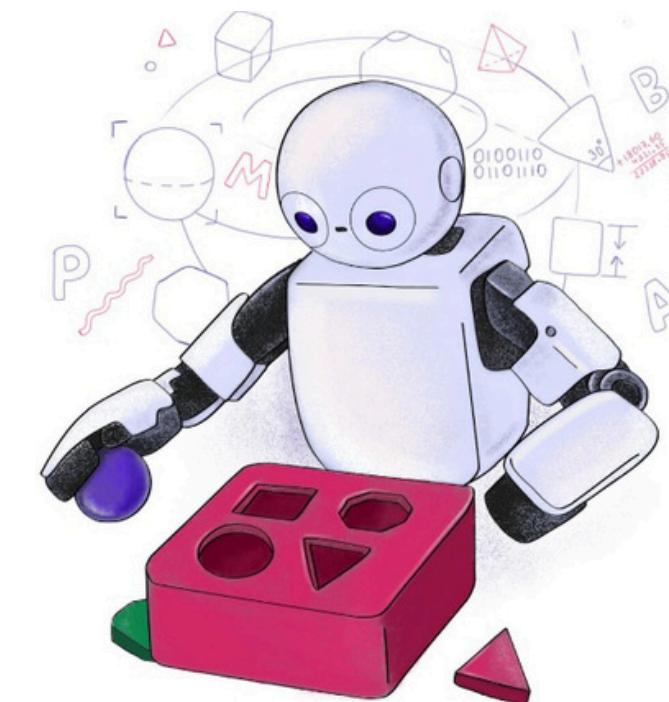


TP558 - Tópicos avançados em Machine Learning:

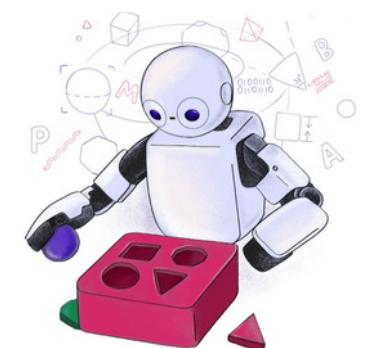
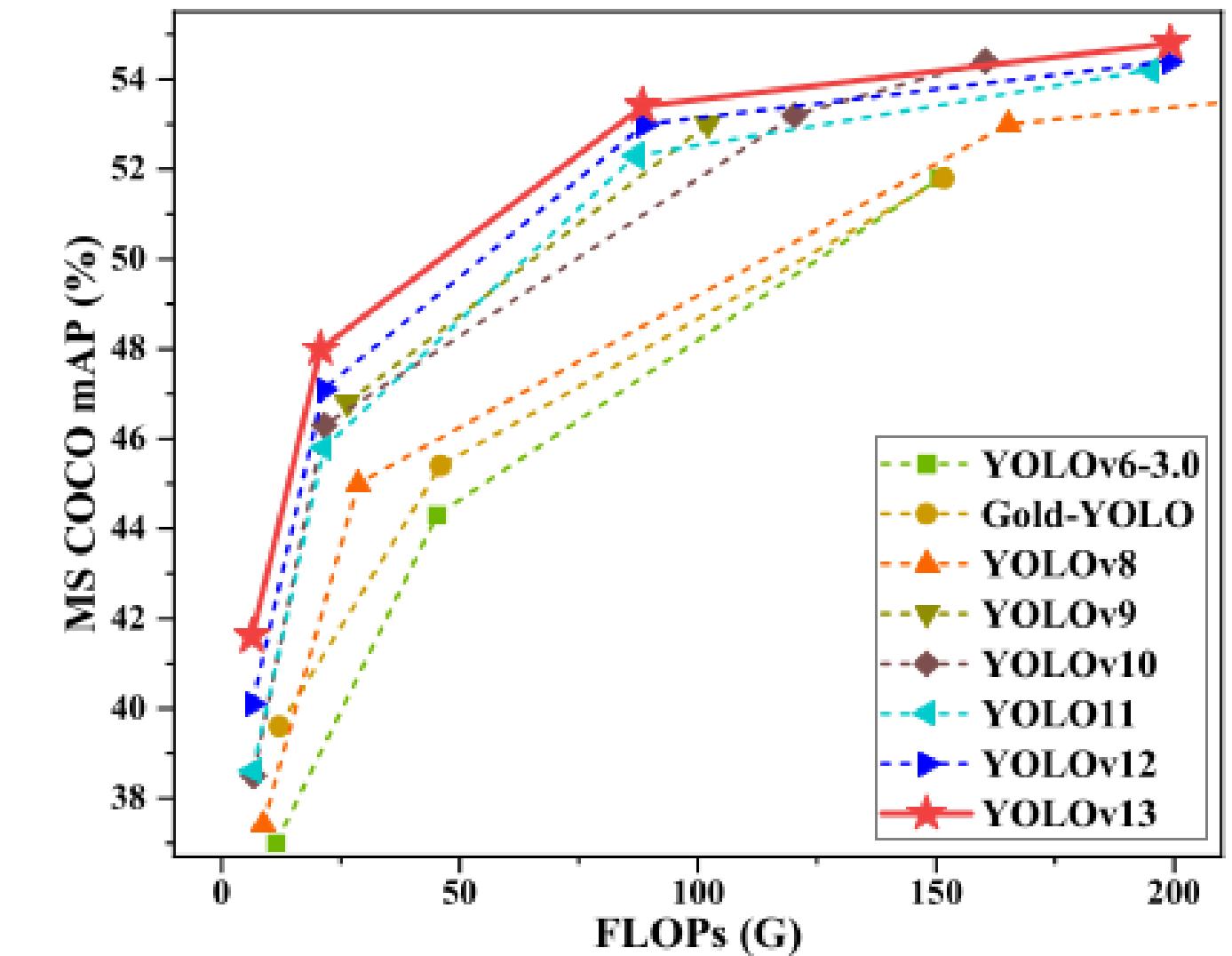
YOLOv13: Real-Time Object Detection with Hypergraph-Enhanced Adaptive Visual Perception



Introdução

Detecção de objetos em tempo real na pesquisa em visão computacional:

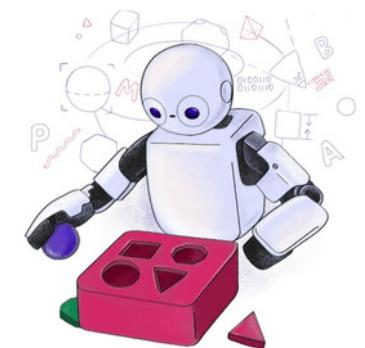
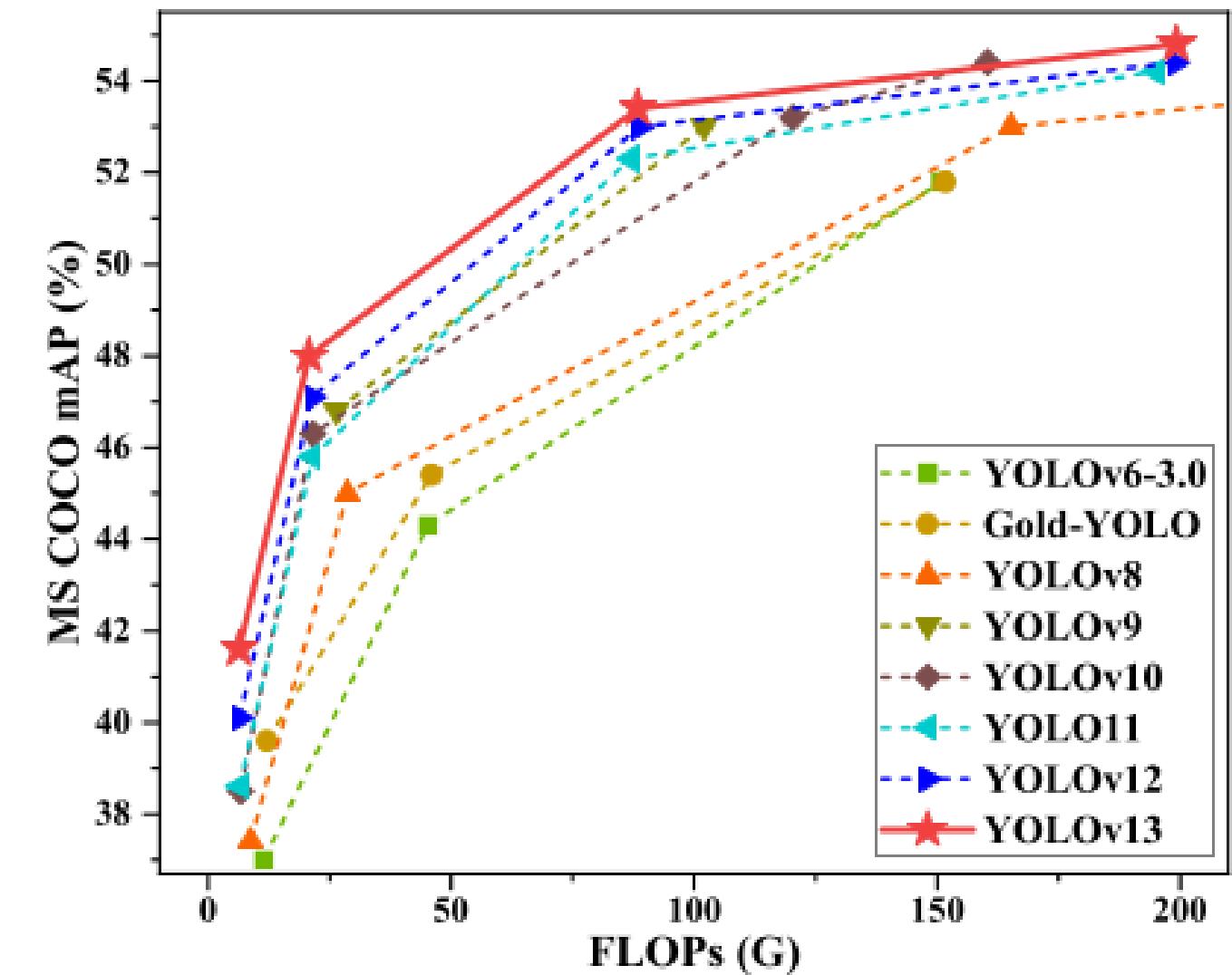
- Predominância de detectores single-stage baseados em CNN;
- Modelo YOLO tornou-se predominante devido ao excelente equilíbrio entre velocidade de inferência e precisão;
- Do YOLO ao YOLO11 as arquiteturas são centradas em convolução;
- YOLO12 faz a junção de convolução e mecanismo de autoatenção baseado em área;
- Do YOLO ao YOLO12 tem capacidade limitada de desempenho em cenários complexos.



Introdução

Proposta para YOLOv13:

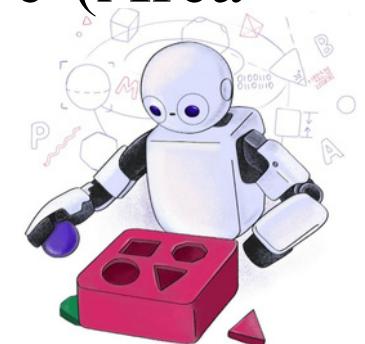
- Detector de objetos end-to-end (todo processo realizado por uma única arquitetura treinada de forma conjunta);
- Uso de hipergrafo adaptativo para explorar potenciais correlações de ordem superior;
- Mecanismo HyperACE com base em computação adaptativa de hipergrafos;
- FullPAD para melhorar o fluxo de informação e sinergia representacional;
- Série de blocos leves baseados em convoluções separáveis em profundidade para substituir blocos convolucionais padrão de Kernel de ordem maior.



Fundamentação teórica

Evolução dos detectores YOLO:

- O modelo original YOLO transformou a tarefa de detecção em um problema de regressão de uma só passada, eliminando a sobrecarga de geração de propostas, conseguindo um ótimo equilíbrio entre velocidade e precisão.
- YOLOv2 introduziu previsões baseadas em âncoras e o backbone DarkNet-19 (rede convolucionar com 19 camadas);
- YOLOv3 utilizou DarkNet-53 (53 camadas convolucionais) e previsões em três escalas para melhorar a detecção de pequenos objetos.
- YOLOv4 até YOLOv8 integraram módulos como CSP, SPP, PANet, suporte multimodo e cabeças sem âncora para melhorar o balanço entre desempenho e throughput.
- YOLOv9 e YOLOv10 focaram em backbone leves e implantação simplificada end-to-end.
- YOLOv11 manteve a modularidade de backbone-neck-head, substituindo o bloco C2f por C3k2 mais eficiente, e adicionou atenção espacial parcial para melhorar a detecção de objetos pequenos e ocluídos.
- YOLOv12 integrou completamente mecanismos de atenção, introduzindo a atenção de área leve (Area Attention) e atenção flash, para otimizar a modelagem semântica global e local.
- Há outras variantes do modelo como YOLOR, YOLO-NAS, Gold-YOLO etc.



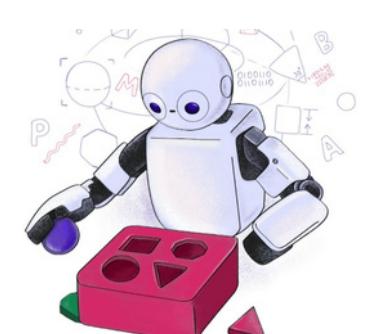
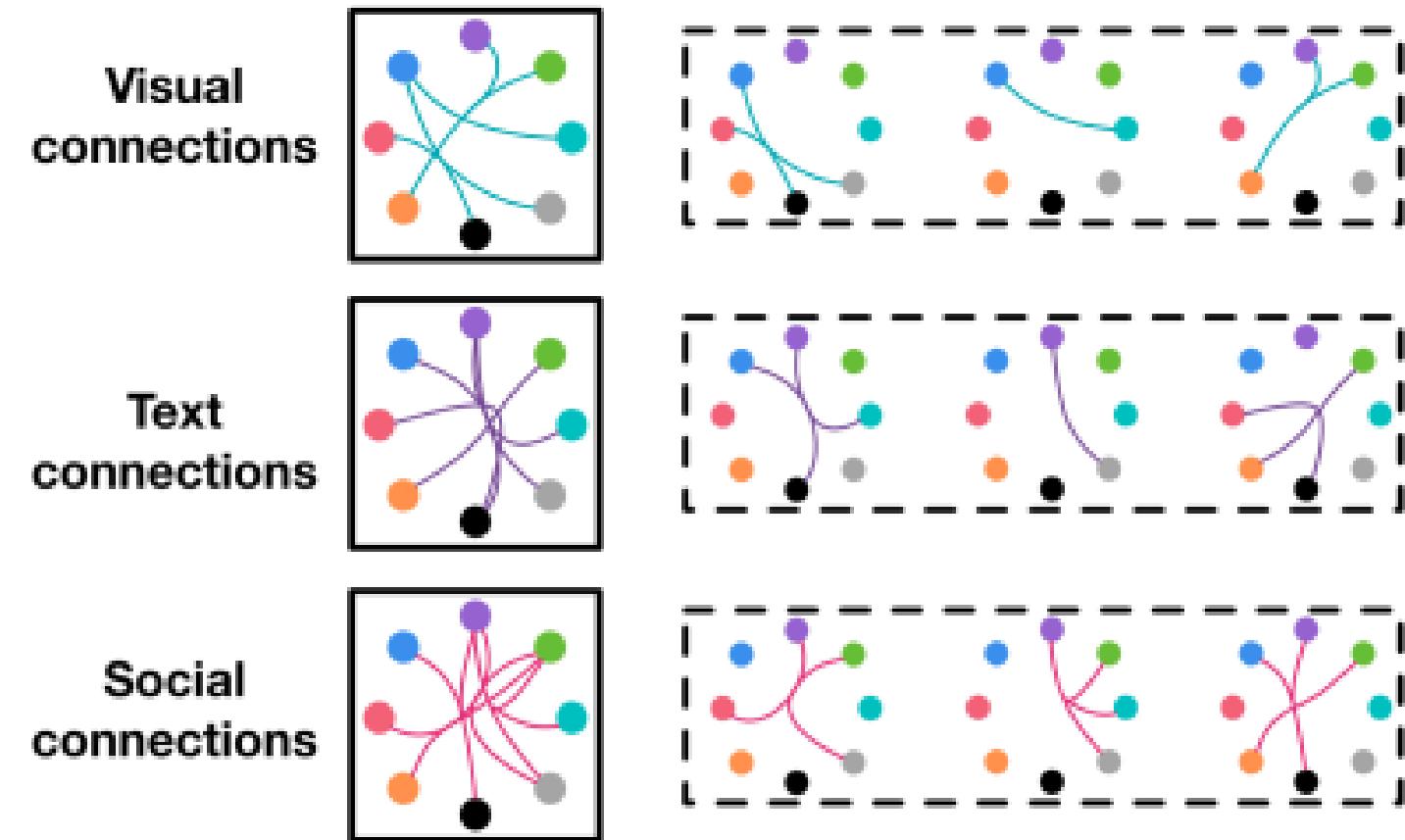
Fundamentação teórica

Entretanto, há uma limitação comum entre as arquiteturas atuais do YOLO, elas modelam apenas correlações locais de pares, impedindo a modelagem global de correlações de alto grau multi-to-multi, o que limita o desempenho em cenários complexos.

