



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Estimativa de Velocidade Veicular: Um Modelo de Aprendizado Profundo com YOLOv8 e 1D-CNN

Trabalho de Conclusão de Curso

Marcelo Henrique Lima Barreto



São Cristóvão – Sergipe

2023

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Marcelo Henrique Lima Barreto

**Estimativa de Velocidade Veicular: Um Modelo de
Aprendizado Profundo com YOLOv8 e 1D-CNN**

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Prof. Dr. Leonardo Nogueira Matos
Coorientador(a): MSc. Rafael Andrade da Silva

São Cristóvão – Sergipe

2023

Resumo

O excesso de velocidade é uma das principais causas de mortes no trânsito global, sendo responsável por mais de 1 milhão de mortes anualmente no mundo, exigindo uma fiscalização eficaz. Este estudo tem como objetivo desenvolver um modelo de aprendizado profundo para estimar velocidades veiculares, visando uma abordagem de fácil implementação para o gerenciamento do tráfego rodoviário. Inicialmente, realizou-se uma revisão abrangente da literatura, explorando abordagens e redes neurais usadas na área. A fundamentação teórica incluiu conceitos como redes neurais convolucionais, detecção de objetos, estimativa de velocidade e métricas de avaliação. Através de um experimento comparativo entre dois modelos, Faster R-CNN e YOLOv8, o YOLOv8 nano foi selecionado devido à sua precisão média superior de 0,5 e menor tempo de inferência de 10,48ms. Posteriormente, desenvolveu-se uma base de dados limitada a 40 km/h para treinar e testar o modelo de estimativa de velocidade, baseado em variações nas áreas das caixas delimitadoras dos veículos. O modelo alcançou um Root Mean Square Error (RMSE) de 1,63 km/h, com erro máximo de 5 km/h, dentro da margem tolerada por fiscalizações eletrônicas de rodovias brasileiras que é de 7km/h ou 10% da velocidade máxima. Essa abordagem versátil pode ser aplicada não apenas em rodovias, mas em diversas áreas com fluxos de veículos. Trabalhos futuros podem expandir o modelo para diferentes condições climáticas e velocidades superiores a 40 km/h, visando uma maior generalização.

Palavras-chave: estimativa de velocidade. aprendizado profundo. visão computacional. detecção de objetos. YOLOv8. 1D-CNN.

Abstract

Excessive speeding is a major contributor to global traffic fatalities, resulting in over one million annual deaths worldwide. This study presents a deep learning model for vehicle speed estimation, offering a readily implementable solution for traffic management. Beginning with a comprehensive literature review, including investigations into various neural network methods, our research delves into essential concepts such as convolutional neural networks, object detection, speed estimation, and evaluation metrics. Through comparative experiments involving Faster R-CNN and YOLOv8 models, YOLOv8 nano emerged as the preferred choice due to its superior average accuracy of 0.5 and shorter inference time of 10.48 ms. Subsequently, we developed a dataset constrained to a 40 km/h speed limit for training and testing the speed estimation model, relying on variations in vehicle bounding box regions. The model achieved an Root Mean Square Error (RMSE) of 1.63 km/h, with a maximum error of 5 km/h, well within the tolerable range for electronic brazilian road detection, i.e., 7 km/h or 10% of the maximum speed. This versatile approach is not limited to highways but is applicable to diverse vehicular traffic scenarios. Future work may expand the model to accommodate different climatic conditions and speeds exceeding 40 km/h, enhancing its adaptability.

Keywords: speed estimation. deep learning. computer vision. object detection. YOLOv8. 1D-CNN.

Listas de ilustrações

Figura 1 – Rede Neural Artificial	20
Figura 2 – Processo de Convolução	22
Figura 3 – Entrada de uma CNN	23
Figura 4 – <i>Feature maps</i> da CNN	23
Figura 5 – Processo do MaxPooling	24
Figura 6 – Processo do AvgPooling	24
Figura 7 – Processo da função ReLU	26
Figura 8 – Matriz de confusão	27
Figura 9 – Matriz de Confusão multi classe	27
Figura 10 – <i>Bounding box</i> original	28
Figura 11 – <i>Bounding box</i> inferida	28
Figura 12 – Técnica de <i>data augmentation</i>	30
Figura 13 – Técnica de NMS	31
Figura 14 – Técnica de <i>fine-tuning</i>	32
Figura 15 – Etapas de construção de modelo para dispositivos de borda	33
Figura 16 – Estrutura de uma CNN	35
Figura 17 – Base de dados MNIST	36
Figura 18 – Autoestrada sem segmentação	36
Figura 19 – Autoestrada segmentada	36
Figura 20 – Modelo U-net	37
Figura 21 – Exemplo de detecção de objetos	37
Figura 22 – Camadas do Faster R-CNN	38
Figura 23 – Estrutura do modelo VGG16	38
Figura 24 – <i>Region Proposal Network</i>	39
Figura 25 – <i>RoI Pooling</i>	40
Figura 26 – Etapas de uma imagem no YOLO	41
Figura 27 – Comparação entre imagem real e imagem desenvolvida por câmeras LiDAR	43
Figura 28 – Matriz de Homografia	44
Figura 29 – Estimativa de Velocidade dos veículos	46
Figura 30 – Cena de teste do 2D DLT	49
Figura 31 – Arquitetura do modelo de estimativa de velocidade desenvolvido por Wu et al. (2021b)	51
Figura 32 – CBBA da velocidade de 3 veículos	53
Figura 33 – Estrutura do ECCNet	56
Figura 34 – Esquema de medição de linha da via	60
Figura 35 – Detecção de objetos com Yolov8m	67

Figura 36 – Detecção de objetos com Yolov8s	67
Figura 37 – Detecção de objetos com Faster R-CNN	68
Figura 38 – Detecção de objetos com Yolov8s	68
Figura 39 – Amostra da base de dados Vs13	70
Figura 40 – Amostra da base de dados UTFPR	71
Figura 41 – Amostra da base de dados UA-DETRAC	72
Figura 42 – Visor dos sensores da fiscalização eletrônica	74
Figura 43 – Trecho do vídeo sem efeito	74
Figura 44 – Trecho do vídeo com efeito	74
Figura 45 – Amostra da base de dados VSE	75
Figura 46 – Fluxo de Trabalho do Projeto	77
Figura 47 – Rastreio de Veículos	78
Figura 48 – Arquitetura do Modelo 1D-CNN	80
Figura 49 – Detecção de multi-veículos	81

