1. INTRODUÇÃO

Neste projeto, foi desenvolvido um sistema de **detecção de objetos** com foco na identificação de elementos presentes em ambientes ferroviários, como trilhos, infraestrutura e vegetação, assim como objetos gerais do cotidiano (pessoas, veículos, etc.).

A motivação principal foi criar uma solução que pudesse **automatizar o monitoramento de vias férreas**, potencialmente aplicável para aumentar a segurança e auxiliar na prevenção de acidentes.

Para alcançar esse objetivo, realizamos o **treinamento e fine-tuning de um modelo YOLOv8**, combinando imagens do dataset **COCO (Common Objects in Context)** com imagens customizadas de trilhos e infraestrutura ferroviária do RailSem19.

O conjunto de dados RailSem19 compreende 8.500 imagens, capturadas da perspectiva de um trem ou bonde. As imagens retratam trilhos em diversos cenários, abrangendo ambientes urbanos, rurais e cruzamentos rodoviários, sob variadas condições climáticas, como chuva e neblina, e diferentes níveis de iluminação. O dataset está disponível publicamente emhttps://www.kaggle.com/datasets/ngohoang0207/railsem19.

As anotações fornecidas destinam-se à segmentação semântica e englobam categorias como trilhos, trens, plataformas, sinais, cruzamentos e pedestres, mantendo compatibilidade com o conjunto de dados Cityscapes. O conjunto inclui tanto anotações geométricas (polígonos) quanto rotulagem densa pixel a pixel.

2. DESENVOLVIMENTO / TÉCNICAS UTILIZADAS

O sistema foi desenvolvido com as seguintes etapas principais:

2.1 Preparação dos Dados

- **Dataset COCO:** Utilizado por conter 80 classes de objetos comuns (pessoas, veículos, sinais, etc.).

- **Dataset RailSem19:** Composto por imagens contendo trilhos, vegetação e infraestrutura.

- **Conversão de Anotações:** As anotações COCO (em formato JSON) foram convertidas automaticamente para o formato YOLO, com o script convert_coco_to_yolo.py, que processou e gerou a pasta coco_yolo_format, contendo os arquivos *images*, *labels* e o *data.yaml*.

- **Reindexação:** Como as classes customizadas eram adicionais, foi utilizado o script reindex_labels.py que reindexou as classes customizadas a partir do índice 80, evitando conflitos com as classes COCO. O script reindex_labels.py está presente no diretório dataset_final.

- **Unificação de Imagens e Labels:** Os dois conjuntos foram combinados em uma estrutura final de pastas (images/train, images/val, labels/train, labels/val) e um único arquivo data.yaml contendo a lista das 85 classes totais.

2.2 Modelo Utilizado

- **YOLOv8s**: Arquitetura de detecção de objetos otimizada para rapidez e boa performance.
- **Fine-Tuning:** Partindo dos pesos pré-treinados no COCO, foi feito o ajuste ao novo domínio ferroviário.

- **Configurações principais:**

    - Épocas: 100 (Early Stopping aplicado)

    - Tamanho de imagem: 640 pixels

    - Batch Size: 12

    - Otimizador: AdamW

    - Augmentações: Ativadas para melhorar generalização (ex.: flip horizontal, variação de brilho)

    - Mixed Precision Training (amp=True) ativado para economia de memória e desempenho.

2.3 Treinamento

O treinamento foi realizado com monitoramento de métricas principais:

