ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Algoritmo Otimizado para Deteção de Passagem de Sinal Vermelho para Implementações em Sistemas Embebidos

TIAGO AGOSTINHO DA SILVA

(Licenciado)

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e Multimédia

Orientadores: Doutor Tiago Miguel Braga da Silva Dias

Doutor Pedro Miguel Torres Mendes Jorge

Júri:

Presidente: Doutor Pedro Emanuel Albuquerque e Baptista dos Santos

Vogais: Doutor Mário Pereira Véstias

Doutor Tiago Miguel Braga da Silva Dias

Novembro 2024

Algoritmo Otimizado para Deteção de Passagem de Sinal Vermelho para Implementações em Sistemas Embebidos

TIAGO AGOSTINHO DA SILVA

(Licenciado)

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e Multimédia

Orientadores: Doutor Tiago Miguel Braga da Silva Dias, DEETC/ISEL
Doutor Pedro Miguel Torres Mendes Jorge, DEETC/ISEL

Júri:

Presidente: Doutor Pedro Emanuel Albuquerque e Baptista dos Santos, DEETC/ISEL
Vogais: Doutor Mário Pereira Véstias, DEETC/ISEL
Doutor Tiago Miguel Braga da Silva Dias, DEETC/ISEL

Novembro 2024

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus orientadores, professores Tiago Dias e Pedro Jorge, não só por me terem dado a oportunidade de realizar esta dissertação, mas também por toda a atenção que me dedicaram ao longo dos últimos meses, encontrando sempre tempo na agenda para me ajudar e reunir comigo, e sobretudo, pela simpatia com que sempre me trataram.

Agradeço à escola, Instituto Superior de Engenharia de Lisboa, pelos anos que lá passei, que, embora tenham sido uma grande luta e desafio, foram, acima de tudo, um período de crescimento pessoal.

Expresso a minha maior gratidão à minha namorada por ter estado sempre presente e ao meu lado, especialmente nos momentos mais difíceis.

Um agradecimento à minha mãe, por toda a educação e valores que sempre me transmitiu, e ao meu pai, pelo apoio e oportunidade de completar este curso.

Agradeço também à Autoridade Nacional de Segurança Rodoviária (ANSR) pelo apoio financeiro na realização deste trabalho através de fundos do projeto de investigação e desenvolvimento (I&D) SINCRO, contrato ANSR CE 19/2022.

Declaração de integridade

Declaro que esta dissertação é o resultado da minha investigação pessoal e independente. O seu conteúdo é original e todas as fontes listadas nas referências bibliográficas foram consultadas e estão devidamente mencionadas no texto. Mais declaro que todas as referências científicas e técnicas relevantes para o desenvolvimento do trabalho estão devidamente citadas e constam das referências bibliográficas.

O autor

Assinado por: **Tiago Agostinho da Silva**
Num. de Identificação: 15146060
Data: 2024.12.23 23:41:24+00'00'

Lisboa, 23 de dezembro de 2024

Resumo

O problema da passagem de semáforo vermelho (*Red Light Running*, RLR) constitui uma das principais causas de acidentes rodoviários, com consequências humanas e económicas significativas. Apesar das várias medidas implementadas ao longo dos anos, a sua frequência permanece elevada, impactando negativamente a segurança nas estradas.

Este trabalho explora técnicas atuais de Inteligência Artificial para o desenvolvimento e implementação de um algoritmo otimizado para deteção de RLR, orientado a implementações em sistemas embebidos de gama média. A solução proposta é baseada em análise de vídeo e utiliza regiões de interesse para delimitar áreas de possível violação RLR e a posição do semáforo, permitindo a classificação da sua cor. O algoritmo disponibiliza dois métodos de classificação da cor dos semáforos, sendo um baseado numa rede neuronal convolucional e o outro na análise de características específicas das imagens. Além disso, faz uso do modelo de deteção de objetos YOLOv8, em conjunto com o modelo de seguimento de objetos ByteTrack, para analisar a trajetória dos veículos. Estes modelos são eficientes e eficazes para um possível processamento local e em tempo real.

O sistema foi avaliado utilizando vídeos reais e implementado num sistema embebido, nomeadamente na plataforma NVIDIA Jetson Orin Nano, simulando um cenário real de operação local em tempo real. Os resultados demonstram a eficácia do algoritmo, tanto na precisão da deteção de RLR quanto no rápido processamento. Ao utilizar a plataforma NVIDIA Jetson Orin Nano, obteve-se um processamento em tempo real de até 23 imagens por segundo, tirando partido do processador gráfico (GPU) e modelos otimizados de TensorRT. Este sistema oferece uma ferramenta promissora para a deteção e prevenção de RLR, com grande potencial de evolução e implementação em cenários reais, contribuindo significativamente para a segurança rodoviária.

Palavras-chave: Passagem de sinal vermelho, Visão computacional, YOLO, Classificador de semáforo, Sistemas embebidos, NVIDIA Jetson Orin Nano

Abstract

The issue of *Red Light Running* (RLR) is one of the main causes of road traffic accidents, resulting in significant human and economic consequences. Despite various measures implemented over the years, the frequency of this violation remains high, negatively affecting road safety.

This dissertation explores current Artificial Intelligence techniques for developing and implementing an optimized RLR detection algorithm, aimed at deployment in mid-range embedded systems. The proposed solution is based on video analysis, employing regions of interest to delineate areas of potential RLR violations and traffic light positioning, allowing its color classification. The algorithm provides two methods for traffic light color classification, one based on a *Convolutional Neural Network* and the other on specific visual image feature analysis. In addition, it uses the YOLOv8 object detection model, combined with the ByteTrack object tracking model, to analyze vehicle trajectories. These models are both efficient and effective, enabling potential real-time local processing.

The system was evaluated using real-world videos and implemented on an embedded system, namely the NVIDIA Jetson Orin Nano platform, simulating a real-world scenario with local real-time operation. The results demonstrate the algorithm's effectiveness, both in the accuracy of RLR violation detection and fast processing times. On the NVIDIA Jetson Orin Nano platform, real-time processing at up to 23 fps was achieved, by exploiting the Graphics Processing Unit (GPU) and TensorRT optimized models. This system offers a promising tool for detecting and preventing RLR, with significant potential for further development and real-world implementation, contributing greatly to road safety.

Keywords: Red light running, Computer vision, YOLO, Traffic light classifier, Embedded systems, NVIDIA Jetson Orin Nano

Índice

Índice de figuras

Índice de tabelas

Lista de siglas e abreviaturas

AA	Aprendizagem Automática xxii, xxiii, 7, 18, 41
AP	Aprendizagem Profunda xxii, xxiii, 14
CNN	*Convolutional Neural Network* xi, xvii, 7, 8, 9, 18, 29, 30, 35, 43, 48, 54, 57, 58, 72, 73, 74
COCO	*Common Objects in Context* xvi, 24, 31, 45, 46, 47, 51, 74
CPU	*Central Processing Unit* xxi, 55, 64, 65, 66, 67, 68, 74
EUA	Estados Unidos da América xv, 42, 43
FCN	*Fully Convolutional Network* 9
FPN	*Feature Pyramid Networks* 11
fps	*Frames Per Second* 5, 16, 41, 65, 66, 67, 68, 69, 70, 71
GELAN	*Generalized Efficient Layer Aggregation Network* 11
GMC	*Global Motion Compensation* 13, 14
GPU	Graphics Processing Unit ix, xi, xxi, 16, 33, 34, 37, 55, 58, 64, 65, 67, 68, 72, 74
HSV	*Hue Saturation Value* 28, 38
IA	Inteligência Artificial ix, xvii, 7, 8, 16, 17, 18, 41
ID	Identificador 40
IDE	*Integrated Development Environment* 37
IoU	*Intersection-over-Union* 13, 40
mAP	*mean Average Precision* 12, 24, 40, 50
MB	*megabyte* 41, 56, 58
MOT	*Multiple Object Tracking* 12
ms	milissegundos 41, 56, 57, 58, 65, 66, 67, 68, 69, 70, 71, 72
NMS	*Non-Maximum Suppression* 12

OMS Organização Mundial da Saúde 2

PGI *Programmable Gradient Information* 11

RF *Random Forest* 7
RLC *Red Light Camera* xv, 3, 4, 15, 75
RLR *Red Light Running* ix, xi, xv, xvi, xviii, 2, 3, 4, 5, 7, 12, 15, 18, 19, 21, 22, 23, 24, 25, 26, 27, 31, 32, 34, 37, 39, 40, 41, 42, 43, 50, 53, 59, 62, 64, 70, 71, 72, 73, 74, 75, 93
ROI *Region of Interest* xv, 9, 21, 22, 23, 24, 27, 31, 32, 34, 59, 60, 70, 73
RPN *Region Proposal Network* 9

s segundos 41, 65, 66, 67, 68, 69, 70, 71
SDK *Software Development Kit* xxiii, 33, 38, 74
SORT *Simple Online and Realtime Tracking* 13, 14
SOT *Single Object Tracking* 12
SSD *Single Shot Multibox Detector* 9
SVM *Support Vector Machine* 7

TOPS *Tera Operations Per Second* 16
TPR *True Positive Rate* 40

YOLO *You Only Look Once* 9, 10, 12, 16, 18, 19, 21, 22, 24, 26, 27, 31, 34, 38, 40, 44, 48, 49, 50, 61, 74, 75
YOLOv8 *You Only Look Once version* 8 ix, xi, xvii, xviii, 4, 5, 12, 15, 24, 25, 26, 34, 48, 49, 50, 51, 59, 60, 61, 63, 68, 69, 70, 71, 73, 74

Glossário

anchor boxes — *Anchor boxes* são caixas utilizadas em modelos de deteção de objetos para prever a localização de objetos. De modo a acelerar o processo, o modelo em vez de fazer previsões na imagem inteira, divide a imagem em retângulos (*anchor boxes*) pré-definidos de diferentes tamanhos e distribuídos pela imagem. Estes retângulos, ou caixas, atuam como pontos de partida para a previsão da localização dos objetos do modelo [33]. 9, 10, 11

batch normalization — *Batch normalization* é uma técnica utilizada para tornar o treino de redes neuronais profundas mais rápido e estável. Funciona ao normalizar as entradas de cada camada da rede, garantindo que a sua média seja zero e a sua variância seja um [108]. 8, 10, 29, 30

dropout — *Dropout* refere-se a uma técnica utilizada para evitar sobreaprendizagem em redes neuronais. Durante o treino, são eliminadas aleatoriamente neurónios da rede, juntamente com as suas conexões, o que força a rede a aprender características mais robustas. Este processo ajuda a evitar que a rede dependa demasiado de qualquer neurónio, melhorando a generalização [117]. 8, 10

kernels — Em computação, *kernel* refere-se a qualquer função ou programa que realiza uma tarefa específica, seja em processadores CPU, processadores GPU ou outros. No contexto de sistemas operacionais, o *kernel* é o núcleo responsável por gerir os recursos do sistema e facilitar a comunicação entre o *hardware* e o *software*. 34

CUDA — *Compute Unified Device Architecture*, ou CUDA, é uma plataforma desenvolvida pela NVIDIA para tirar partido das capacidades de aceleração de processadores gráficos (GPU) – e da computação paralela. A plataforma permite aceder ao poder de processamento dos processadores GPU habilitados para CUDA, ao distribuir tarefas entre milhares de *threads* que operam em paralelo nos múltiplos núcleos do processador [34, 76]. 16, 18

destilação de conhecimento — Destilação de conhecimento — ou em inglês, *knowledge distillation* – é uma técnica de compressão de modelos em que um modelo grande e complexo, conhecido como modelo professor, transfere o seu conhecimento para um modelo menor e mais simples, designado de modelo aluno. O objetivo é treinar o modelo aluno para que ele consiga aproximar o desempenho do modelo professor, mas com uma quantidade muito menor de parâmetros, tornando-o mais eficiente em termos de recursos computacionais [40, 60]. 11

falsos negativos — Em Aprendizagem Automática, falsos negativos referem-se aos casos em que o modelo classifica incorretamente os dados como não pertencentes a uma classe específica, quando na realidade pertencem [74]. 39, 64

falsos positivos — Em Aprendizagem Automática, falsos positivos referem-se aos casos em que o modelo classifica incorretamente os dados como pertencentes a uma classe específica, quando na realidade não pertencem [74]. 39, 64

inferência — Inferência caracteriza o processo de tirar conclusões ou fazer previsões com base em dados ou evidências disponíveis. Em termos de Aprendizagem Automática, refere-se ao uso de um modelo treinado para fazer previsões ou classificações em novos dados. O modelo utiliza os padrões aprendidos durante a fase de treino para inferir resultados sobre dados desconhecidos. 9, 18, 24, 30, 33, 34

LIDAR — *Light Detection and Ranging*, ou LIDAR, é um sistema que utiliza feixes de luz laser para medir a distância entre o sensor e objetos. Medindo o tempo que a luz leva para refletir nos objetos e voltar ao sensor, o sistema cria mapas tridimensionais precisos do local [11, 55]. 15

quantização — Quantização – ou em inglês, *quantization* – é uma técnica utilizada para otimizar a execução de modelos de Aprendizagem Profunda, reduzindo o uso de memória e aumentando a eficiência computacional. Consiste em representar os pesos e as ativações com tipos de dados de baixa precisão, como INT8 (8-bit *Integer*) ou FP16 (16-bit *Floating Point*), em vez de FP32 (32-bit *Floating Point*). Esta abordagem utiliza menos memória, reduz o consumo de energia e acelera operações como a multiplicação de matrizes [35, 60]. 11

sobreaprendizagem — Sobreaprendizagem – ou em inglês, *overfitting* – designa o fenómeno que consiste num modelo de aprendizagem automática se ajustar demasiado aos dados de treino, capturando padrões e ruído irrelevantes. Isto resulta num desempenho superior em dados iguais aos de treino, mas numa menor capacidade de generalização para novos dados [41, 132]. 8, 10, 49

TensorRT — TensorRT é um *Software Development Kit* (SDK) avançado, desenvolvido pela NVIDIA, para otimização de inferência em modelos de Aprendizagem Profunda. O TensorRT é especialmente adequado para aplicações em tempo real, como a deteção de objetos [79]. 16, 33, 34, 59, 60, 64, 65, 74

verdadeiros negativos — Em Aprendizagem Automática, verdadeiros negativos referem-se aos casos em que o modelo classifica corretamente os dados como não pertencentes a uma classe específica [74]. 39

verdadeiros positivos — Em Aprendizagem Automática, verdadeiros positivos referem-se aos casos que o modelo classifica corretamente os dados de uma classe específica. Ou seja, são as previsões realizadas idênticas aos valores reais de uma classe. Em multiclasse a classe real é considerada como classe verdadeira e as restantes classes como classes negativas [74]. 39, 64

1 Introdução

Neste capítulo é abordado o contexto geral que motivou o desenvolvimento deste trabalho, incluindo a relevância do problema e os principais desafios enfrentados. São apresentados os objetivos específicos que se pretenderam alcançar com este trabalho, juntamente com a abordagem metodológica utilizada. Além disso, são referidas as contribuições deste trabalho e apresentada a organização do documento, fornecendo uma visão geral dos conteúdos abordados em cada capítulo.

1.1 Enquadramento

Antigamente, as ruas eram partilhadas de forma relativamente harmoniosa por peões, animais, carroças, charretes e ciclistas. No entanto, no início do século XX (1900), o surgimento dos automóveis trouxe uma nova dinâmica ao tráfego, tornando as vias caóticas e causando um aumento significativo de acidentes, ferimentos e mortes, conforme ilustrado na figura 1.1 que apresenta uma das ruas de Detroit nessa altura. Perante este cenário, tornou-se imperativo estabelecer uma regulação rodoviária de forma a organizar o tráfego e tornar as ruas mais seguras para todos os utentes [106].

Figura 1.1: *Confusão nas ruas de Detroit não regulamentadas [66].*

Em 1909, a Alemanha foi pioneira ao introduzir algumas das primeiras leis de trânsito a nível nacional, incluindo o exame de condução e a emissão de cartas de condução. À medida que as ruas se tornavam mais perigosas, foram surgido outras inovações: as primeiras passadeiras apareceram no Reino Unido na década de 1930 e a Alemanha estabeleceu o seu primeiro limite de velocidade nacional em 1934, fixando-o em 60 km/h nas áreas urbanas [106].

Apesar destas medidas, a segurança rodoviária continuou a ser um desafio global. O crescimento populacional e o aumento do número de veículos nas estradas resultaram num maior número de acidentes, tornando as colisões rodoviárias um dos principais problemas de saúde pública e prevenção de lesões no mundo. Este problema é ainda mais grave pelo facto de as vítimas serem, na maioria dos casos, pessoas saudáveis antes dos acidentes. Segundo a Organização Mundial da Saúde (OMS), mais de um milhão de pessoas perdem a vida nas estradas todos os anos. Um relatório da OMS de 2004 estimou que cerca de 1,2 milhões de pessoas morrem e 50 milhões ficam feridas anualmente em colisões rodoviárias pelo mundo inteiro, sendo os acidentes de trânsito a principal causa de morte entre crianças e jovens de 10 a 19 anos [81]. O mesmo relatório sublinhou que este problema é mais acentuado nos países em desenvolvimento, onde medidas simples de prevenção poderiam reduzir significativamente o número de mortes.

Para enfrentar este desafio crescente, foi necessário implementar medidas simples, mas eficazes, como a regulação do tráfego e a instalação de sinais de trânsito, com o objetivo de garantir um ambiente de tráfego seguro e eficiente nas estradas. Contudo, a desobediência destes sinais e regulamentos pode implicar situações de grande perigo, podendo até mesmo levar a múltiplas fatalidades. Atualmente, uma das principais causas de acidentes rodoviários é o problema de passagem de semáforos com sinal vermelho, habitualmente designado por *Red Light Running* (RLR), tendo este um impacto tanto em termos humanos como económicos [90, 99]. A figura 1.2 mostra um exemplo de infração cometida por RLR.

Do ponto de vista da segurança, esta transgressão representa uma ameaça dupla, colocando

Figura 1.2: *Exemplo de Red Light Running (RLR) [57].*

em perigo tanto os infratores quanto terceiros, outros condutores ou peões, contribuindo significativamente para o aumento do número de lesões graves e mortalidades relacionadas com o tráfego rodoviário. Diversas investigações em diferentes regiões destacam a escala deste problema [30, 46, 100–102, 137, 141]. Estima-se a ocorrência de 260 000 colisões anualmente nos Estados Unidos da América, envolvendo fatalidades em cerca de 750 acidentes [30, 102].

Economicamente, os custos associados ao RLR são substanciais, abrangendo serviços de emergência, despesas médicas, perda de produtividade e danos à propriedade, para além do aumento de trânsito [27].

De modo a minimizar estes problemas, foram realizados múltiplos estudos com o intuito de examinar os principais fatores que influenciam a decisão do condutor. Estes fatores são complexos e envolvem o comportamento do motorista, as condições ambientais e o sistema rodoviário [141]. Fatores pessoais, como demografia e perceção de risco, juntamente com influências ambientais, incluindo o comportamento de outros elementos da estrada e os tempos dos sinais de semáforo, também têm um papel significativo [137].

A análise destes fatores permitiu investigar potenciais contramedidas contra o RLR, progressivamente mais precisas e robustas. Estimar a velocidade dos carros de modo a prever se irá passar o semáforo com o sinal vermelho representa uma contramedida estudada anteriormente [48, 135]. Contudo, esta abordagem mostra-se obsoleta diante de interseções de maior complexidade.

Mais recentemente surgiram outras contramedidas, como a otimização dos tempos de sinalização, ajustando o tempo em que o semáforo permanece com sinal amarelo [17], ou câmaras automáticas de sinal vermelho, normalmente chamadas de *Red Light Camera* (RLC) [111], ilustrada na figura 1.3.

As RLC têm sido o método mais utilizado mundialmente, demonstrando reduções significativas nas violações de RLR [46]. Estes sistemas capturam automaticamente imagens de veículos que entram em interseções após os sinais mudarem para a cor vermelha, dependendo de sensores físicos para detetar a sua passagem. No passado, estes sensores consistiam de tubos estendidos pelas estradas ou sensores indutivos (*indutive loops*). No entanto, foram evoluindo e ficando mais sofisticados, pelo que consistem de radar, laser ou sistemas óticos, permitindo uma deteção mais precisa de infrações. Já as câmaras utilizadas começaram

Figura 1.3: *Exemplo do sistema Red Light Camera (RLC) [92].*

por ser analógicas, capturando os veículos que intersetavam o sinal vermelho, evoluindo, mais tarde, para câmaras digitais onde se começou a fotografar os carros apenas antes e após a interseção com demonstração do sinal vermelho, reduzindo significativamente a complexidade e memória utilizada [65, 69].

Apesar da sua contribuição significativa para a diminuição de acidentes, o propósito e o impacto das câmaras de semáforo têm sido assuntos de debate, disputando entre a melhoria da segurança nas estradas pela redução de violações de trânsito e preocupações sobre o seu uso para ganho financeiro pelas autoridades [37, 105]. Para além disso, existem estudos que indicam que a utilização de câmaras RLC pode aumentar os acidentes traseiros devido às travagens abruptas junto do semáforo e não contempla situações onde pode ser necessário passar o sinal vermelho para prevenir maiores percalços [25, 47, 101].

1.2 Objetivos

Este trabalho foi desenvolvido no contexto do projeto de investigação e desenvolvimento (I&D) SINCRO, que envolve o ISEL e a Autoridade Nacional de Segurança Rodoviária (ANSR), e teve como objetivos estudar, desenvolver e avaliar um algoritmo baseado em inteligência artificial para detetar veículos que desrespeitem semáforos com sinalização vermelha. Este algoritmo deve detetar os veículos presentes nas imagens e acompanhar os seus movimentos nas regiões de interesse para deteção de RLR, para além de classificar corretamente o estado dos semáforos (verde, amarelo ou vermelho).

Pretendeu-se também o desenvolvimento de um sistema para deteção de RLR baseado nesse algoritmo e que fosse adequado a implementações eficientes em sistemas embebidos, possibilitando o processamento automático de forma local e em tempo real. Para isso, foi essencial investigar técnicas que melhorem a eficiência do algoritmo, garantindo o seu desempenho mesmo em sistemas com recursos limitados.

Com um futuro onde cidades inteligentes estão a emergir, espera-se que este sistema contribua para novas ideias e até, possivelmente, a utilização deste sistema nas estradas, ajudando a reduzir o número de acidentes e fatalidades resultantes de violações RLR.

1.3 Abordagem e desafios

A solução proposta tem por base um algoritmo centralizado no modelo YOLOv8, disponibilizado pela Ultralytics [51], sendo este modelo utilizado para deteção dos objetos (neste caso os veículos) em conjunto com algoritmos de seguimento de objetos para obter a sua trajetória. Para classificação da cor dos semáforos foram desenvolvidos dois processos: *i*) avaliação das características normais dos semáforos, como a cor, a localização do brilho ou da saturação, e *ii*) utilização de uma rede neuronal convolucional treinada. No sistema, inicialmente, deve ser selecionada a localização do semáforo para a sua correta classificação e também a região na qual será avaliada a existência de violação. Juntamente com a definição das regiões deve ser indicada a direção em que o movimento do veículo será avaliado. Esta abordagem permite uma avaliação precisa das diferentes direções que os

veículos podem seguir, especialmente em cenários onde múltiplos semáforos controlam o tráfego da mesma faixa de rodagem.

O sistema é implementado num sistema embebido de modo a combinar um processamento automático e local em tempo real, com auxílio de uma câmara, exemplificando um possível sistema económico e otimizado para deteção de RLR.

Contudo, existem sempre desafios que são necessários ultrapassar para cumprir as tarefas descritas, muitas das vezes envolvendo variáveis imprevisíveis. No sistema proposto é inevitável a utilização de uma câmara, por isso, qualquer alteração na imagem capturada, como variação das condições climatéricas do local ou até mesmo danificação da câmara, pode levar ao mau funcionamento do sistema. Em casos onde os objetos ocluam a imagem ou se verifique a existência de uma grande quantidade de objetos próximos, como em situações de maior afluência de trânsito, o funcionamento do algoritmo de deteção pode ser influenciado negativamente. Outros problemas, como motoristas que desempenhem uma condução mais errática, inversões de sentido de marcha ou até mesmo quebras de desempenho do sistema que resultem numa menor captura de imagens por segundo – em inglês, *Frames Per Second* (fps) –, podem ter um impacto no funcionamento do algoritmo de seguimento de objetos.

Apesar de o modelo de deteção de objetos YOLOv8 fornecer modelos pré-treinados altamente otimizados, para maximizar o desempenho desses algoritmos é recomendável integrar e processar dados específicos do domínio em questão para o treino dos modelos, tornando-os mais robustos e adaptados ao contexto. Nesse sentido, a disponibilidade de sistemas computacionais com alta capacidade de processamento é fundamental, tanto para o treino eficaz dos algoritmos quanto para garantir a operação contínua e eficiente do sistema no local de implementação.

1.4 Contribuições

As principais contribuições do trabalho realizado no âmbito desta dissertação são:

- Um novo algoritmo para deteção e seguimento de veículos em vias de trânsito, baseado no modelo YOLOv8 e com capacidade para detetar a violação de passagem do sinal vermelho.

- Um sistema de deteção de RLR baseado no algoritmo proposto e otimizado para implementações na plataforma NVIDIA Jetson Orin Nano. O código fonte deste sistema está disponível num repositório com acesso público e utilização aberta [115].

- Uma nova base de dados com imagens de semáforos exibindo diferentes cores. Esta base de dados está organizada em duas pastas, *train* e *test*, contendo cada uma três pastas com o nome das classes utilizadas, *green*, *yellow* e *red*. No total existem 1383 imagens, divididas em 1161 imagens na pasta de treino e 222 imagens na pasta de teste. Esta base de dados está disponível num repositório com acesso público e utilização aberta [114].

- Uma nova base de dados com vídeos de vias públicas, captando tanto as estradas rodoviárias e veículos, como os semáforos que regulam o trânsito destas. A base de dados contém 24 vídeos de estradas localizadas no Estados Unidos da América e Canadá, e apresentam características diversas, como diferentes horários (dia e noite) e condições climatéricas (chuva e neve). Esta base de dados está disponível num repositório com acesso público e utilização aberta [113].

Foi também escrito um artigo científico apresentando o trabalho realizado e os principais resultados conseguidos, com o título *Red Light Running Detection Using AI-Powered Object Tracking on Embedded Systems*, que foi submetido à 18^{th} *Workshop on Design and Architectures for Signal and Image Processing* (DASIP 2025).

1.5 Organização do documento

Este documento está organizado em cinco capítulos, incluindo este primeiro capítulo, que introduz o problema a ser estudado e os objetivos que se pretenderam alcançar com o trabalho realizado. O capítulo 2 apresenta uma revisão detalhada do estado da arte, explorando os principais estudos e pesquisas relacionados ao tema, proporcionando uma compreensão mais profunda do problema em questão e fundamentando a escolha das ferramentas e metodologias utilizadas no desenvolvimento do projeto. No capítulo 3 é descrita a solução proposta, sendo detalhadas as principais componentes do sistema, as metodologias aplicadas e as ferramentas utilizadas para atingir os objetivos propostos. O capítulo 4 trata dos aspetos relacionados com a implementação do sistema, enquanto o capítulo 5 apresenta e discute os resultados obtidos, analisando-os à luz dos objetivos inicialmente estabelecidos e das hipóteses levantadas ao longo da pesquisa. Por fim, no capítulo 6, são apresentadas as conclusões do trabalho realizado, destacando as suas principais contribuições, e são sugeridos possíveis caminhos para trabalho futuro que podem melhorar ou expandir os resultados alcançados neste projeto.

2 Estado da arte

Neste capítulo analisam-se os avanços tecnológicos e metodológicos aplicados à deteção de infrações rodoviárias, com foco na violação por passagem de sinal vermelho (RLR). São abordadas as evoluções das técnicas de Inteligência Artificial (IA) e Aprendizagem Automática (AA), desde abordagens tradicionais até redes neuronais convolucionais (CNN) e modelos avançados de deteção de objetos, como o YOLO. Discutem-se também modelos e técnicas de seguimento de objetos. Por fim, são analisadas as características de diferentes plataformas *hardware* especializadas, incluindo sistemas embebidos, destinadas a otimizar a eficiência e a conseguir a operação em tempo real de modelos de deteção de objetos baseados em aprendizagem profunda.

2.1 Inteligência Artificial e Aprendizagem Automática

Esta secção aborda a evolução e o estado atual das tecnologias de IA aplicadas à deteção de infrações rodoviárias, com destaque especial na deteção de veículos e violações causadas por RLR. São discutidos os principais desenvolvimentos e desafios enfrentados ao longo dos anos, incluindo as metodologias iniciais baseadas em algoritmos de AA, o impacto das redes neuronais profundas no campo da visão computacional e a progressão para modelos de deteção de objetos eficientes.

2.1.1 Deteção de veículos baseado em Aprendizagem Automática

Os primeiros algoritmos de AA para deteção de infrações rodoviárias como RLR baseavam-se em modelos utilizados para classificação de características, como por exemplo *Random Forest* (RF) [43] ou *Support Vector Machine* (SVM) [26]. Estes modelos foram utilizados para prever se iriam ocorrer infrações ou colisões com base em características extraídas, como a distância da interseção, velocidade, aceleração ou tempo até à interseção do veículo, obtidas através de sensores radar ou câmaras de vídeo [48, 121].

Embora sejam relativamente simples e fáceis de implementar, estes algoritmos apresentam limitações para a classificação de imagens, principalmente por dependerem da qualidade das características extraídas para produzir resultados com melhor desempenho. A capacidade de generalização do modelo também pode ser limitada, o que dificulta a adaptação para dados

complexos ou novos dados que não foram apresentados no conjunto de treino do modelo [5, 56, 71]. Juntando estes constrangimentos ao aparecimento de outras tecnologias de IA, como as redes neuronais [15, 127], os algoritmos deste grupo tornaram-se ultrapassados para tarefas de classificação de imagens [20].

2.1.2 Deteção de veículos baseado em técnicas de Aprendizagem Profunda

O avanço contínuo no campo da IA, especialmente em visão computacional, proporcionou uma análise e classificação mais correta de imagens, possibilitando o desenvolvimento de modelos mais avançados para identificação de cenários mais complexos. O progresso nas redes neuronais convolucionais [59, 110] – em inglês, *Convolutional Neural Network* (CNN) – facultou a criação de modelos eficientes e com necessidade de menos recursos computacionais. Modelos como MobileNet [45], ResNet [42] ou Inception [119], que foram desenvolvidos para implementação em sistemas com capacidade computacional limitada, mostraram ser capazes de alcançar resultados notáveis na classificação e extração de características de imagens, conforme demonstrado nos artigos [6, 9, 38].