Lista de quadros

Quadro 1 – Dados dos modelos do Yolov8 pré-treinado	42
Quadro 2 – Resultados obtidos no trabalho de Wu et al. (2021b)	52
Quadro 3 – Resultados obtidos no experimento de comparação do Faster R-CNN e YOLOv8	67
Quadro 4 – Localização da gravação da base de dados	72
Quadro 5 – Parâmetros da gravação	73
Quadro 6 – Resultados obtidos nos experimentos de estimativa de velocidade	84
Quadro 7 – Comparação entre a velocidade real e a estimada pelo modelo desenvolvido	85

Lista de tabelas

Tabela 1 – Funções de Ativação	25
Tabela 2 – Resultados da pesquisa dos trabalhos relacionados	48
Tabela 3 – Resultados de exclusão dos trabalhos relacionados	48
Tabela 4 – Resultados obtidos no trabalho de Cvijetić, Djukanović e Peruničić (2023) .	53
Tabela 5 – Resultados obtidos no trabalho de Dong, Wen e Yang (2019)	54
Tabela 6 – Resultados obtidos no trabalho utilizando RNN de Cvijetić, Djukanović e Peruničić (2023)	56
Tabela 7 – Resultados obtidos no trabalho de Yu et al. (2022)	57
Tabela 8 – Resultados obtidos no trabalho de Lin, Jeng e Lioa (2021)	58
Tabela 9 – Resultados obtidos no trabalho utilizando áudio dos veículos	59
Tabela 10 – Resultados obtidos no trabalho de Yi, Guan e Li (2021)	61
Tabela 11 – Análise Comparativa dos trabalhos da Revisão Sistemática	63
Tabela 14 – Quantidade de veículos por gravação	75
Tabela 15 – Velocidade dos Veículos	76

Lista de códigos

Código 1 – Preparação dos dados do experimento com Faster R-CNN	100
Código 2 – Criação do modelo Faster R-CNN	101
Código 3 – Funções auxiliares que ajudaram no treinamento do modelo Faster R-CNN	101
Código 4 – Treinamento do modelo Faster R-CNN	102
Código 5 – Funções auxiliares que ajudaram na inferência do modelo Faster R-CNN . .	103
Código 6 – Função de inferência com Faster R-CNN	104
Código 7 – Inferência de dados do experimento com Faster R-CNN	105
Código 8 – Funções auxiliares que ajudaram na preparação dos dados do modelo YOLOv8	106
Código 9 – Preparação dos dados do experimento com YOLOv8	107
Código 10 – Treinamento do modelo YOLOv8	108
Código 11 – Funções auxiliares da inferência dos dados com YOLOv8	109
Código 12 – Função de Inferência do experimento com YOLOv8	110
Código 13 – Inferência dos dados do experimento com YOLOv8	111

Lista de abreviaturas e siglas

ALPR	Automatic License Plate Recognition
ANN	Artificial Neural Network
AP	Average Precision
AE	Absolute Error
CBAM	Convolutional Block Attention Module
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
CUDA	Compute Unified Device Architecture
DLT	Direct Linear Transformation
FN	False Negative
FP	False Positive
FPS	Frames Per Second
GRAMRTM	GARM Road-Traffic Monitoring
GPU	Graphics Processing Unit
IOT	Internet Of Things
IOU	Intersection Over Union
LIDA	Light Detection and Ranging
LNN	Lightweight Neural Network
LSTM	Long Short-Time Memory
MAE	Mean Absolute Error
MAP	Mean Average Precision
MAPE	Mean Absolute Percentage Error
MAVD	Montevideo Audio and Video Dataset

MFCAM	Multi-scale Fusion of Channel Attention Model
MLP	MultiLayer Perceptron
MSE	Mean Square Error
NPU	Neural Processing Unit
NRMSE	Normalized Root Mean Square Error
PAE	Percentage Absolute Error
RAM	Random Access Memory
RANSAC	Random Sample Consensus
RE	Relative Error
RELU	Rectified Linear Unit
RMSE	Root Mean Square Error
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SPP	Spatial Pyramid Pooling
SSD	Single Shot Detector
TANH	Hyperbolic Tangent
TN	True Negative
TOPS	Trillion Operations Per Second
TP	True Positive
YOLO	You Only Look Once

Lista de símbolos

σ Letra grega sigma

μ Letra grega mu

Sumário

1	Introdução	15
1.1	Motivação	16
1.2	Objetivo	16
1.3	Metodologia	17
1.4	Estrutura do Documento	18
2	Conceitos Básicos	19
2.1	Redes Neurais Convolucionais (CNN)	19
2.1.1	Estrutura das CNNs	21
2.1.2	Métricas de avaliação	26
2.1.2.1	Matriz de Confusão	26
2.1.2.2	<i>Intersection over Union</i> (IoU)	28
2.1.2.3	Mean Average Precision(MAP)	29
2.1.3	Técnicas de melhorias de desempenho	29
2.1.3.1	Data Augmentation	29
2.1.3.2	Normalização	30
2.1.3.3	Non Maximum Suppression (NMS)	30
2.1.3.4	Fine-tuning	30
2.1.3.5	Compressão de Modelo	32
2.1.4	Aplicações na Visão Computacional	34
2.1.4.1	Classificação de Imagens	35
2.1.4.2	Segmentação de Imagens	36
2.1.4.3	Detecção de objetos	37
2.1.4.3.1	Faster R-CNN	37
2.1.4.3.2	<i>You Only Look Once</i> (YOLO)	40
2.2	Sistema de Estimativa de Velocidade Veicular	42
3	Trabalhos Relacionados	47
3.1	Análise de Publicações Selecionadas	48
3.1.1	The Vehicle Speed Measuring and Precision Analysis of Video Shot by Moved Camera Based on 2D DLT Algorithm	49
3.1.2	Vehicle Speed Estimation Using Computer Vision And Evolutionary Camera Calibration	50
3.1.3	Design and Implementation of Vehicle Speed Estimation Using Road Marking-based Perspective Transformation	51

