YOLO-Traffic Analytics: Detecção e Contagem de Veículos para Análise de Tráfego Urbano

Vinícius Santos Monteiro

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
São Carlos – SP – Brazil

vini.mon@usp.br

Abstract. *The efficient management of urban traffic is a growing logistical challenge. This study details the development of "YOLO - Traffic Analytics," a computer vision system for the automatic detection, classification, and counting of vehicles in traffic imagery. We utilize a public Kaggle dataset, which presented a severe imbalance across 21 distinct vehicle classes (e.g., 'car', 'suv', 'taxi', 'van'). The core of the proposed approach was the consolidation of these 21 classes into 5 functional categories (Car, Bus, Truck, Motorcycle, Bicycle), cleaning the dataset of noise and focusing the model on the project's objective. We compare two model architectures, YOLOv8s (Small) and YOLOv8m (Medium), to analyze the trade-off between inference speed and precision (mAP). The results demonstrate a robust pipeline for extracting traffic statistics, validating the effectiveness of class fusion.*

Resumo. *O gerenciamento eficiente do tráfego urbano é um desafio logístico crescente. Este estudo detalha o desenvolvimento do "YOLO - Traffic Analytics", um sistema de visão computacional para a detecção, classificação e contagem automática de veículos em imagens de trânsito. Utilizamos um dataset público do Kaggle, que apresentava um severo desbalanceamento com 21 classes de veículos distintas (ex: 'car', 'suv', 'taxi', 'van'). A principal abordagem proposta foi a consolidação dessas 21 classes em 5 categorias funcionais (Carro, Ônibus, Caminhão, Moto, Bicicleta), limpando o dataset de ruídos e focando o modelo no objetivo do projeto. Comparamos duas arquiteturas de modelo, YOLOv8s (Small) e YOLOv8m (Medium), para analisar o trade-off entre velocidade de inferência e precisão (mAP). Os resultados demonstram um pipeline robusto para extração de estatísticas de tráfego, validando a eficácia da fusão de classes.*

1. Introdução, Motivação e Objetivos

O crescimento das cidades trouxe o desafio da gestão do tráfego urbano. É desejado saber como diferentes tipos de veículos aparecem no trânsito para otimizar o fluxo, gerenciar a logística de carga e planejar políticas públicas de mobilidade. Tradicionalmente, essa contagem é feita manualmente, por um funcionário ou com sensores de solo, métodos que são caros, de difícil manutenção e nem sempre muito precisos.

A Visão Computacional, especificamente modelos de detecção de objetos como o YOLO (You Only Look Once) [1], oferece uma solução escalável, rica em dados (contando por tipo de veículo) e adaptável para monitoramento em tempo real.

O código-fonte e o notebook de treinamento podem ser encontrados nos seguintes links: [1] [2]

Os objetivos principais deste trabalho são:

- Investigar um dataset público de tráfego e identificar suas limitações (desbalanceamento, granularidade).
- Desenvolver um pipeline de pré-processamento para mitigar o desbalanceamento e o ruído dos dados por meio da fusão de classes.
- Treinar e comparar dois modelos da família YOLOv8 (Small e Medium) para avaliar a precisão (mAP) em relação ao custo computacional.
- Discutir a viabilidade da solução e os ajustes necessários para uma implementação em um cenário real.

2. Abordagem Proposta (Metodologia)

A solução foi desenvolvida como um pipeline de detecção de objetos utilizando a biblioteca *Ultralytics*. A metodologia foi dividida em quatro etapas principais: (1) Análise Exploratória de Dados, (2) Pipeline de Pré-processamento e Fusão de Classes, (3) Definição da Arquitetura do Modelo e (4) Estratégia de Treinamento.