Conceito e aplicação da modelagem de correlações de alta ordem:

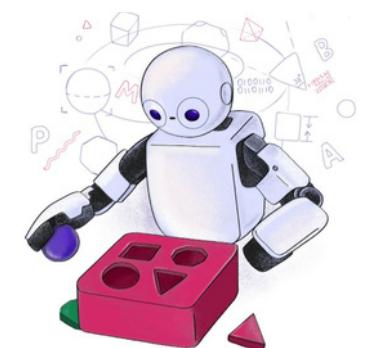
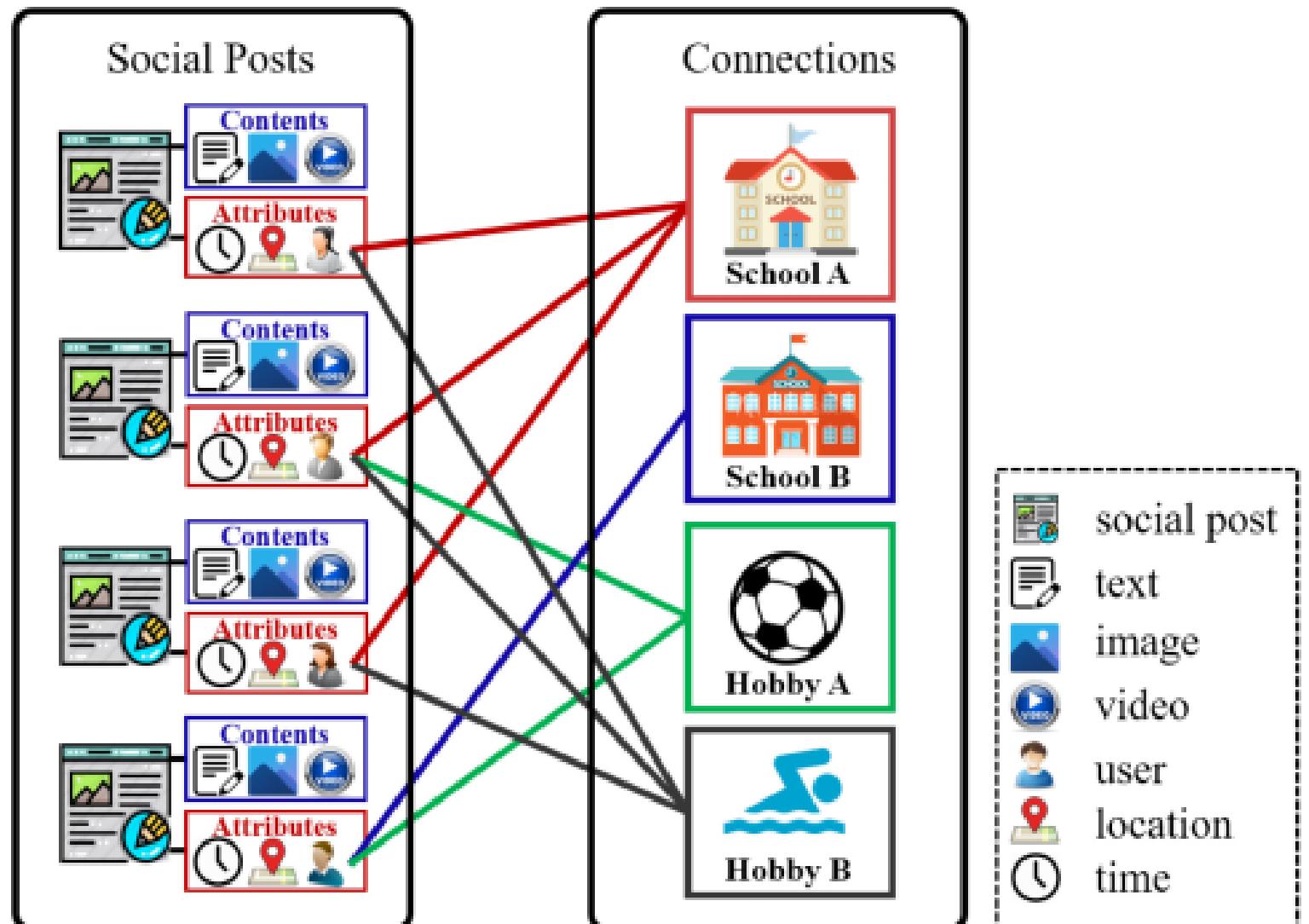
- Correlações mais complexas entre múltiplos elementos, comuns na natureza (conexões neurais, interações proteicas) e na ciência da informação (redes sociais);
- Em dados visuais, objetos diferentes envolvem interações espaciais, temporais e semânticas, que podem ser de baixa ordem (parwise) ou de alta ordem (multi-to-multi);
- Os hipergráficos são uma extensão dos grafos tradicionais que podem representar não apenas correlações par a par, mas também relações múltiplas, conectando múltiplos vértices por hiperares.

Tweets/Microblogs



Fundamentação teórica

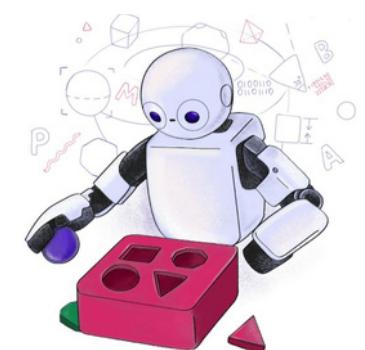
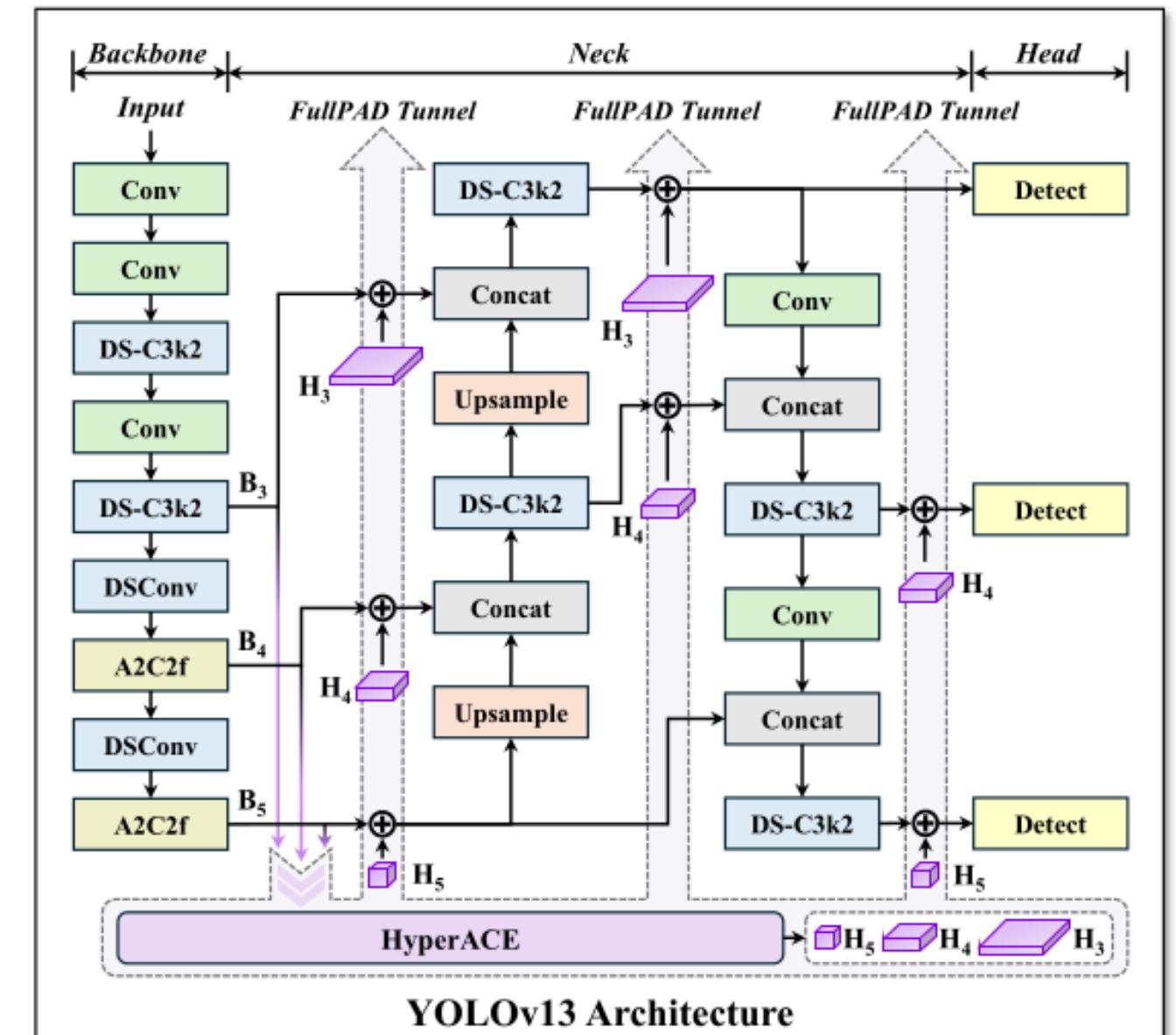
- As redes neurais de hipergráficos (HGNNs) surgiram para modelar essas correlações de alta ordem.
- Trabalhos integrando o HGNN em modelos de detecção (como o primeiro estudo Hyper-YOLO, Feng et al), indicam a necessidade dessa modelagem para melhorar a precisão. No entanto, os métodos existentes utilizam limiares fixos manuais para determinar se pixels estão correlacionados baseados na distância dos recursos, o que compromete a robustez e a precisão da modelagem.
- Para superar essas limitações, o artigo propõe o mecanismo Hypergraph-based Adaptive Correlation Enhancement (HyperACE), que modela de forma adaptativa as correlações latentes globais de alta ordem, explorando interações cruzadas entre locais e escalas, superando a falta de robustez dos métodos com parâmetros manuais.



Arquitetura e funcionamento

Arquitetura Geral:

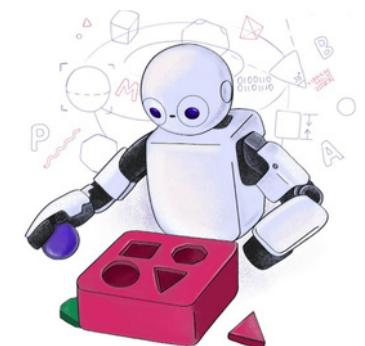
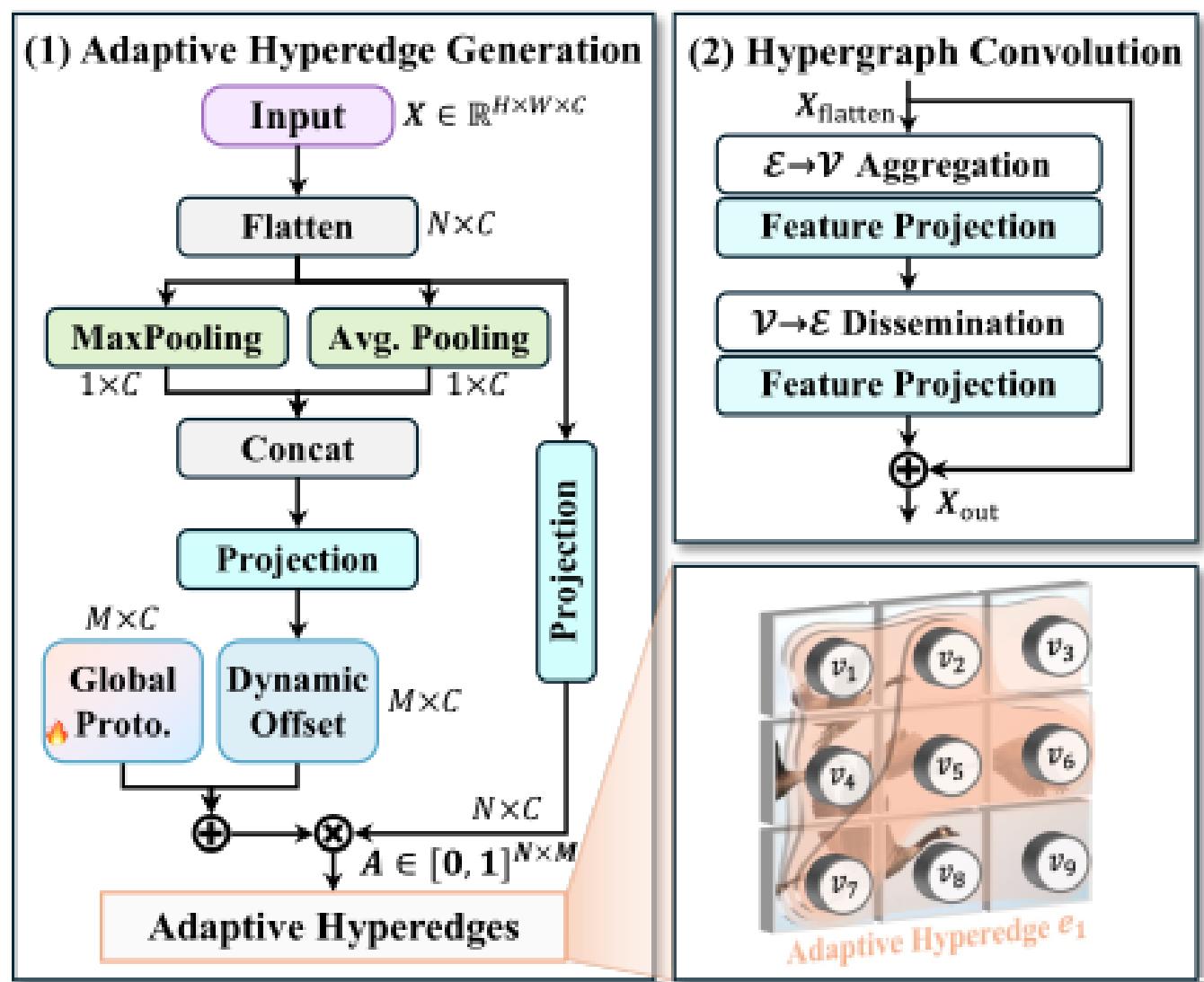
- O YOLOv13 mantém a estrutura tradicional "Backbone → Neck → Head" presente no YOLO.
- **Backbone**: Extrai mapas de características em múltiplas escalas (B1 a B5) de uma imagem.
- A grande inovação: ao invés de enviar diretamente B3, B4 e B5 ao Neck, estas características vão para o módulo HyperACE, que realiza modelagem adaptativa de correlações de alta ordem entre essas características multi-escala.
- O FullPAD é responsável por distribuir as características enriquecidas pelo HyperACE para diversos pontos de ligação entre backbone, neck e head, facilitando uma sinergia melhor na representação e fluxo de dados, e também melhorando a propagação do gradiente no treinamento.
- O backbone substitui convoluções grandes (**large-kernel convolutions**) tradicionais por blocos leves chamados DS-C3k2, que usam convoluções separáveis em profundidade para reduzir o custo computacional