3.1.4	Deep learning-based vehicle speed estimation using the YOLO detector and 1D-CNN	52
3.1.5	Vehicle Speed Estimation Based on 3D ConvNets and Non-Local Blocks	53
3.1.6	Vision-based Vehicle Speed Estimation Using the YOLO Detector and RNN	55
3.1.7	ECCNet: Efficient chained centre network for real-time multi-category vehicle tracking and vehicle speed estimation	56
3.1.8	A Real-Time Vehicle Counting, Speed Estimation, and Classification System Based on Virtual Detection Zone and YOLO	57
3.1.9	Vehicle Speed Estimation From Audio Signals Using 1D Convolutional Neural Networks	58
3.1.10	Intelligent Highway Speed Monitoring UAV System Based on Deep Learning	59
3.2	Análise Comparativa	62
4	Experimentos de Comparaçāo do Faster R-CNN e YOLOv8	65
4.1	Preparação dos dados	65
4.2	Treinamento do modelo	66
4.3	Inferência	66
4.4	Resultados	66
5	Construāo da Base de Dados	69
5.1	Outras bases de dados para estimativa de velocidade	70
5.1.1	Vs13 dataset	70
5.1.2	UTFPR Dataset	70
5.1.3	UA-DETRAC	71
5.2	Base de Dados VSE	72
6	Construāo do Modelo	77
6.1	Modelo de Detecāo e Rastreio dos Veículos	78
6.2	Modelo de Estimativa de Velocidade	79
6.3	Experimentos e Resultados	81
7	Conclusāo	88
	Referências	90

Apêndices **98**

APÊNDICE A Códigos do experimento comparando Faster R-CNN e YOLOv8 . . . **99**

1

Introdução

De acordo com o censo realizado pelo [IBGE \(2022\)](#), o Brasil contava com cerca de 115 milhões de veículos, sendo 60 milhões deles automóveis, o que resulta em um alto volume de tráfego nas rodovias. Esse intenso fluxo de veículos torna o gerenciamento do trânsito essencial, e para isso, são utilizadas diversas medidas, como semáforos, placas de trânsito e fiscalização de velocidade.

De acordo com [Rodrigues \(2018\)](#), estudos realizados pela Organização das Nações Unidas (ONU) e pela Organização Mundial da Saúde (OMS) demonstraram que o excesso de velocidade figura como uma das principais causas de acidentes no trânsito, sendo responsável por mais de 1 milhão de mortes anualmente em todo o mundo. Além disso, esse problema afeta principalmente a faixa etária entre 5 e 29 anos. Estudos demonstram que uma redução de apenas 5% na velocidade média pode resultar em uma redução significativa de até 30% nos acidentes ([ORGANIZATION et al., 2020](#)).

O Brasil figura entre os países com maior número de vítimas de acidentes de trânsito, ficando atrás apenas da Índia, China, Rússia e Estados Unidos ([PEREIRA; TRÓI, 2022](#)). Sendo que, 48,5% das mortes de pedestres e 48,5% das mortes de ciclistas envolvem algum tipo de automóvel em tais acidentes, o que poderia ser evitado caso os motoristas agissem de forma mais prudente no trânsito, evitando comportamentos como dirigir sob influência de álcool, ultrapassar a velocidade permitida, desrespeitar semáforos e cometer outras imprudências.

Dentre as infrações mais frequentes cometidas por motoristas no Brasil, o excesso de velocidade lidera a lista, sendo responsável por cerca de 10% dos acidentes registrados no ano de 2021 ([JUSBRASIL, 2022](#)).

Diante desses dados, nota-se a importância de buscar soluções eficazes para o controle e a conscientização sobre a velocidade nas vias, a fim de promover um trânsito mais seguro e evitar acidentes e suas consequências.

1.1 Motivação

No cenário atual, a estimativa de velocidade desempenha um papel crucial ao contribuir com ferramentas de fiscalização e monitoramento eficientes, tornando-se uma aliada fundamental na aplicação de medidas preventivas e na promoção de um trânsito mais responsável e seguro. Nesse contexto, técnicas como *deep learning* surgem como uma abordagem promissora para realizar essa estimativa.

O *deep learning* tem ganhado popularidade como uma solução eficaz para enfrentar problemas complexos, comprovando seu desempenho eficaz em diversas aplicações, como direção autônoma de veículos e assistentes virtuais (CHOI; KWAK, 2023).

A busca por novas ferramentas de estimativa de velocidade visa encontrar soluções eficientes e com custo reduzido. Dentre os métodos mais utilizados para mensurar a velocidade estão os radares *light detection and ranging* (LiDAR), que utilizam sensores laser para detecção, sensores sob o pavimento da estrada e aparelhos que utilizam o efeito Doppler¹.

Assim, explorar as vantagens do *deep learning* para esse propósito é um caminho promissor em direção a um tráfego mais seguro e uma fiscalização mais acessível. Com a evolução dessas técnicas, espera-se otimizar o uso de recursos e possibilitar a implementação de soluções mais eficientes no monitoramento e controle do trânsito, contribuindo para a redução de acidentes e a melhoria geral da segurança viária.

Além disso, essas técnicas podem ser aplicadas em outras áreas de gerenciamento do tráfego nas estradas. Por exemplo, é possível utilizar a detecção de veículos para verificar se algum carro possui alguma restrição jurídica e alertar as autoridades competentes. Outra possibilidade é no desenvolvimento de veículos autônomos, o que poderia reduzir o impacto das imprudências dos motoristas nos acidentes de trânsito. Essas aplicações adicionais demonstram o potencial do *deep learning* como uma ferramenta versátil e promissora para melhorar a segurança e a eficiência do tráfego nas vias públicas.

1.2 Objetivo

O objetivo deste trabalho é desenvolver um modelo de *deep learning* para estimativa de velocidade que seja tão eficiente quanto as tecnologias existentes, de fácil implementação e acessível, podendo ser aplicado em vias públicas, assim como em locais com fluxos menores, como condomínios ou universidades. Além disso, pretende-se contribuir com uma base de dados para pesquisadores futuros, uma vez que há poucas bases de dados disponíveis sobre esse tema e se busca preencher essa lacuna na disponibilidade de dados sobre esse tema.

¹ O efeito Doppler é um fenômeno físico que ocorre quando há movimento relativo entre a fonte de onda e o observador. Isso resulta no aumento ou diminuição da frequência da onda conforme a distância entre os corpos se altera.

Para alcançar esse objetivo geral, foram definidos alguns objetivos específicos a serem cumpridos, são eles:

- Analisar modelos de estimativa de velocidade já existentes na literatura;
- Criar uma base de dados adequada para o treinamento do modelo proposto;
- Realizar o treinamento do modelo utilizando a base de dados construída;
- Realizar testes para verificar a acurácia e o desempenho do modelo em relação à estimativa de velocidade dos veículos.

Ao cumprir esses objetivos específicos, espera-se obter um modelo de estimativa de velocidade eficiente e preciso, que possa contribuir para a segurança viária e o controle do tráfego nas estradas. Com a conclusão deste trabalho, espera-se também fornecer informações relevantes para a aplicação bem-sucedida de técnicas de *deep learning* em outras áreas de visão computacional e processamento de imagens relacionadas.