As CNN destacam-se pela sua arquitetura composta por camadas convolucionais com diversos neurónios, conforme se ilustra na figura 2.1. Estas camadas aplicam operações de convolução, onde são aplicados filtros aos dados de entrada (como uma imagem) para extrair características relevantes, utilizando o conceito de *weight sharing* para reduzir a quantidade de parâmetros [61]. Cada filtro é uma matriz de pesos que é "deslizada" sobre a entrada, multiplicando os valores dos dados e somando-os para gerar um mapa de características. A introdução de funções de ativação como ReLU, Sigmoid e tangente hiperbólica (Tanh) [91] confere não-linearidade ao modelo, permitindo a aprendizagem de padrões complexos. Para reduzir a dimensionalidade espacial e evitar sobreaprendizagem, são utilizadas camadas de *pooling*, como *max pooling* e *average pooling*. As camadas totalmente ligadas – em inglês, *Fully-Connected Layers* – utilizam as características extraídas nas camadas anteriores para gerar a classificação e produzir as saídas (classes) da rede. Técnicas de normalização, como *batch normalization*, melhoram a eficiência de treino e métodos de regularização, como *dropout*, ajudam a evitar sobreaprendizagem. Estas características tornam as CNN um método indicado na extração de características e classificação em diversas aplicações [38, 61].

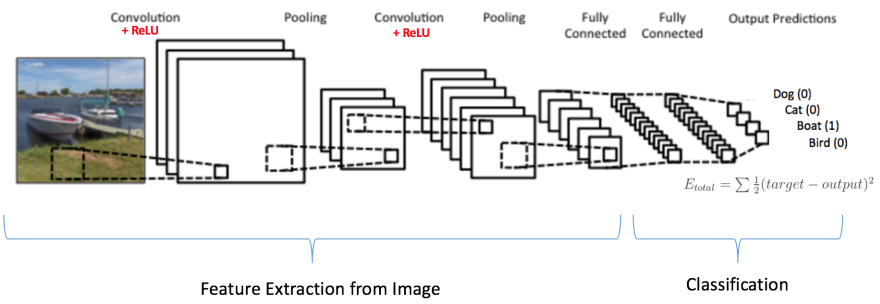


Figura 2.1: *Estrutura básica de uma rede neuronal convolucional [107].*

Com o progresso no estudo das CNN, surgiram os modelos de deteção de objetos baseados nas mesmas, que impulsionaram diferentes abordagens para avaliar imagens no que diz respeito à classificação dos objetos presentes e respetivas localizações [118].

Um dos primeiros algoritmos orientados para deteção de objetos foi o Faster R-CNN (Region-based Convolutional Neural Network) [98]. Este algoritmo compreende duas etapas. A primeira etapa designa-se por *Region Proposal Network* (RPN) e serve para gerar um conjunto de regiões de interesse – em inglês, *Region of Interest* (ROI) – prováveis de conter objetos. Essas ROI são então consideradas na segunda etapa, que é a rede de deteção, onde são classificadas e ajustadas as suas localizações. A RPN utiliza *anchor boxes* e janelas deslizantes para estimar previsões da localização dos objetos, aplicando essas *anchor boxes* na imagem para gerar ROI baseadas nas suas pontuações. A rede de deteção, uma *Fully Convolutional Network* (FCN), recebe as ROI geradas pela RPN e produz as pontuações de classe dos objetos e as caixas delimitadoras, ajustando as localizações dos objetos com base nas classificações de verdadeiros positivos das ROI [98].

Algoritmos posteriores de deteção de objetos focaram-se em melhorar a precisão da classificação e localização dos objetos na imagem, tendo ainda em conta o menor tempo de inferência possível. Entre os algoritmos de deteção de objetos existentes, há que realçar os modelos *You Only Look Once* (YOLO) [94] e *Single Shot Multibox Detector* (SSD) [64] como sendo os mais utilizados. Investigações comparativas destes modelos [72, 85, 87, 96] revelaram que o modelo YOLO é realçado como sendo o melhor na deteção de objetos, em relação ao equilíbrio entre desempenho e tempo de inferência.

2.1.3 Arquitetura do modelo YOLO

A arquitetura habitual dos modelos YOLO consiste nos seguintes blocos:

- *Input* - Camadas de entrada que recebem os píxeis da imagem introduzida.
- *Backbone* - Parte do modelo que codifica as entradas em características.
- *Neck* - Parte adicional do modelo que processa as características codificadas.
- *Head* - Camadas de saída que produzem as previsões.

A primeira versão do modelo, normalmente chamada de YOLOv1, foi apresentada em [94]. A sua arquitetura é baseada na rede GoogLeNet e foi desenhada tendo em conta a utilização de uma única rede convolucional para prever múltiplas caixas delimitadoras e indicar a probabilidade de representar cada classe para cada caixa (a figura 2.2 mostra a arquitetura do modelo YOLOv1). A rede é dividida em duas partes: a primeira extrai características visuais da imagem e a segunda prevê as probabilidades das classes e localizações dos objetos [94]. A imagem é introduzida como entrada no modelo, sendo esta dividida em células de tamanhos semelhantes, cada uma responsável por prever um determinado número de caixas. As caixas com baixa probabilidade de conter objetos são descartadas, enquanto que as caixas com maior confiança são refinadas para englobar o objeto com maior precisão, como apresentado na figura 2.3. Esta abordagem de processamento da imagem, utilizando apenas

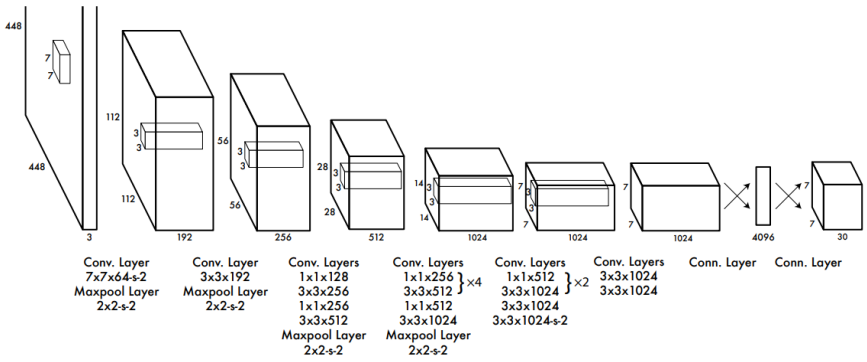


Figura 2.2: *Arquitetura do modelo YOLOv1 [94].*

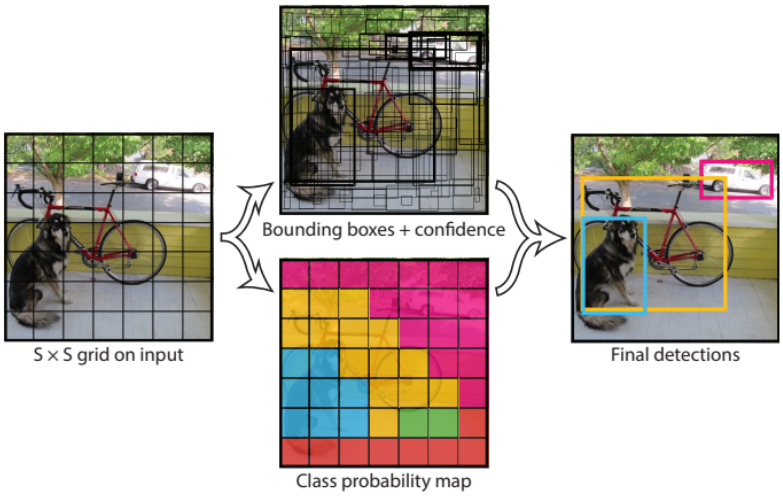


Figura 2.3: *Funcionamento simplificado do modelo YOLOv1 [94].*

uma rede convolucional, reduz bastante os recursos computacionais utilizados, fazendo com que seja adequada para aplicações em tempo real.

Ao longo do tempo, o modelo YOLO passou por várias iterações e melhorias significativas na arquitetura e eficiência. As versões subsequentes do YOLO foram focadas em aumentar a precisão da deteção, reduzir erros de localização, e melhorar a deteção de objetos pequenos e em grupos. As principais melhorias de cada versão posterior incluem:

- YOLOv2 (YOLO9000) [95]: Esta versão foi chamada YOLO9000 por ser capaz de identificar mais de 9000 categorias de objetos. Introduziu *batch normalization* nas camadas convolucionais para melhor convergência, permitindo retirar *dropout* do modelo sem haver sobreaprendizagem. Passou a utilizar *anchor boxes* para previsão das caixas delimitadoras, inspirado na rede Faster R-CNN [98], aumentando significativamente o número de previsões por imagem significativamente. Para além disso, utiliza uma rede diferente como *backbone*, a rede DarkNet [93], e é capaz de processar imagens de maior resolução, melhorando a deteção de objetos menores.

- YOLOv3 [97]: A terceira versão introduz uma rede como *backbone* mais eficiente chamada Darknet-53, contribuindo para maior precisão e rapidez. Melhora a previsão das

caixas delimitadores ao utilizar regressão logística para atribuir pontuações às caixas sem objetos – em inglês, *objectness*. Prevê caixas em três diferentes escalas e extrai características dessas escalas, de forma semelhante ao conceito de *Feature Pyramid Networks* (FPN) [62], melhorando a deteção de objetos de diferentes tamanhos.

- YOLOv4 [16]: Na quarta versão é utilizada a rede CSPDarknet53 como *backbone* melhorando a precisão na deteção de objetos e a capacidade de aprendizagem. Também é incluído o método "*mosaic data augmentation*", onde são misturadas quatro imagens no treino, melhorando o desempenho do modelo.

- YOLOv5 [49]: A versão cinco foi desenvolvida pela Ultralytics [50] usando PyTorch. Disponibiliza um sistema mais intuitivo para o utilizador, com uma implementação mais simples, instalação mais fácil e melhor integração com o ecossistema PyTorch.

- YOLOv6 [60]: A versão seis adota uma abordagem sem *anchor boxes*, ao contrário das arquiteturas anteriores que dependiam destas técnicas para deteção de objetos. Inclui melhorias práticas como treino prolongado, quantização e destilação de conhecimento, melhorando o seu processamento em tempo real.

- YOLOv7 [125]: Na sétima versão é proposto um modelo planeado re-parametrizado, tornando o modelo mais eficiente no treino. Propõe métodos como "*extend*"e "*compound scaling*" [125] permitindo ao modelo ser ajustado para diferentes necessidades de precisão e velocidade. Esta versão conseguiu reduzir efetivamente os parâmetros e a computação em relação a versões anteriores, resultando num modelo mais leve, com maior rapidez de inferência e maior precisão de deteção.

- YOLOv8 [51]: A versão oito incorporou arquiteturas de *backbone* e *neck* mais evoluídas, resultando numa melhor extração de características e desempenho na deteção de objetos. Adotou uma abordagem "*anchor-free split*" [51] na arquitetura da *head*, deu prioridade em manter o balanço entre rapidez e precisão e oferece diferentes variantes do modelo, pré-treinadas com várias dimensões de parâmetros e com suporte de múltiplas tarefas. Estas melhorias permitem que o YOLOv8 alcance melhores resultados comparativamente às versões anteriores, enquanto mantém um custo computacional reduzido e velocidade notável.

- YOLOv9 [126]: Introduzido em 2024, o YOLOv9 dá ênfase à eficiência e preservação de informação com as suas principais inovações: o *framework* de informação de gradiente programável – em inglês, *Programmable Gradient Information* (PGI) – e a rede de agregação de camadas eficiente generalizada – em inglês, *Generalized Efficient Layer Aggregation Network* (GELAN). O PGI aborda a questão da perda de informação em camadas profundas da rede, melhorando a geração de gradientes e a convergência de aprendizagem, enquanto o GELAN melhora a eficiência ao integrar blocos computacionais de forma flexível. O YOLOv9 destaca-se em arquiteturas leves e profundas, apresentando uma precisão superior e uma redução da sobrecarga computacional, superando o YOLOv8 tanto em parâmetros como em métricas de precisão.

- YOLOv10 [123]: Também introduzido em 2024, o YOLOv10 marca um avanço na deteção em tempo real ao eliminar a necessidade de *Non-Maximum Suppression* (NMS) durante o pós-processamento, o que aumenta a velocidade de inferência. Trouxe novas inovações à arquitetura do modelo, como *lightweight classification heads*, *spatial-channel decoupled downsampling* e *rank-guided block design*, cada uma contribuindo para reduções substanciais na latência e no custo computacional [7].

- YOLO11 [52]: A mais recente adição à família YOLO à data, o YOLO11, incorpora arquiteturas melhoradas de *backbone* e *neck*, o que otimiza a capacidade de extração de características para melhorar o desempenho de tarefas complexas e deteção de objetos. A redefinição da rede reduziu 22% dos parâmetros da variante YOLO11m em comparação com o YOLOv8m, ao mesmo tempo que consegue uma pontuação mAP mais elevada. Tal como a versão YOLOv8, o YOLO11 tem suporte a diversas tarefas, como segmentação, classificação de imagem e estimação de pose, para além da deteção de objetos.

Para além destas constantes melhorias, o modelo YOLO é disponibilizado numa biblioteca em Python, facilitando o seu treino e teste, e ainda oferece o suporte de variadas tarefas incluindo classificação de imagem, deteção, segmentação, seguimento – em inglês, *tracking* – e estimação de pose de objetos [51].

2.1.4 Seguimento de objetos

Seguimento de objetos – em inglês, *object tracking* – refere-se ao processo de estimar a trajetória de um objeto em movimento presente num cenário, atribuindo-lhe consistentemente o mesmo identificador ao longo das imagens do vídeo [131].

Normalmente, o objeto detetado é seguido a partir do momento em que surge pela primeira vez numa imagem do vídeo – em inglês, *frame* – até que deixe de ser visível na sequência de imagens. Neste processo, é utilizado um algoritmo de deteção de objetos para obter as coordenadas dos objetos nas imagens [23], sendo essas coordenadas posteriormente utilizadas pelo modelo de seguimento de objetos, em conjunto com outras técnicas, para fazer corresponder os objetos entre imagens, capturando-se assim o movimento dos objetos. O seguimento de objetos pode reportar ao seguimento de um único objeto, denominando-se *Single Object Tracking* (SOT) [31], ou o seguimento de múltiplos objetos, designando-se por *Multiple Object Tracking* (MOT) [67].

No contexto de deteção de diferentes veículos é importante conseguir-se o seguimento de cada um deles, sendo por este motivo necessário recorrer à técnica MOT. Neste caso, não só é necessário obter o identificador atribuído a cada veículo para distingui-los, como também detetar a sua trajetória. No âmbito de deteção RLR, é importante avaliar a trajetória do veículo para a análise de situações envolvendo cruzamentos ou interseções, onde existe mais do que uma possível direção em que o veículo pode seguir.

O desempenho dos algoritmos de seguimento depende de vários fatores que podem afetar a atribuição dos identificadores e o seguimento dos objeto ao longo do vídeo. A dependência

de resultados precisos do algoritmo de deteção ou a possível oclusão dos objetos pode fazer com que sejam atribuídos diferentes identificadores ao mesmo objeto. A qualidade do vídeo, diferenças de iluminação, a velocidade do movimento do objeto, ou até mesmo a possibilidade de variação da forma do objeto pode fazer com que seja mais difícil a associação e previsão entre imagens [23].

Alguns métodos que tentam minimizar os problemas descritos são o filtro de Kalman [54], a métrica *Intersection-over-Union* (IoU) [12] ou o algoritmo *Global Motion Compensation* (GMC) [4].

O filtro de Kalman é utilizado para estimar o estado de um sistema dinâmico ao longo do tempo, mesmo que as medições observáveis estejam contaminadas por ruído. Este método utiliza um modelo matemático para estimar o estado do objeto com base em medições ruidosas e imprecisas, atualizando continuamente a estimativa conforme novas informações são obtidas. A implementação do filtro de Kalman é dividida em duas etapas principais: a previsão e a atualização. Na etapa de previsão o algoritmo utiliza o estado anterior do objeto e um modelo de movimento para estimar a nova posição. Na etapa de atualização o algoritmo ajusta essa estimativa com base na diferença entre a previsão e a observação real do objeto. Esta abordagem torna o filtro de Kalman mais robusto à presença de ruído e mudanças no movimento dos objetos, proporcionando estimativas mais precisas ao longo do tempo [128].

A *Intersection-over-Union* (IoU) é uma métrica utilizada para avaliar a precisão das deteções de objetos e a sua associação no seguimento de objetos. A IoU é calculada como a área da interseção entre duas caixas delimitadoras dividida pela área da união das mesmas caixas. Esta métrica fornece uma medida de similaridade entre a caixa prevista pelo algoritmo e a caixa real do objeto. No contexto de seguimento de múltiplos objetos, a IoU é utilizada para associar deteções de objetos em imagens consecutivas, ajudando a garantir que o mesmo objeto seja identificado corretamente ao longo do tempo. Uma alta IoU indica uma forte sobreposição entre as caixas, sugerindo uma correspondência precisa. Portanto, esta métrica é crucial para minimizar erros de associação, como trocas de identificador, e melhorar a precisão geral do sistema de seguimento [12].

O algoritmo *Global Motion Compensation* (GMC) é utilizado para corrigir os efeitos de movimentos não intencionais da câmara na captura de vídeo, que podem causar deslocamentos significativos na imagem. O GMC é essencial em cenários onde a câmara está em movimento, pois estes movimentos podem distorcer a trajetória dos objetos seguidos. O algoritmo funciona ao identificar pontos de referência estáveis no fundo da cena e modelando o movimento da câmara com base nesses pontos. Com estas informações é possível compensar o movimento da câmara, estabilizando a sequência de imagens do vídeo e permitindo que os algoritmos de seguimento de objetos mantenham a precisão na deteção e associação de objetos. Ao integrar o GMC nos sistemas estes podem lidar melhor com desafios como oclusões e desfoques causados pelo movimento da câmara, resultando numa melhoria significativa na precisão do algoritmo de seguimento de objetos [4].

Entre os algoritmos de seguimento de múltiplos objetos (MOT), o algoritmo *Simple Online*

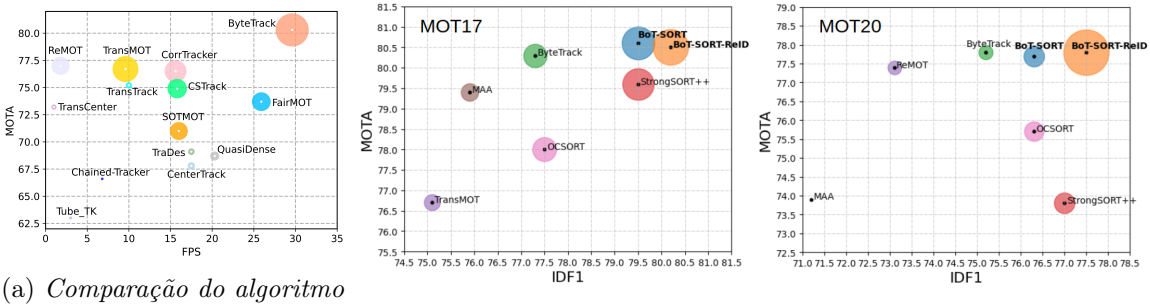
and Realtime Tracking [13] destaca-se por ser uma solução eficiente e simples para o seguimento em vídeos ou aplicações em tempo real. O SORT combina técnicas de deteção e seguimento para estimar o estado de cada objeto de forma contínua, processando as imagens à medida que são recebidas e atualizando as trajetórias dos objetos em tempo real. No entanto, uma das limitações deste algoritmo é a sua dificuldade em lidar com oclusões, o que pode resultar em mudanças de identificador frequentes. Para resolver essas limitações, surgiu o DeepSORT [129], que incorpora Aprendizagem Profunda para reduzir as mudanças de identificador, utilizando descritores de aparência e movimento para manter o seguimento mais consistente. Já o StrongSORT [29], introduziu melhorias no DeepSORT, ao ajustar os métodos de associação e deteção, reforçando a precisão do seguimento de objetos. Contudo, este último. pode não ser o mais indicado para aplicações onde o tempo de processamento deve ser reduzido [1, 73].

O algoritmo ByteTrack [138] introduziu uma abordagem inovadora de associação de objetos chamada BYTE, que tem como objetivo resolver um dos problemas comuns dos algoritmos de seguimento: a interrupção de trajetórias causadas por deteções de baixa confiança ou oclusões. A principal inovação do BYTE é a sua capacidade de associar todas as caixas de deteção, independentemente da sua pontuação de confiança. A maioria dos algoritmos descarta automaticamente caixas de deteção de baixa confiança, mas o ByteTrack utiliza ambas as caixas de alta e baixa pontuação de forma eficaz.

Inicialmente, este algoritmo associa as caixas de deteção de alta confiança às trajetórias dos objetos, com base na similaridade de movimento ou aparência, garantindo a formação de trajetórias estáveis. De seguida, o ByteTrack realiza uma segunda correspondência para as caixas de deteção de baixa confiança, que são tratadas como "pontes" para ajudar a manter a continuidade da trajetória, minimizando a perda de objetos em casos de oclusões ou de imagens onde a deteção é fraca. Esta abordagem permite ao ByteTrack manter as informações de trajetória e lidar melhor com a oclusão dos objetos, resultando numa maior precisão na continuidade do seguimento [86, 138].

O algoritmo BoT-SORT [4] é uma extensão do ByteTrack que adiciona várias melhorias para superar as limitações encontradas nos algoritmos de seguimento de objetos do tipo SORT. Os algoritmos SORT utilizam o filtro de Kalman com a suposição de um modelo de movimento de velocidade constante para prever a caixa delimitadora do objeto na próxima imagem e associá-la à nova deteção. Além disso, o filtro de Kalman é usado para prever o estado do objeto em casos de oclusão ou deteções falhadas. No entanto, esses algoritmos dependem fortemente da qualidade da previsão da caixa de deteção, o que pode resultar em falhas no seguimento, especialmente em cenários complexos, como o movimento da câmara. O BoT-SORT resolve estes problemas integrando um filtro de Kalman modificado, juntamente com o algoritmo GMC, para compensar os movimentos da câmara e melhorar a precisão do seguimento [4, 86].

Estes algoritmos conseguiram atingir bons resultados relativamente a outros algoritmos de renome, como DeepSORT [129], StrongSORT [29] ou OCSORT [21], obtendo altos desempenhos e baixos tempos de processamento em diversas investigações [1, 4, 53, 73, 133, 138]. Alguns destes resultados são apresentados na figura 2.4.



(a) *Comparação do algoritmo ByteTrack [138].*

(b) *Comparação do algoritmo BoT-SORT [4].*

Figura 2.4: *Comparação dos algoritmos ByteTrack e BoT-SORT com outros algoritmos de seguimento de objetos, quanto à sua precisão e identificação corretas [4, 138].*

Para além disso, ambos os algoritmos ByteTrack e BoT-SORT têm integração com o modelo de deteção YOLOv8, simplificando a sua utilização.

2.2 Hardware

A deteção de infrações rodoviárias é uma tarefa complexa que requer a utilização de hardware específico como sensores e câmaras. As câmaras *Red Light Camera* (RLC) e os sensores LIDAR são dois tipos de hardware utilizados hoje em dia para detetar ilegalidades relacionadas com RLR.

As câmaras RLC foram introduzidas pela primeira vez na década de 1960 e ganharam popularidade nos anos 1990, especialmente em países como os Estados Unidos, Reino Unido e Austrália. Estas câmaras são sistemas automatizados projetados para detetar e fotografar veículos que entram numa interseção após o semáforo ficar vermelho. Ao longo do tempo, a tecnologia utilizada nas RLC melhorou, com sistemas mais recentes a utilizar câmaras digitais, tecnologia de radar e sensores avançados para fornecer dados mais precisos [2].

O aparecimento dos sensores LIDAR trouxe novos procedimentos para identificar RLR, a exemplo da fórmula *Stopping Sight Distance* que estabelece a distância necessária para uma travagem segura com base na velocidade do veículo [55]. Porém, tais sensores podem apresentar custos elevados e complexidade operacional, para além de poderem não ser os mais apropriados para a identificação de infrações em condições climatéricas adversas [130, 140].

Quanto ao processamento dos dados, as imagens eram inicialmente enviadas para operadores especializados verificarem a ocorrência de infrações. Mais tarde, os dados passaram a ser processados por sistemas computacionais de alta capacidade que automaticamente verificam as infrações, fazem o cruzamento de informações e, muitas vezes, enviam diretamente as notificações de infração para os condutores via correio postal ou eletrónico. Estas infrações podem ainda ser verificadas remotamente pelas autoridades, eliminando a necessidade de intervenção humana em grande parte do processo, o que reduz o tempo e o custo envolvido na aplicação da Lei [2].

Embora se tenha observado que a utilização de câmaras RLC leva a uma diminuição significativa de violações RLR, constata-se que estes sistemas também têm desvantagens,

como o alto preço de instalação e manutenção, instalações complexas ou o aumento de acidentes devido a embates traseiros [18, 68].

Por outro lado, a crescente evolução e procura na área de IA catalisou o surgimento de sistemas embebidos concebidos para desempenhar funções complexas e especializadas em visão computacional, muitas vezes em tempo real. Esses sistemas embebidos têm-se revelado como um fator determinante na evolução de sistemas cada vez mais otimizados, permitindo ainda um processamento totalmente local.

Estudos comparativos de diversos sistemas embebidos na área de visão computacional [14, 32, 58, 70, 84, 112, 120] visaram aprofundar a compreensão das particularidades e potencialidades dessas plataformas, designadamente no que diz respeito à implementação de modelos de deteção de objetos baseados em aprendizagem profunda, como os da família YOLO. As plataformas Raspberry Pi, Google Coral e a gama NVIDIA Jetson apresentam várias características distintas que as tornam adequadas para aplicações específicas, em virtude das suas capacidades computacionais, eficiência energética e custo-benefício, conforme se mostra na tabela 2.1.

Raspberry Pi tem sido uma solução bastante utilizada na área educativa, ganhando destaque pela sua boa relação custo-benefício. Esta plataforma, nas suas versões 3 e 4, destaca-se como uma solução de baixo custo e consumo energético moderado, capaz de executar algoritmos de deteção de objetos com desempenho, sobretudo quando acoplado com um Intel Neural Compute Stick [32, 70]. Alguns estudos [32, 70] mostram que este dispositivo é particularmente adequado à utilização de modelos mais leves, como o YOLOv3-tiny, oferecendo uma taxa de processamento de imagens (fps) consideravelmente elevada em comparação a variante YOLOv3.

Por outro lado, a NVIDIA introduziu no mercado os seus sistemas embebidos projetados especificamente para IA. Os dispositivos da série NVIDIA Jetson, nomeadamente o Xavier, TX2 e Nano, constituem opções que se destacam pelo equilíbrio entre desempenho computacional e eficiência energética. Estes sistemas estão equipados com processadores gráficos – em inglês, Graphics Processing Unit (GPU) – habilitados para CUDA, proporcionando um terreno fértil para a execução de aplicações de visão computacional. Em particular, o equipamento NVIDIA Jetson Nano destaca-se como um compromisso equilibrado entre desempenho e custo, mostrando-se capaz de atingir taxas de processamento de imagens (fps) elevadas utilizando YOLOv4-tiny e apresentando um potencial de melhoria adicional com o uso de TensorRT [112].

Mais recentemente, a NVIDIA lançou a série Jetson Orin, uma nova linha de sistemas embebidos que redefine o padrão de processamento avançado para aplicações de IA. Com capacidades de até 275 Tera Operações Por Segundo – em inglês, *Tera Operations Per Second* (TOPS) –, estes dispositivos são projetados para enfrentar desafios mais complexos, como análise em tempo real e tarefas de aprendizagem profunda. A arquitetura da série Orin foi otimizada para lidar com cargas de trabalho intensivas, mantendo uma operação fluida e contínua. Este nível de desempenho torna-a uma escolha ideal para projetos que exigem alta capacidade de processamento sem comprometer a eficiência nos recursos.

Tabela 2.1: Comparação de sistemas embebidos para aplicações de IA [39, 78, 88]

Sistema Embebido	**Desempenho de IA**	**Memória**	**Armazenamento**	**Potência**	**Dimensões**	**Preço**
NVIDIA Jetson Nano	Até 0.5 TFLOPS	4GB LPDDR4	16GB eMMC	5W a 10W	70mm x 45mm	≈ $99
NVIDIA Jetson Orin Nano	Até 40 TOPS	8GB LPDDR5	Adicional	7W a 15W	70mm x 45mm	≈ $199
NVIDIA Jetson Xavier	Até 32 TOPS	32GB LPDDR4x	32GB eMMC	10W a 30W	70mm x 45mm	≈ $399
NVIDIA Jetson TX2	Até 1.3 TFLOPS	8GB LPDDR4	32GB eMMC	7.5W a 15W	70mm x 45mm	≈ $149
NVIDIA Jetson AGX Orin	Até 275 TOPS	64GB LPDDR5	64GB eMMC	15W a 60W	100mm x 87mm	≈ $899
Raspberry Pi 4 Model B	Até 1 TFLOPS	Até 8GB LPDDR4	Adicional	3W a 6.5W	85mm x 56mm	≈ $55
Google Coral Dev Board	Até 4 TOPS	4GB LPDDR4	16GB eMMC	2W	85mm x 56mm	≈ $130

Quando comparado com outros sistemas, como o Raspberry Pi 4 e o próprio NVIDIA Jetson Nano, é possível verificar as superiores eficiência e capacidade computacional desta gama. A Jetson Orin destaca-se não apenas pelo seu desempenho bruto, mas também pela sua capacidade para executar tarefas complexas de deteção de objetos com maior rapidez e precisão, conforme mostram alguns estudos recentes [116, 134, 136]. As suas capacidades de processamento permitem uma operação mais rápida do que num computador com CPU Intel i7–7600U, utilizando modelos como o YOLOv5s [136]. Além disso, a incorporação do suporte com CUDA da NVIDIA amplifica o desempenho dos sistemas Jetson Orin em tarefas que requerem processamento paralelo intenso [116].

Em síntese, a escolha de um sistema embebido para aplicações de visão computacional e deteção de objetos em tempo real deve ser guiada por uma avaliação das exigências específicas de cada aplicação. Aspetos como a complexidade do modelo de deteção, a necessidade de eficiência energética, o custo e a facilidade de integração do hardware com os modelos de IA são pontos a ter em conta. Cada uma das plataformas analisadas oferece um conjunto único de vantagens que as tornam adequadas para cenários específicos, desde implementações de baixo custo e energia até soluções que requerem alto desempenho computacional.