Arquitetura e funcionamento

HyperACE:

- Possui dois ramos, a **Percepção global de alta ordem** (usando blocos C3AH que incorporam hipergrafo adaptativo, para captar correlações complexas entre múltiplos pixels/objetos) e **Percepção local de baixa ordem** (usando blocos leves DS-C3k para captar informações locais detalhadas).
- Tradicionalmente, hipergrafos são usados com parâmetros manuais para conectar vértices com base na similaridade. O YOLOv13 propõe um hipergrafo adaptativo que aprende automaticamente o grau de participação de cada vértice (pixel ou região) em cada hiperaresta.
- A geração das hiperarestas é feita em duas etapas, extrai vetores contextuais globais das características usando pooling médio e máximo e concatena, usa uma camada de mapeamento para gerar offsets dinâmicos que, somados a protótipos globais aprendidos, geram protótipos de hiperarestas dinâmicos.
- Cada vértice gera um vetor de consulta. A similaridade entre cada vetor de consulta e cada protótipo define o quanto aquele vértice participa da hiperaresta, calculada via uma média em múltiplos "heads" (subespaços) para diversidade.



Arquitetura e funcionamento

Estrutura do HyperACE:

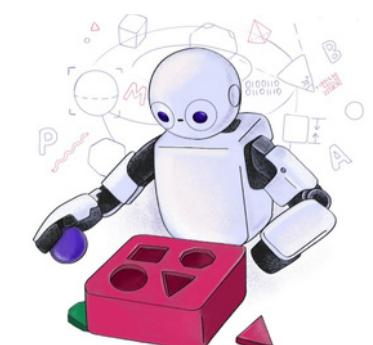
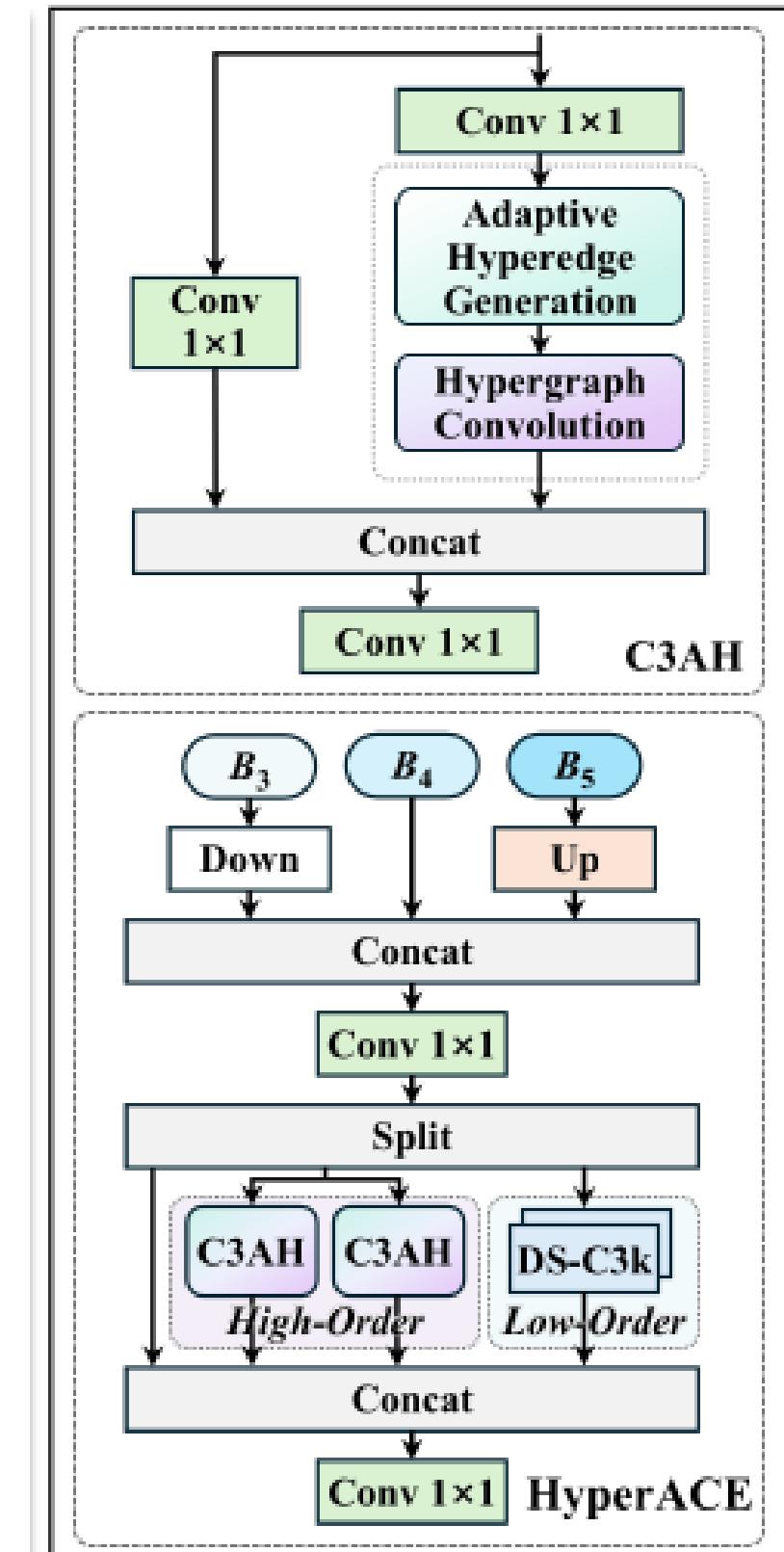
- Recebe os mapas de características B_3 , B_4 e B_5 .
- Redimensiona B_3 e B_5 para o tamanho espacial de B_4 e os combina via convolução para gerar o mapa X_b .
- Divide X_b no canal em três grupos diferentes: X_h para modelagem global de alta ordem (via múltiplos blocos C3AH paralelos), X_l para modelagem local de baixa ordem (via blocos DS-C3k) e X_s atalho (linha direta de dados originais).
- As saídas de cada ramo são então concatenadas, passadas por uma convolução final, formando a saída aprimorada Y .

$$X_h = \text{Concat}(X_h^{(1)}, X_h^{(2)}, \dots, X_h^{(K)}).$$

$$X_l = \text{DS-C3k}(X_b^l).$$

$$X_s = X_b^s.$$

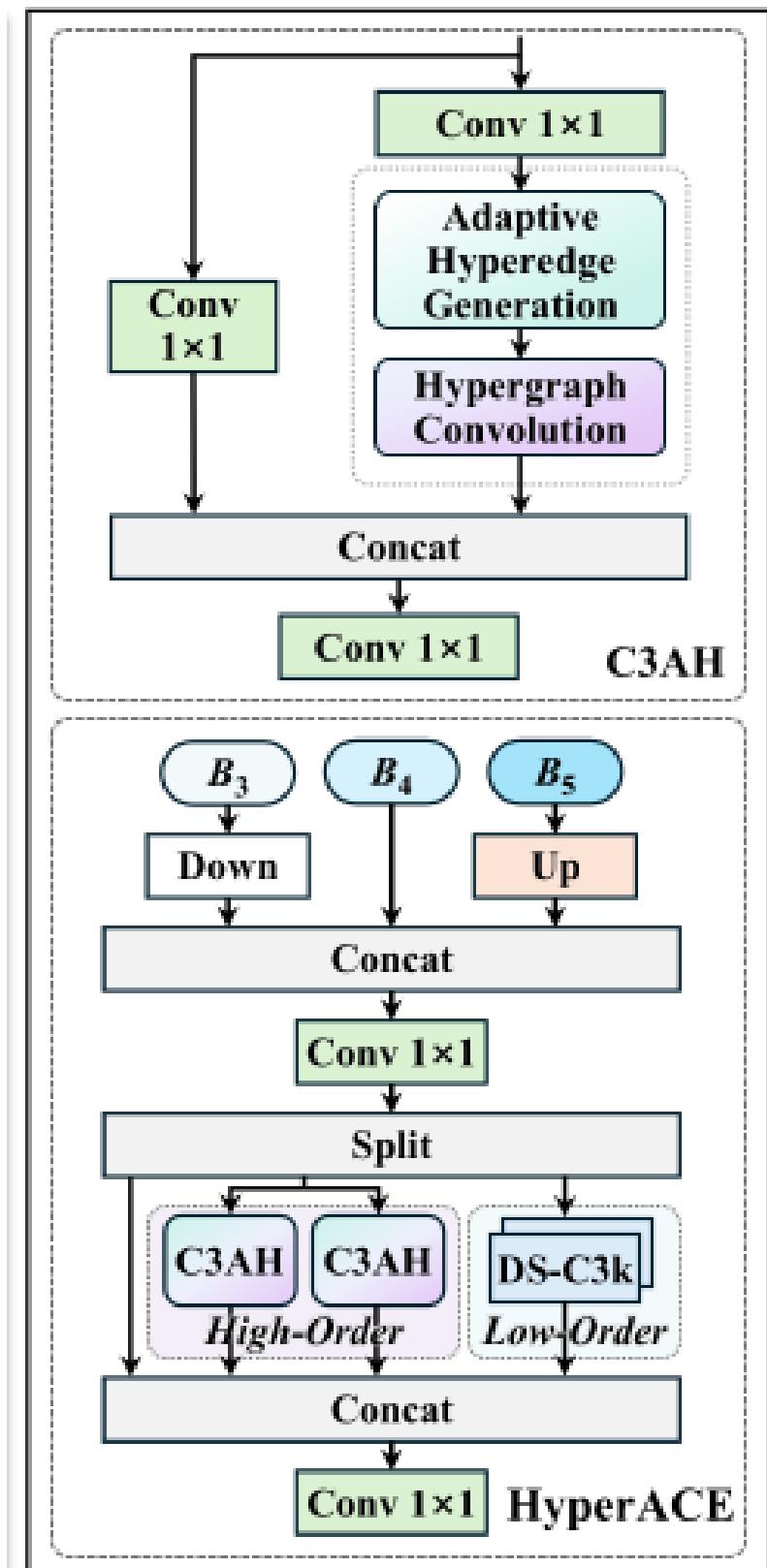
$$Y = \text{Conv}_{1 \times 1}(\text{Concat}(X_h, X_l, X_s)).$$



Arquitetura e funcionamento

C3AH:

- É um bloco que integra o hipergrafo adaptativo ao padrão CSP (Cross Stage Partial) da arquitetura YOLO.
- Recebe o mapa de características, projeta para menor dimensão e divide em dois ramos, um para o processamento do hipergrafo (computação adaptativa e convolução) outro para conexão lateral (mantém informações originais).
- Depois, concatena as saídas e passa por uma convolução saída final do bloco.

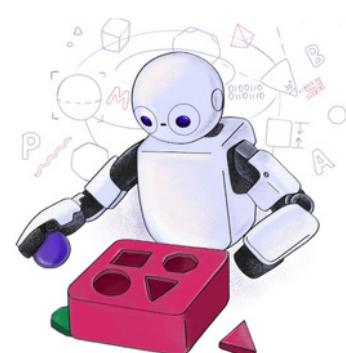


DSConv:

- Aplica uma convolução separada por canal (depthwise)
- Seguido por uma convolução ponto-a-ponto 1 x 1 para combinar canais
- Normalização (batch norm) e ativação (SiLU) são usadas em seguida

DS-Bottleneck:

- Composto por dois DSConv em cascata, o primeiro com kernel 3x3 e o segundo com kernel grande k x k.
- Usa conexão residual quando canais não mudam para preservar informação.



