Detecção, Rastreamento e Contagem de Veículos para Análise de Tráfego Urbano utilizando Métodos de Aprendizagem Profunda

Guilherme de S. Sampaio
guilhermesampaio1997@gmail.com
Instituto Federal de Educação, Ciência e Tecnologia do Amazonas - *Campus* Manaus Zona Leste
Manaus, Amazonas, Brasil

José E. B. de S. Linhares
breno.linhares@ifam.edu.br
Instituto Federal de Educação, Ciência e Tecnologia do Amazonas - *Campus* Manaus Zona Leste
Manaus, Amazonas, Brasil

RESUMO

The constant increase in the number of vehicles around the world has drawn attention to the need to implement effective traffic monitoring and control technologies. In this work, a methodology is presented that uses two algorithms, one that applies the YOLOv8 detector and the other that applies Deep SORT tracking, to detect, track and count vehicles. Validation results during training of the vehicle detection model revealed an overall accuracy of 89, 2% *and mAP50 of* 93, 7% *on the AIC HCMC* 2020 *database. Additionally, when applying the model to a test video, an overall accuracy of* 81, 2% *and mAP50 of* 87, 5% *was achieved. These results highlight the effectiveness of the detection algorithm for use in applications involving vehicle tracking and counting, with significant potential for traffic management.*

KEYWORDS

Urban traffic analysis, YOLO, Deep SORT, Vehicle counting

1 INTRODUÇÃO

A gestão eficiente do tráfego urbano é uma tarefa complexa que vem recebendo cada vez mais atenção, principalmente devido ao crescimento das cidades. Com o aumento do número de veículos nas ruas, tornou-se imprescindível a necessidade de monitorar, analisar e otimizar o fluxo de tráfego. Conforme apresentado por [21], o congestionamento de tráfego é um problema crescente que continua a afligir áreas urbanas com resultados negativos tanto para o público que viaja quanto para a sociedade como um todo. Esses resultados negativos só vão aumentar com o tempo, à medida que mais pessoas se dirigem para áreas urbanas.

Sob uma perspectiva global, o desafio do congestionamento do tráfego transcende as delimitações de cidades e das fronteiras nacionais. Diversas cidades ao redor do mundo enfrentam desafios semelhantes, em consequência do aumento expressivo do número de veículos e da infraestrutura incapaz de atender a essa demanda em constante crescimento. No contexto brasileiro, conforme apontado por dados do estudo anual conduzido pelo Sindipeças (Sindicato Nacional da Indústria de Componentes para Veículos Automotores), a frota automotiva do país expandiu em 3, 7% no ano de 2020 em comparação aos três anos anteriores, totalizando 59, 1 milhões de veículos em circulação. Esse incremento, embora benéfico para o desenvolvimento econômico, também resultou no aumento progressivo dos congestionamentos nas áreas urbanas [18].

A necessidade de abordar de maneira eficaz esse desafio global é evidente. Pois, a utilização de uma solução precisa para detecção, rastreamento e contagem de veículos em tempo real é crucial não só para mitigar congestionamentos, mas também para promover a segurança viária, a sustentabilidade e a qualidade de vida urbana. Nesse contexto, a tecnologia de Aprendizagem Profunda, uma área da Inteligência Artificial, emergiu como uma ferramenta poderosa para enfrentar esses desafios complexos, especialmente no que diz respeito à detecção de veículos. Além disso, os modelos de aprendizado profundo são conhecidos por sua capacidade de lidar com a variabilidade e complexidade inerentes aos ambientes urbanos, como variações na iluminação, condições meteorológicas e tipos de veículos. Essa capacidade adaptativa e robusta dos modelos de aprendizado profundo os torna particularmente adequados para enfrentar os desafios dinâmicos do rastreamento de veículos em ambientes urbanos em constante mudança. A aprendizagem profunda, frequentemente associada a modelos como o YOLO (*You Only Look Once*), tem sido uma abordagem transformadora na análise de imagens e vídeos. O YOLO é um dos mais proeminentes modelos de detecção de objetos baseados em aprendizado profundo, permitindo a detecção precisa de objetos em imagens e vídeos em tempo real.

Neste trabalho, propõe-se explorar e aprofundar o uso do YOLO como uma ferramenta eficaz na identificação e localização precisa de veículos, como parte integrante de um sistema abrangente de análise de tráfego urbano. Ao combinar as capacidades do YOLO para detecção de objetos específicos, o objetivo desta pesquisa é desenvolver uma abordagem escalável para a detecção, rastreamento e contagem em tempo real de veículos em ambientes urbanos. Para isso, é utilizado um algoritmo de contagem de veículos diferenciando-os entre motocicleta, carro, caminhão e ônibus, utilizando um modelo de detecção baseado em redes neurais convolucionais (CNNs).

Nas seções a seguir, serão apresentados em detalhes os trabalhos relacionados (Seção 2), a metodologia proposta (Seção 3), os resultados alcançados (Seção 4) e as considerações finais (Seção 5) com perspectivas para trabalhos futuros em relação à pesquisa apresentada.

2 TRABALHOS RELACIONADOS

No estudo apresentado por [15], foram explorados os modelos YOLOv3 [13] e *Deep* SORT [22] para realizar a detecção e contagem global de veículos em pesquisas de tráfego em rodovias brasileiras. O principal objetivo foi automatizar pesquisas volumétricas no DNIT de maneira não invasiva e de baixo custo. Os pesquisadores alcançaram uma precisão média de 99, 15% na contagem global de veículos, superando as abordagens anteriores. Os conjuntos de dados GRAM-RTM [3] e CD2014 [20] foram utilizados para avaliar o desempenho do modelo, e os hiperparâmetros ideais, como o YOLO Score e a Associação K (aK), foram determinados por análise gráfica. Esses resultados promissores possibilitam a automação da

contagem de veículos por classe, o que pode trazer benefícios significativos em termos de eficiência na obtenção de informações de tráfego.