2.1. Análise Exploratória de Dados (EDA)

O dataset original, obtido do Kaggle, continha 2.704 imagens de treino e 300 de validação, com rótulos para 21 classes. A primeira análise (Figura 1) revelou dois problemas críticos:

- **Severo Desbalanceamento:** Classes como 'car' e 'pickup' dominavam o conjunto de dados com milhares de instâncias, enquanto classes como 'ambulance' ou 'policecar' eram quase inexistentes.
- **Alta Granularidade:** O objetivo do projeto era contar "Carros", mas o conjunto de dados dividia isso em 'car', 'suv', 'taxi', 'van' e 'minivan'. Isso força o modelo a gastar recursos aprendendo a diferenciar classes que, para o objetivo final, são idênticas.

[1] https://www.kaggle.com/code/viniciusjamal/yolo-traffic-analytics
[2] https://github.com/vini-mon/YOLO-Traffic-Analytics

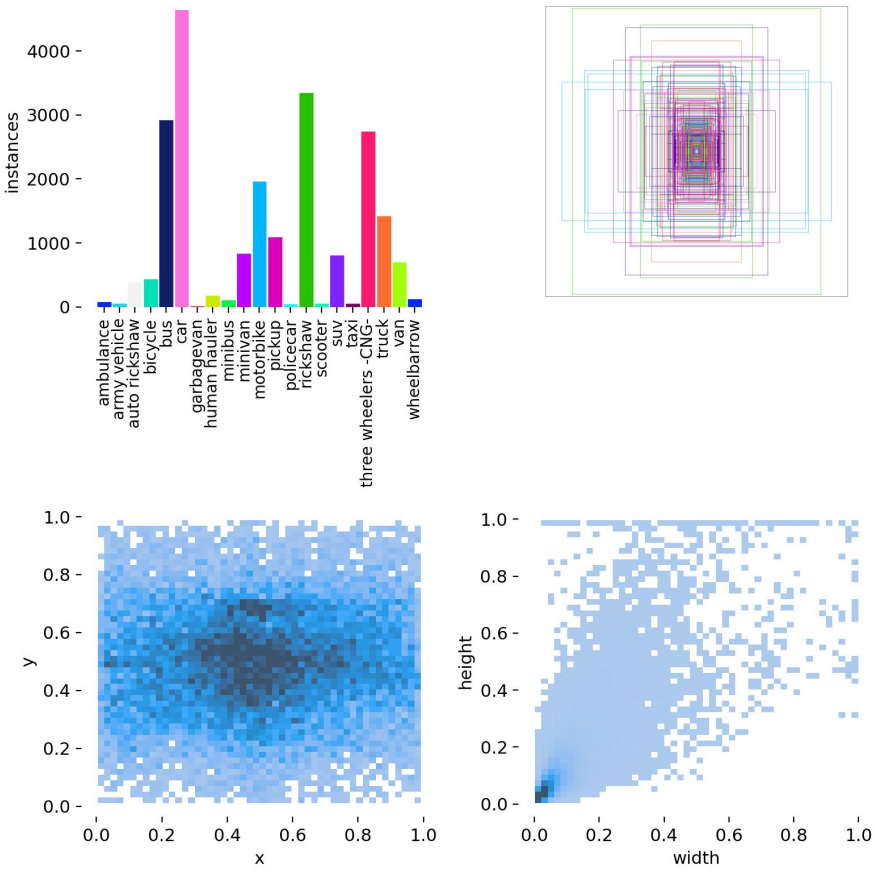

Figure 1. Distribuição original (desbalanceada) das 21 classes.

2.2. Pipeline de Dados: Limpeza e Fusão de Classes

Para resolver os problemas identificados, executamos uma etapa de pré-processamento. Desenvolvemos um script Python que, primeiramente, limpou e validou o *dataset*, removendo arquivos órfãos não essenciais (como os arquivos de *cache*).

Em seguida, implementamos a **fusão de classes**. As 21 classes originais foram mapeadas para 5 super-classes funcionais:

- **'car'**: Fundiu 'car', 'suv', 'taxi', 'van', 'ambulance', 'policecar', 'minivan'.
- **'bus'**: Fundiu 'bus', 'minibus'.
- **'truck'**: Fundiu 'truck', 'pickup', 'garbagevan', 'army vehicle'.
- **'motorcycle'**: Fundiu 'motorbike', 'scooter'.
- **'bicycle'**: Manteve 'bicycle'.