1.3 Metodologia

Para realizar este trabalho, será necessário iniciar com uma revisão sistemática das bibliotecas científicas digitais, como a IEE Xplore Digital Library, ScienceDirect e ACM Digital Library. O objetivo dessa revisão é coletar e investigar as abordagens atualmente utilizadas no desenvolvimento de modelos de detecção de objetos para sistemas de tempo real e modelos de estimativa de velocidade. Através dessa revisão, obter-se-á uma visão do estado-da-arte nessa área, servindo como base para o desenvolvimento do modelo proposto. Além disso, será possível identificar e explorar técnicas que são utilizadas para aprimorar o desempenho desses modelos. Essa revisão sistemática será fundamental para embasar o trabalho e garantir que as soluções propostas estejam alinhadas com as melhores práticas e avanços recentes na área de detecção de objetos em tempo real.

Após a conclusão da revisão sistemática, proceder-se-á com o processamento dos dados a serem utilizados no treinamento do modelo. Esse processamento envolverá a criação da base de dados, incluindo a inserção das *bounding boxes* para cada veículo identificado. Essas *bounding boxes* delimitam a região onde o veículo está presente na imagem e são essenciais para o treinamento adequado do modelo de detecção de objetos. Essa etapa de tratamento dos dados garantirá que o modelo tenha acesso a informações precisas durante o processo de treinamento, contribuindo para o seu desempenho.

Em seguida, o modelo será treinado e testado com o objetivo de encontrar a melhor configuração dos parâmetros que resulte em um desempenho satisfatório para a tarefa em questão.

Após o treinamento e teste do modelo, serão analisados e discutidos os resultados obtidos durante o trabalho. Por fim, apresentação das considerações finais do trabalho, com suas contribuições, e possíveis trabalhos futuros.

1.4 Estrutura do Documento

Este documento está estruturado em capítulos, de forma a facilitar a compreensão do leitor, conforme descrito abaixo:

1. **Capítulo 1 - Introdução:** Apresenta a contextualização do tema, os objetivos do trabalho, a justificativa para a escolha do tema e a organização do documento.
2. **Capítulo 2 - Conceitos Básicos:** Aborda os conceitos fundamentais e relevantes para a compreensão deste trabalho, fornecendo o embasamento teórico necessário para acompanhar os capítulos subsequentes.
3. **Capítulo 3 - Trabalhos Relacionados:** Apresenta uma revisão dos trabalhos acadêmicos e projetos relevantes que foram consultados e que serviram como base e inspiração para o desenvolvimento deste projeto.
4. **Capítulo 4 - Experimentos de Comparação do Faster R-CNN e YOLOv8:** Descreve em detalhes o experimento realizado com o objetivo de comparar o desempenho do modelo Faster R-CNN com alguns modelos do YOLOv8.
5. **Capítulo 5 - Construção da Base de Dados:** Apresenta o passo a passo utilizado na elaboração da base de dados.
6. **Capítulo 6 - Construção do Modelo:** Apresenta o passo a passo utilizado na construção do modelo, detalhando as etapas de preparação dos dados e treinamento do modelo de estimativa de velocidade. Além, dos experimentos realizados e resultados obtidos no trabalho.
7. **Capítulo 7 - Conclusão:** Finaliza este trabalho, resumindo as etapas realizadas, as principais descobertas e conclusões obtidas. Além disso, serão apresentadas as contribuições do trabalho para a área de estimativa de velocidade e possíveis futuros trabalhos relacionados.

2

Conceitos Básicos

Neste capítulo, discute-se alguns conceitos necessários para entendimento do projeto, desde modelos, ferramentas e métricas utilizadas.

Na Seção 2.1, serão apresentados detalhes importantes sobre redes neurais convolucionais que são necessários para a compreensão deste trabalho. Na Seção 2.2, serão apresentadas informações detalhadas sobre as técnicas atualmente empregadas para a estimativa de velocidade de veículos.

2.1 Redes Neurais Convolucionais (CNN)

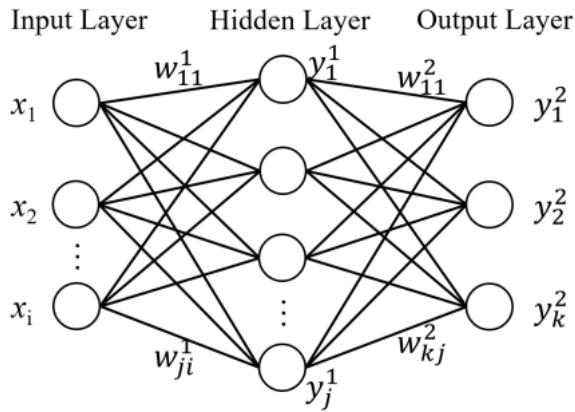
A rede neural convolucional, também conhecida como *convolutional neural network* (CNN), é um tipo de arquitetura de rede neural artificial que utiliza camadas de convolução para processar os dados. A convolução, de acordo com [Viceri-Seidor \(2020\)](#), envolve o somatório dos produtos de duas funções ao longo de uma região de deslocamento. Essa operação de convolução é amplamente utilizada na subárea de processamento de imagens para extrair características relevantes das imagens.

As redes neurais artificiais, também conhecidas como *artificial neural networks* (ANN), são modelos computacionais inspirados nas redes neurais do cérebro humano ([NEILY et al., 1991](#)). As ANNs possuem elementos de processamento chamados de neurônios, que são interligados por sinapses responsáveis pela comunicação entre eles. A ANN é composta por uma função de ativação que determina se os dados de entrada de um neurônio serão enviados para outro neurônio ([LAI; COGHILL, 1991](#)), e pesos que são ajustados visando alcançar o valor de solução do problema.

A [Figura 1](#) exemplifica a estrutura de uma rede neural artificial com suas camadas e conexões entre neurônios. As ANNs são amplamente utilizadas em problemas de classificação, regressão, processamento de imagens e reconhecimento de fala, entre outros ([PEI et al., 2019](#)).

A flexibilidade e eficiência das redes neurais artificiais as tornam uma ferramenta valiosa na solução de problemas complexos de natureza diversa.