Na pesquisa conduzida por [1], foi desenvolvida uma solução para a classificação, contagem e cálculo dos tempos de deslocamento de veículos. A abordagem envolve a utilização de imagens provenientes de câmeras de vigilância já instaladas em áreas de interesse como entrada, empregando técnicas e algoritmos de processamento digital de imagens e reconhecimento de padrões. Essas técnicas permitem a detecção e rastreamento de veículos em interseções urbanas, fazendo uso de recursos e bibliotecas disponíveis na linguagem de programação Python. Neste trabalho, o enfoque principal é o rastreamento de veículos para calcular seus tempos de deslocamento. Utilizou-se a arquitetura YOLOv4, treinada previamente para detectar carros, caminhões e motocicletas, com uma precisão média de 76%. Para aprimorar o rastreamento, definiu-se uma região de deslocamento que leva em consideração distorções de perspectiva, ajustando-se de acordo com a posição da câmera. Para treinar a rede, recorreu-se a imagens da base de dados *Open Images* [7], com 2.000 exemplos de cada classe para treinamento e 500 imagens para testes. As classes incluíram os três veículos que o modelo irá identificar, e todas as imagens foram convertidas para tons de cinza para garantir consistência na análise.

No trabalho apresentado por [8], explorou-se a utilização de técnicas de visão computacional para a contagem e classificação de veículos, bem como o monitoramento de rotas. Dentre as abordagens existentes, destaca-se a aplicação da visão computacional em análise de vídeos de trânsito, com a finalidade de melhorar o planejamento urbano. No contexto específico deste trabalho, é relevante mencionar o uso do sistema *OpenDataCam* [10] e do modelo YOLOv3 para detecção de veículos, bem como a plataforma CUDA (*Compute Unified Device Architecture*) [14] para o processamento acelerado em GPUs. Além disso, a ferramenta *OpenDataCam* desempenha um papel fundamental no rastreio, contagem e geração de dados que alimentam os relatórios e *dashboards* na aplicação desenvolvida, para realizar a análise de imagens de trânsito coletadas por um *drone* do ponto de vista aéreo ortogonal. Os vídeos utilizados para testes foram gravados com durações menores do que 10 minutos com auxílio de um *drone*, posicionado a uma altura média de 95 metros acima de interseções de ruas na cidade de Joinville - SC, e foram carregados para o servidor por meio da interface de *upload* do sistema.

Nesta pesquisa, existem semelhanças em relação aos trabalhos apresentados, porém incorpora contribuições significativas. Primeiramente, utiliza-se a versão mais recente da YOLO, a versão 8, que oferece notáveis avanços em termos de desempenho e precisão. Além disso, para o conjunto de dados, emprega-se o *dataset* AIC HCMC 2020 [4], que inclui quatro classes de veículos distintas: *Motorcycle*, *Car*, *Truck* e *Bus*. Em seguida, realiza-se o treinamento com base nesse *dataset*, aproveitando os pesos pré-definidos que foram originalmente treinados com o *dataset* COCO (*Common Objects in Context*). Isso permitiu aplicar a técnica de *transfer learning* para aprimorar o desempenho do modelo.

Para avaliar o desempenho da solução, empregam-se as métricas de validação mPA50 (*mean Precision Average at* 50) e o mPA50-95 (*mean Precision Average from* 50 *to* 95). O mPA50 leva em conta uma sobreposição de pelo menos 50% entre a detecção e a verdadeira região do objeto para considerar uma detecção como correta, enquanto o mPA50-95 avalia a precisão média para uma faixa de sobreposição de 50% a 95%. Além disso, foram conduzidos testes abrangentes com um conjunto de dados personalizado, criado a partir de um recorte de vídeo disponibilizado na plataforma YouTube, que mostra um trecho da Avenida Torquato Tapajós, nos dois sentidos dessa via, da cidade de Manaus - AM. Essas melhorias refletem o compromisso em aperfeiçoar a aplicabilidade do sistema proposto em comparação aos trabalhos anteriores.

3 METODOLOGIA PROPOSTA

Nessa seção, será apresentada a estrutura geral para desenvolvimento do trabalho. Na Figura 1, apresenta-se o diagrama em blocos do sistema proposto, dividido em duas principais etapas: treinamento e inferência. A etapa de inferência se inicia com a obtenção do modelo de detecção na etapa de treinamento. O sistema proposto é composto por nove blocos. Cada um será descrito, conforme a seguir.

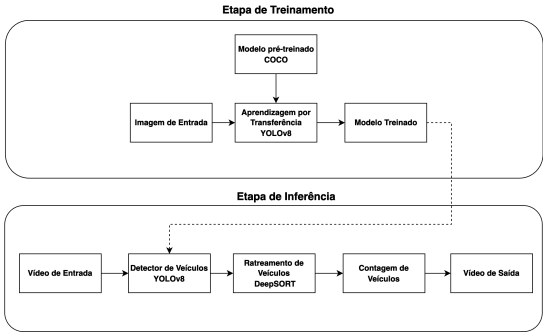

Figura 1: Diagrama em blocos do sistema proposto.

Fonte: Autor, 2023.

3.1 Imagem e Vídeo de Entrada

Nestes blocos, utilizaram-se *datasets* para treinamento e teste do sistema proposto que contivessem cenas com veículos trafegando em ruas e avenidas de duplo sentido, sob um ponto de visão aéreo frontal. Este tipo de visão é característico de câmeras de segurança e monitoramento, em que o objeto de interesse está de frente para a fonte de aquisição, sendo este posicionado a uma altura mais elevada.

Para o conjunto de dados de treinamento, foi utilizado o *dataset* AIC HCMC 2020, que apresenta imagens, em sua maioria, com o mesmo ponto de visão aéreo frontal, onde o sistema será utilizado. Na Figura 2, é possível visualizar amostras de imagens do AIC HCMC 2020.