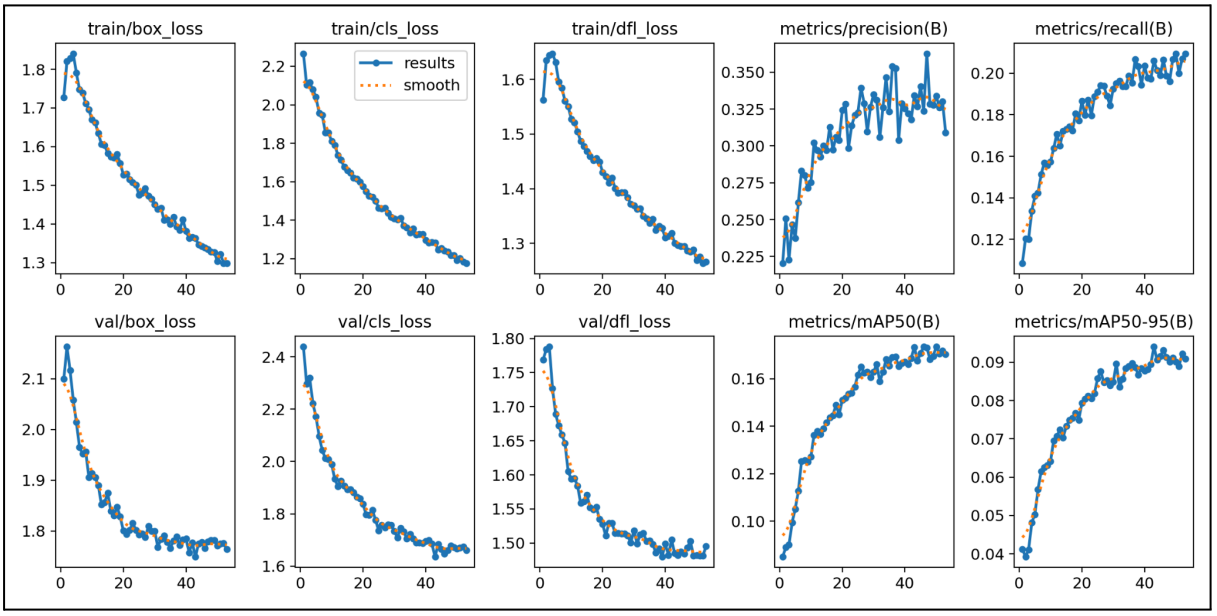
- **Loss (Box, Class, DFL)**

- **mAP50 e mAP50-95** por classe

- **Precisão e Recall**

O melhor modelo foi salvo automaticamente após atingir o maior mAP na validação no diretório projeto2\dataset_final\runs\detect com o nome de rail_detect_yolov8s_v24.

Gráfico 1: **Análise dos Resultados de Treinamento**



Com base no gráfico 1 fornecido, é possível observar uma clara evolução do treinamento. As perdas demonstram uma tendência decrescente e consistente, indicando aprimoramento na localização e classificação de objetos. A perda de caixa (**box_loss**) iniciou em aproximadamente 0.20 e decresceu para cerca de 0.12, enquanto a perda de classificação (**cls_loss**) partiu de um valor em torno de 1.8 e reduziu para aproximadamente 1.3. A perda DFL (**dfl_loss**) também exibiu uma melhoria significativa, começando próximo a 2.2 e finalizando em torno de 1.2, o que sugere uma melhoria na distribuição focal, um aspecto crucial em modelos como o YOLOv8.

No que tange às métricas de validação, a precisão (**precision(B)**) atingiu valores entre 1.6 e 1.3, com alguma flutuação ao longo do treinamento, embora os valores pareçam elevados, o que pode indicar uma escala distinta. O recall (**recall(B)**) variou entre aproximadamente 1.5 e 1.3, revelando uma tendência geral de melhoria, com algumas oscilações. As métricas mAP50 e mAP50-95, por sua vez, apresentaram valores que parecem baixos (entre 0.09 e 0.04 para mAP50 e entre 0.20 e 0.05 para mAP50-95), sendo imperativo verificar a correção da escala, possivelmente multiplicando por 100 para expressar em porcentagem. Essa verificação é crucial para uma interpretação precisa do desempenho do modelo nessas métricas.

2.4 INFERÊNCIA

Após o treinamento, foi feita predição em vídeos reais de trilhos ferroviários. O script de inferência permitiu:

- Visualização ao vivo das detecções.
- Armazenamento dos vídeos anotados.
- Ajuste dinâmico de limiar de confiança (conf=0.5).

3. RESULTADOS

O modelo alcançou os seguintes indicadores finais:

- **mAP50 geral:** ~0,17 . Observa-se que, em média, o modelo demonstra uma precisão de 17% na detecção de objetos, considerando uma intersecção sobre união (IoU) mínima de 50% entre as previsões e as anotações. Este valor é considerado baixo, indicando que o modelo apresenta desafios na localização e classificação precisa de objetos.
- **Classes personalizadas:**
    - *track_custom*: mAP50 ~0,19. Desempenho ligeiramente melhor que a média geral, mas ainda insatisfatório.
    - *vegetation_custom*: mAP50 ~0,24. Melhor classe detectada, possivelmente devido a características mais distintas (cor, forma).
    - *infrastructure_custom*: mAP50 ~0,13 . A pior classe, indicando dificuldade em identificar objetos como postes, placas ou estruturas urbanas.

Os resultados mostraram que, embora o modelo tenha identificado bem alguns objetos (especialmente vegetação e trilhos), ainda há espaço para melhorias, principalmente aumentando o número de imagens customizadas e refinando as anotações.

Nos testes com vídeo, o modelo foi capaz de identificar e rotular diferentes classes com desempenho satisfatório em tempo real.

4. CONSIDERAÇÕES FINAIS

Este projeto demonstrou a viabilidade do uso do YOLOv8 para detecção de objetos ferroviários em conjunto com classes genéricas do COCO, evidenciando como o **fine-tuning de modelos pré-treinados acelera o desenvolvimento de aplicações reais**.

Futuros aprimoramentos incluem:

- Coleta de imagens em cenários variados (dia/noite, diferentes climas).
- Experimentação com arquiteturas maiores (YOLOv8m, YOLOv8l).
- Aplicação de técnicas de Data Augmentation mais avançadas.

5. Referências

- Ultralytics YOLOv8: https://docs.ultralytics.com/
- COCO Dataset: https://cocodataset.org/
- PyTorch: https://pytorch.org/
- RailSem19 dataset: https://www.kaggle.com/datasets/ngohoang0207/railsem19%20