Classes irrelevantes para o projeto (ex: 'rickshaw', 'wheelbarrow', 'human hauler') foram descartadas. Isso reduziu o ruído, simplificou o problema e forçou o modelo a aprender apenas o que era essencial para o objetivo. É possível ver na (Figura 2) o desbalanceamento do *dataset* final.

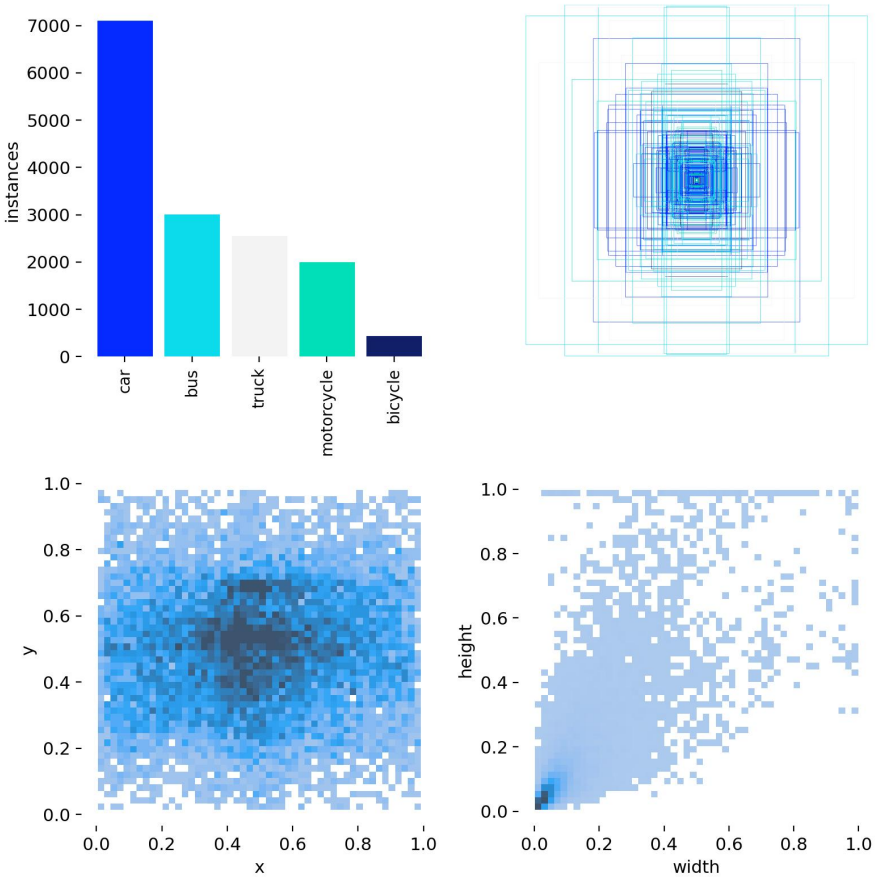

Figure 2. Distribuição final com as 5 classes.

2.3. Arquitetura do Modelo: YOLOv8 (S vs. M)

Escolhemos a família de modelos YOLOv8 pela sua alta performance em detecção em tempo real. Para este estudo, comparamos duas arquiteturas:

- **YOLOv8s (Small):** Um modelo leve (11.1M de parâmetros), ideal para implantação em dispositivos com restrições de processamento.
- **YOLOv8m (Medium):** Um modelo de tamanho médio (25.8M de parâmetros), que oferece um balanço entre precisão e velocidade.

Ambos os modelos foram treinados usando Aprendizado por Transferência, iniciando com os pesos pré-treinados no conjunto de dados COCO [2].

2.4. Estratégia de Treinamento

O treinamento foi conduzido no ambiente Kaggle (GPU Tesla T4). Definimos um máximo de 500 épocas, com um *callback* de *Early Stopping* monitorando a métrica de validação `mAP50-95` com uma paciência (*patience*) de 25 épocas. Isso garantiu que o treino parasse automaticamente quando o modelo não apresentasse mais melhorias, otimizando o tempo de treino e selecionando o melhor *checkpoint* do modelo.