O referido *dataset* contém um total de 26.195 imagens, organizados em duas subpastas: *images* com as imagens e *labels* com as anotações. A divisão de treino e validação é feita por dois arquivos *txt* com uma divisão, respectivamente, de 85% (22.265 imagens) e 15% (3.930 imagens). No entanto, essa estrutura de arquivos estava preparada para ser utilizada em um treinamento com uma arquitetura YOLOv5. Considerando que nesta pesquisa será utilizado o

Figura 2: Amostras de imagens do *dataset* AIC HCMC 2020.

Fonte: [4].

YOLOv8, tornou-se necessário reorganizar a estrutura de arquivos devido a erros de localização ao utilizar a estrutura original. Assim, dividiu-se o *dataset* em duas pastas, sendo uma de treino e a outra de validação, cada uma contendo as duas subpastas *images* e *labels*. Para aplicar a divisão treino-teste, desenvolveu-se um algoritmo para realizar essa distribuição, reorganizando de forma automática e aleatória. Então, realizou-se uma adaptação na divisão desse conjunto de dados, sendo: 80% das imagens destinadas ao treinamento (20.956 imagens) e 20% ao teste (5.239 imagens).

Uma divisão 80:20 é uma escolha comum, pois aumenta a proporção de dados no conjunto de teste e pode ajudar a avaliar o desempenho do modelo com mais dados, levando a estimativas mais confiáveis do desempenho do modelo na prática [16, 17]. A redução do conjunto de treinamento é uma escolha lógica quando se usa um conjunto de dados grande e não se tem recursos computacionais suficientes para treinar um modelo complexo com todo o conjunto de dados. Além disso, aumentar a proporção de dados no conjunto de teste pode ajudar a evitar o sobreajuste (*overfitting*) do modelo ao conjunto de treinamento [9].

Para o conjunto de dados de teste concebido, selecionou-se um vídeo disponibilizado na plataforma YouTube, que mostra um trecho da Avenida Torquato Tapajós, nos dois sentidos dessa via, da cidade de Manaus - AM, e foi realizado um recorte de vídeo em 10 segundos. Após um processo de conversão do vídeo em *frames*, totalizando-se 327 imagens, foi utilizada a ferramenta LabelImg [5] para rotular manualmente os veículos em cada imagem para realizar a avaliação de desempenho do detector de veículos aplicando o YOLOv8. Na Figura 3, apresenta-se uma amostra de imagem do conjunto de dados de teste concebido.

Figura 3: Amostra de imagem do *dataset* de teste concebido.

Fonte: Autor, 2023.

3.2 Modelo pré-treinado COCO

Modelos pré-treinados são modelos de aprendizado de máquina que foram treinados em grandes conjuntos de dados antes de serem ajustados para tarefas específicas. Eles são amplamente utilizados em tarefas de transferência de aprendizado, onde um modelo treinado em uma tarefa geral é afinado ou usado como ponto de partida para tarefas mais específicas. Nesta pesquisa, foram utilizados os pesos pré-treinados do YOLOv8 disponibilizados pela Ultralytics, que contém diferentes desempenhos para detecção, conforme pode ser visto na Tabela 1. Na metodologia proposta, utilizaram-se pesos treinados no *dataset* COCO que contém mais de 118 mil imagens de treino e 5 mil imagens de teste, 80 classes de objetos e treinado por 100 épocas. Essas classes incluem uma variedade de objetos comuns encontrados em contextos cotidianos, tais como pessoas, animais, móveis, eletrodomésticos, entre outros.

Tabela 1: Desempenho dos pesos treinados para detecção com COCO.

Model	*Size (pixels)*	mAP (50-95)	*Speed* CPU ONNX (ms)
YOLOv8n	640	37,3	80,4
YOLOv8s	640	44,9	128,4
YOLOv8m	640	50,2	234,7
YOLOv8l	640	52,9	375,2
YOLOv8x	640	53,9	479,1

Fonte: [19].

3.3 Aprendizado por Transferência

De acordo com o [11], o processo que envolve retreinar um modelo já treinado previamente com um *dataset* maior e mais variado, no caso o *dataset* COCO, com um conjunto de dados específico, como o AIC HCMC 2020, permite ao YOLOv8 ajustar os seus pesos ao problema específico deste trabalho. Nesta pesquisa, foi realizado um treinamento de apenas 10 épocas, devido às limitações de recursos de *hardware* disponíveis. Nesse cenário e conforme Tabela 1, a opção pelos pesos pré-treinados *YOLOv8l.pt* se mostra apropriada, uma vez que essa variante apresenta um mAP acima de 50%, conforme dados da tabela 1. Embora esses pesos geralmente exijam treinamentos mais longos e recursos de *hardware* mais robustos, a escolha se justifica quando o objetivo é obter resultados superiores em um treinamento de curta duração, como o proposto com apenas 10 épocas.

3.4 Rastreamento de Veículos DeepSORT e Contagem de Veículos

Nestes blocos, serão abordados o processo de rastreamento de veículos utilizando o *Deep* SORT, incluindo desde o processamento das informações das caixas delimitadoras, nomes das classes, IDs até as direções dos veículos. Além disso, será apresentado o algoritmo da contagem de veículos e como essas informações são visualizadas no vídeo de saída.

O código original do *Deep* SORT foi desenvolvido no *framework* TensorFlow, dependendo da arquitetura *Faster* R-CNN para seu funcionamento. Dado que, nesta pesquisa foi escolhido o YOLOv8 em PyTorch, necessitou-se de uma versão do *Deep* SORT compatível com essa configuração. Nesse sentido, foi utilizada uma implementação do *Deep* SORT em PyTorch disponibilizada no GitHub por

[12], em que emprega-se o YOLOv3 com o objetivo de detecção e rastreamento de pessoas. Foi necessário realizar a mudança para a versão do detector utilizada nesse trabalho, além de mudar as classes de interesse.