Figura 1 – Rede Neural Artificial



Fonte: [Yang et al. \(2018\)](#)

A estrutura básica de uma rede neural artificial é composta por n camadas, incluindo a camada de entrada, a camada de saída e as camadas intermediárias, conhecidas como camadas escondidas ([PIMENTEL, 2014](#)). A quantidade de camadas e neurônios pode variar de acordo com a rede em questão. Cada neurônio de uma ANN possui a seguinte estrutura:

- X_n : Valor de entrada de um neurônio;
- W_n : Os pesos de cada conexão entre os neurônios, esses pesos são multiplicados com os valores de entrada. Esses pesos servem para ajustar os valores da rede;
- Função de ativação: Função responsável por decidir se um pulso será enviado para o próximo neurônio ou não. Existem diversas funções de ativação, mas em geral, na fórmula delas, elas precisam do somatório dos valores de entrada multiplicado com seus respectivos pesos daquele neurônio;
- Y_n : Valor de saída de cada neurônio, ou seja, o pulso que será enviado para o próximo neurônio da próxima camada.

Os cálculos realizados em uma CNN são independentes entre si, o que possibilita o uso de técnicas de paralelismo para acelerar a sua execução. Nesse contexto, as GPUs (Unidades de Processamento Gráfico) são amplamente utilizadas devido à sua capacidade de realizar cálculos matemáticos de forma rápida e eficiente. As GPUs foram projetadas com mecanismos de paralelismo e alta largura de banda, como mencionado por [Chen e Lin \(2019\)](#), o que as torna uma opção mais eficiente para o aprendizado profundo em comparação com as CPUs. Um exemplo é

a NVidia Tesla K80¹, que possui mais de 8 teraflops de capacidade de processamento, permitindo realizar trilhões de operações de ponto flutuante por segundo. Essa capacidade computacional das GPUs as torna especialmente eficientes em aplicações que exigem intensos cálculos matemáticos.

À medida que os dados e a quantidade de parâmetros em um modelo de aprendizado profundo aumentam, a demanda por recursos de hardware, como memória e poder de processamento, também cresce (KAVARAKUNTZA et al., 2021). Sendo assim, o uso de paralelismo nos cálculos realizados durante o treinamento e a inferência de modelos contribui significativamente para o aumento do desempenho das CNNs, permitindo o treinamento de modelos mais complexos.

Um exemplo dessa aplicação de paralelismo está no cálculo da convolução para cada pixel da imagem. Como o cálculo de cada pixel é independente dos demais, é possível realizar esses cálculos em paralelo, distribuindo o processamento entre vários núcleos de processamento ou dispositivos, como GPUs. Essa abordagem paralela acelera significativamente o processamento das CNNs, permitindo o treinamento mais rápido e a realização de inferências em tempo real.

2.1.1 Estrutura das CNNs

As CNNs são compostas por várias camadas que formam sua estrutura, entre elas, as camadas de convolução e *pooling*. A camada de convolução é o componente principal em uma rede neural convolucional. Nessa camada, um processo de convolução é aplicado aos dados de entrada, utilizando uma matriz auxiliar denominada *kernel* (HUANG et al., 2023). Esse processo tem como objetivo extrair características importantes da imagem, como bordas, texturas e padrões, que são representadas pelos valores gerados na matriz de saída, esses valores são chamados de *feature maps* ou mapa de características.

O processo de convolução consiste em percorrer a imagem com o *kernel*, multiplicando seus valores pelos valores correspondentes da imagem em cada posição, somando esses valores e gerando um único valor na saída da operação. Esse valor é adicionado na matriz de saída da camada de convolução na posição correspondente à posição central do *kernel* na imagem (SUBRAMANIAN, 2018). Esse processo é repetido para cada posição possível do *kernel* na imagem, além disso, o processo é realizado para cada canal da imagem. Os canais são uma das dimensões da imagem, e a imagem geralmente possui 3 dimensões: altura, largura e o número de canais. Por exemplo, uma imagem RGB possui 3 canais, o canal da cor vermelha, da cor verde e da azul, ou seja, a convolução é realizada para cada um desses canais, gerando um conjunto de mapas de características para cada canal.

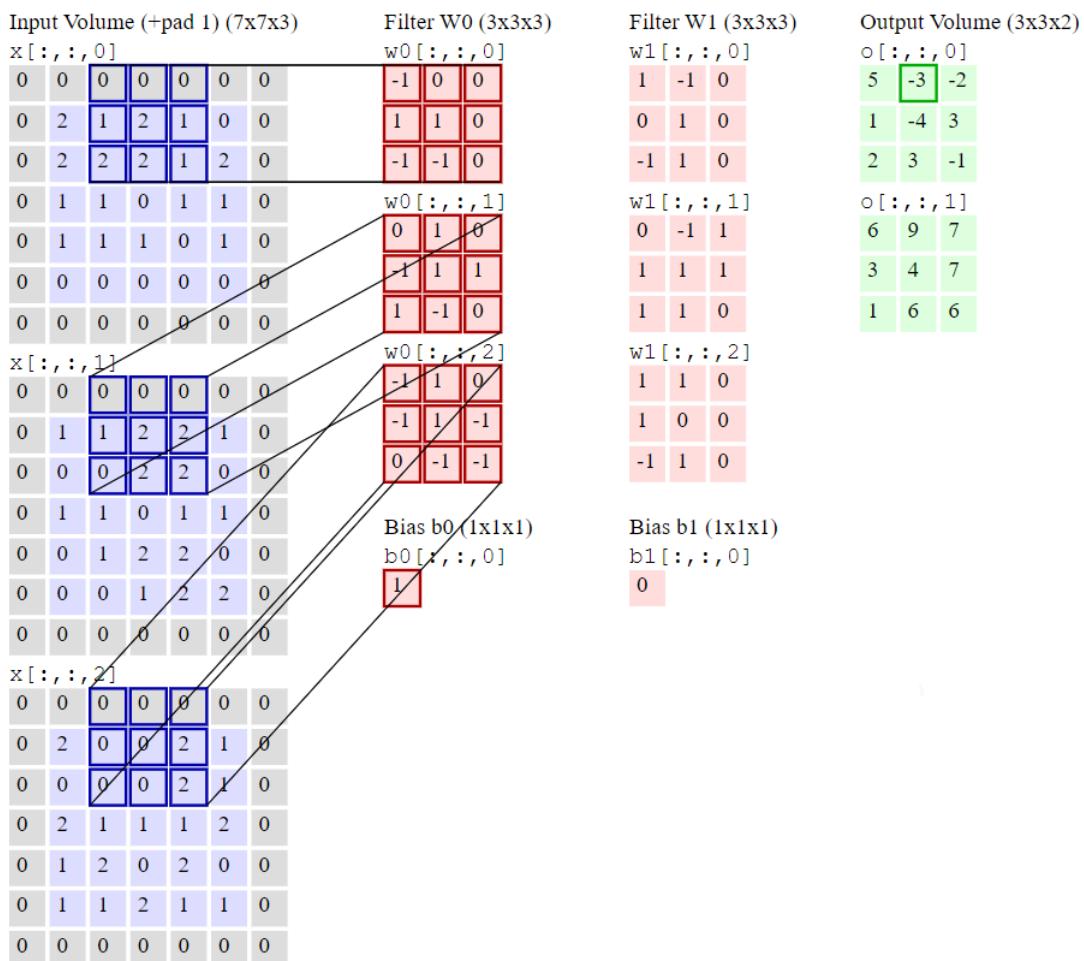
Cada índice do *kernel* possui um peso que é atribuído a ele, sendo responsável por determinar a importância daquela informação para a rede neural. Além dos pesos, outro parâmetro importante é o viés ou *bias*, que tem a função de ajudar a evitar *overfitting* do modelo (KARPATY, 2016). Durante a operação da matriz e do *kernel*, o viés é somado ao resultado. Ao final de cada