2.3 Conclusão

A passagem de semáforos com sinal vermelho (RLR) é um problema sério que representa um risco significativo para a segurança rodoviária. A deteção correta de RLR é um desafio complexo, pois exige a capacidade de identificar diferentes tipos de veículos em movimento e em condições variáveis.

Neste capítulo, foi apresentada uma revisão abrangente sobre o estado da arte na deteção de infrações rodoviárias, com destaque para a violação de passagem de sinal vermelho (RLR). A análise abordou os principais desenvolvimentos na área de IA, mostrando a transição de abordagens tradicionais de AA baseadas em características extraídas, para métodos modernos com redes neuronais convolucionais (CNN) e modelos de deteção de objetos.

Entre os modelos de deteção de objetos, as versões mais recentes do modelo YOLO sobressaíram como uma solução equilibrada entre desempenho e tempo de inferência para deteção de objetos, apresentando melhorias significativas em relação às versões anteriores e a outros algoritmos.

Adicionalmente, foram abordadas técnicas de seguimento de objetos, como ByteTrack e BoT-SORT, que se destacam pelo desempenho na manutenção de trajetórias de objetos em cenários complexos e com tempos de inferência inferiores. A integração destes algoritmos com as versões mais recentes do modelo YOLO também representa uma vantagem significativa, pela simplicidade e desempenho na deteção e seguimento dos objetos conjunta.

No que diz respeito às plataformas de *hardware*, o estudo comparativo evidenciou a crescente importância dos sistemas embebidos no processamento de aplicações de IA e tarefas complexas de visão computacional, com recurso a processadores otimizados e dedicados. De

entre os sistemas embebidos analisados, a gama Orin da série NVIDIA Jetson destacou-se tanto pelo seu desempenho no processamento de aplicações com utilização de modelos de deteção de objetos, como o YOLO, quanto pela sua dimensão compacta e custo relativamente baixo.

A união de algoritmos avançados com *hardware* especializado delineia uma solução promissora para a deteção de RLR.

3 Solução proposta

Neste capítulo aborda-se a solução proposta para detetar violações de passagem do sinal vermelho (RLR). São apresentados os principais objetivos e requisitos do sistema desenvolvido, bem como os algoritmos subjacentes. Nesse contexto, discute-se a definição das ROI, as técnicas aplicadas para classificação de semáforos e a implementação de técnicas de deteção e seguimento de objetos utilizando modelos de visão computacional como YOLO, abordando os desafios enfrentados e as soluções adotadas. Expõe-se ainda o processo de avaliação e atribuição de irregularidades RLR e as técnicas usadas para otimizar o desempenho do algoritmo.

3.1 Arquitetura do sistema

O sistema desenvolvido para detetar RLR é uma solução que se pretende de baixo custo e baseada apenas em técnicas de visão computacional recorrendo a câmaras de vídeo digitais. Considera-se que as câmaras são instaladas estrategicamente junto às intersecções com o intuito de capturarem, simultaneamente, os sinais luminosos de regulação do trânsito – semáforos – e os veículos que circulam nas faixas de rodagem sob monitorização. Mais, o sistema inclui um sistema embebido com capacidade para processar, localmente e em tempo real, o algoritmo de deteção de irregularidades RLR.

O funcionamento do sistema desenvolve-se em duas fases principais. Na primeira fase são definidas as ROI, isto é, as áreas das imagens a avaliar correspondentes aos semáforos, para classificação da sua cor, e das zonas de possível violação RLR. Como uma única câmara pode ser utilizada para monitorizar mais do que um semáforo na mesma faixa de rodagem, para cada semáforo é necessário estabelecer a sua região de violação correspondente. Desta forma, cada potencial evento RLR é analisado com base num par de regiões (semáforo e violação). Para além disso, é necessário definir a direção e o sentido do tráfego que o semáforo está a regular, de modo a ser verificado face ao movimento dos veículos em cada par de regiões.

A segunda fase corresponde ao processamento contínuo das imagens capturadas, verificando-se a ocorrência de irregularidade. Caso seja detetada uma violação RLR, o algoritmo regista

todas as informações relevantes, incluindo a data e hora da ocorrência, o identificador atribuído ao veículo envolvido e o vídeo do incidente.

De modo a facilitar o entendimento do algoritmo foi desenvolvido dois diagramas de fluxo para a primeira e segunda fase, demonstrados nos anexos I e II, respetivamente.

3.2 Regiões de interesse

Para garantir a captura de vídeos que mostrem claramente a ocorrência de RLR, e que também possam servir como prova documental em tribunal, as câmaras devem ser estrategicamente posicionadas de modo a cobrir por completo as faixas de rodagem a monitorizar e os semáforos associados. Contudo, essa disposição também poderá resultar na captura de outras faixas de rodagem ou vias de trânsito, o que torna necessário selecionar ROI que delimitem apenas a área relevante para análise pelo sistema. Esta seleção da ROI, delimitada por uma linha poligonal com quatro vértices, proporciona o foco exclusivo nos veículos que transitam na direção e sentido sob avaliação.

O algoritmo desenvolvido considera que cada ROI de semáforo terá uma e apenas uma ROI de violação associada, por forma a simplificar o funcionamento do sistema. Isto permite que se avaliem as diferentes direções/sentidos que os veículos podem seguir, especialmente em cruzamentos onde poderão existir múltiplos sinais luminosos a controlar o fluxo de trânsito em vias distintas da mesma faixa de rodagem. A figura 3.1 demonstra um exemplo de um cruzamento com diferentes semáforos a controlar o tráfego da mesma faixa de rodagem.

A definição das ROI de violação e semáforo pode ser feita através de um ficheiro de configuração ou usando a interface de utilizador do sistema, que também pode ser usada para ajustar essas regiões à posteriori. Neste caso, o utilizador usa o rato para selecionar, mover e redimensionar as regiões, sendo que o movimento é feito arrastando o centro e o redimensionamento é realizado ajustando os vértices. Para facilitar a definição das ROI dos semáforos é também possível utilizar o modelo de deteção de objetos YOLO para detetar os semáforos, obtendo-se neste caso as regiões dos semáforos com as coordenadas detetadas. Uma vez configuradas todas as regiões, o utilizador tem a opção de guardar essas definições num ficheiro. Este ficheiro pode ser carregado pelo sistema no seu arranque, evitando a necessidade de se redefinir as coordenadas para cada inicialização do sistema.

3.2.1 Região de interesse de violação

No algoritmo desenvolvido, uma ROI de violação é definida pelas suas coordenadas e um conjunto de informações que ajudam a armazenar dados relevantes para a avaliação de RLR. As informações associadas a estas regiões são as seguintes:

- id: Identificador único da região.
- name: Designação da região de violação, identificando-a de forma clara.
- polygon: Coordenadas da região, em formato poligonal.

- reg_color: Cor da linha poligonal da região, convertida no formato BGR (Blue, Green, Red). BGR é o formato utilizado por parte da biblioteca OpenCV para desenho de elementos gráficos.
- text_color: Cor do texto exibido sobre a região no formato BGR.
- counts: Contagem de veículos que se encontram dentro da região, exibido no centro da região.
- violations: Número de violações registadas na região.
- car_count: Número total de veículos que passaram pela região.
- dragging: Variável booleana auxiliar à definição da localização da região, indicando se está a ser movida.

3.2.2 Região de interesse de semáforo

As ROI de semáforo seguem uma estrutura semelhante às ROI de violação, contendo informações como a classificação da cor do semáforo e a direção do fluxo de trânsito que controlam. As informações associadas às regiões de semáforo são as seguintes:

- id: Identificador único da região.
- name: Designação da região de semáforo.
- polygon: Coordenadas da região, em formato poligonal.
- entry_line: Linha por onde entram os veículos relevantes para avaliação RLR, i.e. NORTH, SOUTH, WEST ou EAST.
- direction: Direção do tráfego que o semáforo controla, i.e. NORTH, SOUTH, WEST ou EAST.
- reg_color: Cor da linha poligonal da região no formato BGR.
- text_color: Cor do texto exibido sobre a região no formato BGR.
- text_box_color: Cor da caixa do texto exibida sobre a região no formato BGR, conforme a cor emitida pelo semáforo (verde, amarela, vermelha ou preta se estiver no estado com classificação OFF).
- color: Cor da classificação realizada ao estado do semáforo, i.e. GREEN, YELLOW, RED ou OFF.
- dragging: Variável booleana auxiliar à definição da localização da região, indicando se está a ser movida.

Na figura 3.1 mostram-se duas ROI de semáforo, cada uma associada a uma região de violação e avaliando diferentes direções/sentidos de tráfego rodoviário. A associação dos pares de regiões está representada pelas cores das fronteiras das regiões.

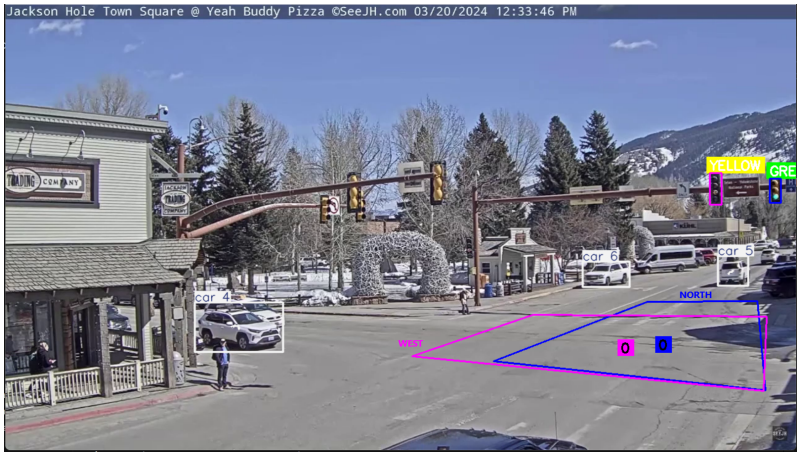


Figura 3.1: *Exemplo de definição dos pares de ROI de violação e semáforo com diferentes direções/sentidos de avaliação numa estrada de Jackson Hole, Estados Unidos da América [44].*

3.3 Modelo de deteção

A deteção de objetos é um dos desafios mais complexos em visão computacional, especialmente quando se requer precisão em tempo real. Um dos modelos mais eficientes e amplamente utilizados para essa tarefa é o YOLO [85], conforme discutido na secção 2.1.2. Desde a sua introdução, o YOLO trouxe uma abordagem inovadora para a deteção de objetos, reformulando o problema como uma única operação de regressão direta, em vez de adaptar classificadores para essa tarefa [94].

O algoritmo de deteção de irregularidades RLR desenvolvido neste trabalho é baseado na oitava versão do YOLO, o modelo YOLOv8. Esta versão foi escolhida por representar o modelo mais avançado e estável disponível no início da fase de implementação do projeto. O YOLOv8 disponibiliza várias variantes, projetadas para atender a diferentes níveis de recursos computacionais e requisitos das aplicações. Essas variantes são definidas pelo tamanho e complexidade do modelo, começando na YOLOv8n (variante *nano*), indicada para ambientes com recursos extremamente restritos, e acabando na YOLOv8x (variante *extra-large*), que disponibiliza o máximo desempenho e precisão disponível [51].

Os modelos menores, como YOLOv8n, possuem menos camadas e parâmetros na estrutura da rede, o que resulta em tempos de inferência mais reduzidos, porém oferecendo menor precisão na deteção de objetos. Em contrapartida, os modelos maiores, como YOLOv8x, proporcionam melhor desempenho em termos de precisão, mas têm um tempo de inferência maior e requerem mais recursos computacionais [50]. Na tabela 3.1 são apresentados os números de parâmetros e camadas de cada variante e na figura 3.2 mostra-se como as diferentes variantes se comparam em termos de precisão (mAP) relativamente aos parâmetros e latência. Os valores apresentados na figura, foram obtidos durante o processo de validação do modelo no conjunto de dados da base de dados COCO [51, 63], notando-se um compromisso entre precisão/número de parâmetros e precisão/latência. As variantes com mais parâmetros apresentam precisão superior, no entanto maior latência. Contrariamente, as variantes com menos parâmetros perdem precisão para obter valores de latência menores.

Tabela 3.1: Número de parâmetros e camadas de cada variante do modelo YOLOv8. Valores obtidos através da execução do modelo com o método 'model.info()'.

Variante	Nº de parâmetros	Nº de camadas
YOLOv8n	3157200	225
YOLOv8s	11166560	225
YOLOv8m	25902640	295
YOLOv8l	43691520	365
YOLOv8x	68229648	365

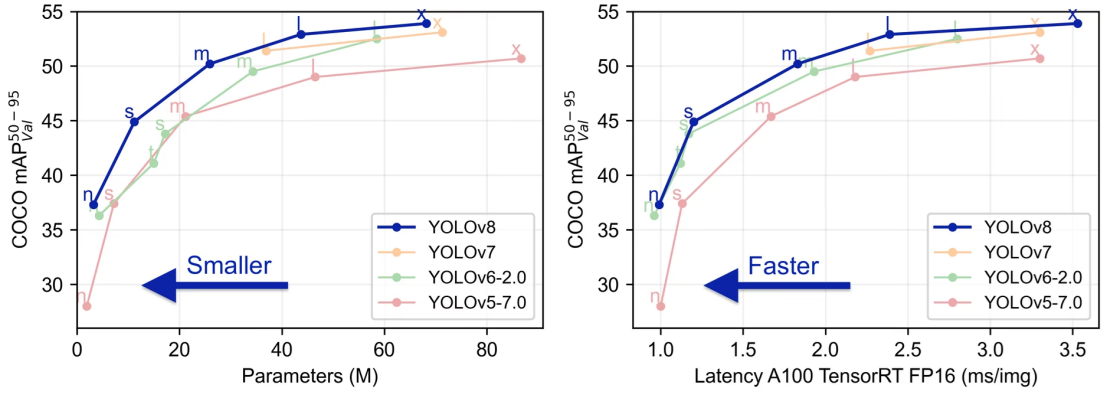


Figura 3.2: *Comparação do modelo YOLOv8 com outras versões relativamente à precisão, parâmetros e latência [51].*

Conclui-se assim que para escolher a variante mais indicada para uma dada aplicação se deve ter em atenção os seguintes parâmetros:

- Tamanho e complexidade - As variantes diferenciam-se em tamanho e complexidade, com modelos maiores a possuir mais parâmetros e camadas, e sendo o tamanho do ficheiro de pesos também maior.
- Rapidez e latência - Os modelos menores são geralmente mais rápidos e atingem valores de latência inferiores, mas podem sacrificar a precisão.
- Precisão - Os modelos maiores tipicamente alcançam maior precisão, mas exigem mais recursos computacionais.
- Requisitos da aplicação - A escolha da variante depende dos requisitos específicos da tarefa, incluindo restrições computacionais, precisão desejada e necessidades de desempenho em tempos reduzidos.

Com base nestes conhecimentos, e considerando que este trabalho tem como intuito a utilização de um sistema embebido com recursos limitados e a realização de processamento em tempo real, decidiu-se considerar apenas as variantes YOLOv8n e YOLOv8s para a implementação do algoritmo de deteção de irregularidades RLR desenvolvido. Estas variantes foram escolhidas por oferecerem um equilíbrio entre a precisão requerida e o baixo impacto na rapidez de processamento e latência, tornando-as adequadas para o sistema proposto.

Por forma a utilizar este modelo, a Ultralytics desenvolveu uma biblioteca em Python, 'ultralytics' [51], onde disponibiliza os modelos YOLO em conjunto com funções úteis para o desenvolvimento de aplicações com recurso a deteção de objetos. Desta forma, para utilizar a variante com menos parâmetros do modelo YOLOv8, por exemplo, basta carregar os pesos referentes ao modelo e utilizar a função destinada para obter os resultados da deteção dos objetos, como apresentado no código da listagem 3.1:

```
1 import ultralytics
2
3 # Carregar o modelo YOLOv8 com variante nano
4 model = YOLO(yolov8n.pt)
5
6 # Fazer deteção dos objetos numa imagem
7 results = model('imagem.png')
```

Listagem 3.1: Código Python de demonstração do procedimento para detetar objetos com o modelo YOLOv8n.

3.4 Seguimento de Objetos

Para utilizar a tarefa de seguimento de objetos no modelo YOLOv8 é necessário indicar que será realizada a tarefa de seguimento de objetos ("*track*"). Assim, o modelo utilizado retorna os resultados com as coordenadas das caixas delimitadoras dos objetos detetados, os identificadores atribuídos a cada objeto e as suas respetivas classes identificadas.

A biblioteca ultralytics permite realizar seguimento de objetos em vídeos completos ou em imagens subsequentes de um vídeo. No caso desta última opção, deve-se indicar que o vídeo a processar será fornecido imagem a imagem através do parâmetro '*persist*' colocado a '*True*'. Um exemplo deste procedimento é apresentado na listagem 3.2.

```
1  import ultralytics
2
3  # Carregar o modelo YOLOv8 com variante nano
4  model = YOLO(yolov8n.pt)
5
6  # Fazer seguimento dos objetos num vídeo
7  results = model.track('video.mp4')
8
9  # Fazer seguimento dos objetos numa sequência de imagens
10 for frame in video:
11     results = model.track(frame, persist=True)
```

Listagem 3.2: Código Python de demonstração do procedimento para seguimento de objetos com o modelo YOLOv8n.

O algoritmo de deteção de irregularidades RLR desenvolvido usa estes dados para definir a trajetória dos veículos. Para cada imagem lida do vídeo capturado é guardado o ponto central de cada veículo, tendo em conta o identificador que lhe foi atribuído. Depois, faz-se a interligação desses pontos para obter a trajetória que o veículo seguiu, servindo esta

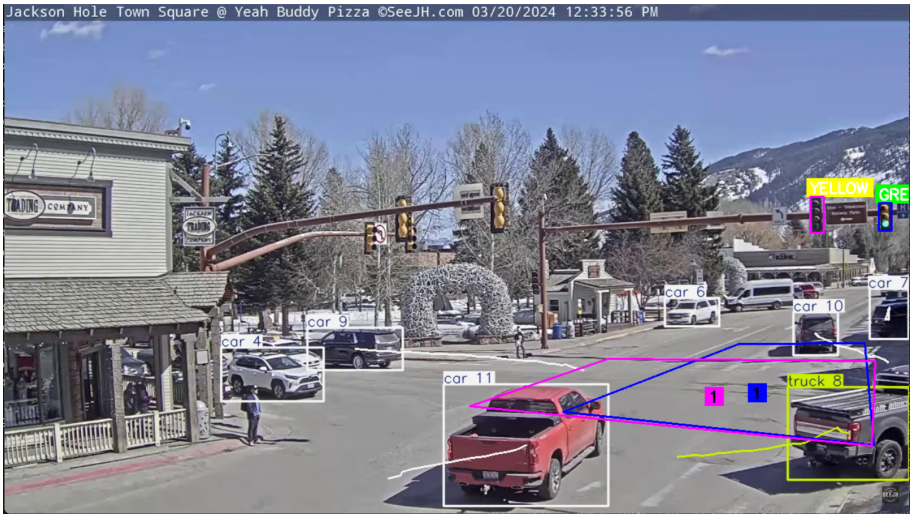


Figura 3.3: *Exemplo onde é possível observar a trajetória dos veículos com os dados obtidos através da tarefa de seguimento de objetos do modelo YOLOv8 com recurso ao algoritmo ByteTrack.*

para avaliar a possibilidade de violação RLR relativamente à ROI em causa. Na figura 3.3 pode-se visualizar as linhas de trajetória dos veículos.

Como referido na secção 2.1.4, os algoritmos ByteTrack e BoT-SORT apresentam vantagens significativas quando comparados com outros algoritmos de seguimento de objetos. Outra importante vantagem destes dois algoritmos é oferecem integração direta com os modelos de deteção YOLO através da biblioteca da Ultralytics [51], o que permite que a tarefa de seguimento de objetos seja configurada com maior simplicidade.

Atendendo a estas vantagens e aos bons resultados conseguidos com a utilização do algoritmo ByteTrack, discutidos na secção 5.3, adotou-se este algoritmo de seguimento para o sistema proposto.

3.5 Classificação da cor do semáforo

A identificação correta da cor do semáforo é essencial para o bom funcionamento de um sistema de deteção de RLR. No caso de não se conseguir identificar a cor vermelha do semáforo, não se deteta nenhuma situação de RLR. Por outro lado, erros na classificação da cor do semáforo podem ter consequências graves, como a deteção incorreta de RLR, comprometendo a confiança dos utilizadores no sistema. Por este motivo, torna-se necessário desenvolver métodos robustos para classificar corretamente as cores dos semáforos.

Para atingir este objetivo, neste trabalho foi desenvolvido um classificador inspirado na proposta de Dewar [28]. Neste artigo propõe-se dois métodos distintos para identificar as cores dos semáforos: um que se baseia nas características específicas das imagens dos semáforos, como a cor, o brilho e a saturação, e outro que utiliza redes neuronais convolucionais.

O classificador desenvolvido para o algoritmo de deteção de irregularidades RLR proposto implementa ambos os métodos com o intuito de avaliar qual apresenta melhor desempenho em termos de precisão e consumo de recursos. Por omissão, o classificador utiliza o método

baseado nas características específicas das imagens para classificar a cor do semáforo. No entanto, é possível optar pelo método da rede neuronal convolucional ao incluir o comando '$--cnn$' na linha de comandos no momento de execução. Nas secções seguintes apresentam-se detalhadamente os dois métodos.

3.5.1 Classificação com base em características específicas da imagem

A classificação baseada na análise de características específicas das imagens dos semáforos compreende as seguintes etapas:

- Pré-processamento das imagens: Todas as imagens devem ter o mesmo formato, por forma a permitir a mesma análise de características para todas as imagens. É importante que as imagens sejam semelhantes para obter características similares [28]. Assim, as imagens são redimensionadas para um tamanho fixo de 32 píxeis de largura por 96 píxeis de altura. Este tamanho da imagem permite manter a disposição vertical dos semáforos, facilitando ainda a divisão uniforme das imagens em três regiões, sendo cada região a área da luz de cada cor do semáforo.
- Análise de brilho: Utilizando o espaço de cores HSV, são calculados os valores de brilho para cada região de cor. A média do brilho em cada uma dessas regiões é então comparada e a região com a cor que apresenta maior valor de média de brilho é identificada como a localização da luz ativa do semáforo (ver imagem 3.4).
- Análise de saturação: De forma semelhante, a saturação é analisada no espaço HSV. A média de saturação de cada região de cor é calculada, e a localização com a maior saturação é correlacionada com a posição da cor do semáforo (ver imagem 3.4).
- Dominância de cor: Adicionalmente, é realizada uma análise da dominância de cor de cada região de cor do semáforo. As proporções das cores vermelho, amarelo e verde são calculadas nas respetivas regiões, permitindo identificar a cor predominante.
- Classificação: Por fim, são analisados os resultados para as três características, ficando aquele que é representado mais vezes. No caso de se obter três resultados diferentes,

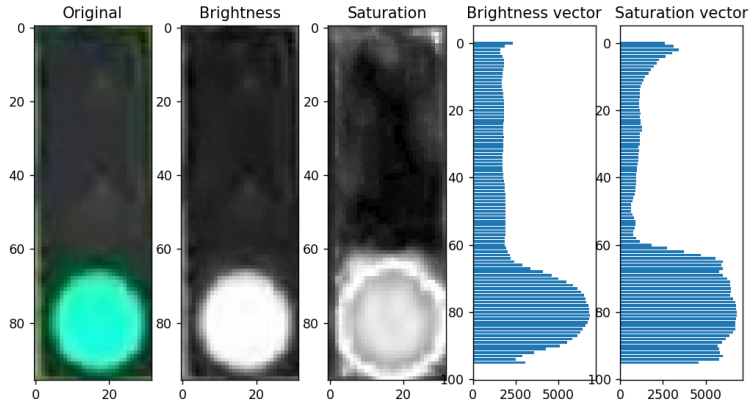


Figura 3.4: *Exemplo de classificação do semáforo com base nas características específicas da imagem.*

ou dos valores obtidos não ultrapassarem suficientemente um valor pré-definido, a classificação final da cor do semáforo fica com o valor "OFF", indicando que o semáforo está desligado e sem apresentação de qualquer cor.

Este método é simples e eficaz em condições normais, com utilização de três características diferentes para confirmação da classificação. Contudo, apresenta limitações em ambientes com condições climáticas adversas ou variações significativas de luminosidade (por exemplo, à noite). Além disto, a sua precisão depende do formato do semáforo, visto que é direcionado para classificar semáforos com a disposição vertical padrão. Semáforos horizontais, com cores diferentes ou com um número diferente de luzes, serão classificados incorretamente. Embora estas limitações, o método baseado nas características pode ser uma alternativa viável em sistemas com recursos computacionais limitados.

3.5.2 Classificação com base em rede neuronal convolucional

O segundo método implementado pelo classificador utiliza uma CNN treinada para classificar a cor que o semáforo emite, ou seja, verde, amarelo ou vermelho. A estrutura desta rede é baseada na proposta de Dewar [28], sendo composta por várias camadas convolucionais seguidas por operações de normalização, *pooling* e camadas densas, como apresentada na figura 3.5. A estrutura completa da rede é a seguinte:

- Camada de Normalização: A rede começa com uma camada de *batch normalization*, que normaliza os valores de entrada, permitindo que o treino ocorra de forma mais rápida e estável.
- Primeira Camada Convolucional (Conv2D): A primeira camada convolucional utiliza 16 filtros de convolução com um tamanho de 3×3. A função de ativação ReLU [3] é usada para introduzir não-linearidade, ajudando a rede a aprender características das imagens. O uso de uma janela de 3×3 permite capturar pequenas características visuais, como bordas e cores localizadas.
- MaxPooling: Após a primeira convolução, é aplicada uma operação de MaxPooling com uma janela de 2×2, que reduz as dimensões espaciais da imagem (*downsampling*) preservando as informações mais importantes e reduzindo o custo computacional.

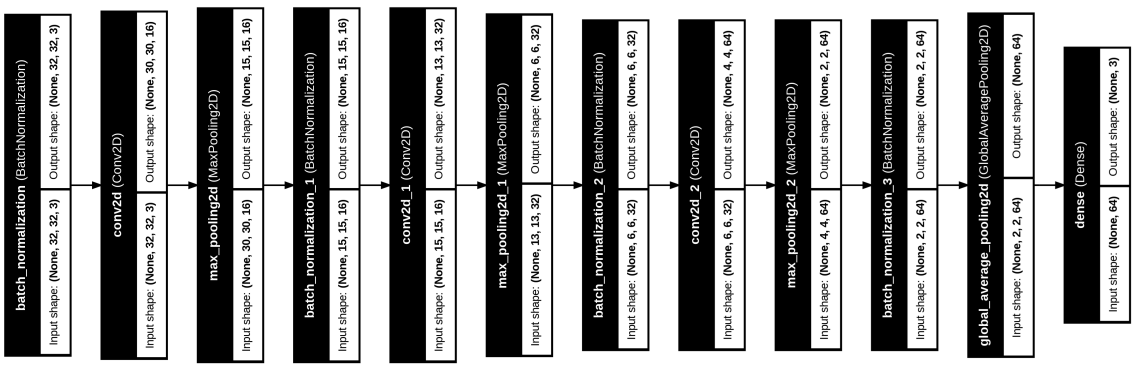


Figura 3.5: *Arquitetura da rede CNN utilizada para classificar as cores do semáforo.*

- Camada de Normalização: Uma segunda camada de *batch normalization* é aplicada para estabilizar e acelerar o processo de treino.

- Segunda Camada Convolucional (Conv2D): A segunda camada convolucional aumenta a profundidade para 32 filtros, também com uma janela de 3×3 e função de ativação ReLU. Isto permite que a rede capture padrões mais complexos, incluindo formas de objetos mais abstratos.

- MaxPooling e Normalização: A operação de MaxPooling é novamente aplicada, seguida de outra camada de normalização para garantir estabilidade no treino.

- Terceira Camada Convolucional (Conv2D): A terceira e última camada convolucional usa 64 filtros de 3×3, capturando características mais detalhadas e específicas das imagens que diferenciem as classes utilizadas.

- Pooling Global (GlobalAveragePooling2D): Após a última camada convolucional, é aplicada uma operação de *Global Average Pooling* que reduz cada mapa de características para um único valor médio, condensando as informações mais relevantes de cada filtro convolucional.

- Camada Densa (*Fully Connected*): Finalmente, a rede termina com uma camada densa que contém três neurónios, correspondentes às três classes (RED, YELLOW e GREEN), utilizando a função de ativação *softmax* [19] para produzir as probabilidades associadas a cada cor do semáforo.

Para treinar a rede é disponibilizado o método "train(train_dir, n_epochs)". Este método permite realizar o treino da rede de forma simples, bastando especificar a diretoria que contém os dados de treino e o número de épocas de treino desejado. O modelo é compilado com o otimizador RMSprop e a função de perda *categorical crossentropy*, adequada para a classificação multiclasse.

Durante a fase de inferência, a rede prevê a cor do semáforo com base nas probabilidades geradas pela função *softmax*. Apenas são consideradas classificações válidas quando a probabilidade está acima de 80%. Caso contrário, a rede atribui a classificação "OFF", indicando que o semáforo pode estar desligado ou que não há certeza suficiente na previsão.

Este método de classificação apresenta vantagens significativas em relação ao primeiro, pois tem a capacidade de se adaptar a diferentes tipos de semáforos e condições ambientais, dependendo do conjunto de treino empregue. Ao fornecer à rede um conjunto diversificado de imagens, incluindo semáforos em diferentes ambientes climáticos e com diferentes características, a CNN tem capacidade de aprender padrões complexos das imagens e classificá-las corretamente em situações mais variadas. Contudo, este método pode exigir mais recursos computacionais, tanto na fase de treino quanto na de inferência, o que deve ser considerado na escolha da aplicação.

3.6 Avaliação de irregularidades RLR

A análise dos vídeos capturados para detetar irregularidades RLR é realizada imagem a imagem, sendo que, para cada imagem do vídeo, o algoritmo executa quatro tarefas principais:

- Classificação da cor dos semáforos presentes nas respetivas regiões definidas.
- Deteção dos objetos presentes, utilizando o modelo de deteção de objetos YOLO.
- Seguimento dos objetos, ligando os seus pontos centrais entre imagens subsequentes para formar uma linha de movimento (trajetória).
- Verificação de possível violação RLR dos veículos detetados.

A utilização do modelo de deteção YOLO pré-treinado com as 80 classes da base de dados COCO [63] leva a que este modelo consiga detetar objetos nas imagens para todas essas classes. Sendo assim, foi necessário aplicar um filtro para considerar apenas as classes de veículos, ou seja, as classes 2-'car', 3-'motorcycle', 5-'bus' e 7-'truck'.