No início da implementação, a etapa de inicialização é executada por meio da chamada da função *init_tracker()*. Essa ação tem o propósito de configurar o rastreador DeepSORT. Durante a inicialização, o sistema carrega as configurações específicas do rastreador a partir do arquivo *deep_sort.yaml* e, em seguida, inicia o modelo DeepSORT de acordo com as configurações definidas.

O rastreamento de veículos ocorre na função *draw_boxes()*. Nessa etapa, o sistema processa as informações essenciais, que incluem:

- As coordenadas das caixas delimitadoras dos objetos detectados, no formato (x_1, y_1, x_2, y_2), conhecidas por *bbox_xyxy*;
- Uma lista contendo os nomes das classes dos objetos identificados, denominada *names*;
- Uma lista de IDs exclusivos atribuídos a cada objeto detectado, referida como *object_id*; e
- Uma lista de identidades rastreadas para os objetos, indicada como *identities*.

O algoritmo *Deep* SORT utiliza as coordenadas das caixas delimitadoras e as informações de confiança associadas a cada detecção para rastrear os objetos em movimento. Ele atribui um ID exclusivo a cada objeto detectado e mantém o rastreamento à medida que os objetos se deslocam pelo vídeo.

Para cada objeto detectado, o sistema realiza as seguintes ações:

- Calcula o centro da parte inferior da caixa delimitadora;
- Atualiza uma fila de pontos de rastreamento com as posições recentes do objeto; e
- Determina a direção do objeto em relação a uma linha de referência definida. Na implementação, essa linha é especificada pelas coordenadas referentes a *line*.

Se um objeto cruzar a linha de referência definida na imagem (*line*), o sistema identifica a direção do cruzamento na direção em que se desloca na cena (norte, sul, leste e oeste). Com base nessa direção, as contagens de objetos são atualizadas nas variáveis *object_counter* e *object_counter1*. A variável *object_counter* é usada para registrar objetos que cruzam a linha de um lado, enquanto que *object_counter1* faz o mesmo para objetos que cruzam do outro lado da linha. Posteriormente, o sistema exibe a caixa delimitadora do objeto, juntamente com sua direção no vídeo. Além disso, ele registra uma trilha que mostra o histórico de movimento do objeto. Por fim, as contagens de objetos por classe são exibidas na imagem em duas áreas separadas, uma para cada direção, conforme pode ser visto na Figura 4.

4 RESULTADOS E DISCUSSÃO

Os resultados deste trabalho são apresentados em quatro seções distintas, cada uma desempenhando um papel fundamental na avaliação da metodologia proposta. Na Seção 4.1, é apresentado o ambiente de desenvolvimento e a implementação, fornecendo contexto sobre os recursos e ferramentas utilizados. Na Seção 4.2, descreve-se o processo de treinamento da YOLO, destacando as estratégias e os hiperparâmetros empregados. Por fim, nas Seções 4.3 e 4.4, apresentam-se os testes realizados com a base de dados de teste

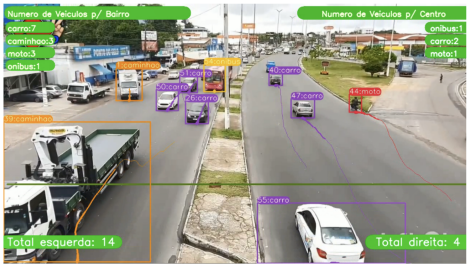

Figura 4: Amostra de imagem do vídeo de saída do sistema proposto.

Fonte: Autor, 2023.

concebida, analisando os resultados da contagem de veículos, proporcionando uma visão completa do desempenho do sistema em cenários práticos.

4.1 *Setup*

Este estudo foi conduzido utilizando o ambiente de desenvolvimento Google Colab com acesso à GPU Testa T4 de 16GB na versão gratuita. A análise e processamento de dados foram realizados por meio desta plataforma, permitindo a execução eficiente de tarefas que requerem intensivo poder de processamento, como o treinamento da YOLO e análise de *frames* de vídeos. Além disso, a integração com o Google Drive garantiu o armazenamento seguro dos dados gerados durante as compilações.

A linguagem de programação utilizada foi Python com o interpretador na versão 3.10.12. Também foi usado o *framework* Ultralytics, na versão 8.0.3, que desempenhou um papel de integração e operação eficiente do sistema na detecção de objetos YOLOv8. Na Tabela 2, apresenta-se a lista das bibliotecas requeridas no projeto.

Tabela 2: Bibliotecas requisitadas no projeto.

Biblioteca	Versão
matplotlib	3.2.2
numpy	1.18.5
opencv-python	4.1.1
Pillow	7.1.2
PyYAML	5.3.1
scipy	1.4.1
Torch	1.7.0
TorchVision	0.8.1
tqdm	4.64.0
requests	2.23.0
tensorboard	2.4.1
pandas	1.1.4
seaborn	0.11.0

Fonte: Autor, 2023.

4.2 Treinamento do modelo de detecção

Conforme apresentado anteriormente, foram utilizados os pesos pré-treinados *YOLOv8l.pt* como parte do processo de *transfer learning* e foi continuado o treinamento com uma nova base de dados (AIC HCMC 2020) que inclui as seguintes classes de veículos: moto, carro, ônibus e caminhão. Essa abordagem é eficaz, pois permite que o modelo pré-treinado aprenda características gerais em um grande conjunto de dados e, em seguida, ajuste essas características para as classes específicas de veículos em uma nova base de dados. Isso economiza tempo e recursos, tornando o treinamento mais eficiente.