¹ Disponível em: <<https://www.nvidia.com/pt-br/data-center/tesla-k80/>>. Acesso em: abril de 2023

época ou *epoch* do modelo, os pesos e vieses de cada neurônio são ajustados de acordo com o erro obtido durante o treinamento. Esse processo é chamado de aprendizado, e é importante para melhorar o desempenho do modelo.

A quantidade de mapas de características gerados por essa operação determinará a nova dimensão de canais da imagem resultante. O tamanho da dimensão de entrada e de saída são passados como parâmetros para a função de convolução.

Figura 2 – Processo de Convolução



Fonte: KARPATHY (2016)

A Figura 2 demonstra o funcionamento da camada de convolução, além de alguns de seus parâmetros. A camada de convolução utiliza o parâmetro *stride* para determinar o número de passos que o *kernel* dará em cada iteração. Por exemplo, se o valor de *stride* for 2, o *kernel* andará dois passos para a direita da imagem a cada iteração. Outro parâmetro utilizado é o *padding*, que é utilizado quando o *kernel* não consegue percorrer todos os dados de entrada, sendo assim, adicionando zeros nas bordas dos tensores para que o *kernel* possa realizar o processo (SUBRAMANIAN, 2018). Tensor é um conceito matemático e físico que representa objetos

multidimensionais ([WANG; SEIGAL, 2023](#)).

Em *frameworks* como o Pytorch, é comum usar uma estrutura de dados denominada tensor para manipular objetos multidimensionais. Esses tensores são representados por variáveis do tipo *nd-tensor*, onde *nd* é o número de dimensões do tensor ou também chamada de *rank*. Escalares são tensores de ordem zero, vetores são tensores unidimensionais, matrizes são tensores bidimensionais, e objetos de três ou mais dimensões são tensores de ordem superior.

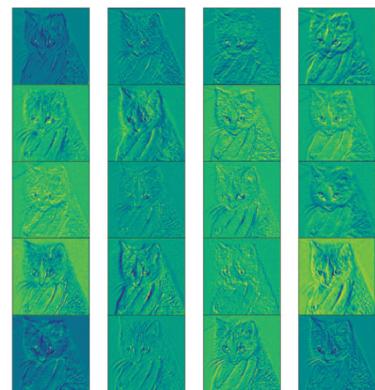
Na [Figura 3](#) e na [Figura 4](#) é possível verificar uma imagem, e seu resultado após passar por uma camada de convolução. Essas figuras permitem visualizar as informações que uma camada de convolução pode extrair de uma imagem.

Figura 3 – Entrada de uma CNN



Fonte: [Subramanian \(2018\)](#)

Figura 4 – *Feature maps* da CNN



Fonte: [Subramanian \(2018\)](#)

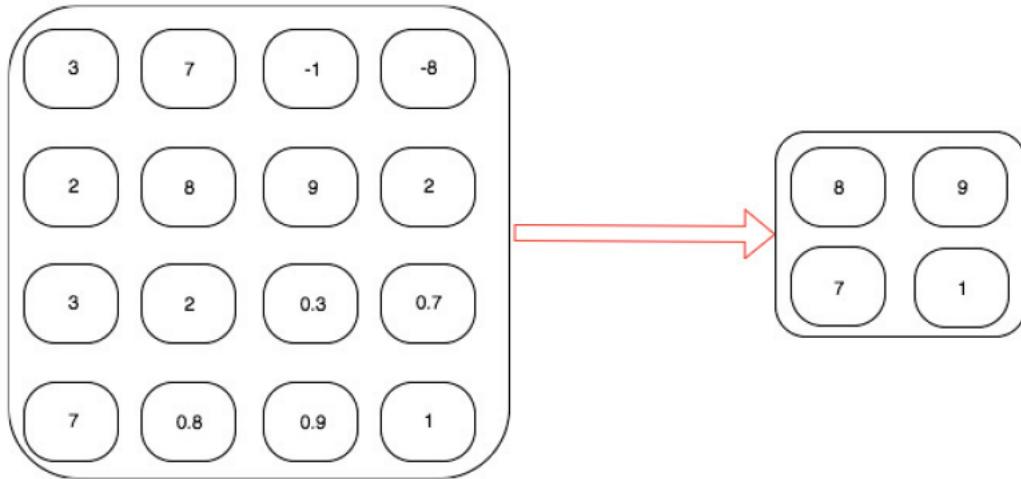
Diferentemente das camadas densas, ou também chamadas de totalmente conectadas, as camadas de convolução mantêm a dimensão espacial e temporal dos dados.

A camada de *pooling* é utilizada para reduzir a dimensionalidade dos dados, o que pode melhorar a eficiência do modelo e reduzir o risco de *overfitting*. A camada de *pooling* é frequentemente usada após as camadas de convolução ([HALAWA; WIBOWO; ERNAWAN, 2019](#)). As variações mais comuns de *pooling* são: *MaxPooling* e *AvgPooling*. No *MaxPooling*, o *kernel* passa pela imagem e, em cada iteração, o maior valor encontrado é inserido na matriz de saída da camada. No *AvgPooling*, o valor inserido na matriz resultante é a média dos elementos que o *kernel* está analisando. Embora existam outras variações de *pooling*, por exemplo, com a média ponderada, essas são as mais utilizadas. A exemplificação do *MaxPooling* e *AvgPooling* é demonstrada na [Figura 5](#) e [Figura 6](#).

No exemplo da [Figura 5](#), o *kernel* utilizado possui tamanho 2. Na primeira iteração o *kernel* analisa os valores 3,7, 2 e 8, e verifica qual é o maior valor desses elementos, nesse caso, o 8, sendo assim, o 8 é inserido na matriz resultante, após isso, o *kernel* verifica os próximos elementos em uma matriz 2x2, nesse caso, -1, -8, 9 e 2. Esse processo é realizado até terminar todos os elementos da matriz de entrada.