Arquitetura e funcionamento

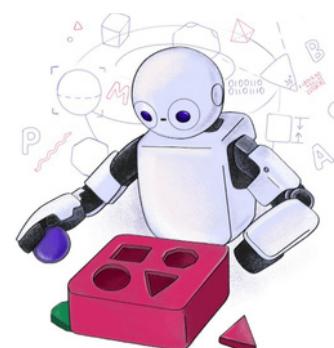
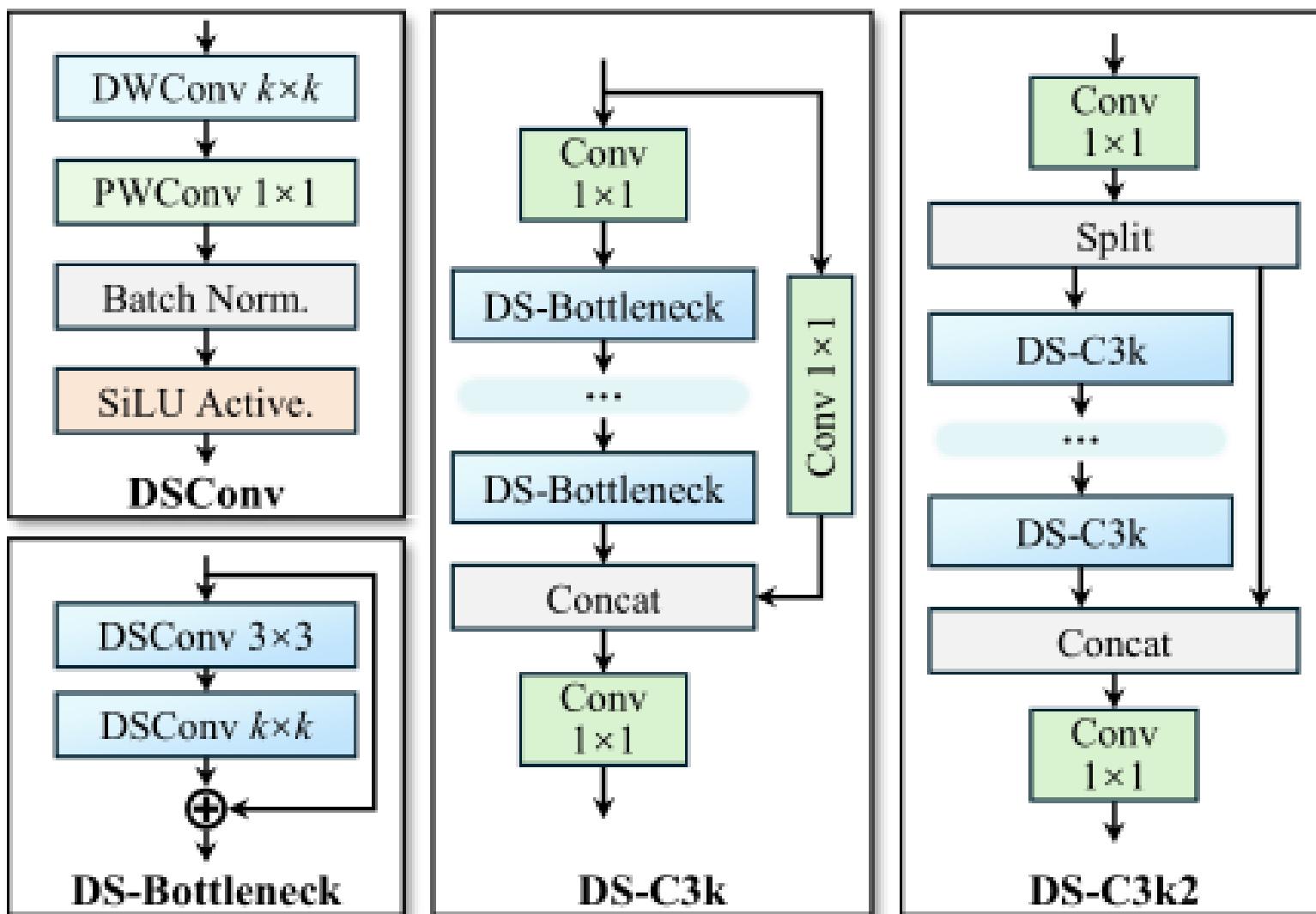
DS-C3k:

- Derivado da estrutura CSP-C3 tradicional, usa blocos DS-Bottleneck para extrair características leves.
- Divide o fluxo em ramificação lateral e em cascata, e depois concatena para manter informação cruzada de canais.

DS-C3k2:

- Baseado em C3k2, aplica um convolução para unificar canais, divide em dois grupos, um passa por múltiplos DS-C3k e o outro é atalho direto.
- Concatenam os dois e uma convolução final une os canais.

O uso extensivo desses blocos no backbone, neck e HyperACE reduz até 30% de parâmetros e 28% de FLOPs para todas as variantes do YOLOv13, mantendo a performance.

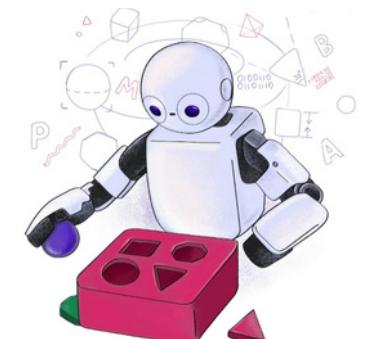
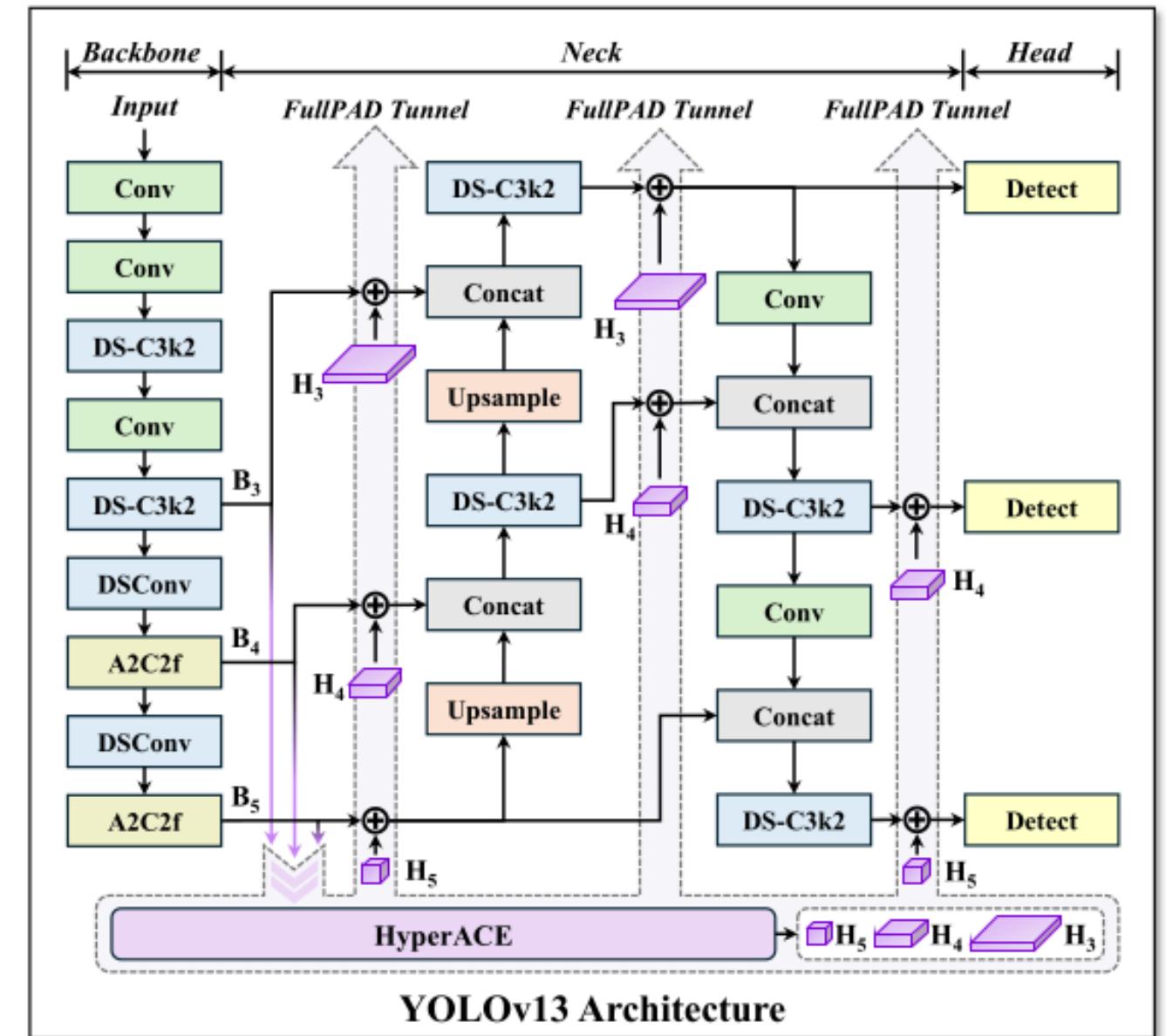


Arquitetura e funcionamento

Inatel

FullPAD:

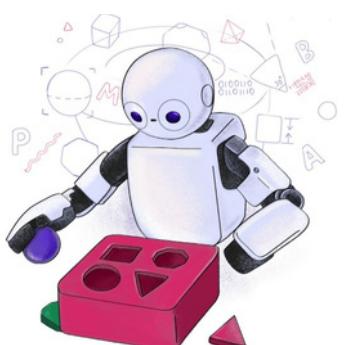
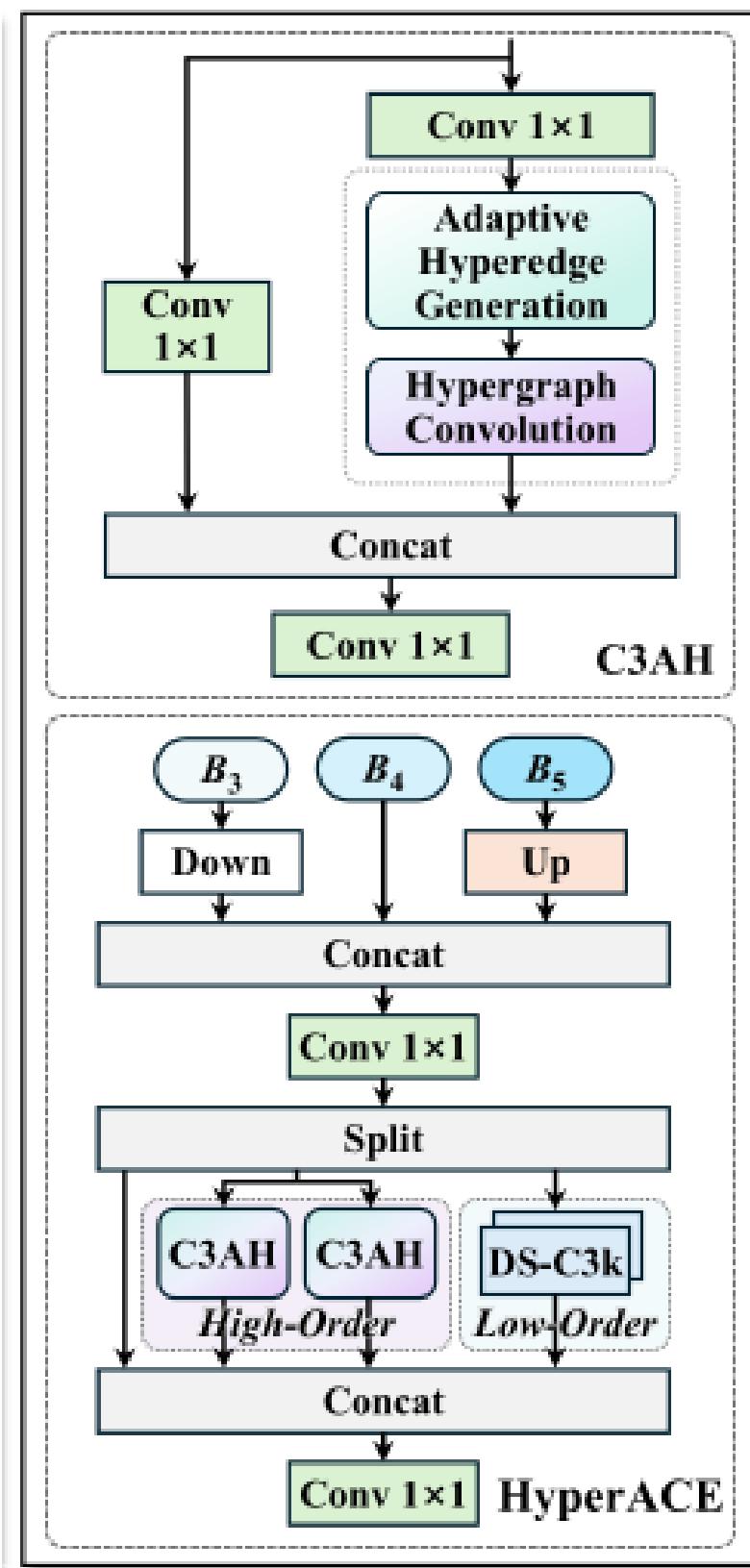
- O objetivo é utilizar ao máximo as características aprimoradas Y geradas pelo HyperACE.
- Redimensiona Y para os tamanhos originais de B3, B4 e B5 usando interpolação e aplica uma convolução 1 x 1 para ajustar canais.
- As características aprimoradas são distribuídas entre backbone e neck, dentro do neck e entre o neck e o head.
- A fusão é feita usando uma soma ponderada com um parâmetro treinável, que equilibra a importância da característica original e da aprimorada.
- Essa distribuição ocorre via três “túneis” separados, garantindo fluxo fino de informações e sinergia de representação em todo o pipeline.
- A prática melhora a propagação do gradiente, facilitando o aprendizado e aumentando o desempenho da detecção.



Arquitetura e funcionamento

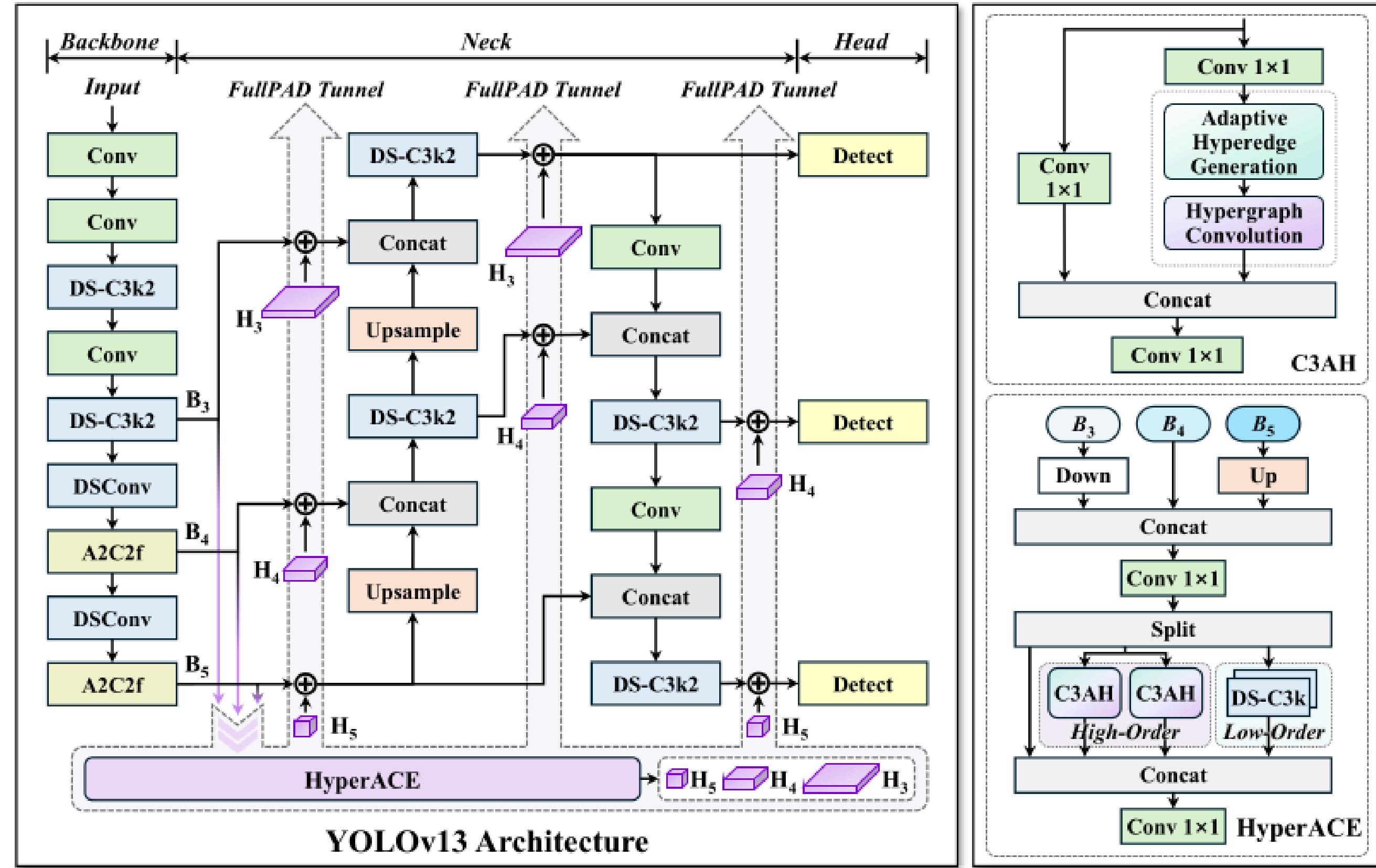
FullPAD:

- O objetivo é utilizar ao máximo as características aprimoradas \tilde{Y} geradas pelo HyperACE;
 - Redimensiona \tilde{Y} para os tamanhos originais de B3, B4 e B5 usando interpolação e aplica uma convolução 1 x 1 para ajustar canais.
 - As características aprimoradas são distribuídas entre backbone e neck, dentro do neck e entre o neck e o head.
 - A fusão é feita usando uma soma ponderada com um parâmetro treinável, que equilibra a importância da característica original e da aprimorada.
 - Essa distribuição ocorre via três “túneis” separados, garantindo fluxo fino de informações e
 - sinergia de representação em todo o pipeline.
 - A prática melhora a propagação do gradiente, facilitando o aprendizado e aumentando o desempenho da detecção.