3. Resultados Experimentais

Ambos os modelos foram treinados no dataset fundido de 5 classes. A métrica primária de avaliação é a **mAP@.50**, que é o padrão para precisão de detecção de objetos (Média das Precisões Médias com IoU de 50%).

3.1. Análise de Treinamento e Convergência

Os gráficos das (Figura 3 e 4) mostram a evolução do treinamento para os modelos Small e Medium.

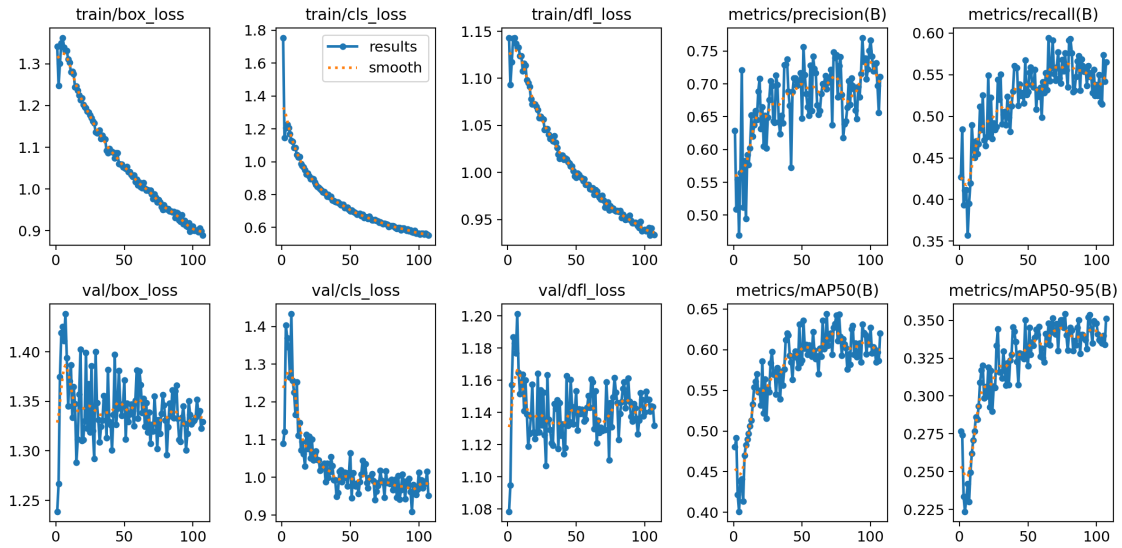

Figure 3. (a) Curvas de Treinamento – YOLOv8 Small

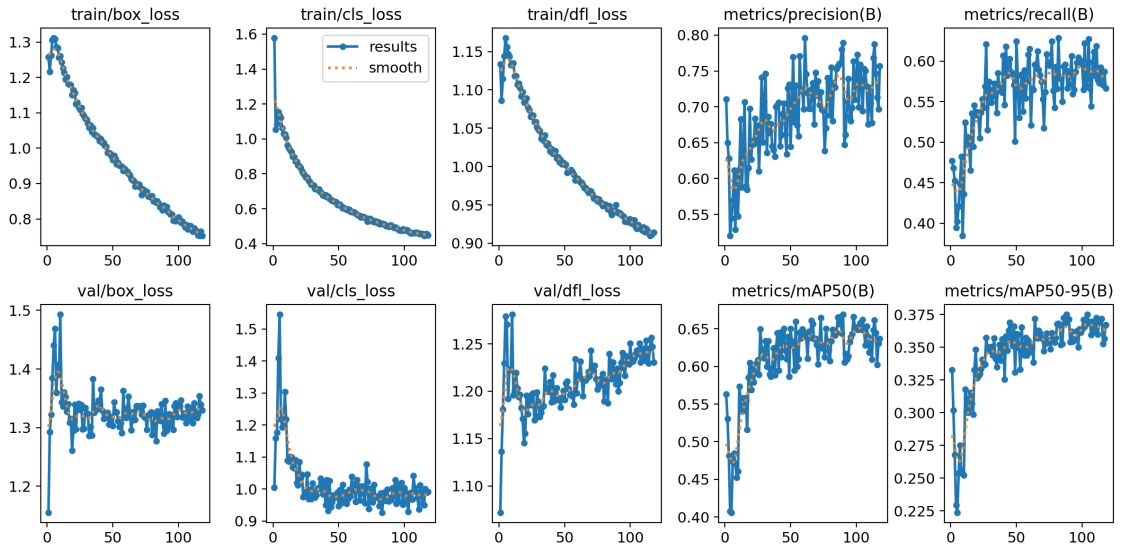