Com os procedimentos de deteção e seguimento dos objetos implementados, é possível obter informações importantes dos veículos, tal como a sua localização na imagem, o identificador único atribuído e a classe detetada. A localização é expressa em caixas delimitadoras, definidas pelas coordenadas (x, y) dos cantos superior esquerdo e inferior direito. A partir destas coordenadas, calcula-se o ponto central da caixa, que é utilizado para verificar se o veículo entrou na ROI de violação definida. Este ponto central é armazenado para cada veículo em cada imagem processada, criando assim um histórico de movimento. Com esta linha de seguimento, o algoritmo desenvolvido pode analisar as interseções entre a trajetória do veículo e as fronteiras da ROI de violação, identificando o momento da interseção e os segmentos da região que foram cruzados.

Importa ter-se presente que nas vias públicas podem transitar veículos noutras faixas de rodagem que não são alvo de avaliação, incluindo situações onde estes entram nas ROI de violação. Para lidar com este desafio, o algoritmo verifica se o veículo entrou na região pela linha de entrada definida, rejeitando aqueles que entram pelas restantes linhas, e garantindo assim que apenas os veículos na direção/sentido corretos sejam considerados como veículos relevantes para análise de possível violação RLR. Por exemplo, na figura 3.6 são considerados como veículos relevantes os que entram pela linha inferior, ou seja, os veículos identificados com os números 6 e 8. Todos os restantes veículos não serão avaliados para possível violação RLR, como é o caso dos veículos identificados pelos números 2, 4 e 9.

Resumindo o funcionamento do algoritmo desenvolvido, no momento em que um veículo interseta a linha de entrada de uma ROI de violação, são registados os seguintes dados:

- O identificador único atribuído ao veículo.
- A linha de movimento realizada pelo veículo.

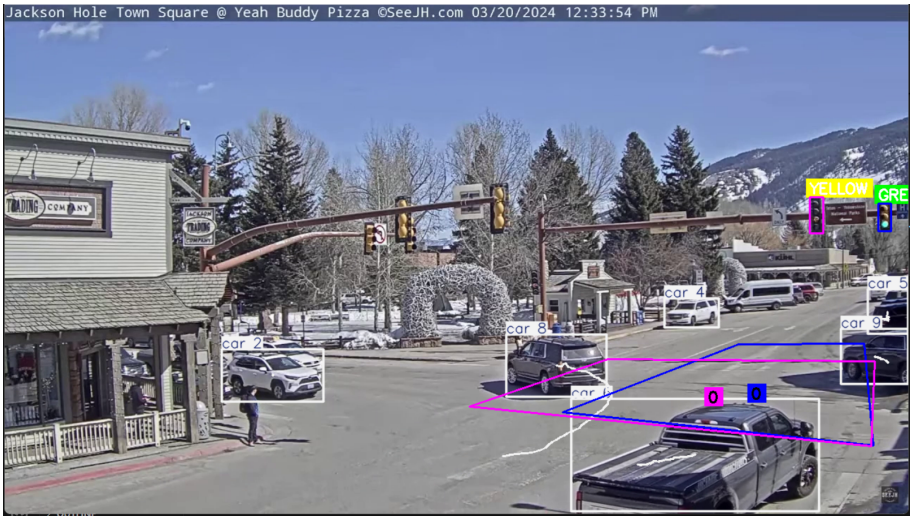


Figura 3.6: *Imagem demonstrativa da interseção entre a trajetória dos veículos e as linhas das ROI.*

- O identificador da ROI de violação.
- A cor do semáforo presente na ROI de semáforo associada à ROI de violação.
- A data e hora em que o evento ocorreu.

Quando o veículo sai dessa ROI de violação é registada a linha pela qual saiu. Nesse instante, o algoritmo verifica se o semáforo correspondente da ROI de violação estava vermelho quando o veículo entrou e se a sua trajetória coincide com a direção controlada pelo par de ROI de violação e semáforo. Se essas condições forem cumpridas, uma violação é atribuída ao veículo e os detalhes da irregularidade, incluindo um vídeo comprovativo, o identificador único atribuído ao veículo e a data e hora do evento são armazenados para análise e revisão futura.

3.7 Otimizações

A deteção de irregularidades no trânsito em tempo real, especialmente recorrendo a sistemas embebidos, exige abordagens otimizadas para maximizar o desempenho e gerir eficientemente os recursos computacionais. Com este propósito, foram implementadas diversas estratégias de otimização no sistema desenvolvido para detetar RLR.

Uma das abordagens adotadas consiste na utilização de estruturas de dados eficientes. O uso de uma estrutura de dados *double-ended queue* (*deque*) para armazenar as imagens dos vídeos de violações num *buffer* circular permite reter apenas um número limitado de imagens na memória, evitando o consumo excessivo de recursos ao processar e guardar vídeos longos. Para além disto, são aplicadas outras estruturas, como *set*, para armazenar identificadores de veículos de forma única, evitando redundâncias e otimizando o processamento.

O processamento de verificação de irregularidades apenas é realizado quando ocorre uma interseção entre as linhas de seguimento dos veículos e as linhas das ROI de violação, e

apenas se essa interseção indicar uma entrada pela linha definida na região. Este processo reduz significativamente a carga de processamento, concentrando os cálculos nas situações mais relevantes.

O algoritmo também beneficia a utilização de bibliotecas otimizadas para operações de visão computacional, como OpenCV e Ultralytics. Estas bibliotecas são conhecidas pela sua eficiência e são projetadas para explorar ao máximo os recursos do sistema, incluindo a aceleração por *hardware* especializado, como GPU, quando disponíveis. Além disso, são concebidas para funcionar em tempo real e com baixa latência, características essenciais para a aplicação proposta.

No caso de sistemas embebidos NVIDIA da gama Jetson, uma utilização eficiente do sistema envolve necessariamente a exportação de modelos de aprendizagem profunda para o formato TensorRT. O TensorRT é um *Software Development Kit* (SDK) criado pela NVIDIA para acelerar a inferência de modelos de aprendizagem profunda em processadores GPU. Este SDK aplica várias técnicas de otimização que transformam o modelo original, ajustando-o para melhorar a rapidez de processamento e reduzir a utilização da memória, sem grande redução na precisão [50, 79]. As principais técnicas utilizadas pelo TensorRT são:

- Fusão de Camadas (*Layer Fusion*): Sempre que possível, são combinadas várias camadas do modelo numa única camada. Por exemplo, uma camada de convolução seguida de uma camada de ativação pode ser unida numa só operação. Isso melhora a rapidez de inferência ao minimizar o acesso à memória.

- Calibração de Precisão (*Precision Calibration*): Esta técnica ajusta o modelo para atingir requisitos de precisão específicos, permitindo o uso de formatos de precisão reduzida, como FP16 e INT8. Esta calibração ajuda a acelerar a inferência, já que operações com precisão reduzida exigem menos tempo de processamento e recursos computacionais do que operações com precisão total (FP32), enquanto ainda mantêm um nível de precisão aceitável. Durante o processo de calibração, o modelo passa por uma análise onde são identificadas as partes que podem ser executadas com precisão reduzida sem comprometer significativamente a qualidade das previsões.

  - FP16 (*Half Precision*): Realiza cálculos com precisão de 16 *bits*, reduzindo a carga computacional e a memória necessária.

  - INT8 (*Integer Precision*): Converte certos cálculos para inteiros de 8 *bits*, reduzindo ainda mais a utilização e consumo de memória, e permitindo inferências mais rápidas. A calibração é ajustada para preservar a precisão do modelo, mesmo com a redução de *bits*.

- Gestão Dinâmica da Memória de Tensores (*Dynamic Tensor Memory Management*): O TensorRT organiza automaticamente o uso da memória dos tensores durante a inferência. Isso significa que a memória é alocada e libertada conforme necessário, evitando desperdícios e maximizando a eficiência em sistemas com recursos limitados.

- Ajuste Automático de *Kernels* (*Kernel Auto-Tuning*): São selecionados automaticamente os melhores *kernels* para as operações do modelo, considerando o *hardware* específico em que está a ser executado. Esse ajuste permite que o modelo use os *kernels* mais rápidos disponíveis para processadores GPU NVIDIA.

O TensorRT é especialmente vantajoso em aplicações de tempo real, onde são necessários tempos de resposta curtos e alto desempenho [50, 79]. No contexto deste projeto, o TensorRT permite otimizar o modelo YOLO, melhorando o tempo de inferência ao utilizar processadores otimizados como os GPU da NVIDIA.

Para utilizar o TensorRT, o modelo YOLO foi primeiramente exportado para um formato compatível, como o ONNX (*Open Neural Network Exchange*). De seguida, o modelo foi convertido para o formato otimizado de inferência com todas as técnicas anteriormente descritas. A conversão para TensorRT é realizada uma vez e, após isso, o modelo convertido pode ser carregado diretamente.

Foram ainda testadas outras possíveis otimizações, como restringir a deteção de classes de veículos pelo modelo YOLO em vez de filtrar as classes ou desativar a visualização gráfica dos resultados com recurso ao OpenCV, com o intuito de tentar reduzir o tempo de processamento do modelo YOLO.

Para além destas otimizações, foi desenvolvida uma versão paralela do algoritmo com recurso a *threading*. Neste caso, o algoritmo utiliza uma *thread* dedicada para cada par de ROI de violação e semáforo, atribuindo a análise de irregularidades de cada par à respetiva *thread*, com uma utilização máxima de seis *threads*. Esta abordagem permite a execução paralela de tarefas, com o intuito de otimizar os recursos disponíveis e acelerar o processamento.

Para assegurar uma coordenação eficiente entre as diferentes *threads*, o algoritmo utiliza mecanismos de sincronização, como filas – em inglês, *Queue* – para a troca segura de dados. Assim é garantido que as informações processadas por uma *thread* estejam disponíveis para outras no momento correto, evitando corrupção de dados e assegurando a integridade do processamento.

3.8 Conclusão

Para concluir este capítulo, é relevante destacar que o sistema proposto apresenta uma solução pioneira de baixo custo para a deteção de violações de passagem de sinal vermelho (RLR), utilizando técnicas de visão computacional em tempo real, projetado para ser implementado em sistemas embebidos de gama média, como a plataforma NVIDIA Jetson Orin Nano. O sistema funciona em duas fases principais: definição de regiões de interesse (ROI) para delimitar as regiões de análise e processamento contínuo das imagens capturadas, examinando a trajetória dos veículos e classificando o estado dos semáforos.

A solução proposta combina técnicas avançadas de deteção e seguimento de objetos com o modelo YOLOv8 para deteção dos veículos e o algoritmo de seguimento ByteTrack

para atribuir identificadores e manter o seguimento dos veículos. Além disso, foram implementados métodos de classificação da cor dos semáforos, com a possibilidade de escolha entre um método baseado em características específicas da imagem ou em redes neuronais convolucionais (CNN).

Por fim, o capítulo também abordou as otimizações realizadas ao algoritmo, de modo a obter resultados desejáveis no menor tempo possível e utilizando recursos limitados.

4 Implementação

Neste capítulo são descritas as ferramentas e bibliotecas utilizadas no desenvolvimento do sistema projetado para deteção de violações de passagem do sinal vermelho (RLR), assim como os processos de criação e seleção das bases de dados utilizadas para o treino e teste dos modelos considerados. São ainda analisados os resultados obtidos com a implementação do algoritmo desenvolvido, avaliando a sua precisão e desempenho em diferentes etapas, como a classificação de cores dos semáforos, a deteção e seguimento de veículos, a deteção de violações RLR e o desempenho computacional do sistema. Essas avaliações empregam métricas que permitem examinar os diferentes aspetos do sistema, fornecendo uma análise abrangente das capacidades do algoritmo em operar nos sistemas disponíveis, destacando as suas forças e limitações.

4.1 Ambientes de desenvolvimento e teste

O algoritmo proposto foi implementado utilizando a linguagem de programação Python na versão 3.9.4 e o desenvolvimento deste *software* foi feito recorrendo ao Ambiente de Desenvolvimento Integrado – em inglês, *Integrated Development Environment* (IDE) – Visual Studio Code. Todo o código desenvolvido encontra-se disponível num repositório público no GitHub [115].

A principal razão para a escolha da linguagem de programação Python reside na sua vasta gama de bibliotecas especializadas, que permitem acelerar a implementação de algoritmos complexos, além de oferecer suporte para exploração de *hardware* especializado como Graphics Processing Unit (GPU), essenciais para conseguir o processamento intensivo de dados em tempo real no contexto deste trabalho.

As bibliotecas utilizadas no desenvolvimento do algoritmo desempenham papéis importantes em diferentes etapas, como o processamento de imagens, o processamento de dados geométricos ou a execução de algoritmos de deteção de objetos. Na tabela 4.1 são apresentadas as principais bibliotecas usadas, destacando as suas versões e funcionalidades.

Tabela 4.1: Bibliotecas Python utilizadas.

Biblioteca	Versão	Descrição
`argparse`	-	Fornece uma interface para analisar argumentos passados via linha de comando.
`collections`	-	Fornece tipos de dados especializados, como listas e dicionários.
`cv2 (OpenCV)`	4.9.0	Biblioteca de visão computacional utilizada para processar e analisar imagens e vídeos.
`numpy`	1.24.2	Biblioteca utilizada para processamento e operações com *arrays* e matrizes multi-dimensionais.
`matplotlib`	3.8.2	Utilizada para criar gráficos, tendo sido empregue para criar gráficos que mostram as características de HSV.
`pathlib`	-	Oferece classes para leitura e processamento de ficheiros e diretorias.
`pickle`	-	Permite guardar e carregar objetos Python de modo eficiente.
`psutil`	5.9.8	Fornece informações sobre processos e utilização de recursos do sistema.
`shapely`	2.0.1	Facilita a utilização de formas geométricas e operações geométricas, como interseções.
`threading`	-	Permite a execução de *threads* em paralelo no código Python.
`time`	-	Inclui funções para processamento de eventos temporais, como medições de tempo de execução ou obtenção da data atual.
`torch`	2.2.1	Biblioteca usada em aprendizagem profunda que fornece suporte à utilização de tensores e redes neuronais.
`ultralytics`	8.2.90	Biblioteca para visão computacional, especialmente usada para deteção de objetos através do modelo YOLO.

No desenvolvimento e teste do protótipo do sistema proposto foram utilizados os recursos de *hardware* ilustrados na figura 4.1, com as seguintes características:

- Portátil ROG Strix G15 (G512LI) equipado com processador CPU Intel i5-10300H, 8 GB de memória RAM e processador GPU NVIDIA GeForce GTX 1650 Ti, operado com o sistema Windows 11.
- Sistema embebido NVIDIA *Developer Kit* Jetson Orin Nano, equipado com processador 6-core Arm® Cortex®-A78AE, 8 GB de memória RAM e processador GPU com arquitetura 1024-core NVIDIA Ampere com 32 Tensor Cores [80], operado com o *software* JetPack 6.1 SDK [77].
- Monitor AOC 27G2U5 com resolução de 1920 × 1080 píxeis e conexões HDMI e DisplayPort (DP) 1.2. Esta última conexão foi utilizada para conectar com o sistema NVIDIA Jetson Orin Nano.
- Camâra PORT Connect *Webcam* com resolução de 1920 × 1080 píxeis.

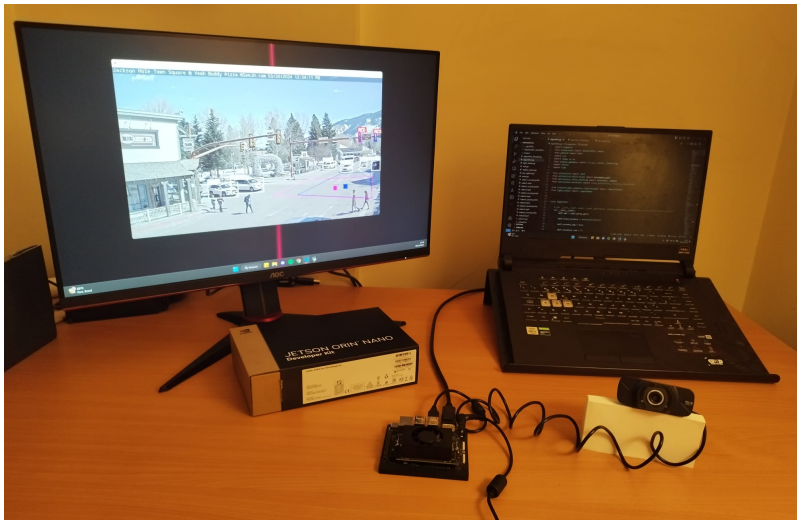


Figura 4.1: *Recursos hardware utilizados para implementar e testar o protótipo do sistema de deteção de RLR. A plataforma NVIDIA com a câmara simula o sistema embebido operando nos cruzamentos.*

4.2 Métricas de avaliação

Por forma a realizar uma análise ao algoritmo proposto e ao desempenho do sistema desenvolvido, foram selecionadas diversas métricas de avaliação frequentemente utilizadas na área [12, 74, 83]. Essas métricas permitem examinar o desempenho do algoritmo na classificação da cor dos semáforos, deteção de objetos, seguimento de objetos, deteção de irregularidades e ainda o desempenho computacional. A escolha das métricas foi feita com base na sua relevância para os objetivos do trabalho, assegurando uma avaliação objetiva e detalhada dos resultados obtidos. No anexo III são apresentadas as expressões matemáticas das métricas consideradas.

4.2.1 Métricas de avaliação do classificador

A avaliação do classificador de cores dos semáforos baseia-se em métricas que permitem medir tanto a eficácia do treino quanto a precisão do modelo.

Durante o treino da rede neuronal, são monitorizados os valores de perda (*loss*) e de eficácia (*accuracy*) [139], que refletem a capacidade da rede aprender a partir dos dados fornecidos. Estes valores ajudam a medir o ajuste do treino do modelo, garantindo que está a convergir corretamente para uma solução eficaz.

Após o treino, o desempenho do modelo é avaliado com base nos dados do conjunto de teste. Ao comparar os resultados previstos pelo modelo com os resultados reais, são calculados os valores de verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN) [109]. Estes valores formam a base para cálculo das métricas de precisão, *recall*, *f1-score* [109] e ainda para a construção de uma matriz de confusão [122], fornecendo uma análise detalhada sobre a capacidade da rede em classificar corretamente as cores dos semáforos.

As métricas precisão e *recall* também são utilizadas para avaliar o desempenho do classificador durante a execução do algoritmo.

4.2.2 Métricas de avaliação de modelos de deteção de objetos

Para monitorizar o treino do modelo YOLO escolheu-se a métrica *mean Average Precision* (mAP) [83, 124], devido à sua capacidade em fornecer uma avaliação do desempenho do modelo quanto à classificação e ajuste da localização dos objetos. Esta métrica, que é a mais utilizada para avaliar a precisão de modelos de deteção de objetos, combina conceitos de precisão, *recall* e *Intersection-over-Union* (IoU) [83] para fornecer uma avaliação abrangente sobre a capacidade do modelo em detetar e localizar objetos.

Relativamente à avaliação do modelo de deteção usando os vídeos de teste, escolheu-se utilizar a métrica eficácia de deteção para simplificar a obtenção de resultados, comparando-se o número de objetos corretamente detetados relativamente ao número total real de objetos.

4.2.3 Métricas de avaliação de modelos de seguimento de objetos

No que diz respeito à avaliação do seguimento de objetos, trata-se de examinar a capacidade do algoritmo em seguir corretamente os objetos ao longo do tempo. Consequentemente, teve-se em consideração os erros mais comuns neste tipo de tarefa [12], como quebras de seguimento do objeto ("Quebras") – quando o modelo deixa de detetar o objeto, voltando a detetá-lo nas imagens seguintes –, atribuição de novo identificador ao objeto ("Trocas ID") – quando são atribuídos identificadores diferentes ao mesmo veículo durante o seguimento e sem sobreposição –, atribuição de múltiplos identificadores ao mesmo objeto ("Múltiplos ID") – quando é atribuído mais do que um identificador ao mesmo objeto simultaneamente – e também mudanças de identificador entre objetos ("Mudanças ID") – quando o algoritmo associa o seguimento de um objeto a outro objeto diferente, ou seja, quando transfere os identificadores entre objetos.

Estas métricas foram escolhidas pela sua simplicidade e objetividade na análise do desempenho do algoritmo, verificando se este consegue não só evitar erros, como a atribuição de múltiplas violações ao mesmo veículo ou deteção de violações incorretas a veículos que não as realizaram, mas sobretudo seguir corretamente os veículos ao longo do vídeo.

4.2.4 Métricas de avaliação de violações RLR

Quanto à análise de violações RLR, o desempenho do algoritmo desenvolvido é avaliado através de métricas que quantificam a sua capacidade em detetar e analisar os veículos relevantes que realizam violações RLR. No contexto deste trabalho, considera-se como veículos relevantes os veículos que circulam nas direções e sentidos monitorizados, estando estes sujeitos à análise de possíveis violações RLR.

A métrica escolhida para avaliar a deteção de violações RLR é a taxa de verdadeiros positivos – em inglês, *True Positive Rate* (TPR) – [109], que mede a proporção de eventos corretamente identificados em relação ao total de eventos reais. Esta métrica permite avaliar o desempenho do algoritmo em identificar corretamente as violações RLR.

4.2.5 Métricas de avaliação do desempenho computacional

Além de se avaliarem as várias etapas do algoritmo e o seu desempenho total, é igualmente importante aferir o seu impacto no desempenho computacional do sistema alvo, pois a eficiência com que o algoritmo processa os dados pode condicionar a sua aplicabilidade quando se coloquem requisitos de operação em tempo real.

As métricas selecionadas para avaliar o desempenho computacional do sistema são as seguintes:

- Tempo de processamento (TP): O tempo de processamento refere-se ao tempo total necessário para que o algoritmo processe um dado conjunto de imagens ou um vídeo completo. Esta métrica é apresentada em segundos (s).
- Tempo de processamento médio (TPM): O tempo de processamento médio indica o tempo necessário, por parte do algoritmo, para processar uma imagem, realizando uma média dos tempos de processamento de todas as imagens processadas. A medida utilizada é milissegundos (ms).
- Tempo de inferência médio (TIM): O tempo de inferência é uma medida do tempo necessário para que o modelo de deteção faça previsões sobre uma única imagem. Na avaliação do algoritmo é apresentada uma média do tempo de inferência em relação ao total de imagens processadas, apresentado em milissegundos (ms). Nas avaliações o tempo associado à tarefa de seguimento é também contabilizado no tempo de inferência.
- Imagens por segundo (FPS): O número de imagens por segundo, ou em inglês *Frame Rate*, indica o número de imagens processadas pelo algoritmo a cada segundo. Esta métrica é medida em *Frames Per Second* (fps).
- Utilização de memória RAM (RAM): A utilização de memória RAM avalia a quantidade de memória utilizada pelo algoritmo durante sua execução. Neste projeto foram avaliadas a utilização de memória RAM média e o máximo da sua utilização, ambas medidas em *megabyte* (MB).

4.3 Base de dados

No contexto de Aprendizagem Automática e Inteligência Artificial, a utilização de conjuntos de dados devidamente anotados e variados é muito importante para o treino e o teste dos algoritmos. Os dados devem ser corretamente identificados, apresentar diversidade suficiente dentro do escopo do problema e incluir um número significativo de amostras que cubram diferentes cenários e condições [24]. Estes fatores influenciam diretamente o desempenho dos algoritmos, permitindo que se generalizem adequadamente e se adaptem a diferentes situações.

No caso de aplicações de deteção de RLR, os conjuntos de dados devem incluir imagens onde estão presentes várias classes de veículos movendo-se em diferentes direções e capturadas

sob diferentes ângulos de visão das câmaras, bem como semáforos em diversos formatos e estados de cor (verde, amarelo e vermelho) e, em ambos os casos, abrangendo diferentes condições de iluminação e cenários meteorológicos. Embora existam diversas bases de dados disponíveis *online* cobrindo uma ampla gama de temas, os conjuntos de dados disponíveis não atendem completamente a estes requisitos, tornando necessária a criação de bases de dados próprias para este trabalho. Assim, para obter os dados e criar as bases de dados, foram utilizados vídeos de estradas localizadas nos Estados Unidos da América (EUA) e no Canadá, disponibilizados publicamente em *websites* próprios [44, 89] ou plataformas como o YouTube [104] e Roboflow Universe [8, 10, 22, 75].

4.3.1 Base de dados para deteção de RLR

Para testar o desempenho do algoritmo desenvolvido [115] na deteção de RLR foi criada uma base de dados nova [113]. Esta base de dados contém vídeos de situações variadas, incluindo cenários noturnos e condições climatéricas adversas, como neve ou chuva. No total, foram reunidos vinte e quatro vídeos obtidos através de câmaras que capturam conjuntamente as estradas rodoviárias e os semáforos que as controlam, incluindo exemplos onde ocorrem irregularidades relacionadas com RLR.

Alguns dos vídeos explorados foram obtidos a partir da plataforma Youtube, em particular de estradas localizadas em Jackson Hole no estado de Wyoming nos EUA, onde se encontra uma câmara a filmar em direto um cruzamento central da cidade [44]. Para além destes vídeos, consideraram-se outros relativos a estradas de Canmore Alberta, um município no Canadá [89]. Ambas as câmaras destas localizações possuem vídeos de alta resolução e com o posicionamento pretendido, como ilustrado nas figuras 4.2 e 4.3, sendo adequadas tanto para o treino dos modelos quanto para o teste do algoritmo.

Embora estes vídeos sejam apropriados para testar o algoritmo na deteção de veículos e classificação da cor do semáforo, não são suficientes para avaliar a capacidade do sistema na identificação de irregularidades RLR. Isso ocorre porque, sendo provenientes de câmaras em tempo real, este tipo de eventos são raros. Para permitir uma avaliação mais completa

Figura 4.2: *Imagem obtida através das câmaras em direto, em Jackson Hole, EUA [44].*

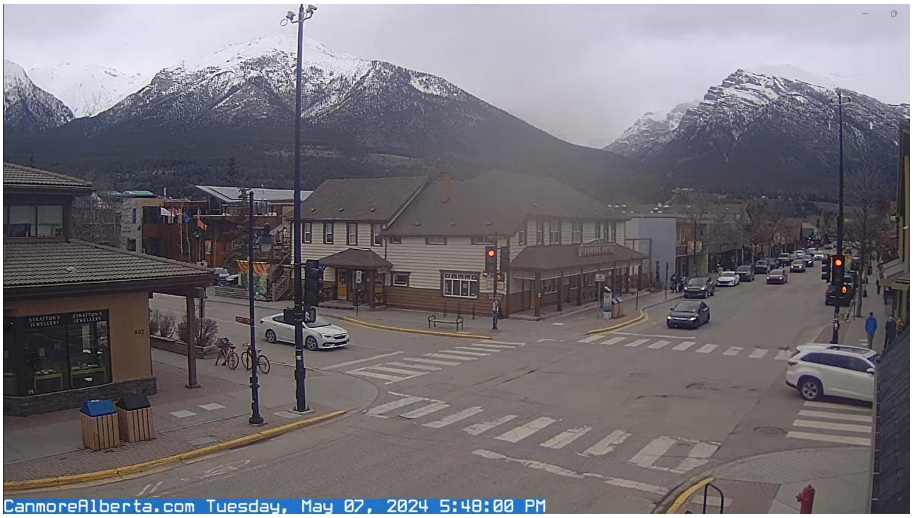


Figura 4.3: *Imagem obtida através das câmaras em direto, em Canmore, Canadá [89].*

do algoritmo em situações de passagem do sinal vermelho, foi necessário recorrer ainda a outros vídeos que apresentassem exemplos claros de violações de RLR.

Assim, foram adquiridos vídeos captados nas estradas Friant e Shepherd, localizadas no condado de Fresno, Califórnia, EUA, exibindo exclusivamente situações onde os sinais luminosos com luz vermelha foram desrespeitados [104].

4.3.2 Base de dados para classificação do semáforo

A partir dos vídeos referidos na subsecção 4.3.1, foram obtidas imagens de diversos semáforos apresentando diferentes cores, com o objetivo de construir uma base de dados que servisse para treinar e testar os classificadores automáticos da cor do semáforo.

De modo a aumentar o número de dados, foram incluídas imagens adicionais da base de dados "Traffic Light Dataset" disponibilizada na plataforma Roboflow Universe [8]. Este aumento no número de dados garante uma maior diversidade de exemplos, contribuindo para a generalização do modelo, no caso do método de classificação com base numa CNN.

Note-se que quando o mesmo conjunto de imagens é utilizado tanto no treino quanto no teste de uma CNN, a rede pode simplesmente "memorizar" essas imagens, levando a um desempenho artificialmente elevado no conjunto de treino e teste, sem refletir a sua verdadeira capacidade de generalização. O objetivo do teste é avaliar como o modelo se comporta em cenários desconhecidos, e, portanto, é importante que as imagens utilizadas para testar a rede sejam diferentes das usadas durante o treino [41, 132].

A base de dados criada para classificação de semáforos [114] compreende uma pasta de treino com 1161 imagens, onde 464 são de semáforos com luz verde, 247 com luz amarela e 450 com luz vermelha. Contém também uma pasta de teste com 222 imagens, sendo 146 de semáforos com luz verde, 18 com luz amarela e 58 com luz vermelha. Na figura 4.4 são apresentados três exemplos de imagens de semáforos em diferentes estados (verde, amarelo e vermelho), presentes na base de dados.

(a) *Semáforo com cor verde.* (b) *Semáforo com cor amarela.* (c) *Semáforo com cor vermelha.*

Figura 4.4: *Exemplos de imagens de semáforos presentes na base de dados [114].*

4.3.3 Base de dados para deteção de objetos

Para melhorar o desempenho do modelo de deteção de objetos YOLO é recomendável treinar-se o modelo com dados referentes ao problema alvo. Desta forma, na maioria dos casos, é possível obter bons resultados sem a necessidade de ajustes nos modelos ou nas configurações de treino, desde que o conjunto de dados seja suficientemente grande e bem anotado [36].

Caso os resultados iniciais não sejam satisfatórios, existem medidas que podem ser tomadas para melhorar os dados de treino e otimizar o treino do modelo YOLO [36], tais como:

- Imagens por classe: É recomendado um mínimo de 1500 imagens por classe.
- Anotações por classe: É recomendado um mínimo de 10.000 objetos anotados por classe.
- Variedade de imagens: As imagens devem representar o ambiente real da aplicação, considerando diferentes horários, condições climatéricas, iluminação, ângulos, e outras características.
- Consistência e precisão nas anotações: Todas as anotações devem estar corretamente identificadas, com as caixas delimitadoras ajustadas ao objeto e sem espaços vazios.
- Verificação das anotações: Deve-se realizar uma verificação e correção das anotações, visualizando amostras durante o início do treino.
- Imagens de fundo: Deve-se utilizar entre 0% e 10% de imagens de cenários sem objetos para reduzir falsos positivos.