As únicas alterações nos parâmetros de treinamento foram as seguintes: definiu-se o número de épocas (*epochs*) como 10, o tamanho do lote (*batch*) como 16 e o tamanho das imagens (*imgsz*) como 640. Todos os outros parâmetros permaneceram os padrões fornecidos pela Ultralytics. A escolha da quantidade de épocas e o tamanho do lote foram realizadas considerando o tempo de treinamento, uma vez que a nova base de dados é relativamente grande e também para aproveitar os recursos da GPU fornecidos pelo Google. Durante o treinamento, cada época teve uma duração média de 19 minutos e utilizou cerca de 11 gigabytes de recursos da GPU, conforme pode ser visto na Figura 5 que mostra a primeira e última época. O tamanho das imagens como 640 é o padrão utilizado pelo YOLOv8 [19]. Manteve-se esse padrão para equilibrar a qualidade e a eficiência do treinamento, pois imagens maiores contêm mais detalhes, porém requerem mais recursos computacionais, o que pode tornar o treinamento mais lento, necessitando, assim, de *hardware* mais poderoso. Por outro lado, imagens muito pequenas podem perder informações essenciais. Esse tamanho foi selecionado para manter um equilíbrio adequado da qualidade das previsões.

Figura 5: *Logs* do treinamento do modelo.

Fonte: Autor, 2023.

Com o treinamento concluído, obtiveram-se os pesos salvos no drive, juntamente com a matriz de confusão e os gráficos resultantes da análise dos *logs* gerados durante o treinamento (ver Figura 6).

Esses *logs* do treinamento fornecem uma visão detalhada do desempenho do modelo de detecção de veículos, destacando métricas como *Precision*, que mede a precisão das caixas delimitadoras usadas para identificar objetos; *Recall*, que avalia a capacidade do modelo em encontrar objetos reais; mAP50, que calcula a média da precisão para todas as classes a um nível de confiança de 50%, oferecendo uma visão global do desempenho; e mAP50-95, métrica que considera um intervalo de confiança de 50% a 95%, proporcionando uma análise das detecções com maior confiança.

Na Figura 7, é possível visualizar o desempenho geral do modelo a partir do gráfico de *Precision-Recall*, com curvas mais próximas

Class	Images	Instances	Precision	ReCall	mAP50	mAP50-95
all	5239	49996	0.892	0.89	0.937	0.667
moto	5239	35644	0.92	0.881	0.951	0.591
carro	5239	8670	0.898	0.906	0.947	0.69
onibus	5239	1674	0.884	0.876	0.921	0.703
caminhao	5239	4008	0.868	0.895	0.928	0.685

Figura 6: *Logs* de validação do modelo treinado.

Fonte: Autor, 2023.

de 1 (i.e., 100%) indicando resultados satisfatórios, que ressalta a consistência do modelo na detecção dos objetos.

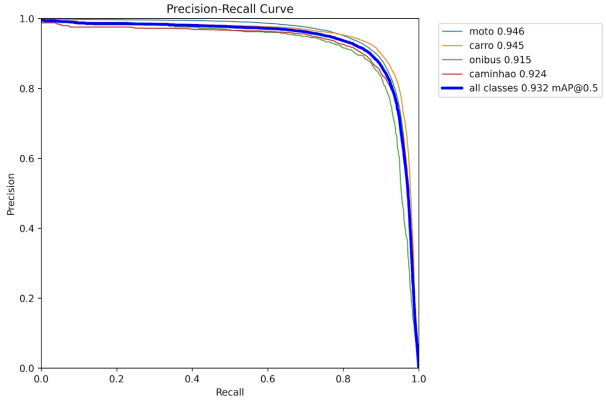

Figura 7: Gráfico *Precision-Recall* do modelo de detecção de veículos.

Fonte: Autor, 2023.

A rótulo *all* engloba todas as classes, e conforme resultados apresentados na Figura 6, verifica-se que modelo possui uma precisão de 0, 892, indicando que a maioria das detecções está correta. O *Recall* de 0, 890 significa que a maioria dos objetos reais foi detectada; o mAP50 de 0, 937, indica um bom desempenho global na detecção de objetos com uma confiança de 50%; e o mAP50-95 de 0, 667, pode indicar alguma variação na confiança das detecções.

Realizando uma análise do volume de dados por classe no *dataset* AIC HCMC 2020, é possível observar, conforme Figura 8, que o comportamento desses resultados varia de acordo com a quantidade de dados, por esse *dataset* estar desbalanceado entre as classes. Tem-se a classe *moto* com o melhor desempenho por possuir a maior quantidade de amostras. As demais classes possuem menos amostras, mas mesmo assim apresentam um desempenho aparentemente bom em termos de precisão, uma vez que há menos exemplos negativos para gerar falsos positivos. Portanto, torna-se necessário realizar uma análise da matriz confusão do modelo que foi gerado durante o treinamento. Isso permitirá visualizar o número de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos para cada classe.

A matriz de confusão fornece informações valiosas sobre o desempenho do modelo na tarefa de classificação. Cada linha da matriz

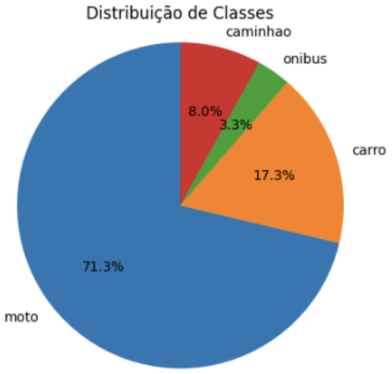

Figura 8: Distribuição de classes no *dataset* AIC HCMC 2020.

Fonte: Autor, 2023.

representa a classe real, e cada coluna representa a classe prevista. Os valores nas células indicam a proporção de instâncias classificadas corretamente (verdadeiros positivos) e incorretamente (falsos positivos) em relação à classe real [2]. Ao analisar a matriz confusão do modelo gerada na Figura 9, percebe-se que a classe *background* representa a classe que não foi definida explicitamente durante o treinamento, e o modelo a identificou como uma categoria separada nas predições. Isso é comum em sistemas de detecção de objetos, onde o *background* representa regiões da imagem que não contém nenhum objeto de interesse.