Figure 4. (b) Curvas de Treinamento – YOLOv8 Medium

Ambos os modelos convergiram com sucesso, com o *Early Stopping* interrompendo o treino após a estabilização das métricas. O modelo Medium (b) alcançou

um `mAP50-95` (métrica de paciência) ligeiramente superior, atingindo um pico de 0.37, contra 0.35 do modelo Small (a).

3.2. Performance de Detecção (Curva PR)

A Curva de Precisão-Recall (Figuras 5 e 6) são a avaliação principal.

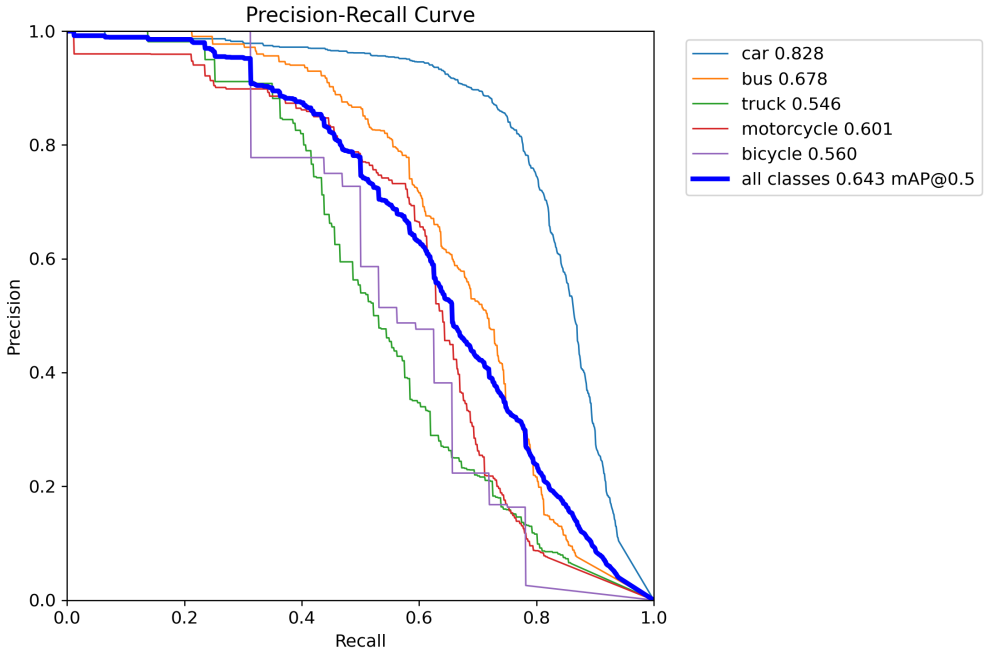

Figure 5. Curva PR - YOLOv8 Small (mAP@.50 = 0.643)