Arquitetura e funcionamento

Inatel

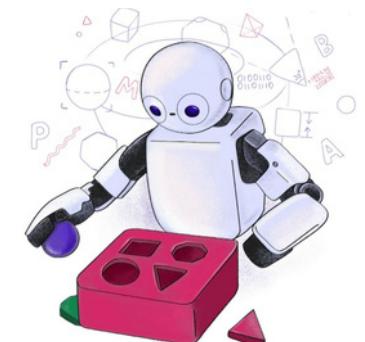
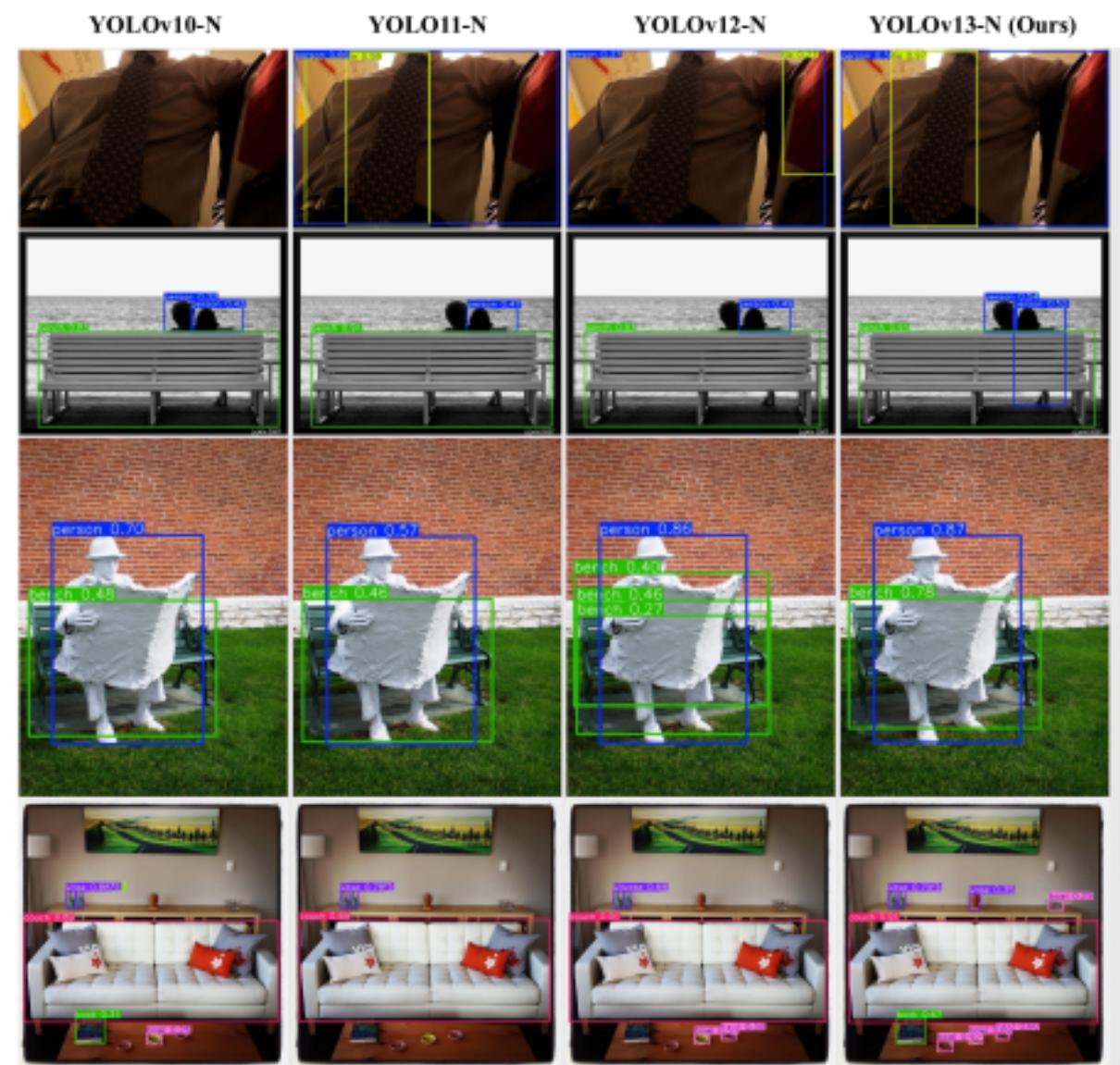


Treinamento e otimização

O modelo YOLOv13 foi treinado com os seguintes hiperparâmetros principais:

- Número de épocas: 600 épocas.
- Tamanho do batch (batch size): 256.
- Taxa inicial de aprendizado (initial learning rate): 0.01.
- Otimizador: Stochastic Gradient Descent (SGD).
- Scheduler de decaimento do learning rate: decaimento linear.
- Warm-up: linear warm-up aplicado nas primeiras 3 épocas.
- Tamanho da imagem de entrada: 640×640 pixels.
- Técnicas de aumento de dados usadas: Mosaic e Mixup.
- Quantidade de GPUs usadas para treinamento: 4 e 8 GPUs RTX 4090 para os modelos Nano e Small, respectivamente, e 4 e 8 GPUs A800 para os modelos Large e Extra-Large.
- Para avaliação de latência, usaram uma GPU Tesla T4 com TensorRT FP16.

Além disso, para garantir uma comparação justa, os modelos YOLOv11 e YOLOv12 foram reproduzidos com as mesmas configurações e no mesmo hardware que o YOLOv13. Os modelos YOLOv13 vêm em quatro variantes, com o número de hiperarestas (hyperedges) definidas para cada variante: 4 para Nano, 8 para Small e Large, e 12 para Extra-Large.



Treinamento e otimização

Inatel

Avaliação da importância do FullPAD e do HyperACE:

- Ao retirar a distribuição de recursos do FullPAD (equivalente a remover o HyperACE), o desempenho do modelo YOLOv13-Small caiu 0,9% no AP val 50:95 e 1,1% no AP val 50.
- Quando o FullPAD distribuía recursos parcialmente para backbone-neck, in-neck ou neck-head, houve perdas menores no desempenho (entre 0,2% e 0,4%), mostrando que o esquema de distribuição completa é necessário para o melhor desempenho.

Análise do número de hiperarestas (hyperedges) M:

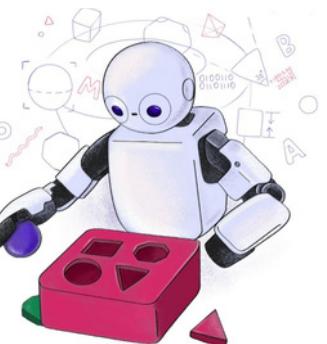
- Variações no número de hiperarestas usadas no HyperACE mostraram que poucas hiperarestas reduzem os parâmetros e custos computacionais, mas prejudicam a performance (menor capacidade de modelar as correlações da cena).
- Com mais hiperarestas (até 16), o desempenho melhora ligeiramente, porém com aumento de parâmetros e custo computacional.

Impacto dos blocos DS (depthwise separable convolutions):

- As substituições dos blocos convolucionais convencionais pelos blocos DS no HyperACE levaram a redução significativa do número de parâmetros (até 30%) e dos FLOPs (até 28%) sem perda relevante da acurácia.
- Isso mostra que o HyperACE suporta eficientemente o uso desses blocos leves mantendo o desempenho.

Visualização das hiperarestas adaptativas:

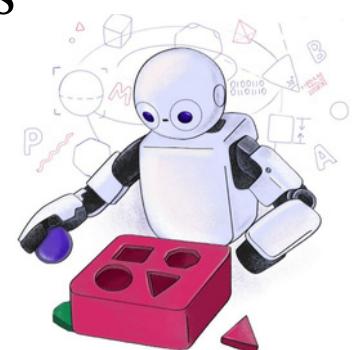
- Através da visualização das hiperarestas adaptativas geradas pelo HyperACE, foi possível observar que o mecanismo efetivamente modela correlações de ordem alta entre múltiplos objetos e cenas, indo além das correlações binárias típicas de mecanismos anteriores.



Treinamento e otimização

Inatel

- Na avaliação no conjunto de dados MS COCO, todas as variantes do YOLOv13 atingem desempenho estado-da-arte enquanto permanecem leves.
- O YOLOv13-Nano (Nano) supera o YOLOv12-Nano em 1,5% no AP_val_50:95 e 1,8% no AP_val_50, e supera o YOLOv11-Nano em 3,0% no AP_val_50:95.
- O YOLOv13-Small (Small) também tem ganhos significativos: 0,9% e 1,0% em AP_val_50:95 e AP_val_50 sobre o YOLOv12-S, respectivamente.
- O YOLOv13-S melhora em AP_val_50:95 em relação a modelos DETR baseados em ViT (exemplo RT-DETRv2-R18) enquanto utiliza menos parâmetros (redução de 55%) e FLOPs (redução de 65,3%).
- Em modelos maiores (Large e Extra-Large), o YOLOv13 mantém vantagem em mAP e eficiência em comparação com YOLOv12 e outros, com menor consumo de FLOPs e parâmetros.
- Qualitativamente, o YOLOv13 detecta melhor objetos pequenos e em cenas complexas, onde modelos anteriores falham, graças à modelagem adaptativa de correlações de alta ordem pelo HyperACE.
- Na generalização cruzada, o YOLOv13 treinado no MS COCO tem melhor desempenho que o YOLOv12 e outros ao ser testado diretamente no Pascal VOC 2007, o que indica robustez e capacidade de generalização superiores.
- Latência nos diferentes hardwares mostra que o YOLOv13 é rápido, com tempos de inferência competitivos mesmo em GPUs mais modestas.