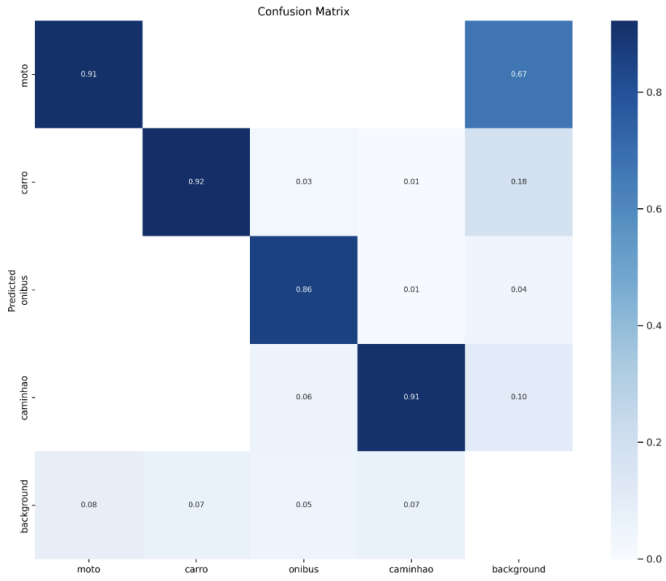

Figura 9: Matriz de confusão do modelo de detecção de veículos.

Fonte: Autor, 2023.

Além disso, para a classe *moto*, 91% foram classificadas corretamente como moto, o que representa uma taxa alta de verdadeiros positivos. Não houve classificações equivocadas de motos como carro, ônibus ou caminhão, resultando em um percentual de falsos positivos de 0% para essas classes. No entanto, 67% das motos foram incorretamente identificadas como *background*, sugerindo que o modelo tende a atribuir a classe *background* a regiões vazias da imagem, e isso pode estar relacionado ao desbalanceamento de instâncias da classe moto em comparação com outras classes. Esse desbalanceamento pode levar o modelo a gerar mais falsos positivos para *background* ao encontrar áreas na imagem que não correspondem a nenhuma classe específica, uma vez que o modelo tenta ajustar a desigualdade entre as classes.

Em resumo, a análise realizada destaca o desempenho do modelo no treinamento, tanto os acertos como os erros, e aponta a importância de considerar o balanceamento de classes na otimização do modelo, bem como ajustes para minimizar confusões entre as diferentes categorias. Destaca-se a técnica proposta por [6], conhecida como *Focal Loss*, como uma abordagem eficaz para lidar com o desbalanceamento de classes em tarefas de detecção de objetos. Embora esta técnica não tenha sido aplicada neste estudo, sua introdução, neste contexto, revela a importância a ser dada aos exemplos de treinamento difíceis, reduzindo o impacto das regiões *background* abundantes, o que pode aprimorar o desempenho geral do modelo.

4.3 Teste com a base concebida

Nesta seção, será realizada a análise do desempenho do modelo treinado, utilizando um conjunto de testes próprio. Isto é, será avaliado como o modelo se comporta ao lidar com novos dados, fornecendo novas observações sobre sua capacidade de generalização.

O *dataset* de teste foi concebido a partir de um recorte de 10 segundos de um vídeo do YouTube e as anotações foram geradas manualmente usando a ferramenta LabelImg. Conforme ilustrado na Figura 10, os critérios adotados para realizar as anotações em 327 *frames* do vídeo são descritos a seguir:

- Definiu-se uma área de detecção com limites representados por linhas amarelas. Esses limites foram estabelecidos realizando testes em vídeos com o modelo, percebendo que objetos muito distantes, como no caso do vídeo de teste, são muito pequenos e não ocorrem detecções, enquanto que os objetos que vão saindo da cena, são cortados, gerando falsas detecções;
- A prioridade de anotação foi dada aos veículos na via, nas direções norte e sul, enquanto que objetos em outras direções, como carros estacionados, foram excluídos; e
- Objetos que foram cortados, ocultando a maioria de suas características, não receberam rótulos. Por exemplo, a Kombi no lado esquerdo estava ultrapassando o limite inferior estabelecido, e o que sobrava da sua detecção poderia causar confusão na classificação.

Ao analisar os dados das anotações manuais e automáticas geradas pelo modelo, observa-se que a maioria das classes possui volumes de dados semelhantes. No entanto, há uma diferença notável na classe *onibus*, que apresenta quase o dobro de detecções nos dados gerados automaticamente pelo modelo. Isso sugere a possibilidade de ocorrência de falsos positivos ou falsos negativos nas detecções dessa classe em específico.

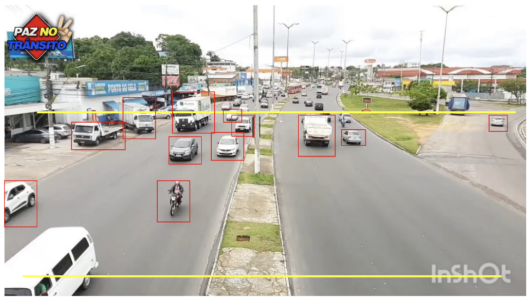

Figura 10: *Frame* do vídeo de 10 segundos utilizado no *dataset* de teste.

Fonte: Autor, 2023.

Os resultados detalhados na etapa de inferência podem ser vistos na Figura 11 a seguir:

Class	Images	Instances	Precision	ReCall	mAP50	mAP50-95
all	327	3900	0.812	0.8	0.875	0.544
moto	327	521	0.961	0.757	0.832	0.403
carro	327	2368	0.912	0.735	0.907	0.606
onibus	327	106	0.464	0.978	0.845	0.632
caminhao	327	905	0.91	0.728	0.918	0.534

Figura 11: *Logs* gerados no *dataset* de teste com o modelo de detecção de veículos.

Fonte: Autor, 2023.

Analisando a classe *all* que contém a média geral das classes, verifique-se que a precisão é de 0, 812, significando que a maioria das detecções é precisa. O *recall* em 0, 800 indica que a maioria dos objetos detectados é verdadeiro. A métrica mPA50 ficou em 0, 875, representando a média da precisão com uma sobreposição mínima de 50% entre caixas delimitadoras verdadeiras e preditas. O mAP50-95 é de 0, 544, indicando o desempenho em uma faixa mais ampla de sobreposições.