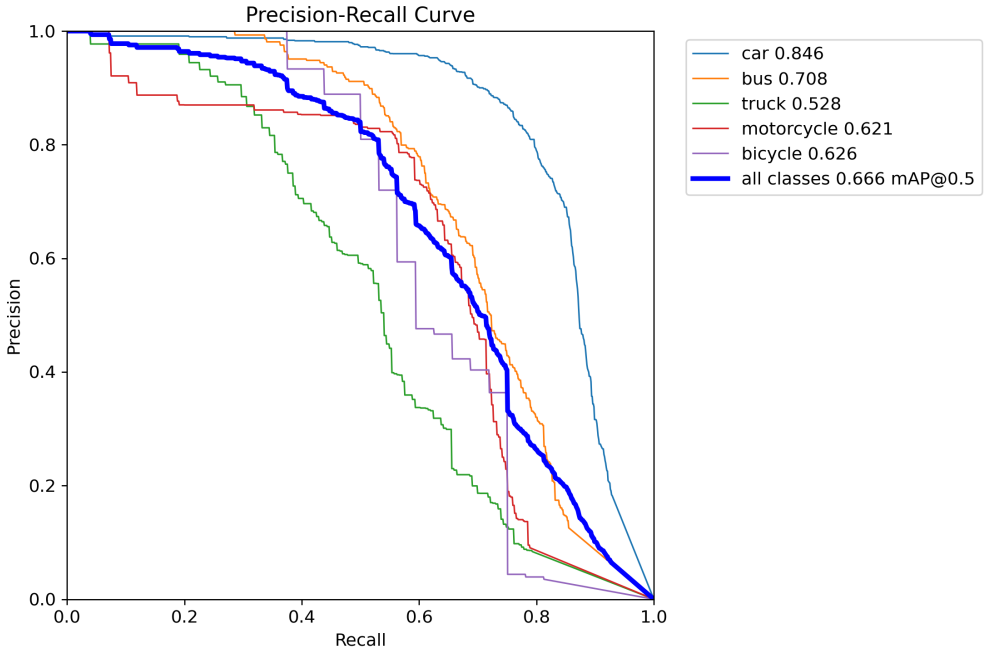

Figure 6. Curva PR - YOLOv8 Medium (mAP@.50 = 0.666)

O modelo YOLOv8m (Medium) alcançou um **mAP@.50 de 0.666**, um resultado 3.6% (relativo) superior ao YOLOv8s (Small), que alcançou **0.643**.

Uma análise por classe revela um *trade-off* interessante:

- O modelo **Medium** foi superior na detecção de 'car' (0.846 vs 0.828), 'bus' (0.708 vs 0.678), 'motorcycle' (0.621 vs 0.601) e 'bicycle' (0.626 vs 0.560).
- Surpreendentemente, o modelo **Small** foi superior na detecção de 'truck' (0.546 vs 0.528). Isso pode ser uma anomalia estatística ou indicar que o modelo Medium estava generalizando excessivamente características de 'truck' com 'car'.

3.3. Análise de Custo Computacional (Tempo de Treino)

Embora o modelo *Medium* tenha apresentado um ganho de performance, é crucial analisar o custo computacional associado. O tempo total de treinamento (utilizando *Early Stopping* em GPU Tesla T4) apresentou uma diferença significativa:

- **YOLOv8s (Small):** 1 hora, 28 minutos e 52 segundos.
- **YOLOv8m (Medium):** 3 horas e 17 minutos e 0 segundos.

O modelo *Medium* exigiu aproximadamente **2.2x mais tempo de processamento** para alcançar seu pico de performance. Esta informação é vital para a decisão de implementação, ponderando o ganho de 3.6% em mAP@.50 contra um aumento de 120% no tempo de treino.

3.4. Análise de Erros (Matriz de Confusão)

As Matrizes de Confusão (Figuras 7 e 8) nos permite analisar os tipos de erro que os modelos cometeram no set de validação.