A aplicação destes princípios neste trabalho resulta que a base de dados deve ser composta por imagens de veículos capturados em diferentes ângulos e posições. Além disso, os veículos nessas imagens devem estar devidamente identificados com as suas classes e com as coordenadas das caixas delimitadoras corretamente selecionadas e ajustadas.

Com este propósito, foram exploradas bases de dados com estas características disponibilizadas na plataforma Roboflow Universe [103], por ser uma plataforma dedicada a fornecer publicamente bases de dados na área de visão computacional e com anotações orientadas para diferentes versões do modelo YOLO. No estudo realizado foram encontradas três bases de dados com foco na deteção de veículos: 'N_O_3' [75], 'cctv_detection' [10] e '*Cars*

detecting and how many' [22]. Além disso, também foi explorada a base de dados COCO, possibilitando uma análise comparativa dos seus conteúdos em relação às anteriores.

4.3.3.1 Base de dados da plataforma Roboflow Universe com foco na deteção de veículos

A base de dados denominada por 'N_O_3' contém 10.053 imagens, com 70 diferentes classes de objetos de veículos e de sinais regulamentares de trânsito, tais como carro, peão, mota, sinais de limite de velocidade ou de sentido proibido.

As imagens disponibilizadas têm alturas e larguras variadas e apresentam alta resolução. No entanto, apesar de os objetos estarem devidamente anotados, a maior parte das imagens deste conjunto apresenta objetos com dimensões reduzidas, como ilustrado na figura 4.5.

Por outro lado, as bases de dados 'cctv_detection' e '*Cars detecting and how many*' contemplam um menor número de imagens, todas com uma dimensão fixa de 640×640 píxeis. A primeira base de dados contém 1.729 imagens de veículos, com duas classes identificadas, '*car*' e '*motorbike*', enquanto a segunda possui 3.496 imagens anotadas apenas com a classe 'car'.

As imagens contidas em ambas as bases de dados apresentam veículos capturados de diferentes ângulos e a curta distância. Embora estas imagens apresentem os objetos com maiores dimensões, possuem uma resolução inferior relativamente à base de dados 'N_O_3' e a maioria exibe apenas um veículo por imagem, como ilustrado nas figuras 4.6 e 4.7.

(a) (b) (c) (d)

Figura 4.5: *Exemplos de imagens contidas na base de dados 'N_O_3' [75].*

(a) (b)

Figura 4.6: *Exemplos de imagens contidas na base de dados 'cctv_ detection' [10].*

(a) (b)

Figura 4.7: *Exemplos de imagens contidas na base de dados 'Cars detecting and how many' [22].*

4.3.3.2 Base de dados COCO

A base de dados *Common Objects in Context* (COCO) é uma das principais bases de dados utilizadas para pesquisa e desenvolvimento de algoritmos de visão computacional, especialmente para as tarefas de deteção de objetos, segmentação e geração de legendas automáticas para imagens [63].

Esta base de dados é caracterizada pela sua diversidade de dados e anotações, possuindo aproximadamente 330 mil imagens. Dessas, cerca de 200 mil contêm anotações detalhadas para as tarefas descritas. Contempla 80 classes de objetos, que variam desde objetos comuns como carros, bicicletas e animais, até objetos mais específicos como guarda-chuvas, malas e equipamentos desportivos. As imagens presentes são diversificadas, incluindo cenários complexos com múltiplos objetos e ambientes variados, demonstradas nos exemplos da figura 4.8.

Cada uma das imagens que contém anotações está devidamente anotada com as caixas delimitadoras para a tarefa de deteção de objetos, máscaras de segmentação para delimitar a segmentação e legendas descritivas para a tarefa de geração de texto automático, de cada objeto.

Comparando os dados desta base de dados com os dados das bases referidas anteriormente, nota-se que as imagens presentes na base de dados COCO apresentam maior quantidade

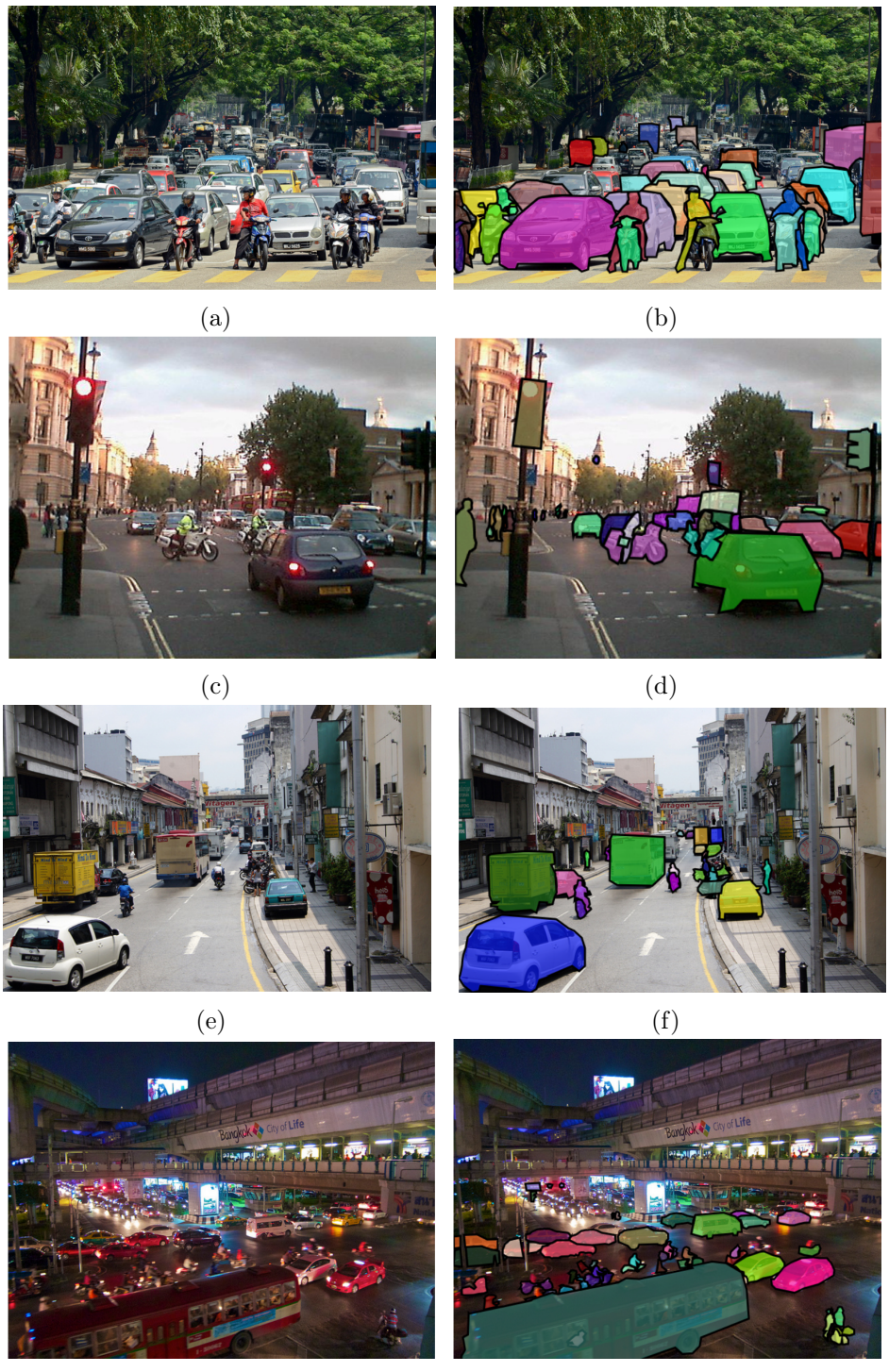


Figura 4.8: *Exemplos de imagens contidas na base de dados COCO [63]. À esquerda encontram-se as imagens originais e à direita as imagens com as anotações dos objetos.*

e diversidade de exemplos com diferentes cenários e ambientes, como imagens capturadas durante a noite, ilustradas nas subfiguras 4.8g e 4.8h. As imagens apresentam uma dimensão fixa igual à utilizada pelo modelo YOLO por padrão e apresentam maior resolução comparativamente às bases de dados 'cctv_detection' e '*Cars detecting and how many*'.

4.4 Treino dos modelos

Com as bases de dados selecionadas e preparadas, procedeu-se com o treino dos modelos utilizados no sistema proposto, direcionando o foco para a classificação das cores do semáforo com base numa CNN e para a deteção de veículos com o modelo YOLOv8. Os dados de treino dessas bases foram utilizados para ajustar os modelos, e o desempenho foi avaliado tanto nos conjuntos de teste quanto nos vídeos reais coletados [113], representativos do cenário de aplicação.

4.4.1 Treino do modelo de classificação

Para treinar a CNN responsável pela classificação das cores do semáforo utilizou-se o conjunto de dados da pasta de treino da base de dados criada para o efeito [114], separando 20% dos dados para validação da rede. Esta separação permite obter uma perspetiva do ajuste dos pesos da rede, ao avaliar os valores da classificação prevista nos dados de validação.

Na figura 4.9 é possível examinar a evolução do desempenho da rede ao longo das 100 épocas de treino. O gráfico mostra quatro curvas: perda do treino (*Training Loss*), perda da validação (*Validation Loss*), eficácia do treino (*Training Accuracy*) e eficácia da validação (*Validation Accuracy*). A análise destas curvas oferece uma visão sobre a convergência da rede e a sua capacidade de generalização.

A duração de 100 épocas foi escolhida como valor experimental inicial. Contudo, ao avaliar o gráfico da figura 4.9, é possível perceber que o treino estabiliza por volta da décima quinta época, indicando que um número menor de épocas seria suficiente.

A análise da figura 4.9 também permite verificar que tanto as perdas de treino como de validação diminuem rapidamente nas primeiras épocas, estabilizando-se em valores similares,

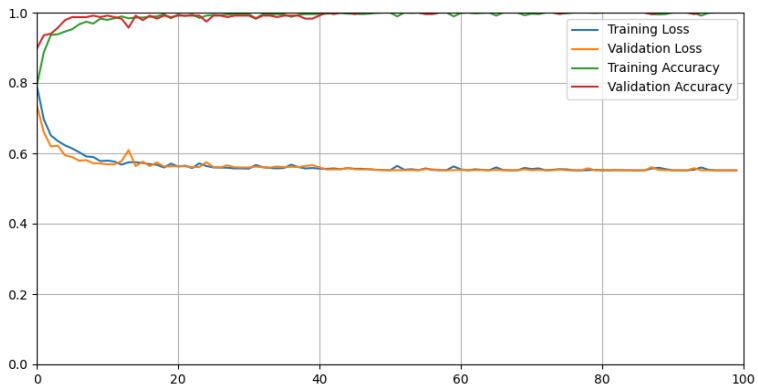


Figura 4.9: *Evolução do desempenho da rede quando treinada durante 100 épocas.*

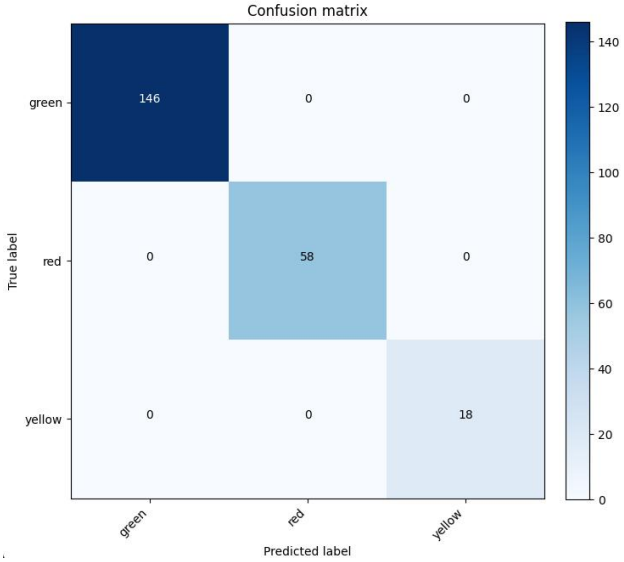


Figura 4.10: *Matriz de confusão do teste do classificador com base numa rede neuronal convolucional.*

o que indica que a rede consegue aprender as características dos dados e sugere que o modelo não está com sobreaprendizagem. Pode-se, portanto, inferir que a rede deverá conseguir generalizar bem para dados não vistos.

A eficácia do treino e da validação atingem valores próximos de 1.0 (ou 100%), o que significa que a rede consegue classificar corretamente a maioria dos dados do conjunto de treino e de validação após cerca de dez épocas. A eficácia alta junto com perdas estáveis demonstra que a rede treinada está bem ajustada aos dados.

Para confirmar o desempenho da rede treinada, foi avaliada a sua classificação nos dados do conjunto de teste [114], cujos resultados estão presentes na tabela 4.2 e na matriz de confusão da figura 4.10. Estes resultados mostram que a rede atingiu uma precisão de 100% na classificação de todas as classes, o que comprova a eficácia da rede neuronal convolucional treinada.

Tabela 4.2: Desempenho do classificador com base numa rede neuronal convolucional no conjunto de dados de teste.

Classe	**Precisão**	***Recall***	***F1-Score***	**Nº de dados**
GREEN	1.00	1.00	1.00	146
YELLOW	1.00	1.00	1.00	18
RED	1.00	1.00	1.00	58

4.4.2 Treino do modelo de deteção de objetos

O modelo YOLOv8 foi treinado com as bases de dados descritas na subsecção 4.3.3.1, utilizando as configurações padrão do YOLO, durante 20 épocas. Este valor foi utilizado inicialmente, não se obtendo melhores resultados no decorrer de um maior número.

Na base de dados 'N_O_3', os resultados do treino do modelo apresentaram valores de

mAP entre 0.196, para a variante com menos parâmetros, e 0.482, para a variante com mais parâmetros. Apesar destes resultados levarem a crer que seriam necessárias mais épocas no treino para se conseguir que o modelo se ajustasse melhor aos dados, a continuação do seu treino por outras 20 épocas não revelou melhorias significativas. Esse comportamento deve-se, em parte, à presença de múltiplas classes (70 classes) e à qualidade das anotações, que frequentemente incluem objetos de dimensões mínimas, como ilustrado na figura 4.5.

Ao testar este modelo treinado nos vídeos de teste adquiridos (que simulam cenários realistas do sistema a ser implementado) [113], foi constatado que o modelo era capaz de detetar veículos menores (mais distantes), mas falhava na deteção de veículos maiores (mais próximos). Essa limitação pode estar relacionada com a predominância de anotações de objetos de pequenas dimensões nos dados de treino e a variação nas dimensões das imagens. Como o modelo YOLO, por padrão, transforma os dados para dimensões de 640×640 píxeis, é aconselhado utilizar imagens do mesmo tamanho no treino, teste e inferência, para obter melhores resultados.

O treino com as bases de dados 'cctv_detection' e '*Cars detecting and how many*' mostrou valores elevados de mAP, superiores a 0.900, nos conjuntos de treino e validação para várias variantes do modelo. Porém, ao testá-los nos vídeos de teste, verificou-se que o modelo não conseguiu detetar nenhum dos veículos presentes nas imagens. Essa limitação pode estar associada à baixa qualidade das imagens, ao número reduzido de classes utilizadas e ainda por apenas apresentarem uma anotação na maioria das imagens, como demonstram as figuras 4.6 e 4.7. Note-se que as bases de dados 'cctv_detection' e '*Cars detecting and how many*', para além de possuírem um número reduzido de imagens e classes identificadas em relação ao sugerido, também carecem de variedade e de cenários realistas que simulem adequadamente o sistema proposto, no qual são capturadas imagens da via pública com múltiplos veículos em trânsito.

Por outro lado, a versão pré-treinada do modelo YOLOv8, disponibilizada pela Ultralytics [51] e ajustada com a base de dados COCO, apresenta valores de mAP de 53.9 para a variante com mais parâmetros [51]. Ao testar este modelo pré-treinado nos vídeos de teste observou-se que o modelo apresentou um desempenho significativamente superior na deteção de veículos em comparação com os modelos treinados nas bases de dados anteriores. Mesmo utilizando a variante com menos parâmetros, o modelo foi capaz de identificar objetos com maior precisão e consistência na identificação dos veículos, destacando a importância da qualidade e diversidade dos dados para treinar o YOLOv8.

Diante disso, optou-se por utilizar o modelo YOLOv8 com os pesos pré-treinados através da base de dados COCO nos testes subsequentes do sistema.

4.5 Conclusão

Neste capítulo foram introduzidas as métricas utilizadas para avaliar a solução proposta para deteção de violações de passagem do sinal vermelho (RLR), incluindo o algoritmo e os modelos utilizados, bem como a preparação e criação de bases de dados.

Dada a inexistência de conjuntos de dados públicos que atendessem às necessidades deste trabalho, foram criadas bases de dados próprias para treinar e testar os modelos utilizados no algoritmo e ainda avaliar a capacidade de deteção de violações de passagem de sinal vermelho. A elaboração da base de dados de avaliação do algoritmo focou-se na simulação de cenários de trânsito reais, incluindo vídeos de estradas que englobam o tráfego de veículos e os semáforos correspondentes, visando criar um sistema pioneiro e simular situações próximas das reais. A partir desta base de dados, foram extraídas imagens dos semáforos presentes nos vídeos para desenvolver a base de dados de treino e teste dos classificadores, complementadas com imagens de outra base de dados pública para melhorar a generalização do modelo. Para treinar o modelo de deteção de objetos YOLOv8, foram inicialmente utilizadas três bases de dados com imagens de veículos da plataforma Roboflow Universe. Porém, o modelo pré-treinado com a base de dados COCO apresentou melhores resultados, optando-se por utilizá-lo no desenvolvimento do algoritmo.

5 Resultados experimentais

Neste capítulo são apresentados os resultados obtidos pela solução desenvolvida para detetar violações de passagem do sinal vermelho usando dois sistemas distintos, um computador portátil equipado com um processador Intel i5-10300H e um sistema embebido NVIDIA *Developer Kit* Jetson Orin Nano. Discute-se ainda os vídeos especificamente selecionados para testar o desempenho dos sistemas em diferentes condições, avaliando-se também a capacidade do algoritmo proposto em detetar e seguir corretamente os veículos, a precisão na classificação das cores dos semáforos e, principalmente, o desempenho na deteção de violações RLR.

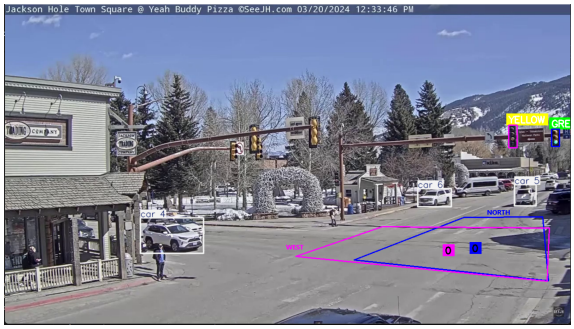
5.1 Seleção de dados para análise

Para avaliar o desempenho do algoritmo proposto para deteção de irregularidades do tipo RLR, foram selecionados cinco vídeos distintos da base de dados descrita na secção 4.3.1 [113]. De modo a abranger diversas condições, escolheu-se um vídeo diurno de Jackson Hole [44], outro de Canmore [89] e ainda um vídeo noturno de Jackson Hole. Como nenhum desses vídeos contém violações RLR, consideraram-se também dois vídeos da região de Fresno [104], que incluem especificamente eventos RLR e, desse modo, permitem avaliar a capacidade do algoritmo em detetar este tipo de incidentes.

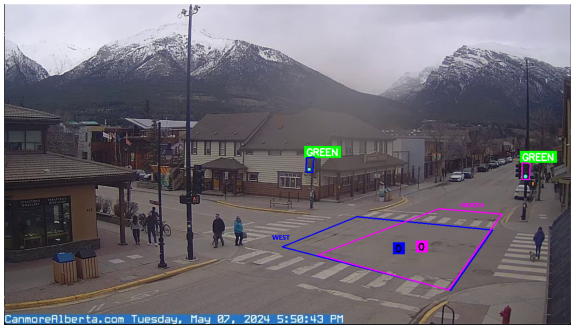
Os vídeos escolhidos incluem vários tipos de veículos e diferentes sinais luminosos acesos com cores distintas. Além disso, as câmaras que capturaram esses vídeos estão posicionadas de modo a abranger a via pública e os semáforos presentes. Todos os vídeos têm uma resolução de 1280×720 píxeis e uma resolução temporal de 30 imagens por segundo. Mais detalhes sobre estes vídeos são fornecidos na tabela 5.1.

Tabela 5.1: Características dos vídeos selecionados [113].

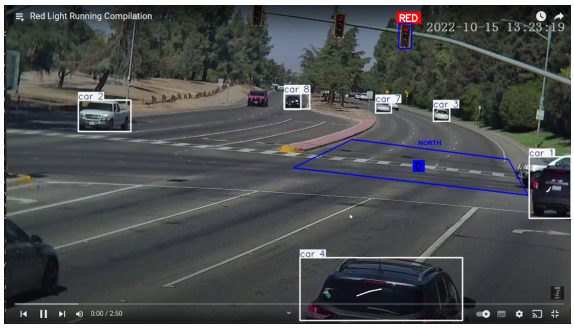
Vídeo	A	B	C	D	E
Duração (segundos)	62	115	8	17	96
N.º de imagens	1861	3459	251	536	2902
Localização	Jackson Hole, EUA	Canmore, Canadá	Fresno, EUA	Fresno, EUA	Jackson Hole, EUA
Horário	Diurno	Diurno	Diurno	Diurno	Noite
Interseção	Cruzamento	Cruzamento	Entroncamento	Entroncamento	Cruzamento
N.º de violações	0	0	1	1	0
N.º semáforos ROI	2	2	1	1	2



(a) *Regiões definidas no vídeo A.*

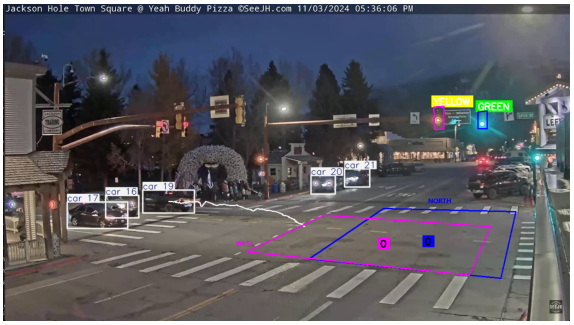


(b) *Regiões definidas no vídeo B.*



(c) *Região definida no vídeo C.*

(d) *Região definida no vídeo D.*



(e) *Regiões definidas no vídeo E.*

Figura 5.1: *Imagem das regiões definidas nos vídeos selecionados para avaliação do algoritmo.*

Para cada vídeo, foram definidas as regiões de interesse adequadas, conforme ilustrado pelas caixas de cores rosa e azul, na figura 5.1.

5.2 Avaliação dos classificadores

Atendendo a que no algoritmo proposto se faz a classificação da cor dos semáforos para cada imagem do vídeo, os dois classificadores desenvolvidos – classificador baseado nas características específicas da imagem (ver secção 3.5.1) e classificador baseado numa rede CNN (ver secção 3.5.2) – foram avaliados considerando as suas previsões para cada umas das imagens de cada um dos cinco vídeos selecionados. Porém, nos vídeos A e E há um semáforo que não foi tido em conta na avaliação realizada.

Trata-se de um semáforo com quatro luzes, em que as duas luzes intermédias são amarelas e uma delas está no estado intermitente, como ilustrado na figura 5.2. Este semáforo não foi contabilizado pelo facto dos métodos de classificação utilizados não consideraram o estado

Figura 5.2: *Imagem do semáforo não utilizado para avaliação dos classificadores.*

intermitente das luzes nem semáforos com mais de três cores para o classificador baseado em características da imagem, que é a situação típica verificada em Portugal.

Para melhor aferir os recursos computacionais exigidos durante o processamento do algoritmo com cada classificador, os testes foram realizados tanto no computador portátil como no sistema embebido NVIDIA *Developer Kit* Jetson Orin Nano (ver secção 4.1) utilizando o modelo YOLOv8n em conjunto com o algoritmo de seguimento de objetos ByteTrack. Os elementos computacionais utilizados em cada sistema foram os seguintes: *i*) apenas o processador Intel i5-10300H (CPU) do computador portátil; *ii*) o CPU e o GPU no sistema embebido da NVIDIA.

5.2.1 Avaliação do classificador baseado em características da imagem

Os resultados obtidos no teste do classificador baseado em características da imagem, para cada cor exibida pelos semáforos, são apresentados nas tabelas 5.2, 5.3 e 5.4.

Analisando os valores constantes destas tabelas observa-se que o classificador apresenta precisão e *recall* de 100% na classificação das cores dos semáforos presentes nos vídeos capturados durante o dia (vídeos A, B, C e D). Nos casos em que não são apresentados dados, tal deve-se ao facto de o semáforo não acender todas as cores nos vídeos. Já no caso do vídeo E, capturado à noite, o classificador apresentou erros na classificação da cor amarela, atribuindo-lhe incorretamente as classes OFF, ou GREEN e RED numa única imagem.

Tabela 5.2: Classificação com classe GREEN das cores dos semáforos presentes em cada vídeo, utilizando o classificador baseado nas características da imagem.

Vídeo	**VP**	**FP**	**FN**	**Precisão**	**Recall**
A	1016	0	0	100%	100%
B	3078	0	0	100%	100%
C	-	0	-	-	-
D	-	0	-	-	-
E	2020	1	0	99.95%	100%

Tabela 5.3: Classificação com classe YELLOW das cores dos semáforos presentes em cada vídeo, utilizando o classificador baseado nas características da imagem.

Vídeo	**VP**	**FP**	**FN**	**Precisão**	**Recall**
A	95	0	0	100%	100%
B	422	0	0	100%	100%
C	-	0	-	-	-
D	-	0	-	-	-
E	0	0	95	0%	0%

Tabela 5.4: Classificação com classe RED das cores dos semáforos presentes em cada vídeo, utilizando o classificador baseado nas características da imagem.

Vídeo	VP	FP	FN	Precisão	Recall
A	750	0	0	100%	100%
B	3420	0	0	100%	100%
C	251	0	0	100%	100%
D	536	0	0	100%	100%
E	787	1	0	99.87%	100%

Apesar de não se avaliar o semáforo com quatro luzes, no vídeo A observou-se que este classificador transitava entre as classes OFF e RED quando o semáforo acendia a luz intermédia intermitente com cor amarela, e atribuía a classe RED quando apresentava a luz intermédia não intermitente. Quanto às restantes luzes do semáforo, o classificador consegue realizar uma classificação correta da cor apresentada. No vídeo E, a classificação deste semáforo alterna entre OFF, YELLOW e GREEN.

Nas tabelas 5.5 e 5.6 apresentam-se os valores de desempenho conseguidos pelos dois sistemas. Para processar os vídeos em tempo real, o sistema teria de ter um tempo de processamento de 33,3 ms por imagem, e como se pode observar, os tempos obtidos são bastante superiores a esse objetivo. Apresentam-se ainda os valores de pico máximo de utilização da memória RAM durante o processamento do algoritmo, que foi 1.213,0 MB no caso da execução no computador portátil e 1.997,8 MB para o sistema embebido NVIDIA Jetson. Estes valores indicam que o sistema alvo deve dispor de, pelo menos, 2.000 MB de memória RAM para conseguir processar o algoritmo desenvolvido, no caso de se usar o classificador baseado nas características da imagem.

Tabela 5.5: Desempenho computacional no computador portátil durante a execução do algoritmo utilizando o classificador baseado nas características da imagem.

Vídeo	RAM máxima [MB]	RAM média [MB]	Tempo de Processamento Médio [ms]
A	1213.0	1145.4	330.2
B	1212.7	1149.6	340.0
C	993.4	704.2	265.6
D	1047.4	895.1	229.4
E	1163.4	1032.5	334.7

Tabela 5.6: Desempenho computacional no sistema embebido NVIDIA Jetson durante a execução do algoritmo utilizando o classificador baseado nas características da imagem.

Vídeo	RAM máxima [MB]	RAM média [MB]	Tempo de Processamento Médio [ms]
A	1949.1	1882.6	102.3
B	1997.8	1951.3	100.1
C	1821.7	1487.3	111.0
D	1972.8	1739.6	93.7
E	1947.3	1904.4	98.9

5.2.2 Avaliação do classificador baseado em rede neuronal convolucional

Os resultados obtidos na avaliação do classificador baseado em CNN constam nas tabelas 5.7, 5.8 e 5.9.

Como se pode observar, este classificador apresenta uma precisão e *recall* de 100% na classificação de todos os semáforos que acendem de forma não intermitente. Contudo, o classificador não mantém esta precisão ao incluir-se o semáforo com estado intermitente dos vídeos A e E. No vídeo A, a luz amarela intermitente é sempre considerada como YELLOW, não chegando a ser classificada como OFF quando o semáforo não emite nenhuma cor. No vídeo E, esta luz é classificada alternando entre todas as diferentes cores (GREEN, YELLOW e RED). A classe GREEN é prevista quando o semáforo não emite luz, a classe YELLOW quando emite a luz amarela e a classe RED quando existe a transição de desligado para amarelo.

Durante o processamento, foi analisado o desempenho computacional dos dois sistemas considerados, sendo os resultados dessa análise apresentados nas tabelas 5.10 e 5.11.

Com este classificador, o desempenho computacional melhorou para ambos os sistemas em termos de tempo de processamento, conseguindo-se ganhos entre 30 ms e 120 ms para a utilização do computador portátil e entre 20 ms e 30 ms quando se utilizou o sistema

Tabela 5.7: Classificação com classe '*GREEN*' das cores dos semáforos presentes em cada vídeo utilizando o classificador baseado em CNN.

Vídeo	**VP**	**FP**	**FN**	**Precisão**	**Recall**
A	1016	0	0	100%	100%
B	3076	0	0	100%	100%
C	0	0	0	0%	0%
D	0	0	0	0%	0%
E	2020	0	0	100%	100%

Tabela 5.8: Classificação com classe '*YELLOW*' das cores dos semáforos presentes em cada vídeo utilizando o classificador baseado em CNN.

Vídeo	**VP**	**FP**	**FN**	**Precisão**	**Recall**
A	95	0	0	100%	100%
B	422	0	0	100%	100%
C	0	0	0	0%	0%
D	0	0	0	0%	0%
E	95	0	0	100%	100%

Tabela 5.9: Classificação com classe '*RED*' das cores dos semáforos presentes em cada vídeo utilizando o classificador baseado em CNN.

Vídeo	**VP**	**FP**	**FN**	**Precisão**	**Recall**
A	750	0	0	100%	100%
B	3420	0	0	100%	100%
C	251	0	0	100%	100%
D	536	0	0	100%	100%
E	787	0	0	100%	100%

Tabela 5.10: Desempenho computacional no computador portátil durante a execução do algoritmo utilizando o classificador baseado em CNN.