Exemplos de aplicação

Toda documentação e códigos proposto para o modelo podem ser encontradas no endereço:
<https://github.com/iMoonLab/yolov13>

```
✓ 22s [1] !git clone https://github.com/iMoonLab/yolov13.git
%cd yolov13
!pip install -U pip
!pip install -r requirements.txt
!pip install -e .

>Show output
Mostrar saída oculta

✓ 0s [2] !mkdir -p weights
!wget -O weights/yolov13n.pt https://github.com/iMoonLab/yolov13/releases/download/yolov13/yolov13n.pt

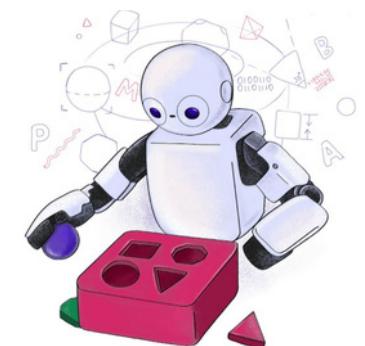
>Show output
Mostrar saída oculta

✓ 14s [3] !unzip /content/barcos.zip
>Show output
Mostrar saída oculta

✓ 4s [4] !pip install ultralytics
>Show output

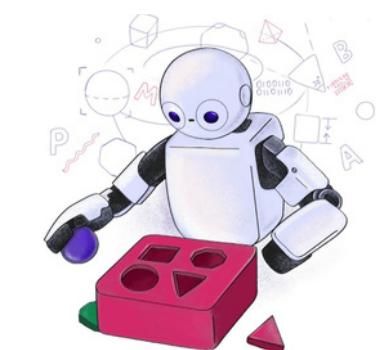
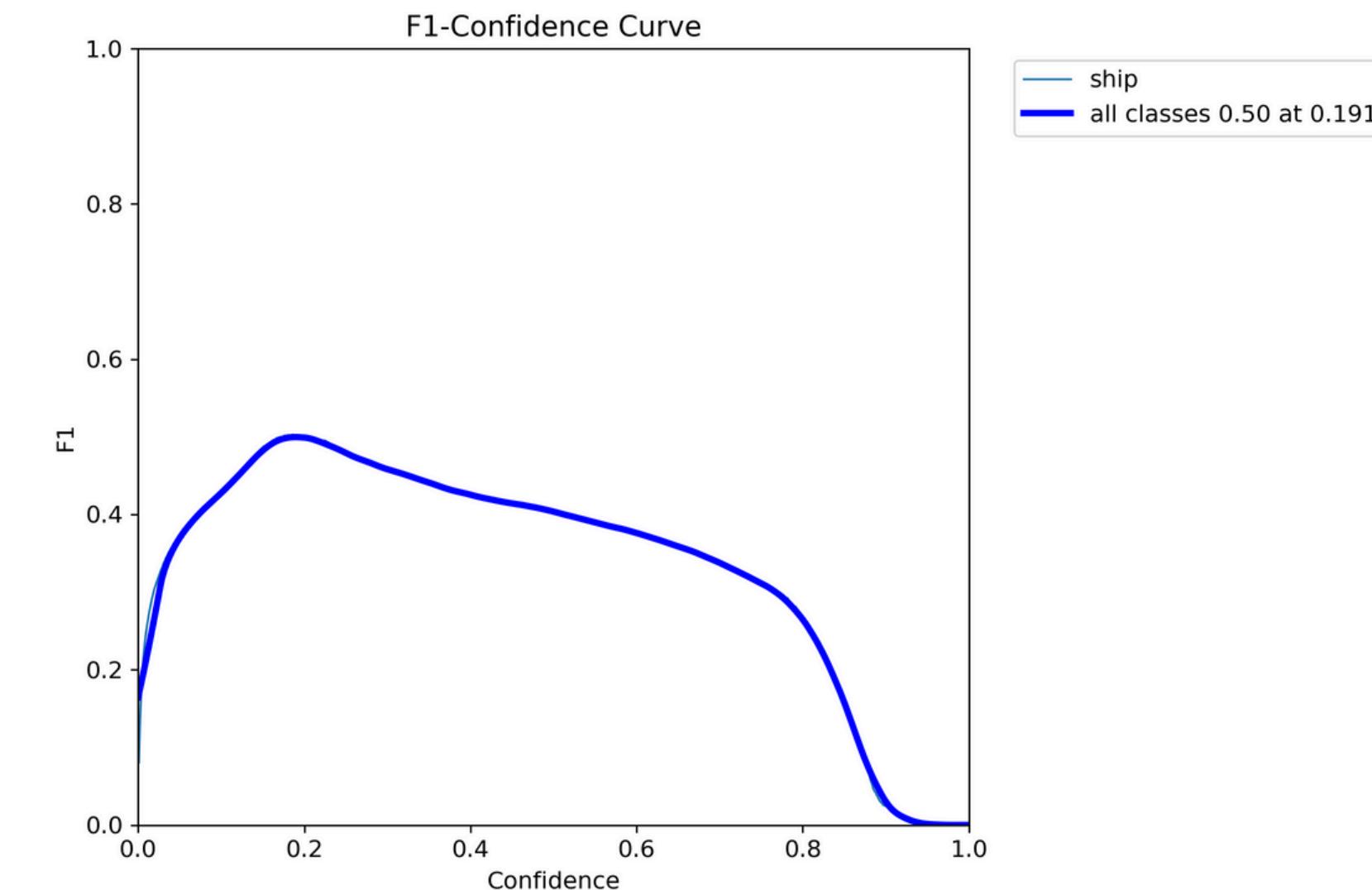
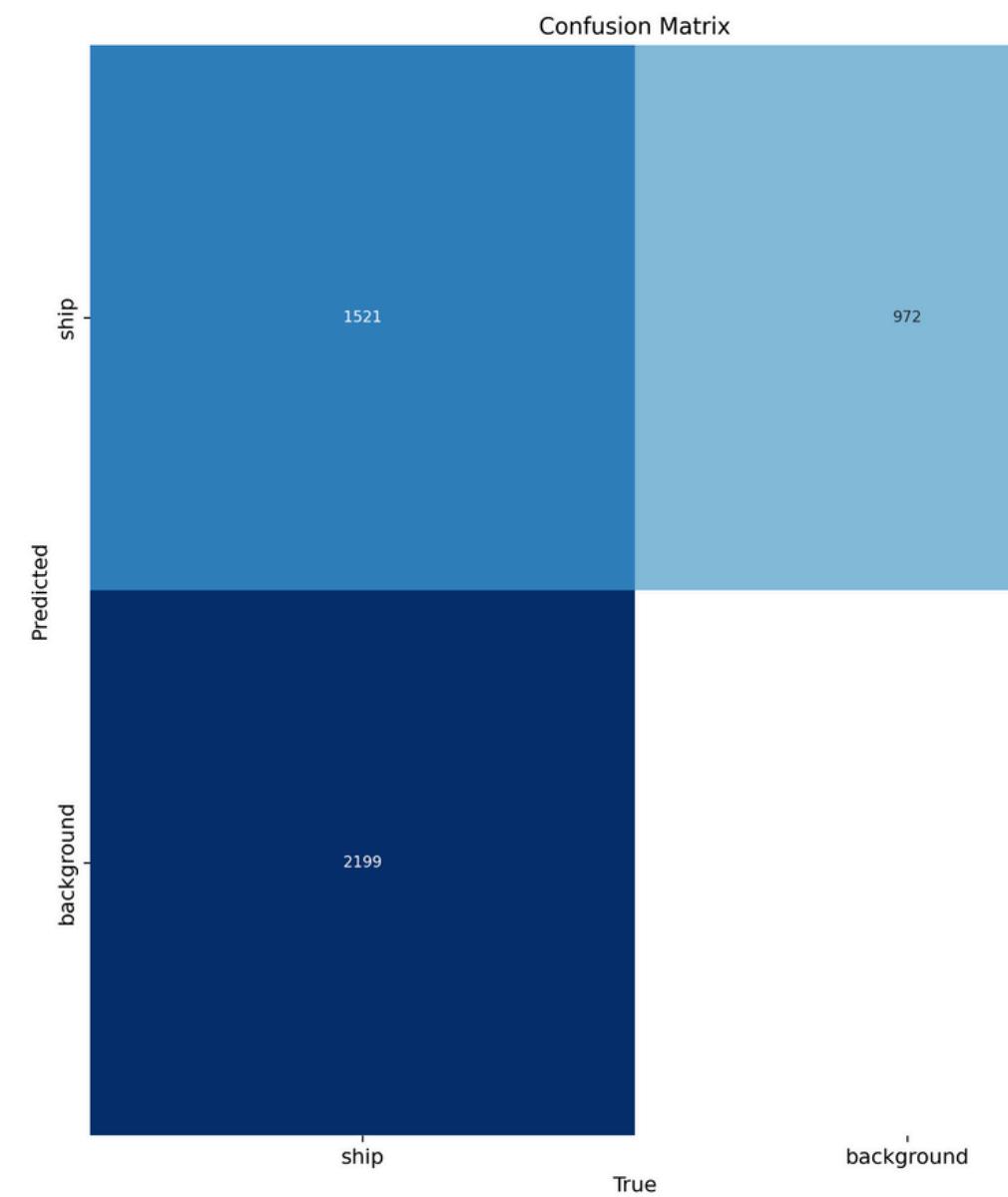
from ultralytics import YOLO

model = YOLO("yolov13n.yaml") # ou troque para s/l/x
results = model.train(
    data="/content/yolov13/ships-aerial-images/data.yaml",
    epochs=100,      # o README usa 600 para COCO; para começar, 100-300 é ok
    batch=32,        # ajuste conforme sua GPU
    imgsz=640,
    scale=0.5,       # S:0.9; L/X:0.9 (do README)
    mosaic=1.0,
    mixup=0.0,        # S:0.05; L:0.15; X:0.2 (do README)
    copy_paste=0.1,   # S:0.15; L:0.5; X:0.6 (do README)
    device=0          # no Colab geralmente "0"
)
```



Exemplos de aplicação

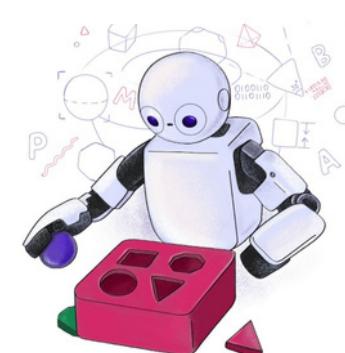
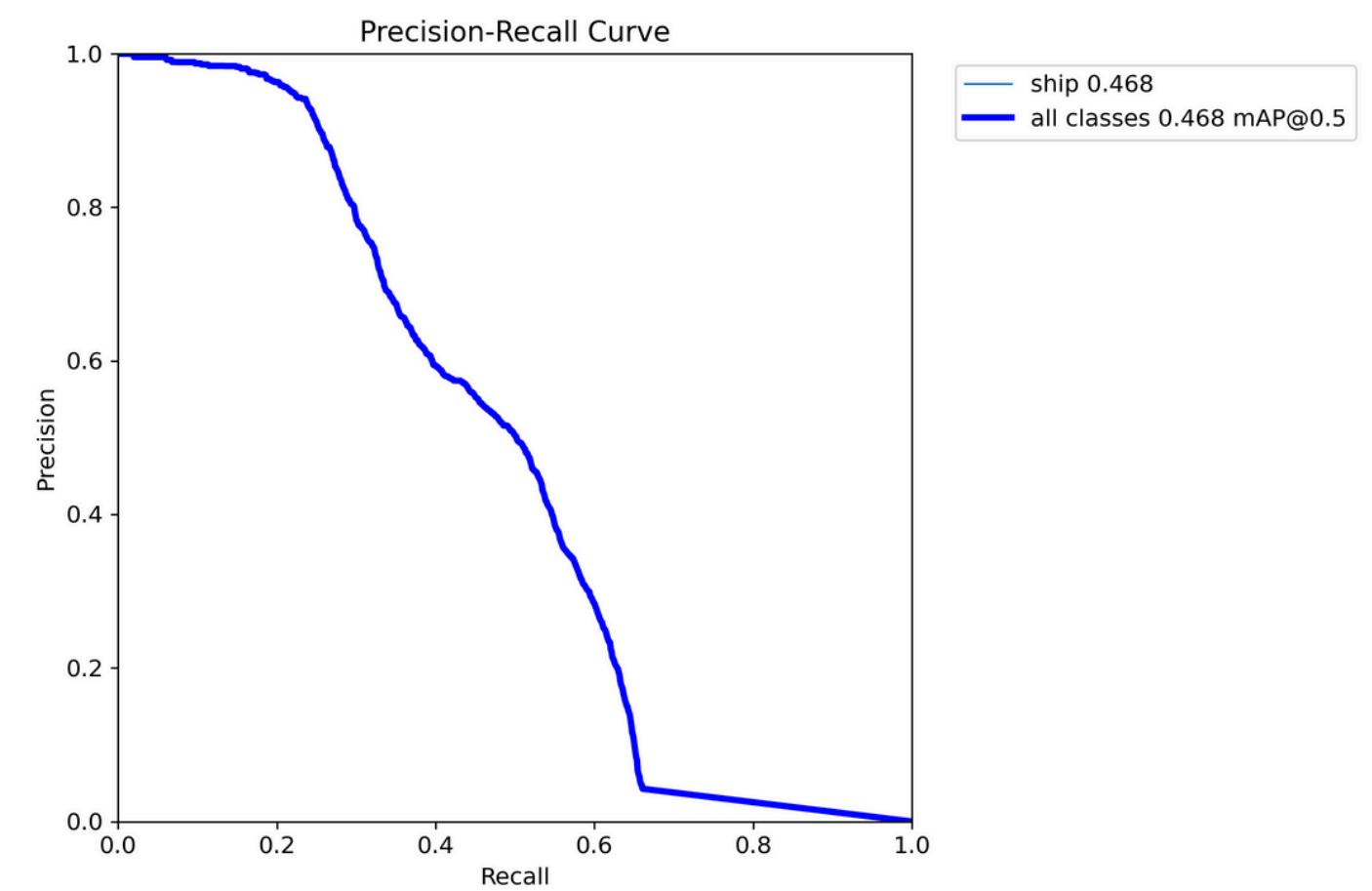
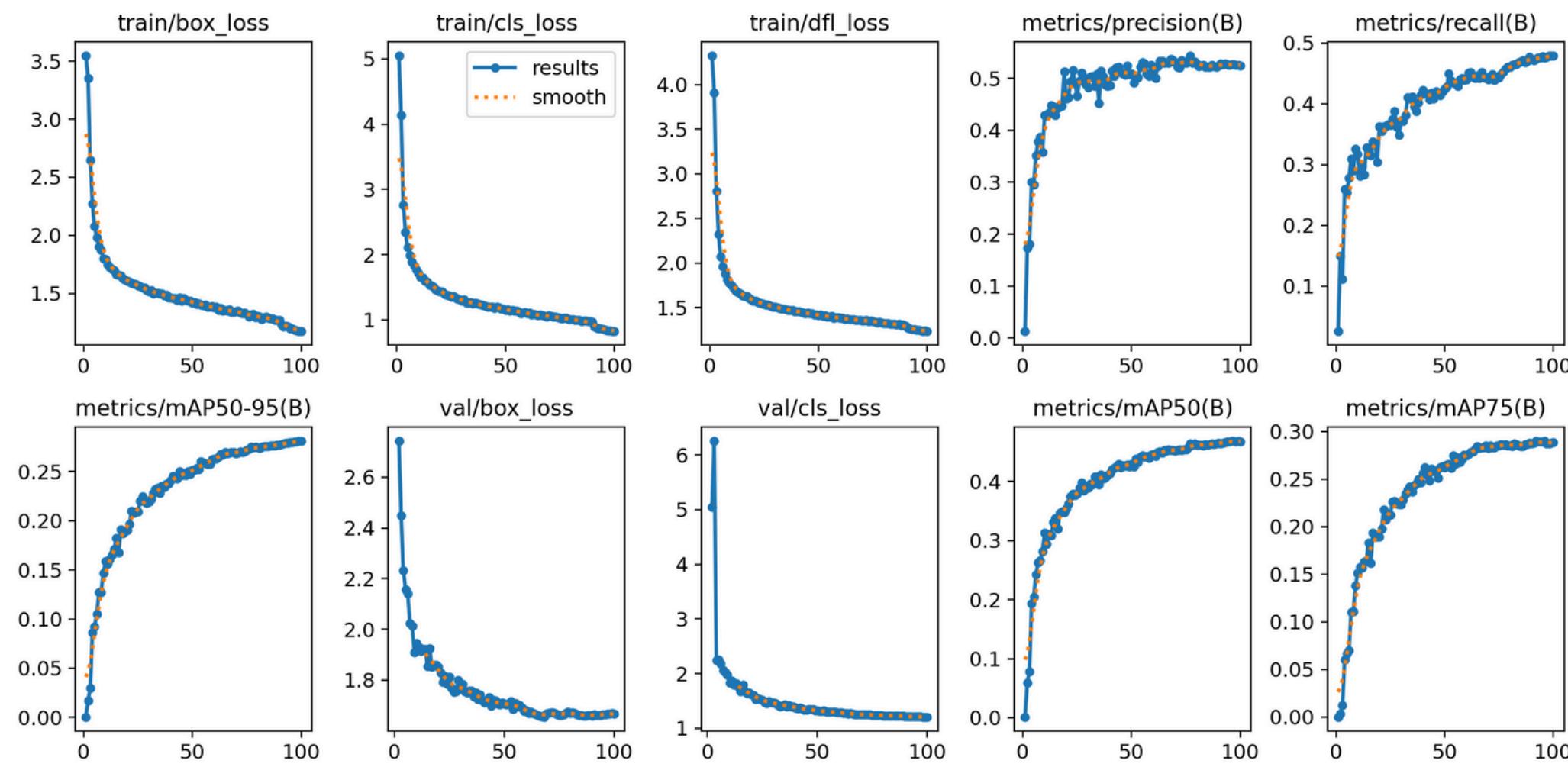
Para um exemplo de aplicação foi utilizado um dataset disponível na plataforma Kaggle com imagens aéreas de barcos, dataset com apenas 1 classe. O F1 máximo foi em torno de 0.50 (50%) com threshold ≈ 0.19 de confiança.