Em resumo, os resultados indicam um desempenho geral satisfatório do modelo na detecção de objetos, com maior destaque para a classe *moto*, em que a precisão é notavelmente maior. No entanto, há variações na capacidade de detectar objetos reais em algumas classes, como em *onibus*. Além disso, as métricas mAP50 e mAP50-95 analisadas fornecem uma avaliação abrangente do desempenho em diferentes níveis de sobreposição entre as caixas delimitadoras verdadeiras e preditas.

4.4 Resultados da contagem de veículos

Na Tabela 3, apresentam-se os resultados obtidos durante a experimentação do algoritmo de contagem de veículos, quando utilizado o *dataset* de teste. Conforme apresentado anteriormente, essa base foi criada pelo autor e consiste em um período de 10 segundos de vídeo. Durante esse tempo, um total de 24 veículos foram registrados, dos quais 14 são carros, 3 são caminhões, 5 são motocicletas e 2 são ônibus. Esses números fornecem uma visão geral do conteúdo e da composição da base de dados utilizada para avaliar o desempenho da contagem de veículos.

Tabela 3: Resultados da contagem no recorte de vídeo.

Base de Dados	Tempo	Contagem total	Contagem por classe			
			Carro	Caminhão	Motocicleta	Ônibus
Teste (Autor)	10s	24	14	3	5	2

Fonte: Autor, 2023.

A análise do experimento foi realizada durante os 10 segundos do vídeo, revelando diversos pontos de interesse. Primeiramente, a contagem de veículos inicia apenas após sua passagem pela linha verde na imagem. Observou-se que a contagem foi iniciada com a passagem de dois veículos do lado esquerdo, indicando que o algoritmo rastreador estava ativo e funcionando. No entanto, à medida que os objetos se distanciavam em direção ao norte, ou seja, mais ao fundo da cena, o algoritmo perdia a capacidade de detecção. Isso se deve ao fato de os objetos perdem a maioria de suas características, conforme vão ficando menores na cena.

Além disso, foram identificados erros no processo de classificação dos objetos. O algoritmo equivocadamente classificou um objeto da classe *carro* como *caminhão*. Outras situações de erro de classificação foram observadas na experimentação, resultando em falsos positivos, o que não impactou na contagem total de veículos, mas afetou a contagem por classe.

Em relação ao rastreamento e à atribuição de IDs a cada detecção realizados pelo *Deep* SORT, observaram-se situações notáveis. Em certos momentos, o algoritmo retificou erros anteriores de classificação para uma detecção específica, inicialmente rotulada como carro e posteriormente corrigida para caminhão. Essa correção ocorreu à medida que mais partes do objeto se tornaram visíveis quando ele se aproximava da câmera, vindo do fundo da cena.

No entanto, também foram identificados erros de classificação. O algoritmo confundiu o objeto que já havia sido registrado com um ID e já havia deixado a cena, com um objeto que ainda estava na cena, porém não havia sido detectado devido à oclusão. Além disso, houve erros de classificação devido a uma situação na qual apenas uma parte lateral do carro estava visível na cena, enquanto o restante estava parcialmente cortado. O detector classificou erroneamente essa porção visível como uma moto, e o rastreador atribuiu um ID mantendo essa classificação à medida que o objeto se movia pela cena. Essa situação destaca a importância de melhorar o modelo de detecção, a fim de melhor lidar com detecções desafiadoras e situações onde apenas partes dos objetos estão visíveis.

A classe *ônibus* resultou em duas contagens incorretas devido ao detector confundir uma van e um carro com essa classe, já que apenas esses dois tipos de veículos atravessaram a linha que marca a contagem. Essa confusão indica que a precisão da detecção para essa classe está resultando em um número significativo de falsos positivos.

No entanto, no que diz respeito à contagem geral de veículos que cruzam a linha e contribuem para o total, os resultados foram precisos, quando analisados visualmente. Vale ressaltar que a regra para efetuar a contagem é que a parte inferior das caixas delimitadoras dos objetos detectados deve cruzar a linha, permitindo, assim, a

coleta de informações sobre a contagem total, a direção dos objetos e a contagem por classe. Na Figura 12 a seguir, apresenta-se o último *frame* do vídeo e as informações exibidas diretamente na cena.

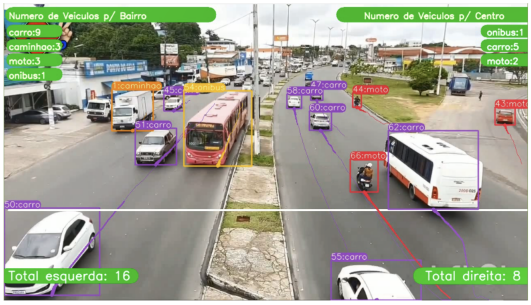

Figura 12: Resultados da contagem de veículos por via.

Fonte: Autor, 2023.

5 CONCLUSÃO

Neste trabalho, foi apresentada uma metodologia para desenvolver uma abordagem escalável para a detecção, rastreamento e contagem de veículos em ambientes urbanos. Para isso, foram empregados algoritmos que consistem na detecção, rastreamento e contagem de veículos por linha virtual, diferenciando-os entre motocicleta, carro, caminhão e ônibus. As métricas de precisão, *recall* e mAP foram utilizadas para a avaliação do modelo de detecção de veículos apresentado.

As análises dos resultados da contagem enfatizam a complexidade inerente ao processo de rastreamento e classificação de objetos em tempo real. Essa complexidade se manifestou de forma notável na classe ônibus, que apresentou uma precisão relativamente baixa, resultando em um número significativo de falsos positivos. Essa questão ressalta a necessidade de aprimoramentos no modelo de classificação, com foco especial na classe ônibus.