Vídeo	RAM máxima [MB]	RAM média [MB]	Tempo de Processamento Médio [ms]
A	1215.0	1147.1	247.0
B	1209.3	1149.1	224.2
C	1053.1	727.1	235.5
D	1206.4	984.5	197.3
E	1139.3	1097.1	214.1

Tabela 5.11: Desempenho computacional no sistema NVIDIA Jetson durante a execução do algoritmo utilizando o classificador baseado em CNN.

Vídeo	RAM máxima [MB]	RAM média [MB]	Tempo de Processamento Médio [ms]
A	1955.7	1889.6	78.1
B	2002.7	1955.7	71.8
C	1825.5	1490.8	87.4
E	1953.0	1739.6	73.9
D	1984.9	1914.6	74.8

embebido da NVIDIA. Note-se que tempos menores correspondem a vídeos com menor número de imagens e onde se avalia apenas um semáforo sempre no estado vermelho, enquanto os tempos maiores reportam aos vídeos com maior número de imagens e dois semáforos que apresentam os vários estados possíveis. Por outro lado, os valores máximos de utilização da memória RAM registados para os dois sistemas foram 1.215,0 MB e 2.002,7 MB, indicando que, para utilizar este classificador, poderá já não ser suficiente utilizar uma memória RAM de 2.000 MB com o uso de GPU.

Comparando os resultados obtidos para os dois métodos de classificação, pode-se concluir que o classificador baseado em CNN se destaca pela precisão, acertando consistentemente na classificação de todas as cores dos semáforos analisados. No que se refere ao desempenho computacional, embora o método baseado nas características da imagem utilize menos memória RAM na maioria dos casos, o tempo de processamento médio de cada imagem é inferior com o classificador baseado em CNN. Apesar de este último requerer um pouco mais de memória RAM, a diferença é pouco significativa, sendo compensada pelos tempos de processamento consideravelmente mais curtos.

Estes resultados evidenciam as vantagens de se utilizar uma rede neuronal convolucional para a classificação das cores dos semáforos. Ao ser treinada com uma diversidade de dados, a CNN é capaz de extrair características importantes das imagens, generalizando melhor para diferentes tipos e condições de semáforos. Como um dos objetivos deste trabalho é garantir uma classificação precisa de todos os semáforos com execução do algoritmo no menor tempo possível e como os sistemas usados oferecem memória RAM suficiente para suportar ambos os métodos, optou-se por adotar o classificador baseado em CNN nos testes subsequentes.

5.3 Avaliação do modelo de deteção e seguimento de objetos

Outra funcionalidade essencial para o correto funcionamento do algoritmo são os modelos de deteção e seguimento de objetos. No caso do modelo de deteção de objetos, este deve classificar corretamente os objetos como veículos e obter a sua localização na imagem. É através desta localização que o algoritmo de seguimento realiza as suas tarefas, como a atribuição de identificadores e o acompanhamento do objeto, sendo estes aspetos muito importantes para a correta deteção de violações.

A avaliação dos modelos de deteção e seguimento de objetos é complexa, especialmente ao utilizar-se vídeos como material de teste. De forma a avaliar efetivamente o modelo de deteção, os veículos captados nos vídeos de teste foram divididos em duas categorias: veículos relevantes e veículos não relevantes. Veículos relevantes são aqueles que podem potencialmente cometer violações RLR. Esta categoria inclui veículos que entram numa ROI de violação ao cruzar a sua linha de entrada e mover-se na direção controlada pelo semáforo correspondente. Por oposição, veículos não relevantes são todos os outros veículos em movimento que não cumprem esses critérios. Veículos que estavam estacionados ou imóveis durante todo o vídeo não foram contabilizados nestas avaliações, não pertencendo a nenhum das duas categorias supramencionadas.

Na análise do modelo de deteção YOLOv8, focando exclusivamente na tarefa de deteção sem incluir o algoritmo de seguimento, observou-se que todas as variantes testadas, YOLOv8n e YOLOv8s, detetavam corretamente os veículos em movimento de todas as imagens. Com base nesses resultados, decidiu-se testar o desempenho destas variantes ao combinar a sua deteção com o seguimento dos veículos, pois o algoritmo de seguimento apenas utiliza as deteções com uma confiança acima do limiar definido. O limiar de confiança utilizado como padrão na tarefa de seguimento do YOLOv8 está definido como 0,3, ou seja, 30% com a utilização de ambos os algoritmos Bot-SORT e ByteTrack, sendo este o valor utilizado para as avaliações. Para além disso, foram também examinadas as variantes YOLOv8n e YOLOv8s com o modelo ajustado através de TensorRT, uma vez que esta otimização pode afetar a eficácia na deteção de objetos. Ao utilizarem-se os formatos de precisão reduzida FP16 e INT8 os cálculos numéricos são simplificados para melhorar o desempenho computacional, podendo haver perda de detalhes em representações críticas, especialmente em objetos pequenos ou ocluídos, reduzindo assim a confiança de algumas previsões.

Os valores apresentados na tabela 5.12 mostram um alto desempenho do modelo YOLOv8 na deteção dos veículos, especialmente quando utilizada a variante YOLOv8s, com mais parâmetros. No entanto, para a variante YOLOv8n observaram-se algumas limitações na deteção de veículos menores (capturados a uma distância maior) e daqueles parcialmente visíveis na imagem ou com alguma obstrução por parte de outros objetos ou veículos. Ainda assim, de forma geral, a deteção destes veículos ocorre no decorrer das imagens, à medida que os veículos se aproximam da câmara ou se tornam maioritariamente visíveis. A registar ainda que no vídeo E se observou que um veículo não foi detetado para ambas as variantes do modelo. Trata-se de uma mota que pode ser facilmente confundida com uma bicicleta, um tipo de veículo que não é considerado no algoritmo desenvolvido.

Tabela 5.12: Desempenho do modelo YOLOv8 na deteção de veículos presentes em cada vídeo, nas variantes YOLOv8n e YOLOv8s.

Vídeo	**Variante**	**Total de veículos**		**Veículos relevantes**	
		Presentes	**Detetados**	**Presentes**	**Detetados**
A	YOLOv8n	26	25	12	12
	YOLOv8s	26	26	12	12
B	YOLOv8n	13	12	2	2
	YOLOv8s	13	13	2	2
C	YOLOv8n	21	20	5	5
	YOLOv8s	21	21	5	5
D	YOLOv8n	21	21	10	10
	YOLOv8s	21	21	10	10
E	YOLOv8n	22	21	12	12
	YOLOv8s	22	21	12	12

Similarmente, as tabelas 5.13 e 5.14 mostram o desempenho do modelo YOLOv8 na deteção dos veículos, só que desta vez são utilizadas as variantes ajustadas através do TensorRT e usando formatos de precisão reduzida FP16 e INT8. Os resultados obtidos indicam que a deteção dos objetos permanece inalterada ao utilizar-se o formato de redução de precisão FP16, não se notando diferenças nas deteções durante o processamento do algoritmo. Contudo, quando utilizado o formato de redução de precisão INT8, existem variações significativas, perdendo-se alguma eficiência na deteção dos veículos e observando-se maior dificuldade na deteção de veículos menores e parcialmente tapados relativamente aos outros modelos. No caso do vídeo E, com a utilização do formato INT8, grande parte dos veículos relevantes eram apenas detetados já dentro das ROI de violação e houve mesmo veículos relevantes que passaram a não ser detetados.

Embora o modelo já seja bastante preciso com ambas as variantes YOLOv8n e YOLOv8s, em caso de necessidade poderá sempre utilizar-se variantes do modelo YOLOv8 com mais parâmetros, ganhando em termos de precisão, mas necessitando de mais recursos

Tabela 5.13: Desempenho do modelo YOLOv8 na deteção de veículos presentes em cada vídeo, nas variantes YOLOv8n e YOLOv8s ajustadas para formatos de precisão reduzida FP16 através do TensorRT.

Vídeo	**Variante**	**Total de veículos**		**Veículos relevantes**	
		Presentes	**Detetados**	**Presentes**	**Detetados**
A	YOLOv8n	26	25	12	12
	YOLOv8s	26	26	12	12
B	YOLOv8n	13	12	2	2
	YOLOv8s	13	13	2	2
C	YOLOv8n	21	20	5	5
	YOLOv8s	21	21	5	5
D	YOLOv8n	21	21	10	10
	YOLOv8s	21	21	10	10
E	YOLOv8n	22	21	12	12
	YOLOv8s	22	21	12	12

Tabela 5.14: Desempenho do modelo YOLOv8 na deteção de veículos presentes em cada vídeo, nas variantes YOLOv8n e YOLOv8s ajustadas para formatos de precisão reduzida INT8 através do TensorRT.

Vídeo	**Variante**	**Total de veículos**		**Veículos relevantes**	
		Presentes	**Detetados**	**Presentes**	**Detetados**
A	YOLOv8n	26	24	12	12
	YOLOv8s	26	24	12	12
B	YOLOv8n	13	12	2	2
	YOLOv8s	13	12	2	2
C	YOLOv8n	21	19	5	5
	YOLOv8s	21	19	5	5
D	YOLOv8n	21	19	10	10
	YOLOv8s	21	19	10	10
E	YOLOv8n	22	19	12	11
	YOLOv8s	22	21	12	12

computacionais e maior tempo de inferência. Alternativamente, alterar a disposição das câmaras para uma posição mais alta poderia também contribuir para a deteção dos veículos mais facilmente, ao reduzir o número de oclusões. Outra opção, não testada neste trabalho, seria treinar o modelo YOLO com dados próprios, possivelmente aumentando a precisão e generalização na deteção dos veículos.

No que diz respeito à tarefa de seguimento de objetos, esta foi analisada com o intuito de avaliar e comparar o desempenho dos algoritmos BoT-SORT e ByteTrack no seguimento contínuo dos veículos, ambos utilizados em conjunto com o modelo de deteção YOLOv8. As tabelas 5.15 e 5.16 mostram as diferenças no seguimento dos veículos ao utilizar-se diferentes algoritmos de seguimento de objetos.

Os valores presentes nas duas tabelas denotam que, em alguns casos, ao utilizar a variante YOLOv8s, existe um aumento do número de quebras, trocas e atribuição de múltiplos identificadores. Isso ocorre porque a variante YOLOv8s realiza mais deteções por imagem, incluindo objetos difíceis de identificar e parcialmente ocluídos.

Durante o processamento do algoritmo, observou-se que as quebras no seguimento e as

Tabela 5.15: Desempenho do modelo YOLOv8 em conjunto com o algoritmo BoT-SORT no seguimento de veículos de cada vídeo.

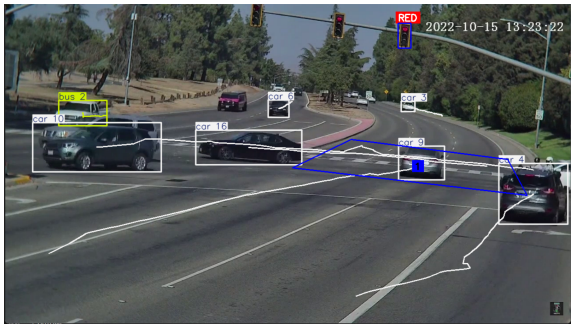
Vídeo	**Variante**	**Quebras**	**Trocas de ID**	**Múltiplos ID**	**Mudanças ID**
A	YOLOv8n	3	3	2	4
	YOLOv8s	2	4	0	3
B	YOLOv8n	2	1	0	0
	YOLOv8s	3	2	1	0
C	YOLOv8n	5	3	0	0
	YOLOv8s	6	4	0	1
D	YOLOv8n	4	2	0	0
	YOLOv8s	4	2	0	1
E	YOLOv8n	9	7	3	0
	YOLOv8s	8	6	2	0

Tabela 5.16: Desempenho do modelo YOLOv8 em conjunto com o algoritmo ByteTrack no seguimento de veículos de cada vídeo.

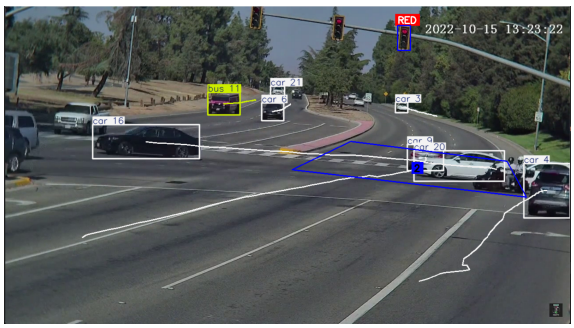
Vídeo	Variante	Quebras	Trocas de ID	Múltiplos ID	Mudanças ID
A	YOLOv8n	3	2	1	2
	YOLOv8s	2	1	2	2
B	YOLOv8n	3	2	0	0
	YOLOv8s	3	2	1	0
C	YOLOv8n	3	1	0	0
	YOLOv8s	3	1	0	0
D	YOLOv8n	5	3	0	1
	YOLOv8s	5	3	1	2
E	YOLOv8n	10	8	3	0
	YOLOv8s	9	6	2	1

trocas de identidade surgem principalmente em veículos que ainda não estão completamente visíveis na imagem, isto é, que estão a entrar em cena/na imagem, ou então em veículos que se sobrepõem a outros, causando alguma obstrução. A maioria das trocas de identificador estão associadas com as quebras no seguimento, havendo normalmente troca de identificador quando existem estas quebras. No entanto, também existem trocas de identificador causados por haverem múltiplos identificadores atribuídos ao mesmo objeto.

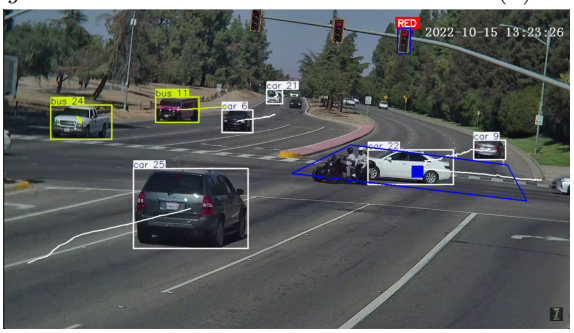
No vídeo C existe uma quebra no seguimento do veículo que realiza RLR, mas ambos os algoritmos conseguem recuperar o seguimento desse veículo com o mesmo identificador anteriormente atribuído, como ilustrado na figura 5.3. Pelo contrário, no vídeo D o seguimento do veículo que causa a passagem de sinal vermelho não se mantém correto, havendo mesmo



(a) *Imagem 1.*



(b) *Imagem 2.*

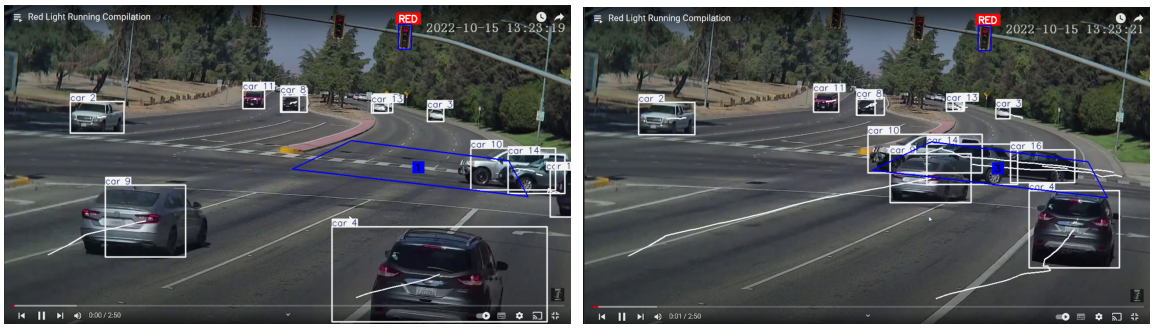


(c) *Imagem 3.*

Figura 5.3: *Imagens de exemplo de seguimento no vídeo C, onde existe quebra no seguimento do veículo com identificador 9, mas este mantém o identificador no decorrer do vídeo sem haver troca.*

(a) *Imagem 1.* (b) *Imagem 2.*

Figura 5.4: *Imagens do erro de seguimento no vídeo D, causado pelo veículo com identificador 13.*



(a) *Exemplo do ajuste inicial das caixas delimitadoras do algoritmo ByteTrack.* (b) *Exemplo do ajuste das caixas delimitadoras do algoritmo ByteTrack no decorrer do vídeo.*

Figura 5.5: *Imagens de exemplo do ajuste das caixas delimitadores realizado pelo algoritmo ByteTrack.*

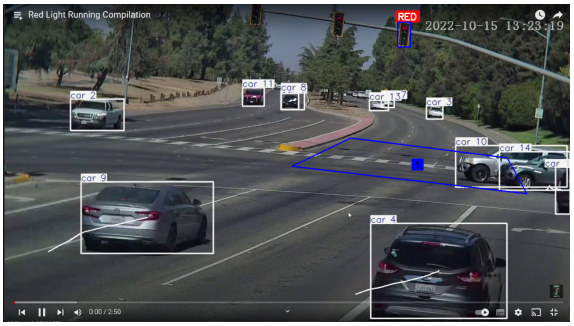


Figura 5.6: *Imagem de exemplo do ajuste das caixas delimitadores realizado pelo algoritmo BoT-SORT.*

uma mudança de identificadores entre veículos sobrepostos, onde o veículo infrator cruza com vários outros veículos e acaba assumindo o identificador do último, como ilustrado na figura 5.4.

Comparando os resultados dos dois algoritmos, observa-se que o ByteTrack tende a ajustar as caixas delimitadoras de forma imprecisa no início do seguimento dos objetos, mas melhora progressivamente ao longo da sequência de imagens, como mostra a figura 5.5. Contrariamente, o algoritmo BoT-SORT consegue localizar precisamente o veículo logo que este entra em cena, como demonstra a figura 5.6. Este detalhe pode ser atribuído ao facto de o algoritmo ByteTrack "aproveitar" todas as deteções do modelo YOLOv8, mesmo aquelas com menor confiança.

5.4 Avaliação de violações RLR

O algoritmo desenvolvido tem como principal objetivo a deteção automática de violações de passagem de sinal vermelho. Como tal, a sua avaliação compreendeu a utilização dos vídeos C e D para testar os casos onde são corretamente identificadas violações RLR (verdadeiros positivos) e os casos onde não são detetadas essas violações apesar de existirem (falsos negativos), atendendo a que apenas estes vídeos contêm incidentes reais de RLR. Os restantes vídeos foram utilizados para verificar se o algoritmo identificaria incorretamente violações de RLR, testando assim a sua tendência a falsos positivos.

Os resultados apresentados na tabela 5.17 mostram que o algoritmo deteta com sucesso a violação existente no vídeo C. No entanto, falha em identificar a violação registada no vídeo D devido a um erro de seguimento, conforme discutido na subsecção 5.3. Para resolver este problema, é importante melhorar o posicionamento das câmaras para minimizar obstruções, o que deve melhorar tanto o desempenho de deteção como de seguimento. Nos vídeos restantes, o algoritmo conseguiu rastrear corretamente os veículos, detetar com precisão a cor dos semáforos e confirmar que não ocorreram violações. Como resultado, não detetou falsamente quaisquer violações RLR, demonstrando a sua robustez e bom desempenho nesta tarefa.

Tabela 5.17: Desempenho do algoritmo desenvolvido quanto à deteção de violações RLR.

Vídeo	VP	FN	FP	TPR
A	-	-	0	-
B	-	-	0	-
C	1	0	0	100%
D	0	1	0	0%
E	-	-	0	-

5.5 Avaliação do desempenho computacional

Sendo um dos objetivos deste trabalho conseguir-se a execução do algoritmo em sistemas embebidos e com processamento em tempo real, os teste realizados também versaram a avaliação do desempenho computacional nos dois sistemas disponíveis. Para facilitar a apresentação e comparação dos resultados, os testes foram separados e categorizados da seguinte forma:

Sistema 1 – Portátil com utilização exclusiva do CPU Intel i5-10300H.

Sistema 2 – NVIDIA Jetson Orin Nano com utilização exclusiva do CPU.

Sistema 3 – NVIDIA Jetson Orin Nano com utilização adicional do processador GPU.

Sistema 4 – NVIDIA Jetson Orin Nano com utilização adicional do processador GPU e com recurso ao modelo otimizado para precisões FP16 e INT8 através de TensorRT.

Adicionalmente, o sistema NVIDIA Jetson Orin Nano pode ser configurado para aproveitar o máximo desempenho dos seus processadores CPU e GPU. Para isso, são ativadas duas funcionalidades:

- "*MAX Power Mode*": Esta opção permite que o sistema utilize todos os núcleos dos processadores CPU e GPU simultaneamente, oferecendo a máxima capacidade de processamento disponível na plataforma. Este modo é ativado ao executar o comando '*sudo nvpmodel -m 0*' na linha de comandos da plataforma NVIDIA Jetson.
- "*Jetson Clocks*": Esta configuração força todos os núcleos dos processadores a funcionarem na sua frequência máxima. Para ativar esta configuração, executa-se o comando '*sudo jetson_clocks*' na linha de comandos da plataforma NVIDIA Jetson.

Com estas otimizações, foram definidas mais duas categorias de testes:

Sistema 5 – NVIDIA Jetson Orin Nano com utilização adicional do processador GPU e tirando o máximo partido do desempenho dos processadores.

Sistema 6 – NVIDIA Jetson Orin Nano com utilização adicional do processador GPU e com recurso ao modelo otimizado para precisões FP16 e INT8 através de TensorRT, tirando o máximo partido do desempenho dos processadores.

De modo a perceber se existem diferenças na utilização dos algoritmos de seguimento BoT-SORT e ByteTrack, realizaram-se os testes nos sistemas 1 e 3 e analisou-se o tempo total de processamento (TP), o tempo médio de inferência (TIM), o tempo médio de processamento (TPM) e o número de imagens processadas por segundo (FPS). Nas tabelas 5.18, 5.19, 5.20 e 5.21 apresentam-se os valores obtidos nesses testes.

A comparação dos valores apresentados nas tabelas 5.18, 5.19, 5.20 e 5.21 revela que o algoritmo ByteTrack apresenta tempos de processamento e inferência aproximadamente duas vezes menores que o algoritmo BoT-SORT. Estes resultados estão alinhados com as expectativas, visto que o algoritmo ByteTrack foi projetado para aplicações em tempo

Tabela 5.18: Desempenho computacional do sistema 1 no processamento do algoritmo desenvolvido, utilizando o algoritmo de seguimento BoT-SORT.

Vídeo	**Variante**	**TP [s]**	**TIM [ms]**	**TPM [ms]**	**FPS [fps]**
A	YOLOv8n	456.5	200.7	240.8	4.08
	YOLOv8s	925.6	444.3	493.7	2.01
B	YOLOv8n	833.8	209.3	237.8	4.15
	YOLOv8s	1585.6	426.6	455.5	2.18
C	YOLOv8n	59.2	200.9	231.6	4.24
	YOLOv8s	122.3	447.4	483.2	2.05
D	YOLOv8n	117.7	197.6	215.8	4.55
	YOLOv8s	249.0	437.3	460.8	2.15
E	YOLOv8n	764.5	228.4	259.6	3.80
	YOLOv8s	1422.8	450.0	486.5	2.04

Tabela 5.19: Desempenho computacional do sistema 1 no processamento do algoritmo desenvolvido, utilizando o algoritmo de seguimento ByteTrack.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	467.0	207.2	247.0	3.99
	YOLOv8s	892.8	427.5	475.1	2.08
B	YOLOv8n	787.6	196.7	224.2	4.39
	YOLOv8s	1666.7	436.9	477.3	2.08
C	YOLOv8n	60.1	202.9	235.5	4.18
	YOLOv8s	107.4	389.4	423.2	2.34
D	YOLOv8n	107.7	178.5	197.3	4.98
	YOLOv8s	245.8	427.1	454.0	2.18
E	YOLOv8n	633.9	185.3	214.1	4.58
	YOLOv8s	1356.1	424.3	462.8	2.14

Tabela 5.20: Desempenho computacional do sistema 3 no processamento do algoritmo desenvolvido, utilizando o algoritmo de seguimento BoT-SORT.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	239.4	90.3	124.0	7.77
	YOLOv8s	255.4	96.0	132.0	7.29
B	YOLOv8n	414.0	85.6	118.5	8.36
	YOLOv8s	462.9	96.2	127.7	7.47
C	YOLOv8n	42.0	99.9	134.1	5.98
	YOLOv8s	47.7	110.3	144.0	5.26
D	YOLOv8n	63.6	92.2	110.6	8.43
	YOLOv8s	68.4	100.7	120.0	7.84
E	YOLOv8n	362.1	91.1	121.7	8.01
	YOLOv8s	394.0	98.0	132.6	7.37

Tabela 5.21: Desempenho computacional do sistema 3 no processamento do algoritmo desenvolvido, utilizando o algoritmo de seguimento ByteTrack.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	151.6	44.5	78.1	12.28
	YOLOv8s	171.3	53.1	88.9	10.86
B	YOLOv8n	259.5	42.7	71.8	13.33
	YOLOv8s	299.6	51.7	83.4	11.55
C	YOLOv8n	23.0	51.8	87.4	10.91
	YOLOv8s	25.6	61.4	98.2	9.80
D	YOLOv8n	41.7	43.3	73.9	12.85
	YOLOv8s	44.7	50.0	82.3	11.99
E	YOLOv8n	226.6	43.8	74.8	12.81
	YOLOv8s	261.0	52.1	87.1	11.12

real, enquanto que o algoritmo BoT-SORT abdica de rapidez para obter maior eficácia no seguimento dos objetos, como é referido em vários artigos científicos [1, 4]. Devido a este melhor desempenho, optou-se por utilizar o algoritmo ByteTrack na execução dos testes subsequentes.

Para aferir o desempenho computacional oferecido pelo CPU Arm que equipa a plataforma

Tabela 5.22: Desempenho computacional do sistema 2 no processamento do algoritmo desenvolvido.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	896.1	444.9	478.5	2.08
	YOLOv8s	2200.9	1144.1	1179.5	0.85
B	YOLOv8n	1725.5	421.6	458.6	2.00
	YOLOv8s	4180.4	1132.9	1164.2	0.83
C	YOLOv8n	98.8	419.7	451.3	2.54
	YOLOv8s	276.2	1110.0	1135.0	0.91
D	YOLOv8n	224.4	432.9	472.8	2.39
	YOLOv8s	618.6	1104.2	1140.5	0.87
E	YOLOv8n	1393.9	442.8	477.1	1.50
	YOLOv8s	3432.6	1145.1	1179.7	0.61

NVIDIA Jetson Orin Nano e compreender o impacto da utilização deste sistema embebido na implementação de um sistema de deteção de violações de passagem do sinal vermelho baseado no algoritmo proposto foram realizados testes usando os sistemas 1 e 2. Nas tabelas 5.19 e 5.22 apresentam-se os valores obtidos nos testes realizados.

A análise das tabelas 5.19 e 5.22 mostra que o computador portátil com o CPU Intel i5-10300H permite obter tempos de processamento e inferência mais baixos, evidenciando as diferenças de desempenho entre os CPU dos dois sistemas. Os processadores Intel, baseados na arquitetura x86, são projetados para tarefas de alto desempenho, oferecendo frequências de trabalho mais altas, mais núcleos e *caches* maiores em comparação com processadores Arm, como o que equipa a plataforma NVIDIA Jetson Orin Nano. Isso torna os processadores da Intel mais adequados para aplicações computacionalmente intensivas, pois não é limitado por restrições de energia como os processadores Arm. Em contrapartida, os processadores Arm têm a eficiência energética como prioridade e são otimizados para tarefas específicas e computacionalmente menos exigentes [82]. Ainda assim, nenhum dos sistemas consegue cumprir com o requisito de processamento em tempo real, i.e. processar acima de 20 fps.

Para aferir o contributo do processador GPU para a melhoria do desempenho computacional do sistema embebido, foram realizados outros testes usando o sistema 3. Na tabela 5.21 apresentam-se os resultados obtidos.

A análise dos valores apresentados nas tabelas 5.19, 5.21 e 5.22 evidencia diferenças consideráveis entre sistemas que utilizam processadores GPU e aqueles que dependem exclusivamente de CPU. O sistema 1, que não conta com processadores GPU dedicados para cálculos complexos, requer tempos de processamento mais longos para executar o algoritmo. Em contrapartida, o sistema 3, equipado com um processador GPU, consegue alcançar tempos de processamento e inferência aproximadamente três vezes menores, comprovando as vantagens da utilização destes processadores para a execução do algoritmo desenvolvido. Ainda assim, a redução conseguida nos tempos de processamento e inferência não é suficiente para se atingir o processamento em tempo real.

Para atingir esse objetivo pode fazer-se a otimização dos modelos no formato TensorRT,

Tabela 5.23: Desempenho computacional do sistema 4 no processamento do algoritmo desenvolvido, utilizando o modelo YOLOv8 ajustado para formatos de precisão reduzida FP16 através do TensorRT.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	160.9	42.5	82.5	11.57
	YOLOv8s	184.4	51.6	95.0	10.09
B	YOLOv8n	263.2	37.4	44.7	13.14
	YOLOv8s	298.9	44.7	82.3	11.57
C	YOLOv8n	23.7	49.9	90.3	10.59
	YOLOv8s	25.3	54.9	96.6	9.92
D	YOLOv8n	46.5	44.6	82.0	11.53
	YOLOv8s	47.1	49.0	83.3	11.38
E	YOLOv8n	257.6	44.9	84.2	11.27
	YOLOv8s	276.0	48.4	90.9	10.51

Tabela 5.24: Desempenho computacional do sistema 4 no processamento do algoritmo desenvolvido, utilizando o modelo YOLOv8 ajustado para formatos de precisão reduzida INT8 através do TensorRT.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	146.4	37.8	74.7	12.71
	YOLOv8s	152.5	40.2	77.9	12.2
B	YOLOv8n	262.3	36.8	71.4	13.19
	YOLOv8s	272.9	39.6	74.4	12.67
C	YOLOv8n	21.0	42.9	79.6	11.95
	YOLOv8s	21.4	45.2	81.2	11.73
D	YOLOv8n	46.2	43.2	81.3	11.60
	YOLOv8s	49.3	47.3	86.9	10.87
E	YOLOv8n	238.7	39.2	77.7	12.13
	YOLOv8s	271.1	46.1	88.6	10.70

como é destacado na página Ultralytics [51]. Nas tabelas 5.23 e 5.24 apresentam-se os valores obtidos para os testes realizados usando o sistema 4, que tira partido das otimizações ao modelo usando os formatos de precisão reduzida FP16 e INT8. Comparando estes valores com os apresentados na tabela 5.21 percebe-se que os ganhos são modestos, especialmente quando se utiliza o formato de precisão FP16. No caso do formato INT8, este proporciona uma redução adicional nos tempos de inferência, embora essa melhoria seja relativamente limitada.