Exemplos de aplicação

Inatel

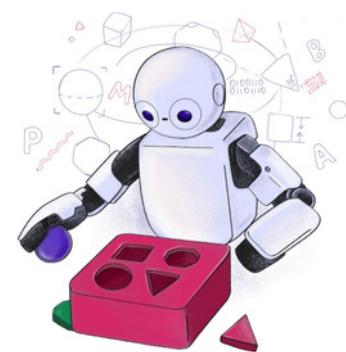
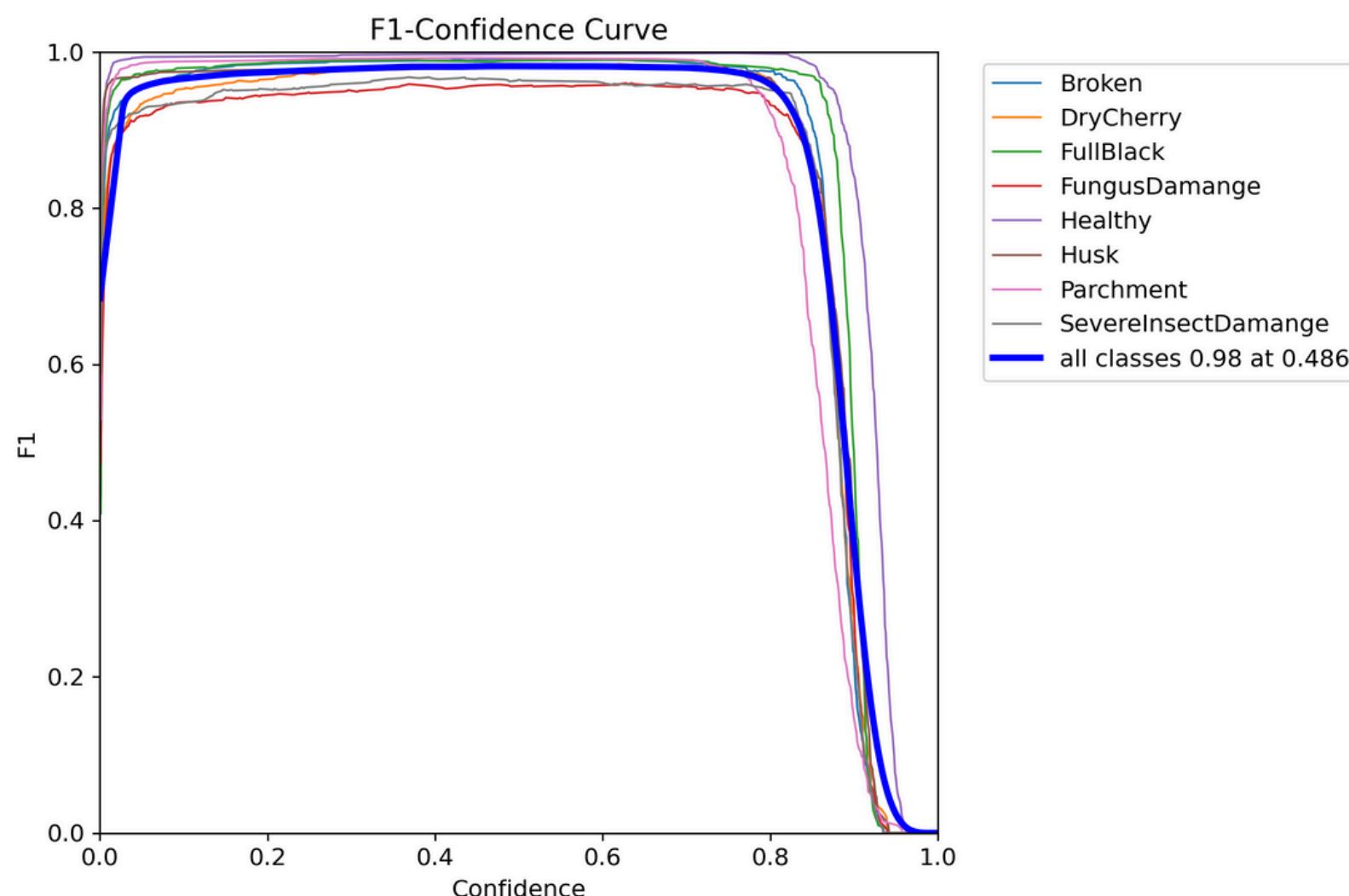
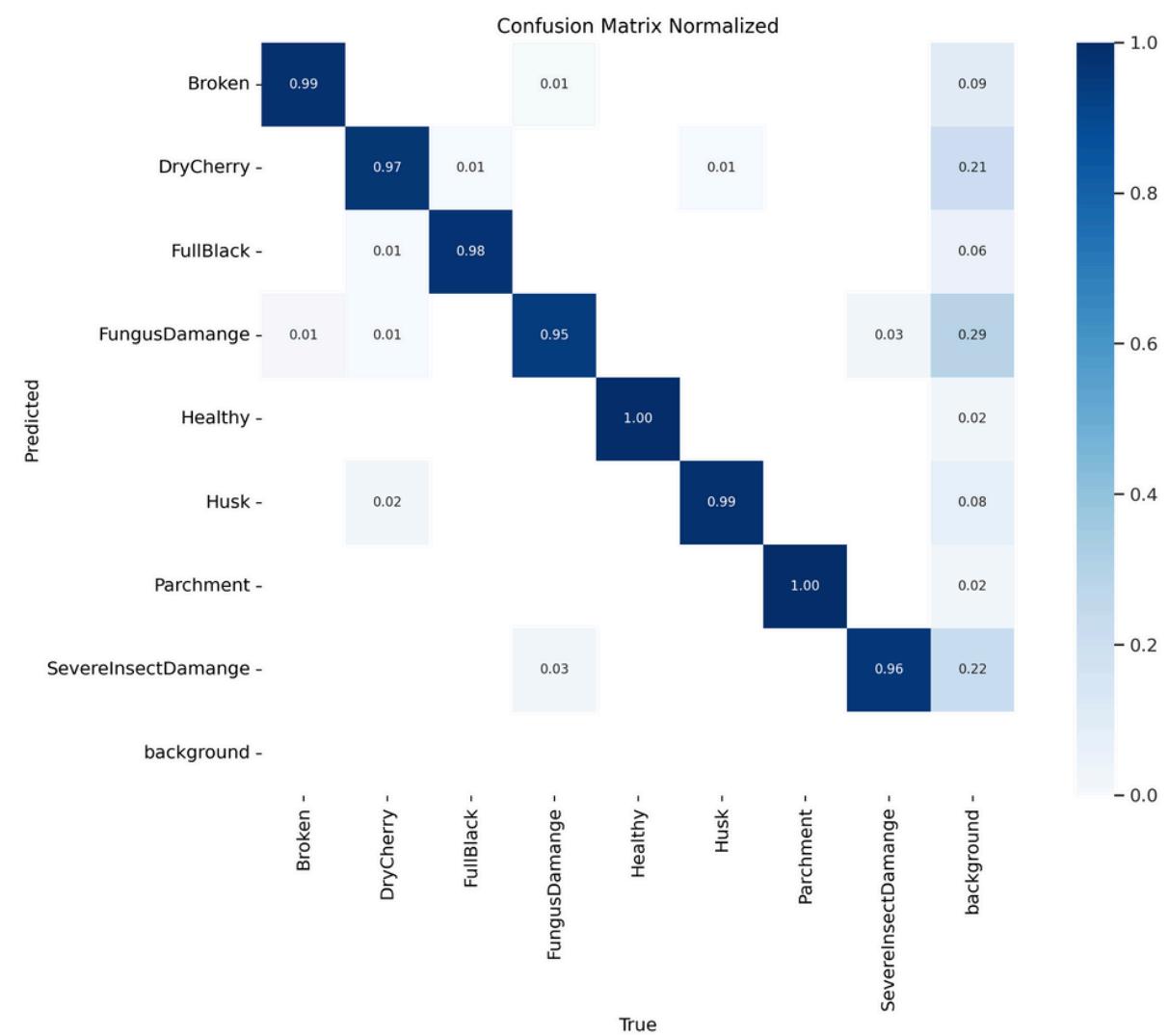
Para um exemplo de aplicação foi utilizado um dataset disponível na plataforma Kaggle com imagens aéreas de barcos. train/box_loss, train/cls_loss, train/dfl_loss , todos caindo de forma estável, sem overfitting aparente. val/box_loss, val/cls_loss também decrescem, mas estabilizam em patamar relativamente alto. Isso mostra que o modelo aprendeu algo, mas não conseguiu reduzir bem os erros de classificação (provavelmente pela dificuldade com background).



Exemplos de aplicação

Inatel

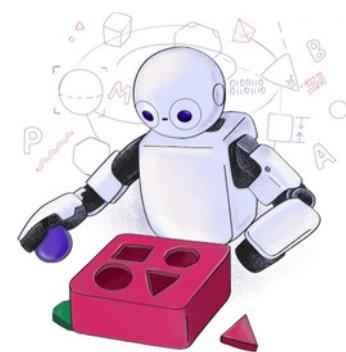
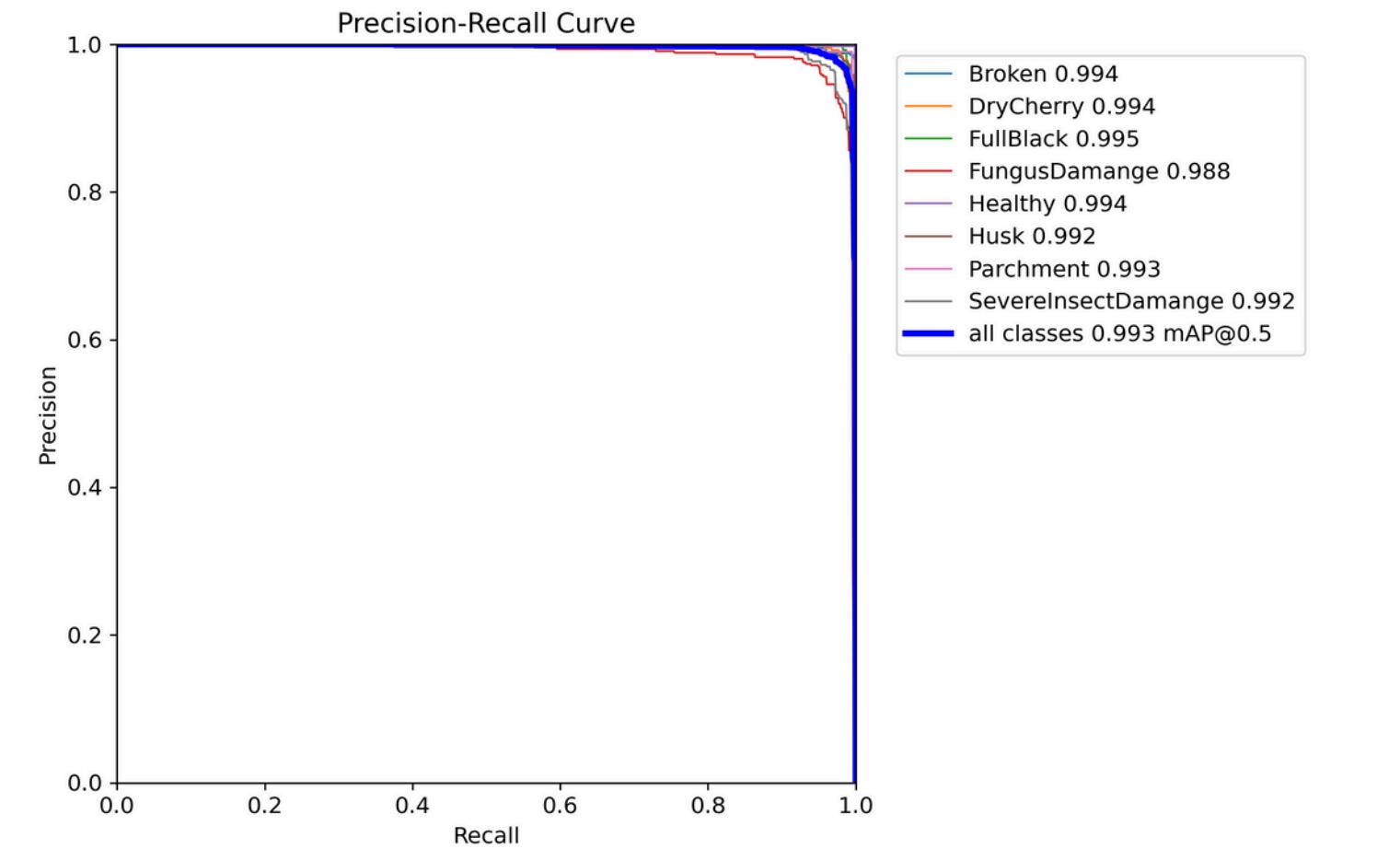
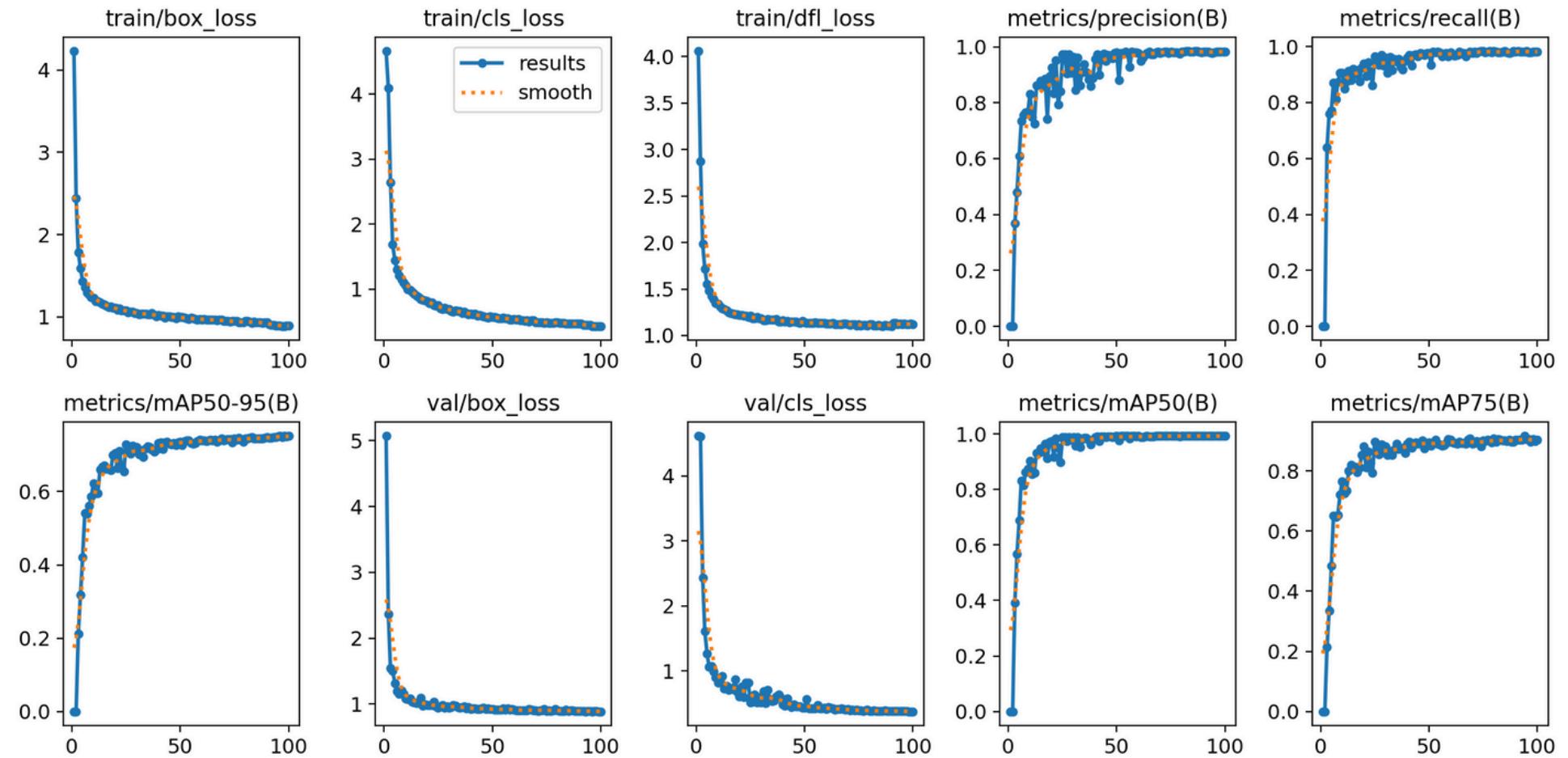
Um segundo exemplo de aplicação utilizando dataset pessoal com mosaicos de imagens de grãos de café, dataset com 8 classes. Nesse verdadeiros positivos são bem maiores, assim como F1 máximo em torno de 0.98 (98%) com threshold ≈ 0.49 de confiança.