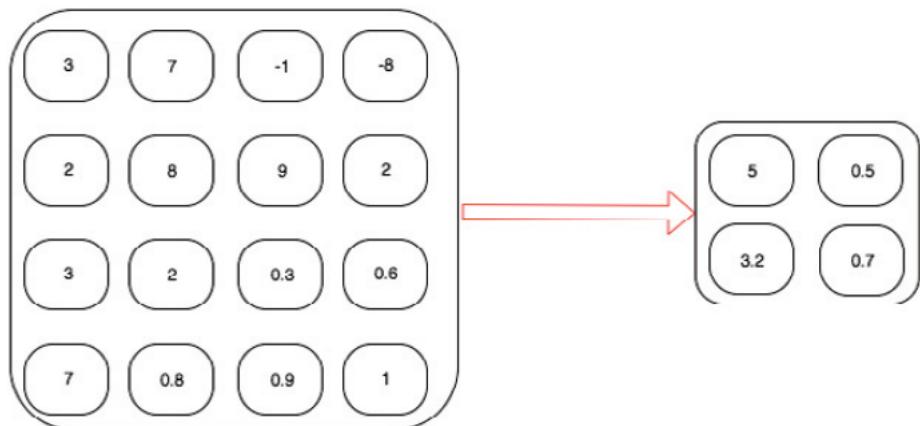
O processo exemplificado na [Figura 6](#) é semelhante ao exemplo da [Figura 5](#), a diferença

Figura 5 – Processo do MaxPooling



Fonte: Subramanian (2018)

Figura 6 – Processo do AvgPooling



Fonte: Subramanian (2018)

é que na técnica de *AvgPooling*, ele retorna a média dos valores analisados.

A função de ativação desempenha um dos elementos mais importantes para os modelos de aprendizado profundo, pois ela determina se um nó de uma camada deve passar informações para a próxima camada. A escolha de uma função de ativação influencia na eficiência do modelo (S.; A; K, 2021). Dentre algumas funções de ativação estão:

Sigmoide: É uma função de ativação não-linear, ou seja, permite que os resultados não precisem ser linear, assim abrangendo mais problemáticas que podem ser resolvidas com modelos de aprendizado profundo. No entanto, ela tem um escopo limitado de 0 a 1 (LI et al., 2019a), o que a torna inadequada para problemas que requerem saídas negativas. Além

Tabela 1 – Funções de Ativação

Função	Fórmula
Sigmoide	$ f(x) = \frac{1}{1+e^{-x}} $
Tanh	$ f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} $
ReLU	$\begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$
Degrau unitário	$\begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$
Sinal	$\begin{cases} +1, x \geq 0 \\ -1, x < 0 \end{cases}$

Fonte: Autor

disso, quando os valores se aproximam muito de 0, a função sigmoide pode apresentar problemas que afetam o aprendizado dos neurônios.

Tanh : É semelhante à função sigmoide e é frequentemente usada em problemas de classificação binária. Uma diferença entre elas, é que a função tanh tem um escopo de -1 a 1 (LI et al., 2019a), permitindo saídas negativas, tornando-a uma escolha para problemas que precisam de saídas negativas.

ReLU : É uma função não-linear com baixo custo computacional. No entanto, ela é limitada a valores positivos, tornando-a inadequada para problemas que exigem saídas negativas.

Degrau unitário : Igualmente à função ReLU possui um baixo custo computacional, pois só precisa verificar se o valor de entrada está acima ou abaixo do limiar, sendo 1 caso o valor seja maior ou igual a 0, e sendo 0 caso o valor seja menor que 0. Essa função é pode ser utilizada em problemas de classificação e probabilidade.

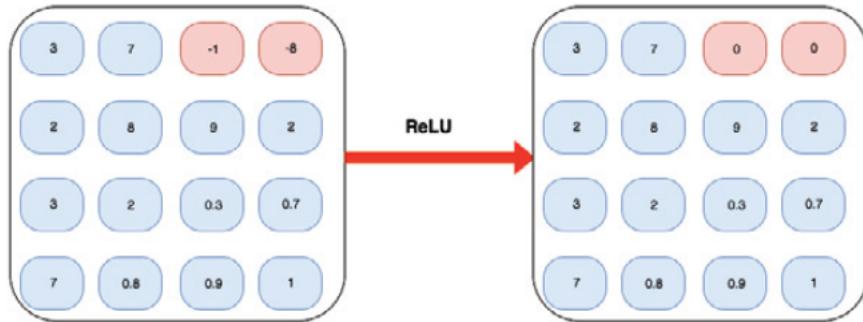
Sinal : É uma função semelhante à degrau unitário, pois ela também possui um limiar para determinar a saída da função, mas permite valores negativos. Portanto, é mais adequada para problemas que precise de representação negativa do que a função degrau unitário.

Softmax : Função geralmente utilizada para problemas de classificação com várias classes, pois tem como saída a probabilidade do elemento pertencer a cada uma dessas classes.

Segundo Subramanian (2018) é uma boa prática e bastante comum utilizar uma camada não-linear após a camada de *pooling* ou convolução, sendo o ReLU um dos mais utilizados, devido ser considerada computacionalmente barata.

A função ReLU é aplicada a cada elemento da imagem, e se o valor for menor que zero, será substituído por zero no mapa de características resultante, caso contrário, será mantido sem modificação. Isso é demonstrado na [Figura 7](#).

Figura 7 – Processo da função ReLU



Fonte: [Subramanian \(2018\)](#)

A capacidade das CNNs de extrair informações de imagens tornou essa rede muito popular na área de Computação Visual ([SUBRAMANIAN, 2018](#)).

2.1.2 Métricas de avaliação

As métricas de avaliação são utilizadas para verificar a eficácia do modelo em relação a sua capacidade de fazer previsões precisas. Elas são importantes para avaliar o desempenho do modelo e qualidade de suas previsões.

Existem diversas métricas de avaliação, entre elas, as da [subseção 2.1.2.1](#), da [subseção 2.1.2.2](#) e da [subseção 2.1.2.3](#).