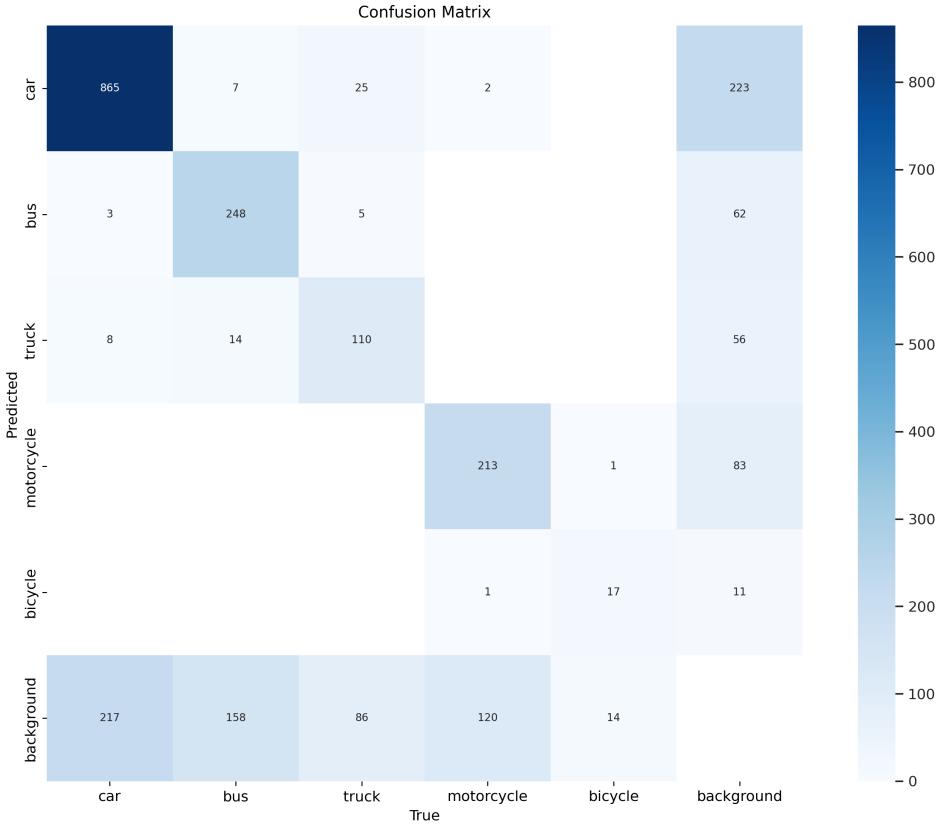

Figure 7. Matriz de Confusão - YOLOv8 Small

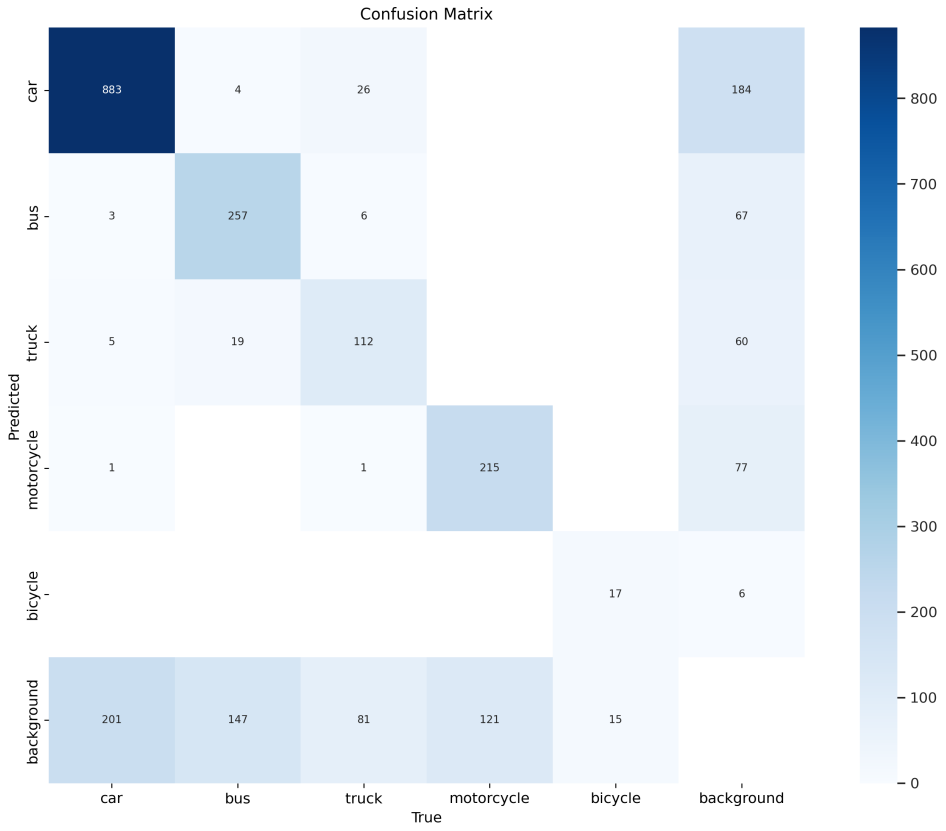

Figure 8. Matriz de Confusão - YOLOv8 Medium

A análise das matrizes revela que o principal tipo de erro para **ambos** os modelos não é a confusão entre classes (ex: prever 'car' quando era 'truck'), mas sim **falsos negativos** (prever 'background' quando havia um objeto).

- **Small:** Perdeu 217 carros, 158 ônibus e 86 caminhões para o 'background'.
- **Medium:** Foi ligeiramente melhor, perdendo 201 carros, 147 ônibus e 81 caminhões.

Isso indica que o modelo Medium é um pouco mais "confiante" em suas detecções. A confusão entre classes, como 'truck' sendo previsto como 'car' (25 no *Small*, 26 no *Medium*), foi mínima e muito similar entre os modelos.

4. Discussão e Limitações

Os resultados do modelo são a base para um Sistema de Tráfego Inteligente (ITS). A contagem em tempo real permite a análise de fluxo (para ajuste dinâmico de semáforos) e o planejamento urbano (identificando quais vias são mais usadas por caminhões vs. carros).

No entanto, o dataset e o modelo apresentam limitações críticas para uma implantação no mundo real:

- **Viés Geográfico (Região Única):** O dataset original (com 'rickshaws') foi capturado em uma região específica (provavelmente Sul da Ásia). Os modelos

de veículos (ônibus, caminhões) são visualmente diferentes dos encontrados no Brasil.

- **Condições Climáticas:** A maioria das imagens foi capturada em condições ideais (dia, tempo bom). O modelo provavelmente falharia à noite, com chuva intensa ou neblina.