Os testes realizados usando o sistema 6 permitiram aferir o desempenho da plataforma NVIDIA Jetson Orin Nano quando configurada para disponibilizar a máxima capacidade de processamento, ou seja utilizando simultaneamente todos os núcleos dos processadores CPU e GPU e a operarem nas suas frequências máximas.

Como se pode ver ao comparar os resultados apresentados nas tabelas 5.21 e 5.25, se se tirar partido do desempenho máximo do sistema NVIDIA Jetson, é possível reduzir um pouco

Tabela 5.25: Desempenho computacional do sistema 5 no processamento do algoritmo desenvolvido.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	145.4	42.1	74.7	12.80
	YOLOv8s	155.9	45.4	80.7	11.94
B	YOLOv8n	235.3	35.8	64.8	14.70
	YOLOv8s	276.7	45.6	76.9	12.50
C	YOLOv8n	21.3	45.8	81.0	11.78
	YOLOv8s	23.4	53.7	89.6	10.73
D	YOLOv8n	39.8	40.0	70.3	13.47
	YOLOv8s	40.5	48.0	70.4	13.23
E	YOLOv8n	201.6	36.3	66.2	14.39
	YOLOv8s	238.8	45.3	79.2	12.15

os tempos de processamento e de inferência, permitindo processar uma imagem completa adicional por segundo, no caso do vídeo 2 com a variante menor. No entanto, quando se considera a utilização dos modelos otimizados, o desempenho é significativamente melhorado para ambos os formatos FP16 e INT8, como mostram os resultados apresentados nas tabelas 5.26 e 5.27. Daqui conclui-se que os modelos otimizados somente conseguem atingir o máximo desempenho quando são utilizados todos os recursos do sistema, possibilitando o processamento de quase 20 imagens por segundo.

Ainda que estas melhorias sejam já significativas, foram testadas outras estratégias de otimização para minimizar os tempos de processamento e inferência e, assim, conseguir o processamento em tempo real, tais como o uso de *threads*, a configuração do modelo de deteção para identificar apenas classes de veículos em vez de se aplicar filtragem dos objetos detetados posteriormente, e a exclusão da renderização gráfica dos resultados com a biblioteca OpenCV. Estas configurações foram estudadas apenas para o sistema 6 e utilizando o modelo YOLOv8 na sua variante com menos parâmetros e ajustada para o formato de precisão INT8, conjuntamente com o algoritmo ByteTrack para seguimento de objetos.

Tabela 5.26: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido, utilizando o modelo YOLOv8 ajustado para formatos de precisão reduzida FP16 através do TensorRT.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	116.7	26.9	59.6	15.95
	YOLOv8s	128.8	30.7	66.1	14.45
B	YOLOv8n	192.9	24.1	52.7	17.93
	YOLOv8s	218.3	29.4	60.1	15.85
C	YOLOv8n	17.4	32.3	66.2	14.43
	YOLOv8s	19.1	38.0	72.9	13.14
D	YOLOv8n	32.1	26.9	56.8	16.70
	YOLOv8s	33.2	32.8	57.0	16.14
E	YOLOv8n	167.3	25.0	54.6	17.35
	YOLOv8s	192.4	28.8	63.2	15.08

Tabela 5.27: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido, utilizando o modelo YOLOv8 ajustado para formatos de precisão reduzida INT8 através do TensorRT.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	106.7	24.0	54.2	17.44
	YOLOv8s	113.9	27.3	58.1	16.34
B	YOLOv8n	176.9	21.4	48.1	19.55
	YOLOv8s	187.7	24.3	51.2	18.43
C	YOLOv8n	16.2	29.4	61.3	15.49
	YOLOv8s	16.4	31.1	62.0	15.30
D	YOLOv8n	31.2	25.4	55.1	17.18
	YOLOv8s	32.6	27.4	57.6	16.44
E	YOLOv8n	156.3	21.7	50.8	18.57
	YOLOv8s	175.7	26.2	57.4	16.52

A tabela 5.28 mostra os resultados obtidos com a execução de uma versão paralela do algoritmo utilizando *threads*. Neste caso, em vez de se processar as informações dos veículos todas num único ciclo, as informações de cada par de regiões de violação e semáforo são divididas por diferentes *threads* de maneira a realizar o processamento de verificação de RLR de cada região de violação em paralelo. Seria expectável que este processo diminuísse o tempo de processamento, porém os resultados obtidos mostram que este acabou por aumentar ligeiramente os tempos de processamento e inferência. Uma possível explicação para este comportamento pode ser atribuída ao facto de haver poucos pares de ROI, existindo apenas dois pares nos vídeo A, B e E, e um par nos vídeos C e D. Assim, o tempo gasto na criação e sincronização das *threads* supera os benefícios de execução paralela, indicando que o paralelismo no processamento de deteção de RLR só traria vantagens consideráveis em cenários com um número maior de pares de regiões de violação e semáforo, onde o algoritmo poderia aproveitar mais eficientemente a divisão de trabalho entre múltiplas *threads*.

Relativamente à reconfiguração do modelo de deteção para identificar apenas classes de veículos, os resultados apresentados na tabela 5.29, indicam que esta otimização não tem impacto significativo no tempo de inferência, provavelmente por os vídeos utilizados conterem poucos objetos diferentes de veículos ou porque a realização de filtragem dos objetos detetados no algoritmo resulta num tempo semelhante ao pré-processamento do modelo de deteção quando utilizadas classes exclusivas. No entanto, a eliminação da renderização

Tabela 5.28: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido, utilizando o algoritmo com *threads*.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	110.2	25.4	56.1	16.89
B	YOLOv8n	182.9	23.2	49.8	18.91
C	YOLOv8n	17.5	32.4	65.3	14.34
D	YOLOv8n	33.5	26.3	58.1	16.00
E	YOLOv8n	162.2	24.2	53.7	17.89

Tabela 5.29: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido, apenas detetando classes de veículos por parte do modelo YOLOv8.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	106.7	23.9	54.2	17.44
B	YOLOv8n	176.5	21.3	47.9	19.60
C	YOLOv8n	16.4	30.1	62.1	15.30
D	YOLOv8n	31.6	26.0	55.8	16.96
E	YOLOv8n	157.1	22.1	51.1	18.47

gráfica para a apresentação dos resultados permite reduzir em cerca de 10ms o tempo de processamento de uma imagem conforme mostram os dados apresentados na tabela 5.30, sendo que esta redução se refere ao tempo do algoritmo sem contar com o tempo de inferência do modelo YOLOv8. Os resultados obtidos combinando todas estas otimizações são muito próximos dos valores apresentados na tabela 5.30, conforme se apresenta na tabela 5.31, atendendo a que a renderização gráfica é a parte que mais afeta o tempo de processamento do algoritmo.

Fazendo uma análise geral de todos os resultados apresentadas nesta secção, é possível concluir-se que o vídeo C foi o que obteve maiores tempos de processamento médio das imagens. Isto deve-se ao facto de ser o único vídeo para o qual o algoritmo deteta uma violação RLR, necessitando assim de mais processamento (e tempo) para guardar as informações dessa ocorrência. Contrariamente, o vídeo B é o que apresenta menores tempos de processamento e inferência por conter um número inferior de veículos por imagem, diminuindo também o número de deteções necessárias por parte do modelo de deteção YOLOv8. Também é notável que o tempo de processamento de uma imagem, sem incluir o tempo de inferência do modelo de deteção e seguimento, é, em média, aproximadamente 30 ms para

Tabela 5.30: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido sem utilização de renderização gráfica da biblioteca OpenCV.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	88.9	22.3	44.7	20.93
B	YOLOv8n	148.1	20.4	39.8	23.36
C	YOLOv8n	13.6	28.3	51.1	18.46
D	YOLOv8n	26.2	23.5	45.8	20.46
E	YOLOv8n	131.5	20.6	42.3	22.07

Tabela 5.31: Desempenho computacional do sistema 6 no processamento do algoritmo desenvolvido, apenas detetando classes de veículos por parte do modelo YOLOv8 e sem utilização de renderização gráfica da biblioteca OpenCV.

Vídeo	Variante	TP [s]	TIM [ms]	TPM [ms]	FPS [fps]
A	YOLOv8n	86.7	21.9	44.3	21.46
B	YOLOv8n	149.2	20.4	40.0	23.18
C	YOLOv8n	13,7	28.7	51.3	18.32
D	YOLOv8n	25.0	24.1	46.1	21.44
E	YOLOv8n	131.9	21.2	42.9	22.00

os sistemas que utilizam o GPU. Este tempo é reduzido para cerca de 20 ms ao excluir-se a renderização gráfica dos resultados.

Em conclusão, com a implementação de todas as otimizações referidas, conseguiu-se um ritmo de processamento de 23 imagens por segundo utilizando vídeos com uma resolução temporal de 30 imagens por segundo, o que indica que o algoritmo desenvolvido pode ser utilizado para processamento perto do tempo real.

5.6 Conclusão

Neste capítulo são apresentados e discutidos os resultados obtidos com a avaliação do algoritmo em dois sistemas computacionais distintos: um computador portátil equipado com um processador Intel i5-10300H e um sistema embebido NVIDIA *Developer Kit* Jetson Orin Nano.

Esta avaliação indicou que o classificador baseado em redes neuronais convolucionais (CNN) supera o método baseado em características da imagem em termos de precisão, além de proporcionar menores tempos de processamento. O modelo de deteção, em conjunto com o algoritmo de seguimento ByteTrack, demonstrou eficiência na deteção da maioria dos veículos presentes nos vídeos, com um desempenho superior quando se utiliza a variante YOLOv8s. Também foi testado o impacto de ajustes de precisão FP16 e INT8 através de *TensorRT*, verificando-se que o formato FP16 manteve a eficiência de deteção dos modelos base. Contudo, o formato INT8 apresentou uma redução na precisão de deteção dos veículos, chegando a falhar na deteção de um veículo relevante anteriormente identificado pelos outros modelos. No seguimento dos veículos, foram observados alguns desafios, como quebras no seguimento e trocas de identificadores, especialmente em situações de oclusão ou sobreposição de veículos, tendo-se concluído que o algoritmo ByteTrack apresenta menores tempos de inferência relativamente ao algoritmo BoT-SORT.

A avaliação das violações de RLR evidenciou a capacidade do algoritmo proposto para as detetar corretamente. Ainda assim, registou-se uma falha na deteção da violação no vídeo D, sendo esta resultante de erros no seguimento do veículo, onde ocorrem várias oclusões e acaba por haver troca de identificadores com outro veículo. Estas limitações sugerem a necessidade de melhorias no algoritmo de seguimento ou na configuração do sistema, por exemplo, para ajustar o posicionamento das câmaras para ângulos que minimizem oclusões.

Quanto ao desempenho computacional, os testes realizados em diferentes sistemas mostraram que a utilização de processadores gráficos (GPU) e a otimização dos modelos podem melhorar significativamente o tempo de processamento e de inferência. Quando todas as otimizações propostas foram aplicadas, o sistema conseguiu processar 23 imagens por segundo, estando no limite do que se pode considerar processamento em tempo real.

6 Conclusões

O trabalho de investigação vertido nesta dissertação teve como principal objetivo o estudo e desenvolvimento de um sistema para deteção automática de violações de passagem de sinal vermelho (RLR), utilizando exclusivamente técnicas de visão computacional e visando a implementação em sistemas embebidos. A motivação para o desenvolvimento deste sistema reside na elevada incidência de acidentes causados por RLR, que representam um desafio contínuo para a segurança rodoviária e com elevados custos humanos e financeiros associados. A solução proposta tem como premissas a otimização para processamento em sistemas embebidos, em tempo real, com baixo custo de implementação, considerando-se como um sistema piloto para uma possível futura aplicação em cenários reais.

Na base do algoritmo desenvolvido está o modelo YOLOv8 , que se destaca pelo seu desempenho superior na deteção de objetos quando comparado a outros modelos disponíveis na data de início do trabalho. Para além da deteção de objetos, o algoritmo desenvolvido também faz partido de um classificador da cor do semáforo, com recurso a dois métodos diferentes, sendo um deles baseado nas características específicas da imagem, como brilho, saturação e cor, e o outro numa rede neuronal convolucional (CNN). Para complementar a tarefa de deteção de objetos, a solução proposta utiliza o algoritmo de seguimento de objetos ByteTrack para realizar uma análise da trajetória dos veículos. Essa trajetória é importante na avaliação de violações RLR por fornecer a direção que o veículo seguiu ao intersetar a sua linha da trajetória com ROI definidas, conseguindo-se assim avaliar interseções e cruzamentos onde os veículos podem seguir em mais do que uma direção.

Para suprir a ausência de bases de dados públicas adequadas a este caso de estudo, foram criadas bases de dados específicas para treinar e testar os modelos e avaliar a capacidade de deteção de violações RLR. A primeira base de dados criada contém vídeos de estradas nos Estados Unidos da América e Canadá, simulando cenários reais de trânsito com captura do tráfego de veículos e dos semáforos presentes, incluindo violações explícitas de RLR. A outra base de dados, dedicada à classificação das cores dos semáforos, foi construída a partir de imagens extraídas desses vídeos complementadas com outras de bases de dados públicas, por forma a aumentar a generalização do modelo.

No treino do modelo YOLOv8 foram exploradas bases de dados dedicadas para a tarefa

de deteção de objetos que apresentassem imagens de veículos. Contudo, os resultados do teste do modelo treinado com estes dados foram inferiores comparativamente ao modelo YOLOv8 com os pesos pré-treinados na base de dados COCO. Também foram analisadas as várias variantes do YOLOv8, onde são apresentados diferentes números de parâmetros e capacidade de inferência. Ao examinar as características de cada variante, adotaram-se as variantes YOLOv8n e YOLOv8s, por apresentarem um tempo de inferência menor mantendo uma boa precisão na deteção de objetos.

Relativamente ao desempenho do algoritmo, os resultados obtidos mostraram que o classificador baseado em CNN supera o método baseado em características da imagem, tanto em precisão como em tempo de processamento. O modelo de deteção, aliado ao algoritmo ByteTrack, demonstrou eficiência na identificação e seguimento da maioria dos veículos presentes nos vídeos. Contudo, foram identificados desafios no seguimento dos veículos, tais como falhas no seguimento e trocas de identificadores em cenários com oclusões ou sobreposição de veículos.

Quanto ao desempenho computacional do algoritmo em sistemas com diferentes capacidades e características, verificou-se uma vantagem significativa na utilização de processadores GPU, que apresentaram tempos de processamento substancialmente menores em comparação com processadores CPU. A otimização do modelo YOLOv8 com o SDK TensorRT trouxe melhorias significativas no tempo de inferência, especialmente utilizando o formato INT8. No entanto, a adoção deste formato resultou numa ligeira redução da eficácia na deteção dos veículos. Não obstante, o modelo otimizado aliado ao uso pleno dos recursos da plataforma NVIDIA Jetson Orin Nano, permitiu maximizar o desempenho computacional do sistema e atingir um processamento de aproximadamente 23 imagens por segundo, que se pode considerar de tempo real.

6.1 Trabalhos futuros

Apesar dos resultados obtidos serem promissores na classificação da cor dos semáforos e na deteção de veículos e violações RLR, o trabalho realizado não abordou alguns desafios interessantes de analisar em investigações futuras.

Uma possível direção para trabalho futuro, é o estudo do funcionamento do sistema em condições ambientais adversas, como chuva intensa ou nevoeiro, que podem afetar negativamente a precisão do sistema na classificação da cor do semáforo e na deteção e seguimento de objetos, resultando em possíveis falhas.

Outro desafio interessante prende-se com a utilização do sistema com densidades elevadas de tráfego, que podem dificultar a capacidade do algoritmo seguir veículos de forma contínua, especialmente em situações de oclusão.

O modelo de deteção de objetos YOLO tem estado em constante evolução, tendo-se verificado uma tendência constante de melhoria da sua precisão e tempo de inferência. Assim, trabalhos futuros devem continuar a explorar novos métodos para reduzir o tempo de inferência do modelo, quer seja com a utilização de novas versões do modelo YOLO, quer

seja com outros métodos de otimização do modelo.

Neste trabalho não houve possibilidade de criar uma base de dados com dados próprios para treinar o modelo de deteção YOLO. Assim, uma outra possibilidade de trabalho futuro consiste em examinar os efeitos do treino do modelo com dados semelhantes aos cenários que se pretendem avaliar, verificando se melhora a precisão na deteção dos veículos e, sobretudo, se melhora o tempo de inferência do modelo.

Neste trabalho apenas foram avaliados os objetos categorizados como veículos, mas o tema pode ser aberto para abranger outras classes como pessoas e ciclistas. Nesses casos, o sistema poderia realizar o processamento necessário para verificar se pessoas se encontram à espera nas passadeiras ou ciclistas na estrada. Este tipo de análises, em conjunto com a avaliação de fluxo rodoviário, poderão servir para uma gestão mais eficiente e inteligente do tráfego rodoviário naquilo que atualmente se designa por cidades inteligentes.

Como referido no capítulo 1, um dos problemas do uso de câmaras RLC é o aumento dos acidentes traseiros, devido às travagens abruptas junto do semáforo. Com a adição de métodos matemáticos mais avançados, o sistema poderá evoluir para controlar os semáforos e atender a estas situações fazendo a análise da distância entre veículos no momento da ocasião, verificando a necessidade de passagem do sinal vermelho para prevenir maiores percalços.

O sistema também poderá identificar veículos de emergência ao obter a classificação atribuída ao objeto, dando-lhes permissão para realizar RLR ou avaliar infrações de limite de velocidade ao analisar a velocidade dos veículos.

Bibliografia

[1] F. Z. A. H. Aadi, A. Sadiq e Z. Labd. "Comparing Object Tracking Algorithms for Real-Time Applications: Performance Analysis and Implementation Study". Em: *10th Int. Conf. Wirel. Netw. Mob. Commun.* IEEE. 2023, pp. 1–5 (ver pp. 14, 66).

[2] A. Aeron-Thomas e S. Hess. "Red-light cameras for the prevention of road traffic crashes". Em: *Cochrane Database of Systematic Reviews* 2 (2005) (ver p. 15).

[3] A. F. Agarap. "Deep learning using rectified linear units (relu)". Em: *arXiv preprint arXiv:1803.08375* (2018) (ver p. 29).

[4] N. Aharon, R. Orfaig e B.-Z. Bobrovsky. *BoT-SORT: Robust Associations Multi-Pedestrian Tracking*. 2022. arXiv: 2206.14651 [id='cs.CV' full_name='Computer Vision and Pattern Recognition' is_active=True alt_name=None in_archive='cs' is_general=False description='Covers image processing, computer vision, pattern recognition, and scene understanding. Roughly includes material in ACM Subject Classes I.2.10, I.4, and I.5.'] (ver pp. 13–15, 66).

[5] O. Al-Jarrah, A. Siddiqui, M Elsalamouny, P. D. Yoo, S. Muhaidat e K. Kim. "Machine-learning-based feature selection techniques for large-scale network intrusion detection". Em: *2014 IEEE 34th international conference on distributed computing systems workshops (ICDCSW)*. IEEE. 2014, pp. 177–181 (ver p. 8).

[6] A. K. Alhaq, T. Ahmad e H. Studiawan. "Forensic Analysis of Car Accident Using MobileNet and Optical Flow". Em: *2023 2nd International Conference on Computer System, Information Technology, and Electrical Engineering (COSITE)*. IEEE. 2023, pp. 78–83 (ver p. 8).

[7] M. A. R. Alif e M. Hussain. *YOLOv1 to YOLOv10: A comprehensive review of YOLO variants and their application in the agricultural domain*. 2024. DOI: 10.48550/arXiv.2406.10139. arXiv: 2406.10139 [cs.CV]. URL: https://arxiv.org/abs/2406.10139 (ver p. 12).

[8] amrsayed. *Traffic Light Dataset*. Open Source Dataset. Acedido em 2024. 2024. URL: https://universe.roboflow.com/amrsayed/traffic-light-1cygo-bvchu (ver pp. 42, 43).

[9] Álvaro Arcos-García, J. A. Álvarez García e L. M. Soria-Morillo. "Evaluation of deep neural networks for traffic sign detection systems". Em: *Neurocomputing* 316 (2018), pp. 332–344 (ver p. 8).

[10] Baem. *cctv$_d$etectionDataset*. https://universe.roboflow.com/baem/cctv_detection. Open Source Dataset. 2023. URL: https://universe.roboflow.com/baem/cctv_detection (ver pp. 42, 44, 46).

[11] B. Behroozpour, P. A. Sandborn, M. C. Wu e B. E. Boser. "Lidar system architectures and circuits". Em: *IEEE Communications Magazine* 55.10 (2017), pp. 135–142 (ver p. xxii).

[12] K. Bernardin e R. Stiefelhagen. "Evaluating multiple object tracking performance: the clear mot metrics". Em: *EURASIP Journal on Image and Video Processing* 2008 (2008), pp. 1–10 (ver pp. 13, 39, 40).

[13] A. Bewley, Z. Ge, L. Ott, F. Ramos e B. Upcroft. "Simple online and realtime tracking". Em: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016. DOI: 10.1109/icip.2016.7533003. URL: http://dx.doi.org/10.1109/ICIP.2016.7533003 (ver p. 14).

[14] A. Biglari e W. Tang. "A Review of Embedded Machine Learning Based on Hardware, Application, and Sensing Scheme". Em: *Sensors* 23.4 (2023). ISSN: 1424-8220. DOI: 10.3390/s23042131. URL: https://www.mdpi.com/1424-8220/23/4/2131 (ver p. 16).

[15] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995 (ver p. 8).

[16] A. Bochkovskiy, C.-Y. Wang e H.-Y. M. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV] (ver p. 11).

[17] J. A. Bonneson e K. H. Zimmerman. "Effect of Yellow-Interval Timing on the Frequency of Red-Light Violations at Urban Intersections". Em: *Transportation Research Record* 1865.1 (2004), pp. 20–27. DOI: 10.3141/1865-04 (ver p. 3).

[18] M Bourne e R Cooke. "Victoria's speed camera program". Em: *Crime prevention studies* 1 (1993), pp. 177–192 (ver p. 16).

[19] J. S. Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition". Em: *Neurocomputing: Algorithms, architectures and applications*. Springer, 1990, pp. 227–236 (ver p. 30).

[20] D. T. Bui, P. Tsangaratos, V.-T. Nguyen, N. Van Liem e P. T. Trinh. "Comparing the prediction performance of a Deep Learning Neural Network model with conventional machine learning models in landslide susceptibility assessment". Em: *Catena* 188 (2020), p. 104426 (ver p. 8).

[21] J. Cao, J. Pang, X. Weng, R. Khirodkar e K. Kitani. *Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking*. 2023. arXiv: 2203.14360 [cs.CV]. URL: https://arxiv.org/abs/2203.14360 (ver p. 14).

[22] Cars. *Cars detecting and how many Dataset*. https://universe.roboflow.com/cars-fjcrk/cars-detecting-and-how-many. Open Source Dataset. 2024. URL: https://universe.roboflow.com/cars-fjcrk/cars-detecting-and-how-many (ver pp. 42, 45, 46).

[23] F. Chen, X. Wang, Y. Zhao, S. Lv e X. Niu. "Visual object tracking: A survey". Em: *Computer Vision and Image Understanding* 222 (2022), p. 103508. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2022.103508. URL: https://www.sciencedirect.com/science/article/pii/S1077314222001011 (ver pp. 12, 13).

[24] J. Cho, K. Lee, E. Shin, G. Choy e S. Do. *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* 2016. arXiv: 1511.06348 [cs.LG]. URL: https://arxiv.org/abs/1511.06348 (ver p. 41).

[25] E. G. Cohn, S. Kakar, C. Perkins, R. Steinbach e P. Edwards. "Red light camera interventions for reducing traffic violations and traffic crashes: A systematic review". Em: *Campbell Systematic Reviews* 16.2 (2020), e1091. DOI: https://doi.org/10.1002/cl2.1091. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cl2.1091. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cl2.1091 (ver p. 4).

[26] C. Cortes e V. Vapnik. "Support-vector networks". Em: *Machine learning* 20.3 (1995), pp. 273–297 (ver p. 7).

[27] N. S. Council. *Motor Vehicle Overview*. URL: https://injuryfacts.nsc.org/contact/ (ver p. 3).

[28] M. Dewar. *ITSDC Project : Traffic Light Classifier*. 2018. URL: https://alyxion.github.io/Udacity_IntroToSelfDrivingCarsNd/8_2_Project_5_Traffic_Light_Classifier/Traffic_Light_Classifier.html (ver pp. 27–29).

[29] Y. Du, Z. Zhao, Y. Song, Y. Zhao, F. Su, T. Gong e H. Meng. *StrongSORT: Make DeepSORT Great Again*. 2023. arXiv: 2202.13514 [cs.CV]. URL: https://arxiv.org/abs/2202.13514 (ver p. 14).

[30] N. Elmitiny, X. Yan, E. Radwan, C. Russo e D. Nashar. "Classification analysis of driver's stop/go decision and red-light running violation". Em: *Accident Analysis & Prevention* 42.1 (2010), pp. 101–111. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2009.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0001457509001729 (ver p. 3).

[31] H. Fan, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu, H. Bai, Y. Xu, C. Liao e H. Ling. "LaSOT: A High-Quality Benchmark for Large-Scale Single Object Tracking". Em: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (ver p. 12).

[32] H. Feng, G. Mu, S. Zhong, P. Zhang e T. Yuan. "Benchmark Analysis of YOLO Performance on Edge Intelligence Devices". Em: *Cryptography* 6.2 (2022). ISSN: 2410-387X. DOI: 10.3390/cryptography6020016. URL: https://www.mdpi.com/2410-387X/6/2/16 (ver p. 16).

[33] M. Gao, Y. Du, Y. Yang e J. Zhang. "Adaptive anchor box mechanism to improve the accuracy in the object detection system". Em: *Multimedia Tools and Applications* 78 (2019), pp. 27383–27402 (ver p. xxi).

[34] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang e V. Volkov. "Parallel computing experiences with CUDA". Em: *IEEE micro* 28.4 (2008), pp. 13–27 (ver p. xxi).

[35] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney e K. Keutzer. "A survey of quantization methods for efficient neural network inference". Em: *Low-Power Computer Vision*. Chapman e Hall/CRC, 2022, pp. 291–326 (ver p. xxii).

[36] B. Glenn Jocher Muhammad Rizwan. *Tips for Best Training Results*. 2023. URL: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/ (ver p. 44).

[37] D. Goldstein. "Are Police Tricking People Into Paying 'Snitch' Tickets?" Em: *KCAL News* (February 13, 2011). URL: https://www.cbsnews.com/losangeles/news/goldstein-investigation-are-police-tricking-people-into-paying-snitch-tickets/ (ver p. 4).

[38] J. C. de Goma, R. J. Bautista, M. A. J. Eviota e V. P. Lopena. "Detecting Red-Light Runners (RLR) and Speeding Violation through Video Capture". Em: *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*. 2020, pp. 774–778. DOI: 10.1109/ICIEA49774.2020.9102059 (ver p. 8).

[39] Google. *Dev Board datasheet | Coral*. URL: https://coral.ai/docs/dev-board/datasheet/#features (ver p. 17).

[40] J. Gou, B. Yu, S. J. Maybank e D. Tao. "Knowledge distillation: A survey". Em: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819 (ver p. xxii).

[41] D. M. Hawkins. "The problem of overfitting". Em: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12 (ver pp. xxiii, 43).

[42] K. He, X. Zhang, S. Ren e J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (ver p. 8).

[43] T. K. Ho. "Random decision forests". Em: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282 (ver p. 7).

[44] S. J. Hole. *Jackson Hole Wyoming USA Town Square Live Cam - SeeJH.com*. Acedido em 2024. URL: https://www.seejh.com/webcams/jackson/town-square-broadway (ver pp. 24, 42, 53).

[45] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto e H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV] (ver p. 8).

[46] W. Hu, A. T. McCartt e E. R. Teoh. "Effects of red light camera enforcement on fatal crashes in large US cities". Em: *Journal of Safety Research* 42.4 (2011), pp. 277–282. ISSN: 0022-4375. DOI: https://doi.org/10.1016/j.jsr.2011.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0022437511000612 (ver p. 3).

[47] H. Huang, H. C. Chin e A. H. H. Heng. "Effect of Red Light Cameras on Accident Risk at Intersections". Em: *Transportation Research Record* 1969.1 (2006), pp. 18–26. DOI: 10.1177/0361198106196900103. URL: https://doi.org/10.1177/0361198106196900103 (ver p. 4).

[48] A. Jahangiri, H. A. Rakha e T. A. Dingus. "Adopting Machine Learning Methods to Predict Red-light Running Violations". Em: *IEEE 18th Int. Conf. Intell. Transp. Syst.* 2015, pp. 650–655. DOI: 10.1109/ITSC.2015.112 (ver pp. 3, 7).

[49] G. Jocher. *Ultralytics YOLOv5*. Versão 7.0. 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5 (ver p. 11).

[50] G. Jocher, A. Chaurasia e J. Qiu. *Ultralytics*. URL: https://docs.ultralytics.com/ (ver pp. 11, 24, 33, 34).

[51] G. Jocher, A. Chaurasia e J. Qiu. *Ultralytics YOLOv8*. Versão 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics (ver pp. 4, 11, 12, 24–27, 50, 68).

[52] G. Jocher e J. Qiu. *Ultralytics YOLO11*. Versão 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics (ver p. 12).

[53] S. S. Kadam, T. Jadhav, L. Patil, P. Saw e A. Landage. "Crowd counting using yolov8 and various tracking algorithms". Em: *2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0*. IEEE. 2024, pp. 1–9 (ver p. 14).

[54] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". Em: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. URL: https://doi.org/10.1115/1.3662552 (ver p. 13).

[55] N. Kim, J. Kim, H. Kim, K. Lim, Y. Ko, N. Jeong, A. H. Smith e H. A. McNally. "Red Light Running Prediction System using LIDAR". Em: *2019 IEEE Sensors Applications Symposium (SAS)*. 2019, pp. 1–5. DOI: 10.1109/SAS.2019.8706098 (ver pp. xxii, 15).

[56] A. Krizhevsky, I. Sutskever e G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". Em: *Advances in Neural Information Processing Systems*. Ed. por F. Pereira, C. Burges, L. Bottou e K. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (ver p. 8).

[57] A. K. I. Lawyers. *Red Light Running Deaths Hit An All-Time High*. URL: https://arashlaw.com/red-light-running-deaths-hit-an-all-time-high/ (ver p. 2).