Exemplos de aplicação

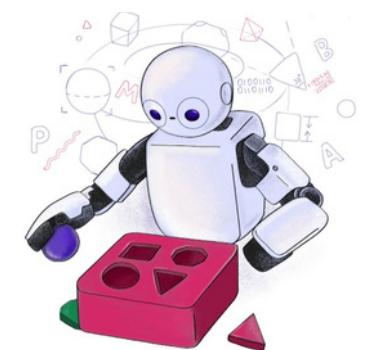
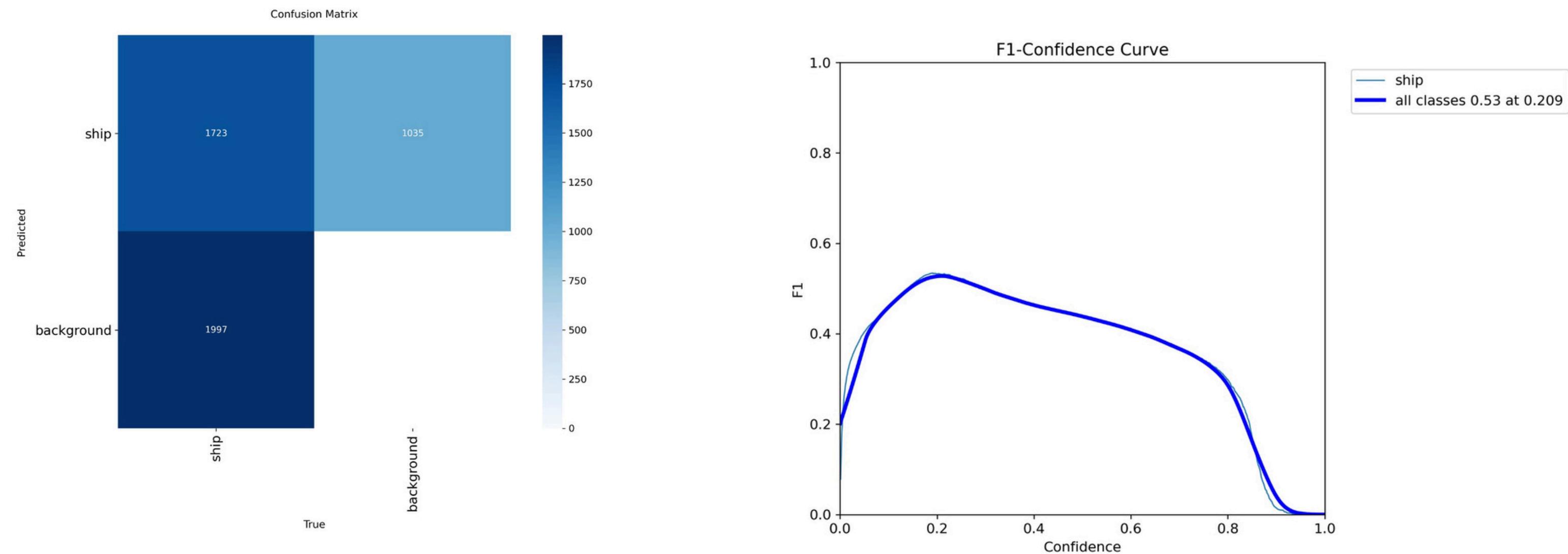
Inatel

Também apresenta bom resultado de aprendizado, sem overfitting e alta precisão.



Comparação com outros algoritmos

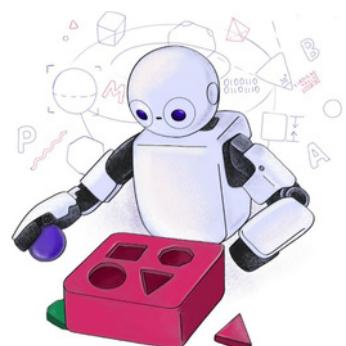
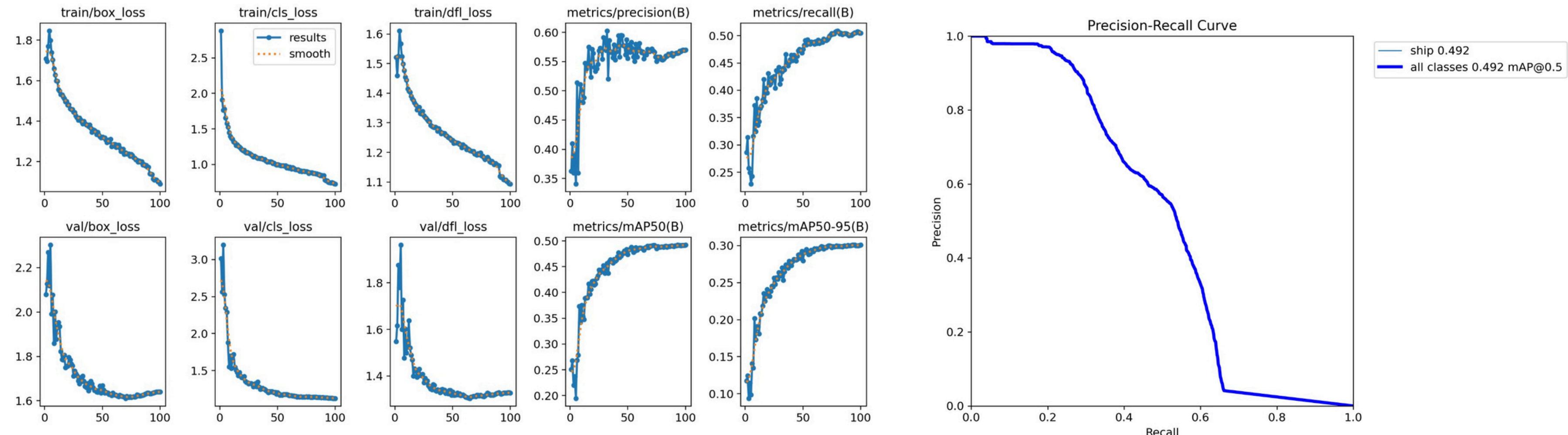
Resultados do treinamento com dataset com imagens de barcos pelo modelo YOLO11n utilizando os mesmos parâmetros do modelo YOLO13n. Modelo YOLO11n obteve F1 máximo foi em torno de 0.53 (53%) com threshold ≈ 0.21 de confiança, valores maiores comparado ao YOLO13n, além de mais acertos em verdadeiros positivos.



Comparação com outros algoritmos

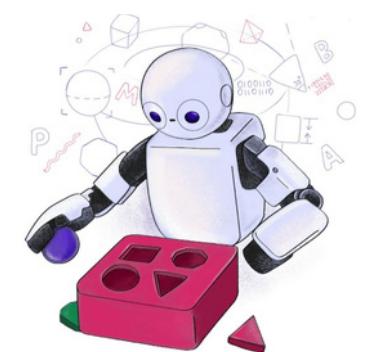
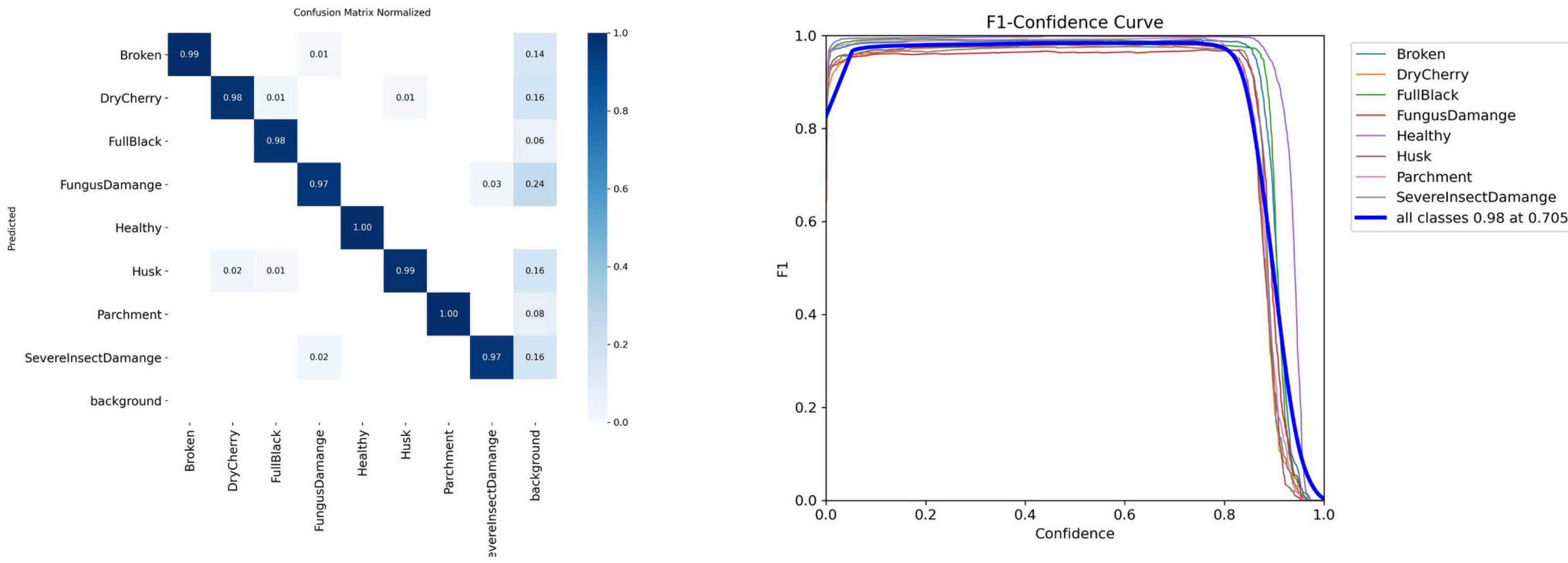
Inatel

YOLO11n também apresenta maior precisão.



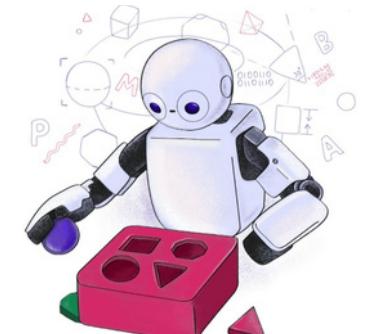
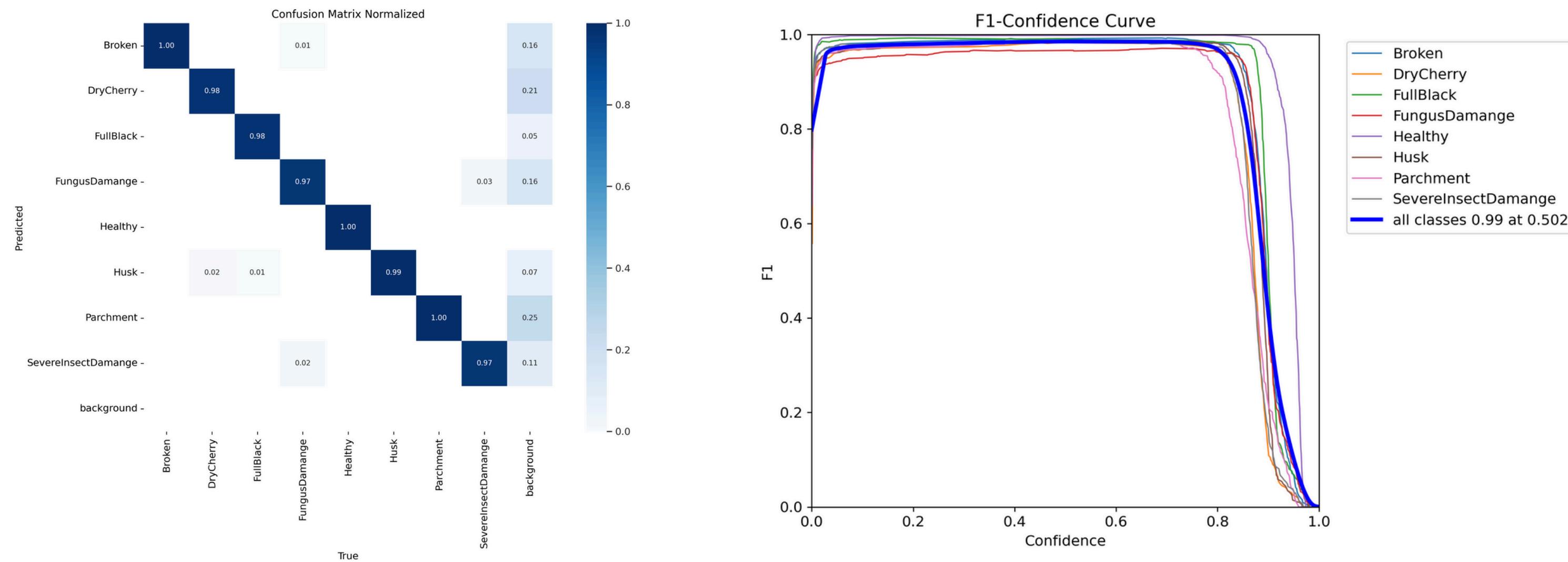
Comparação com outros algoritmos

Resultados do treinamento com dataset com imagens de grãos de café pelo modelo YOLO11n utilizando os mesmos parâmetros. Modelo YOLO11n obteve F1 máximo foi em torno de 0.98 (98%) com threshold ≈ 0.70 de confiança (maior confiança), valores maiores comparado ao YOLO13n, além de mais acertos em verdadeiros positivos.



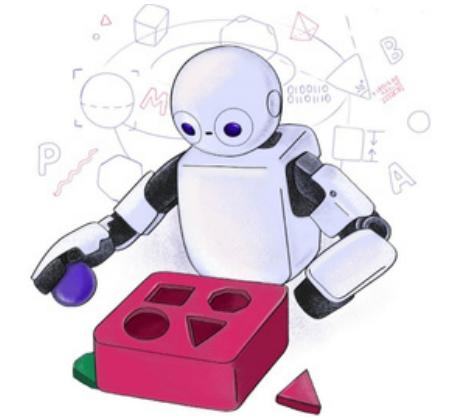
Comparação com outros algoritmos

Resultados do treinamento com dataset com imagens de grãos de café pelo modelo YOLO8n utilizando os mesmos parâmetros. Modelo YOLO8n obteve F1 máximo foi em torno de 0.99 (99%) com threshold ≈ 0.50 de confiança, valores maiores comparado ao YOLO13n, além de mais acertos em verdadeiros positivos.



Perguntas?

Inatel



Referências

- [1] Y. Feng, J. Huang, S. Du, S. Ying, J.-H. Yong, Y. Li, G. Ding, R. Ji, and Y. Gao, “Hyper-YOLO: When Visual Object Detection Meets Hypergraph Computation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 47, no. 4, pp. 2388–2401, 2025.
- [2] J. Fixelle, “Hypergraph vision transformers: Images are more than nodes, more than edges,” in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2025, pp. 9751–9761.
- [3] H. Wang, S. Zhang, and B. Leng, “HGFormer: Topology-aware vision transformer with hypergraph learning,” *IEEE Trans. Multimedia*, 2025.
- [4] Lei, M., Li, S., Wu, Y., Hu, H., Zhou, Y., Zheng, X., Ding, G., Du, S., Wu, Z., & Gao, Y. (2025). YOLOv13: Real-Time Object Detection with Hypergraph-Enhanced Adaptive Visual Perception. *arXiv preprint arXiv:2506.17733*.

Obrigado!

Inatel