- **Perspectiva da Câmera:** Quase todas as imagens tiradas na perspective de um cidadão ou por câmeras dentro dos veículos. O modelo não foi treinado para detecção ao nível da rua, ou de pontos elevados, onde a oclusão (um veículo na frente do outro) é um problema muito maior.

Para aplicar este sistema em outra cidade (ex: São Paulo), seria necessário coletar e rotular imagens locais (dia/noite, chuva/sol) e aplicar *Fine-Tuning* (Ajuste Fino) no modelo YOLOv8m. Para tempo real, o modelo precisaria ser otimizado (ex: TensorRT) e executado em hardware de inferência dedicado (ex: NVIDIA Jetson).

5. Conclusões

Este trabalho demonstrou um pipeline completo para a criação de um detector de veículos robusto. Provou-se que a etapa de análise e pré-processamento de dados foi fundamental; a decisão de **fundir 21 classes em 5** foi o fator crucial para o sucesso do projeto, eliminando ruído e focando o esforço de aprendizado do modelo.

A comparação entre o YOLOv8s e o YOLOv8m revelou que o modelo Médio (YOLOv8m) oferece um ganho de precisão de 3.6% (mAP@.50 de 0.666 vs 0.643), com um **custo computacional 120% maior** (3h 17m vs 1h 28m).

Concluímos que o YOLOv8m é a escolha ideal para implementações em servidor onde a precisão máxima é a prioridade. Em contrapartida, o YOLOv8s, com performance muito próxima, é o candidato superior para dispositivos de borda (*edge*) onde o custo computacional e a velocidade de inferência são restrições críticas.

References

[1] Jocher, G., Chabot, L., & Shojma, T. (2024). *YOLOv8: Ultralytics YOLOv8*. Disponível em: `https://github.com/ultralytics/ultralytics`

[2] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). *Microsoft COCO: Common Objects in Context*. In European Conference on Computer Vision (ECCV).