[58] I. Lazarevich, M. Grimaldi, R. Kumar, S. Mitra, S. Khan e S. Sah. "YOLOBench: Benchmarking Efficient Object Detectors on Embedded Systems". Em: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. 2023, pp. 1169–1178 (ver p. 16).

[59] LeCun, Yann and Bengio, Yoshua and Hinton, Geoffrey. "Deep learning". Em: *Nature* 521.7553 (2015), p. 436 (ver p. 8).

[60] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei e X. Wei. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: 2209.02976 [cs.CV] (ver pp. xxii, 11).

[61] Z. Li, F. Liu, W. Yang, S. Peng e J. Zhou. "A survey of convolutional neural networks: analysis, applications, and prospects". Em: *IEEE transactions on neural networks and learning systems* 33.12 (2021), pp. 6999–7019 (ver p. 8).

[62] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan e S. Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV] (ver p. 11).

[63] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár e C. L. Zitnick. "Microsoft COCO: Common Objects in Context". Em: *CoRR* abs/1405.0312 (2015). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (ver pp. 24, 31, 46, 47).

[64] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu e A. C. Berg. "SSD: Single Shot MultiBox Detector". Em: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325 (ver p. 9).

[65] F. Lo e P. Michael. "Red-light cameras click into action". Em: *South China Morning Post* (2004) (ver p. 4).

[66] B. Loomis. "1900-1930: The years of driving dangerously". Em: *The Detroit News* (2015). URL: https://eu.detroitnews.com/story/news/local/michigan-history/2015/04/26/auto-traffic-history-detroit/26312107/ (ver p. 1).

[67] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu e T.-K. Kim. "Multiple object tracking: A literature review". Em: *Artificial intelligence* 293 (2021), p. 103448 (ver p. 12).

[68] R. P. Maccubbin, B. L. Staples, A. E. Salwin et al. "Automated enforcement of traffic signals: A literature review". Em: *Mitretek Systems* (2001) (ver p. 16).

[69] K. Manaa, M. Rabee'a, L. Khalaf et al. "Traffic control by digital imaging cameras". Em: *Emerging Trends in Image Processing, Computer Vision and Pattern Recognition*. Elsevier, 2015, pp. 231–247 (ver p. 4).

[70] V. Mazzia, A. Khaliq, F. Salvetti e M. Chiaberge. "Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application". Em: *IEEE Access* 8 (2020), pp. 9102–9114. DOI: 10.1109/ACCESS.2020.2964608 (ver p. 16).

[71] A.-A. Nahid e Y. Kong. "Involvement of machine learning for breast cancer image classification: a survey". Em: *Computational and mathematical methods in medicine* 2017.1 (2017), p. 3781951 (ver p. 8).

[72] A. Neelopant, D. S. V. Viraktamath e P. Navalgi. "Comparison of YOLOv3 and SSD Algorithms". Em: *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT)* (2021). DOI: 10.17577/IJERTV10IS020077 (ver p. 9).

[73] J. J. Ng, K. O. M. Goh e C. Tee. "Traffic Impact Assessment System using Yolov5 and ByteTrack". Em: *Journal of Informatics and Web Engineering* 2.2 (2023), pp. 168–188 (ver p. 14).

[74] N. J. Nilson. "Introduction to Machine Learning". Em: *AN EARLY DRAFT OF A PROPOSED TEXTBOOK. Robotics Laboratory. Department of Computer Science Stanford University. Stanford. USA. Recuperado de: http://ai. stanford. edu/˜ nilsson/MLBOOK. pdf* (1998) (ver pp. xxii, xxiii, 39).

[75] NO. *N_ O_ 3 Dataset*. https://universe.roboflow.com/no-vabwp/n_o_3. Open Source Dataset. 2024. URL: https://universe.roboflow.com/no-vabwp/n_o_3 (ver pp. 42, 44, 45).

[76] NVIDIA. *CUDA Toolkit*. URL: https://developer.nvidia.com/cuda-toolkit (ver p. xxi).

[77] NVIDIA. *JetPack SDK | NVIDIA Developer*. URL: https://developer.nvidia.com/embedded/jetpack (ver p. 38).

[78] NVIDIA. *Jetson Orin for Next-Gen Robotics | NVIDIA*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (ver p. 17).

[79] NVIDIA. *NVIDIA TensorRT SDK*. Acedido em 2024. URL: https://developer.nvidia.com/tensorrt (ver pp. xxiii, 33, 34).

[80] NVIDIA. *View Jetson Orin Technical Specifications*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (ver p. 38).

[81] W. H. Organization. *World report on road traffic injury prevention*. 2004. URL: https://www.who.int/publications/i/item/world-report-on-road-traffic-injury-prevention (ver p. 2).

[82] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Ylä-Jääski e P. Hui. "Energy- and Cost-Efficiency Analysis of ARM-Based Clusters". Em: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. 2012, pp. 115–123. DOI: 10.1109/CCGrid.2012.84 (ver p. 67).

[83] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto e E. A. B. da Silva. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". Em: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279 (ver pp. 39, 40, 95).

[84] F. Pasti, N. Bellotto et al. "Evaluation of Computer Vision-Based Person Detection on Low-Cost Embedded Systems". Em: *Proceedings of the 18th International Conference on Computer Vision Theory and Applications (VISAPP)*. 2023 (ver p. 16).

[85] K. Pavani e P. Sriramya. "Comparison of KNN, ANN, CNN and YOLO algorithms for detecting the accurate traffic flow and build an Intelligent Transportation System". Em: *2nd Int. Conf. Innov. Pract. Technol. Manag.* Vol. 2. 2022, pp. 628–633. DOI: 10.1109/ICIPTM54933.2022.9753900 (ver pp. 9, 24).

[86] M. Petersson e N. Kifle Solomon. *Object Tracking Evaluation: BoT-SORT & ByteTrack with YOLOv8: A Comparison of Accuracy and Computational Efficiency*. 2024 (ver p. 14).

[87] M. Phadtare, V Choudhari, R Pedram e S Vartak. "Comparison between YOLO and SSD mobile net for object detection in a surveillance drone". Em: *Int. J. Sci. Res. Eng. Man.* 5 (2021), pp. 1–5. DOI: 10.13140/RG.2.2.34029.51688 (ver p. 9).

[88] R. Pi. *Raspberry Pi 4 Model B specs*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (ver p. 17).

[89] C. A. community portal. *Canmore's Live Streaming Webcam "Main Street* . Acedido em 2024. URL: https://canmorealberta.com/webcams/main-street (ver pp. 42, 43, 53).

[90] A. F. W. R A Retting R G Ulmer. "Prevalence and characteristics of red light running crashes in the United States". Em: *Accid Anal Prev* 31.6 (1999), pp. 687–94. DOI: 10.1016/s0001-4575(99)00029-9 (ver p. 2).

[91] P. Ramachandran, B. Zoph e Q. V. Le. "Searching for activation functions". Em: *arXiv preprint arXiv:1710.05941* (2017) (ver p. 8).

[92] J. Ravi. *Red light camera*. In Wikipedia. 2024. URL: https://en.wikipedia.org/wiki/Red_light_camera (ver p. 3).

[93] J. Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016 (ver p. 10).

[94] J. Redmon, S. Divvala, R. Girshick e A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". Em: *2016 IEEE Conf. Comput. Vis. Pattern Recognit.* 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (ver pp. 9, 10, 24).

[95] J. Redmon e A. Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV] (ver p. 10).

[96] J. Redmon e A. Farhadi. "YOLOv3: An Incremental Improvement". Em: *arXiv* (2018) (ver p. 9).

[97] J. Redmon e A. Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV] (ver p. 10).

[98] S. Ren, K. He, R. Girshick e J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (ver pp. 9, 10).

[99] S. Report. "Over a thousand lives lost every year as red-light running rages on". Em: *Times Leader* (2024). URL: https://www.timesleader.com/news/1663465/over-a-thousand-lives-lost-every-year-as-red-light-running-rages-on (ver p. 2).

[100] R. A. Retting, S. A. Ferguson e C. M. Farmer. "Reducing red light running through longer yellow signal timing and red light camera enforcement: Results of a field investigation". Em: *Accident Analysis & Prevention* 40.1 (2008), pp. 327–333. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2007.06.011. URL: https://www.sciencedirect.com/science/article/pii/S000145750700111X (ver p. 3).

[101] R. A. Retting, S. A. Ferguson e A. S. Hakkert. "Effects of Red Light Cameras on Violations and Crashes: A Review of the International Literature". Em: *Traffic Injury Prevention* 4.1 (2003), pp. 17–23. DOI: 10.1080/15389580309858 (ver pp. 3, 4).

[102] R. A. Retting, R. G. Ulmer e A. F. Williams. "Prevalence and characteristics of red light running crashes in the United States". Em: *Accident Analysis & Prevention* 31.6 (1999), pp. 687–694. ISSN: 0001-4575. DOI: https://doi.org/10.1016/S0001-4575(99)00029-9. URL: https://www.sciencedirect.com/science/article/pii/S0001457599000299 (ver p. 3).

[103] Roboflow. *Explore the Roboflow Universe*. URL: https://universe.roboflow.com/ (ver p. 44).

[104] F. Roulette. *Red Light Running Compilation*. Acedido em 2024. 2022. URL: https://www.youtube.com/watch?v=TR-C8egkjP8 (ver pp. 42, 43, 53).

[105] D. Russell. "The Big Picture Behind Red-Light Cameras". Em: *Philadelphia Daily News* (July 1, 2003). URL: https://web.archive.org/web/20120910030653/http://www.motorists.org/red-light-cameras/big-picture (ver p. 4).

[106] C. P. P. S.A. *O passado, o presente e o futuro da segurança rodoviária*. URL: https://www.continental-tires.com/pt/pt/b2c/stories/road-safety-history/ (ver pp. 1, 2).

[107] P. Sanagapati. *A Simple CNN Model Beginner Guide !!!!!!* 2020. URL: https://www.kaggle.com/code/pavansanagapati/a-simple-cnn-model-beginner-guide (ver p. 8).

[108] S. Santurkar, D. Tsipras, A. Ilyas e A. Madry. "How does batch normalization help optimization?" Em: *Advances in neural information processing systems* 31 (2018) (ver p. xxi).

[109] T Saranya, S Sridevi, C Deisy, T. D. Chung e M. A. Khan. "Performance analysis of machine learning algorithms in intrusion detection system: A review". Em: *Procedia Computer Science* 171 (2020), pp. 1251–1260 (ver pp. 39, 40, 95).

[110] J. Schmidhuber. "Deep Learning in Neural Networks: An overview". Em: *Neural Networks* 61 (2015), 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003 (ver p. 8).

[111] K. Shaaban e A. Pande. “Evaluation of red-light camera enforcement using traffic violations”. Em: *Journal of Traffic and Transportation Engineering (English Edition)* 5.1 (2018), pp. 66–72. ISSN: 2095-7564. DOI: https://doi.org/10.1016/j.jtte.2017.04.005. URL: https://www.sciencedirect.com/science/article/pii/S2095756416301659 (ver p. 3).

[112] D.-J. Shin e J.-J. Kim. “A Deep Learning Framework Performance Evaluation to Use YOLO in Nvidia Jetson Platform”. Em: *Applied Sciences* 12.8 (2022). ISSN: 2076-3417. DOI: 10.3390/app12083734. URL: https://www.mdpi.com/2076-3417/12/8/3734 (ver p. 16).

[113] T. Silva. *Dataset for Traffic Evaluation*. 2024. DOI: 10.5281/zenodo.14041920. URL: https://doi.org/10.5281/zenodo.14041920 (ver pp. 6, 42, 48, 50, 53).

[114] T. Silva. *Dataset for Traffic Light Classification*. 2024. DOI: 10.5281/zenodo.14032895. URL: https://doi.org/10.5281/zenodo.14032895 (ver pp. 5, 43, 44, 48, 49).

[115] T. Silva. *Red Light Running Detection Algorithm*. Versão 1.0. 2024. URL: https://github.com/Tiago13Silva/Red-Light-Running-Detection-Algorithm (ver pp. 5, 37, 42).

[116] P. S.K., V. Hegde, K. Patchava, A. Das e Y. Simmhan. “ Performance Characterization of Containerized DNN Training and Inference on Edge Accelerators ”. Em: *IEEE 30th Int. Conf. High Perform. Comput. Data Analyt.* Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 127–131. DOI: 10.1109/HiPC58850.2023.00028. URL: https://doi.ieeecomputersociety.org/10.1109/HiPC58850.2023.00028 (ver p. 18).

[117] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever e R. Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. Em: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (ver p. xxi).

[118] F. Sultana, A. Sufian e P. Dutta. “A review of object detection models based on convolutional neural network”. Em: *Intelligent computing: image processing based applications* (2020), pp. 1–16 (ver p. 9).

[119] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke e A. Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV] (ver p. 8).

[120] A. A. Süzen, B. Duman e B. Şen. “Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN”. Em: *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. 2020, pp. 1–5. DOI: 10.1109/HORA49412.2020.9152915 (ver p. 16).

[121] B. Vaishali e A. Jeyapriya. “A survey on intelligent system for automobile collision avoidance at intersection of roads”. Em: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. 2017, pp. 1–3. DOI: 10.1109/ICIIECS.2017.8275891 (ver p. 7).

[122] S. Visa, B. Ramsay, A. L. Ralescu e E. Van Der Knaap. "Confusion matrix-based feature selection." Em: *Maics* 710.1 (2011), pp. 120–127 (ver p. 39).

[123] A. Wang, H. Chen, L. Liu e et al. "YOLOv10: Real-Time End-to-End Object Detection". Em: *arXiv preprint arXiv:2405.14458* (2024) (ver p. 12).

[124] B. Wang. *A Parallel Implementation of Computing Mean Average Precision*. 2022. arXiv: 2206.09504 [cs.CV]. URL: https://arxiv.org/abs/2206.09504 (ver pp. 40, 95).

[125] C.-Y. Wang, A. Bochkovskiy e H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV] (ver p. 11).

[126] C.-Y. Wang e H.-Y. M. Liao. "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information". Em: *Arxiv* (2024) (ver p. 11).

[127] Wang, Sun-Chong. "Artificial Neural Network". Em: *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US, 2003, pp. 81–100. ISBN: 978-1-4615-0377-4. DOI: 10.1007/978-1-4615-0377-4_5. URL: https://doi.org/10.1007/978-1-4615-0377-4_5 (ver p. 8).

[128] G. Welch, G. Bishop et al. "An introduction to the Kalman filter". Em: (1995) (ver p. 13).

[129] N. Wojke, A. Bewley e D. Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. arXiv: 1703.07402 [cs.CV]. URL: https://arxiv.org/abs/1703.07402 (ver p. 14).

[130] A. Yasar, M. Adnan, W. Ectors e G. Wets. "Comparison review on LIDAR technologies vs. RADAR technologies in speed enforcement system". Em: *Personal and Ubiquitous Computing* 27.5 (2023), pp. 1691–1700 (ver p. 15).

[131] A. Yilmaz, O. Javed e M. Shah. "Object tracking: A survey". Em: *ACM Comput. Surv.* 38.4 (2006), 13–es. ISSN: 0360-0300. DOI: 10.1145/1177352.1177355. URL: https://doi.org/10.1145/1177352.1177355 (ver p. 12).

[132] X. Ying. "An overview of overfitting and its solutions". Em: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022 (ver pp. xxiii, 43).

[133] L. You, Y. Chen, C. Xiao, C. Sun e R. Li. "Multi-Object Vehicle Detection and Tracking Algorithm Based on Improved YOLOv8 and ByteTrack". Em: *Electronics* 13.15 (2024). ISSN: 2079-9292. DOI: 10.3390/electronics13153033. URL: https://www.mdpi.com/2079-9292/13/15/3033 (ver p. 14).

[134] S. Yun, H. Chen, R. Masukawa, H. E. Barkam, A. Ding, W. Huang, A. Rezvani, S. Angizi e M. Imani. *HyperSense: Accelerating Hyper-Dimensional Computing for Intelligent Sensor Data Processing*. 2024. arXiv: 2401.10267 [cs.AR] (ver p. 18).

[135] N. Yung e A. Lai. "An effective video analysis method for detecting red light runners". Em: *IEEE Transactions on Vehicular Technology* 50.4 (2001), pp. 1074–1084. DOI: 10.1109/25.938581 (ver p. 3).

[136] M. E. Yücel, S. Topaloğlu e C. Ünsalan. *Embedded Planogram Compliance Control System*. 2024. arXiv: 2401.06690 [cs.CV] (ver p. 18).

[137] W. Zhang, K. Wang, L. Wang, Z. Feng e Y. Du. "Exploring factors affecting pedestrians' red-light running behaviors at intersections in China". Em: *Accident Analysis & Prevention* 96 (2016), pp. 71–78. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2016.07.038. URL: https://www.sciencedirect.com/science/article/pii/S0001457516302676 (ver p. 3).

[138] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu e X. Wang. "ByteTrack: Multi-object Tracking by Associating Every Detection Box". Em: *Computer Vision – ECCV 2022", year="2022*. Ed. por S. Avidan, G. Brostow, M. Cissé, G. M. Farinella e T. Hassner. Cham: Springer Nature Switzerland, pp. 1–21. ISBN: 978-3-031-20047-2 (ver pp. 14, 15).

[139] H. Zhou, R. Myrzashova e R. Zheng. "Diabetes prediction model based on an enhanced deep neural network". Em: *EURASIP Journal on Wireless Communications and Networking* 2020 (2020), pp. 1–13 (ver p. 39).

[140] J. Zhou. "A Review of LiDAR sensor Technologies for Perception in Automated Driving". Em: *Academic Journal of Science and Technology* 3.3 (2022), 255–261. DOI: 10.54097/ajst.v3i3.2993. URL: https://drpress.org/ojs/index.php/ajst/article/view/2993 (ver p. 15).

[141] D. Zhu, N. Sze e L. Bai. "Roles of personal and environmental factors in the red light running propensity of pedestrian: Case study at the urban crosswalks". Em: *Transportation Research Part F: Traffic Psychology and Behaviour* 76 (2021), pp. 47–58. ISSN: 1369-8478. DOI: https://doi.org/10.1016/j.trf.2020.11.001. URL: https://www.sciencedirect.com/science/article/pii/S1369847820305635 (ver p. 3).

I Diagrama de fluxo da fase de seleção de regiões do algoritmo

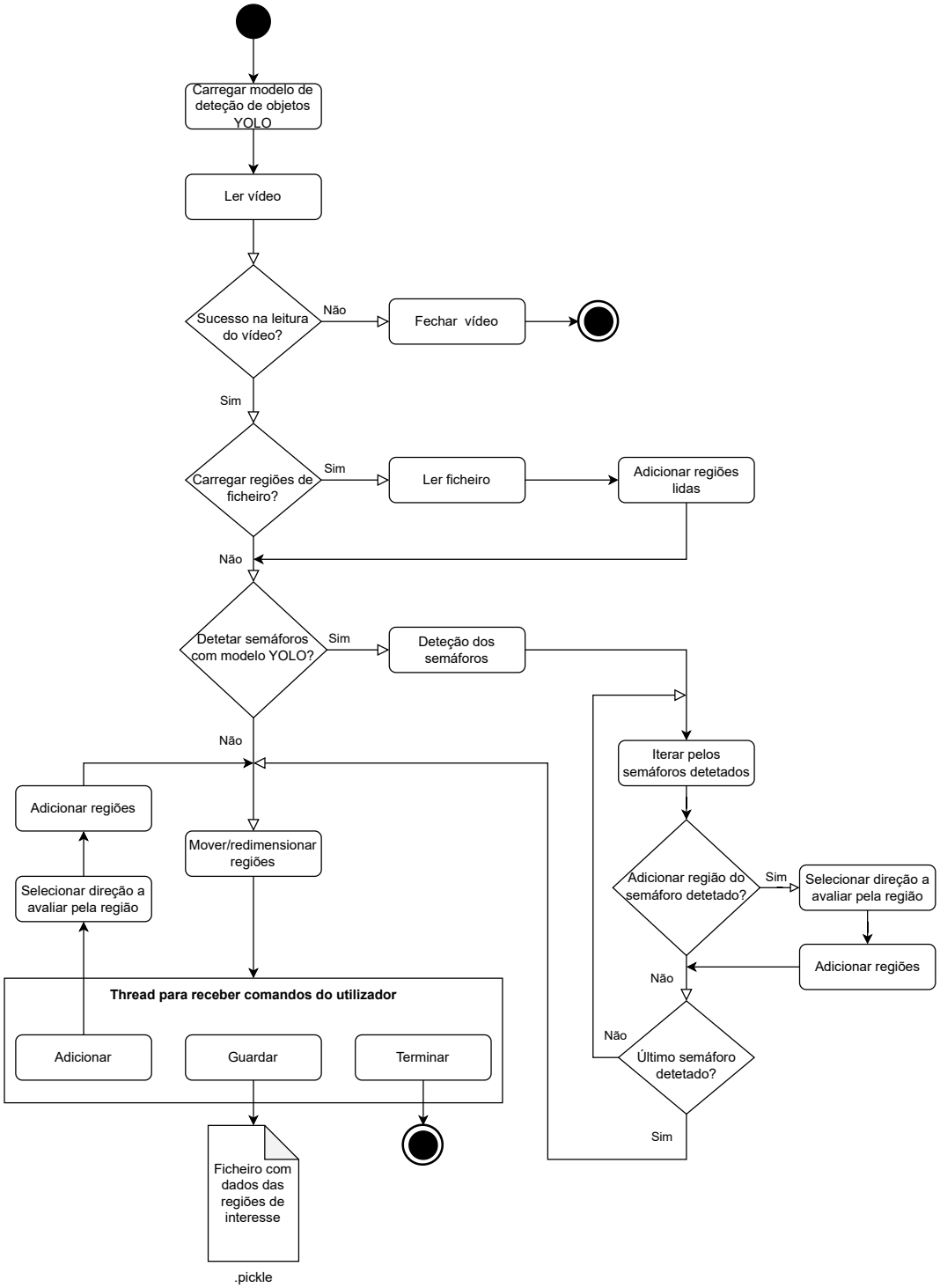
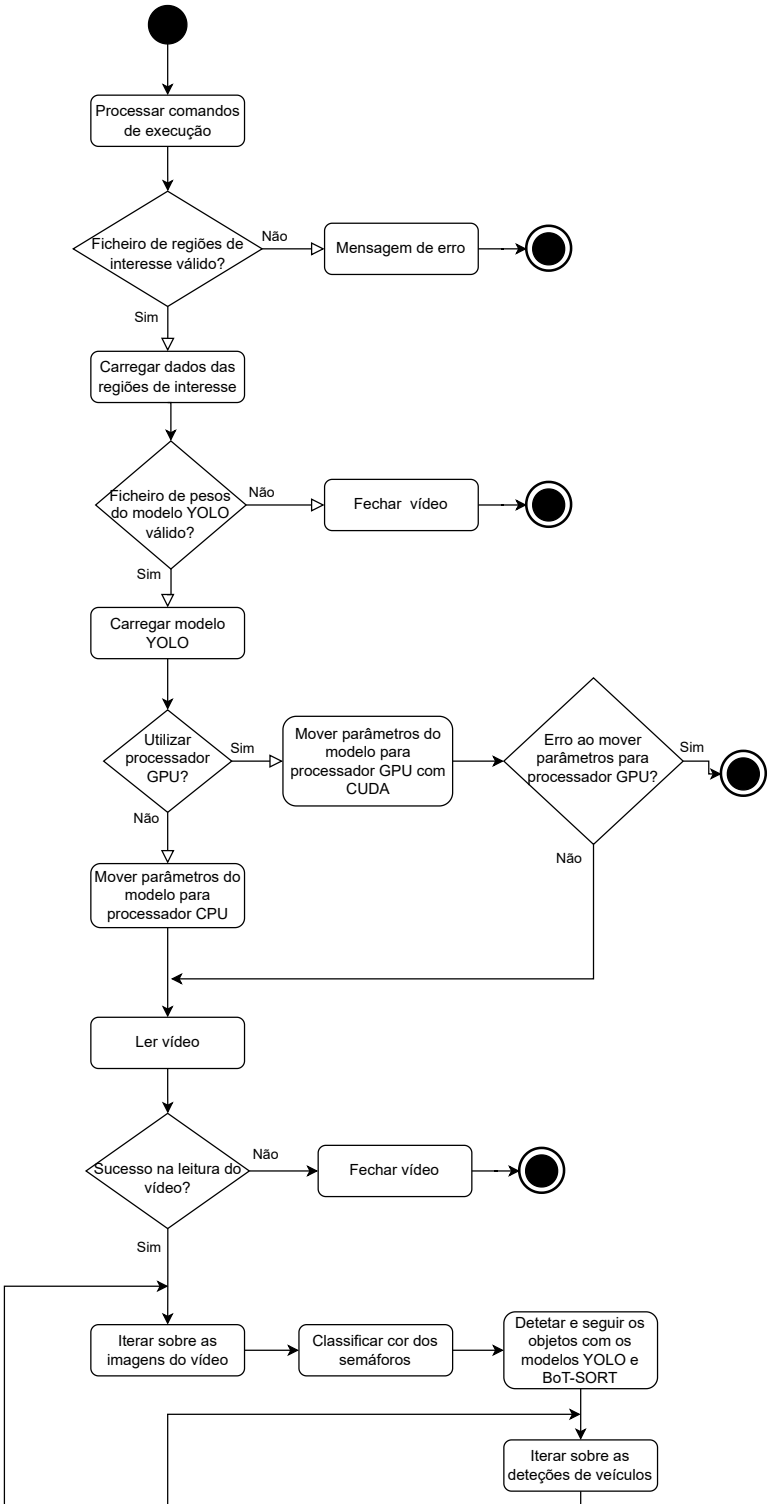


Figura I.1: *Diagrama de fluxo da fase de seleção de regiões do algoritmo.*

II Diagrama de fluxo da fase de processamento de violações RLR do algoritmo


Processar comandos de execução
Ficheiro de regiões de interesse válido?
Não
Mensagem de erro
Sim
Carregar dados das regiões de interesse
Ficheiro de pesos do modelo YOLO válido?
Não
Fechar vídeo
Sim
Carregar modelo YOLO
Utilizar processador GPU?
Sim
Mover parâmetros do modelo para processador GPU com CUDA
Erro ao mover parâmetros para processador GPU?
Sim
Não
Não
Mover parâmetros do modelo para processador CPU
Ler vídeo
Sucesso na leitura do vídeo?
Não
Fechar vídeo
Sim
Iterar sobre as imagens do vídeo
Classificar cor dos semáforos
Detetar e seguir os objetos com os modelos YOLO e BoT-SORT
Iterar sobre as deteções de veículos

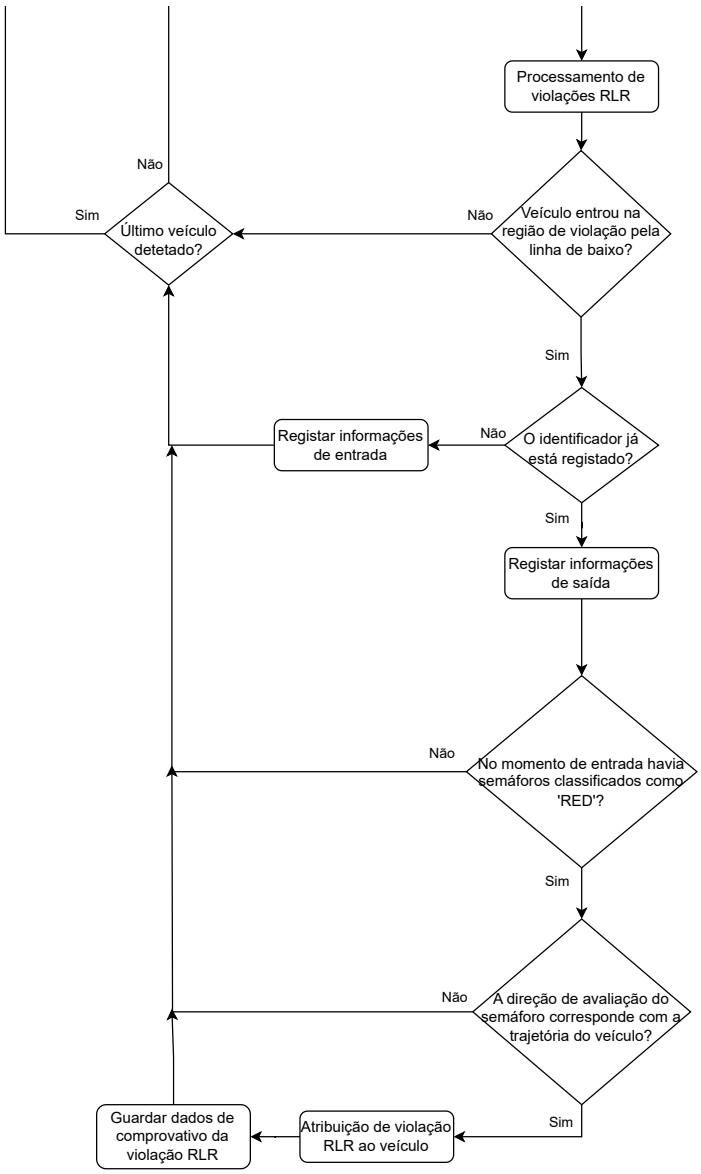


Figura II.1: *Diagrama de fluxo da fase de processamento de violações RLR do algoritmo.*

III Expressões matemáticas das métricas utilizadas

- **Precisão** [109]:

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}} \tag{III.1}$$

- ***Recall*** [109]:

$$\textit{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}} \tag{III.2}$$

- ***F1-score*** [109]:

$$\textit{F1-score} = \frac{2 * \text{Precisão} * \textit{Recall}}{\text{Precisão} + \textit{Recall}} \tag{III.3}$$

- **Eficácia** [109]:

$$\text{Eficácia} = \frac{\text{Verdadeiros Positivos} + \text{Verdadeiros Negativos}}{\text{Verdadeiros Positivos} + \text{Verdadeiros Negativos} + \text{Falsos Positivos} + \text{Falsos Negativos}} \tag{III.4}$$

- ***mean Average Precision*** [83, 124]:
  - C: Número total de classes.
  - AP_t: Precisão Média (*Average Precision*) para a classe t.

$$\text{mAP} = \frac{1}{C} \sum_{t=1}^{C} \text{AP}_t \tag{III.5}$$

- ***Intersection-over-Union*** [83]:

$$\text{IoU} = \frac{\text{Área de Interseção}}{\text{Área de União}} \tag{III.6}$$

- **Taxa de verdadeiros positivos** [109]:

$$\text{TPR} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}} \tag{III.7}$$

- **Imagens por segundo**:

$$\text{FPS} = \frac{\text{Número Total de Imagens Processadas}}{\text{Tempo Total de Execução}} \tag{III.8}$$