É importante observar que esse desafio de classificação está relacionado ao desbalanceamento das classes no conjunto de dados utilizado no treinamento, que se baseou no *dataset* AIC HCMC 2020. A classe ônibus é particularmente afetada devido à sua representação limitada nesse conjunto de dados. Esse desbalanceamento pode prejudicar a capacidade do modelo de aprender efetivamente as características da classe ônibus e, consequentemente, resultando em uma precisão reduzida.

Considerando os resultados obtidos, é possível apresentar propostas para a continuação deste trabalho como, por exemplo: empregar técnicas que visem aprimorar o desempenho dos algoritmos, principalmente na etapa de detecção e classificação dos veículos por classe, através do balanceamento do *dataset* utilizado; realizar mais testes de validação em cenários diversificados; e melhorar a interface onde os resultados são apresentados, podendo, assim, fornecer informações valiosas para o controle de tráfego urbano.

AGRADECIMENTOS

Este artigo é resultado do projeto de pesquisa e desenvolvimento ARANOUÁ financiado pela Samsung Eletrônica da Amazônia Ltda nos termos da Lei Federal nº 8.387/1991, de acordo com o art. 21 do Decreto nº 10.521/2020. Agradecemos, também, ao Campus Manaus Zona Leste do Instituto Federal do Amazonas (IFAM) pelo suporte e incentivos para a realização deste trabalho.

REFERÊNCIAS

[1] Larissa Barbado, Lucas Medeiros Reinaldet dos Santos, Miguel Diogenes Matrakas, and Jasmine Moreira. 2022. Aplicação da rede convolucional Yolo para análise de fluxo de veículos. In *Anais do XIX Congresso Latino-Americano de Software Livre e Tecnologias Abertas*. SBC, 43–49.

[2] G A Domene. 2020. Abordagem Computacional para Detecção Automatizada por Imagem do Uso de Cinto de Segurança em Condutores baseado em Redes Neurais Convolucionais. , 23 pages.

[3] Ricardo Guerrero-Gómez-Olmedo, Roberto J López-Sastre, Saturnino Maldonado-Bascón, and Antonio Fernández-Caballero. 2013. Vehicle tracking by simultaneous detection and viewpoint estimation. In *5th International Work-Conference on the Interplay Between Natural and Artificial Computation, (IWINAC)*. Springer, 306–316.

[4] Nguyen Tien Hunng. 2021. Vehicle Counting AIC HCMC 2020. Disponível em: https://www.kaggle.com/datasets/hungkhoi/vehicle-counting-aic-hcmc-2020/data. Acesso em: 24 de outubro de 2023.

[5] T. Lin. 2015. LabelImg. https://github.com/tzutalin/labelImg.

[6] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the International Conference on Computer Vision*. 2980–2988.

[7] Google LLC. 2018. Overview of Open Images V4. Disponível em: https://storage.googleapis.com/openimages/web/factsfigures_v4.html. Acesso em: 16 de outubro de 2023.

[8] André Vitor Maus, Tatiele Martins Razera, Roney Kuntz, Valdir Linhares Junior, and Ademir Camillo Junior. 2021. Contagem e Classificação de Veículos por Visão Computacional. *Anais do Computer on the Beach* 12 (2021), 049–056.

[9] Anna Milani, Fábio S da Silva, Elloá B Guedes, and Ricardo Rios. 2023. A Deep Learning Application for Psoriasis Detection. In *Anais do XX Encontro Nacional de Inteligência Artificial e Computacional*. SBC, 315–329.

[10] OpenDataCam. 2021. OpenDataCam: An Open Source Tool to Collect and Visualize Traffic Data. Disponível em: https://github.com/opendatacam/opendatacam. Acesso em: 24 de outubro de 2023.

[11] T Pedro, A Oliveira, J Cintra, and N Garcia. 2023. Aprendizagem Automática para Detecção de Embarcações em Sistemas Radar. In *Anais do Décimo Quarto Simpósio de Informática (INForum 2023)*. 309–313.

[12] Z Pei. 2021. Deep SORT with PyTorch. Disponível em: https://github.com/ZQPei/deep_sort_pytorch. Acesso em: 20 de outubro de 2023.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*. 779–788.

[14] Shane Ryoo, Christopher I Rodrigues, Sara S Baghsorkhi, Sam S Stone, David B Kirk, and Wen-mei W Hwu. 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th Symposium on Principles and Practice of Parallel Programming*. 73–82.

[15] Adson M Santos, Carmelo JA Bastos-Filho, Alexandre MA Maciel, and Estanislau Lima. 2020. Counting vehicle with high-precision in brazilian roads using yolov3 and deep sort. In *33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 69–76.

[16] Hellen Geremias dos Santos, Carla Ferreira do Nascimento, Rafael Izbicki, Yeda Aparecida de Oliveira Duarte, Porto Chiavegatto Filho, and Alexandre Dias. 2019. Machine learning para análises preditivas em saúde: exemplo de aplicação para predizer óbito em idosos de São Paulo, Brasil. *Cadernos de Saúde Pública* 35 (2019), e00050818.

[17] Matheus Henrique Santos et al. 2023. Classificação do perfil dos clientes através de técnicas de mineração de dados. (2023).

[18] Sindipeças. 2021. Relatório da Frota Circulante. Disponível em: https://www.sindipecas.org.br/sindinews/Economia/2021/RelatorioFrotaCirculante_Marco_2021.pdf. Acesso em: 16 de outubro de 2023.

[19] Ultralytics. 2021. YOLO: A Brief History. Disponível em: https://docs.ultralytics.com/. Acesso em: 11 de outubro de 2023.

[20] Yi Wang, Pierre-Marc Jodoin, Fatih Porikli, Janusz Konrad, Yannick Benezeth, and Prakash Ishwar. 2014. CDnet 2014: An expanded change detection benchmark dataset. In *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops*. 387–394.

[21] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. 2019. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117* (2019).

[22] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 International Conference on Image Processing (ICIP)*. IEEE, 3645–3649.