2.1.2.1 Matriz de Confusão

Matriz de confusão é uma métrica utilizada para avaliar o desempenho de um modelo de classificação, que permite verificar o número de acertos e erros de classificação do modelo para cada classe. Ela é composta por quatro atributos:

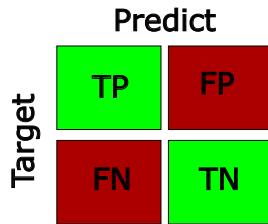
Verdadeiro positivo (TP): O modelo prediz que o resultado é verdadeiro, e realmente é verdadeiro;

Falso positivo (FP): O modelo faz uma previsão que o resultado é verdadeiro, porém é falso;

Falso negativo (FN): O modelo prediz que o resultado é falso, mas é verdadeiro;

Verdadeiro negativo (TN): O modelo faz uma previsão que o resultado é falso, e a previsão é confirmada como falsa.

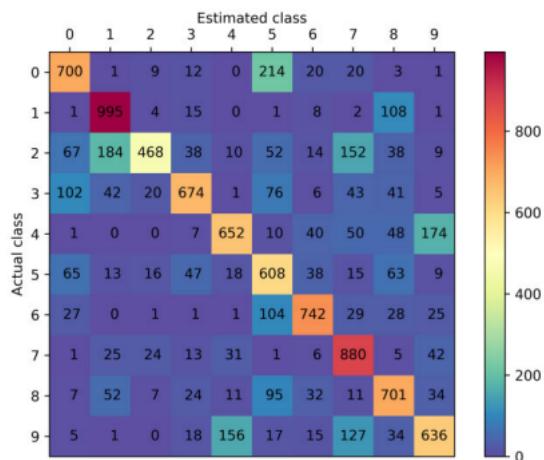
Figura 8 – Matriz de confusão



Fonte: Produzida pelo autor

A Figura 8 exibe o formato da matriz de confusão para uma classificação binária. Para classificações multi classe, a matriz seria expandida com linhas e colunas adicionais para cada classe, resultando em uma matriz com dimensões NxN, onde N é o número de classes. Na matriz de confusão, a diagonal principal corresponde aos acertos do modelo para cada classe, enquanto as outras células indicam os erros de classificação. A Figura 9 ilustra uma matriz de confusão multi classe (LUQUE et al., 2022).

Figura 9 – Matriz de Confusão multi classe



Fonte: Luque et al. (2022)

A partir desses valores, é possível calcular outras métricas, como a acurácia, precisão, *recall*, e *F1-score*, os quais permitem avaliar diferentes aspectos da eficácia do modelo (CHEN; LIN, 2019). Essas métricas são descritas abaixo:

Acurácia: Métrica que demonstra a quantidade de acertos total do modelo.

$$\frac{TP + TN}{TP + FP + FN + TN} \quad (2.1)$$

Precisão: Verifica quais dos elementos classificados como positivos são realmente positivos, ou seja, proporção de verdadeiros positivos em relação ao total de detecções positivas;

$$\frac{TP}{TP + FP} \quad (2.2)$$

Recall: Verifica quais dos objetos positivos realmente foram classificados como positivos, ou seja, proporção de verdadeiros positivos em relação ao total de objetos reais.

$$\frac{TP}{TP + TN} \quad (2.3)$$

F1-score: É a sumarização da precisão e a *recall*, fornece o resultado das 2 métricas em um único valor. O resultado é um valor entre 0 e 1. Quanto mais próximo de 1, melhor é o desempenho.

$$\frac{2 \times precisao \times recall}{precisao + recall} \quad (2.4)$$

2.1.2.2 Intersection over Union(IoU)

Intersection over Union (IoU) é uma métrica utilizada para avaliar o desempenho de um modelo de detecção de objetos ou segmentação de imagem (GOYZUETA; CRUZ; MACHACA, 2021). Para calcular a IoU necessita das *bounding boxes* da imagem, e as *bounding boxes* resultantes do modelo. O cálculo é realizado pela divisão da área de interseção das *bounding boxes* pela área de união das mesmas, resultando em um valor entre 0 e 1. Quanto mais próximo de 1, melhor é o desempenho do modelo.

Figura 10 – *Bounding box* original



Fonte: [sshikamaru \(2023\)](#)

Figura 11 – *Bounding box* inferida



Fonte: [sshikamaru \(2023\)](#)

As *bounding boxes* são caixas que delimitam o objeto na imagem. A Figura 10 e a Figura 11 ilustram uma *bounding box* delimitando um carro. Nesse exemplo, seria calculado a divisão da interseção pela união da área das caixas que delimitam os carros da imagem original, e da imagem inferida pelo modelo. A fórmula do IoU é dada por:

$$\frac{Intersection(bbox_1, bbox_2)}{Union(bbox_1, bbox_2)} \quad (2.5)$$

2.1.2.3 Mean Average Precision(MAP)

A mAP é uma métrica utilizada na avaliação do desempenho de modelos de detecção de objetos. Para realização do cálculo da mAP, é preciso obter os resultados de outras métricas, citadas anteriormente, nas [subseção 2.1.2.1](#) e [subseção 2.1.2.2](#).

Para cada classe de objeto, são calculadas as métricas de precisão, *recall* e a IoU. A primeira métrica calculada é a IoU, se a IoU for maior do que um valor pré-definido, então, considera-se que essa *bounding box* é um TP, caso contrário, é considerada FP. Em seguida, são calculados a precisão e *recall*. Com base na curva de precisão por *recall*, é calculada a área sob a curva de precisão e *recall*, conhecida como *average precision* (AP) ([WANG et al., 2021](#)). Após isso, calcula-se a mAP, que é a média das APs.

$$\frac{\sum_{i=1}^n AP_i}{N} \quad (2.6)$$

A mAP pode ser calculada para um valor específico do IoU pré-definido, ou então, para um conjunto de IoUs. Por exemplo, map@0.5:0.05:0.95 significa que será calculado a mAP com o IoUs variando de 0.5 até 0.95, em intervalos de 0.05. Nesse caso, o resultado da mAP seria a média dos valores das mAPs calculadas para cada IoU. Sendo assim, o valor da IoU utilizado para calcular a mAP interfere significativamente no resultado, já que dependendo desse IoU, uma *bounding box* pode ser considerada válida ou não.

2.1.3 Técnicas de melhorias de desempenho

Existem diversas técnicas que podem ser empregadas para aprimorar o desempenho de modelos de redes neurais, tais como melhorar a velocidade de inferência, aumentar a acurácia do modelo, evitar *overfitting*, ou seja, evitar que o modelo decore os dados de treino, pois quando o modelo decora esses dados, ele não consegue generalizar, sendo assim, não consegue ter um bom desempenho com os dados de teste.

2.1.3.1 Data Augmentation

A técnica de *Data augmentation* é bastante útil em visão computacional para aumentar a quantidade de dados para o treinamento de um modelo, sem a necessidade de adicionar novos arquivos à base de dados. Esse processo melhora o desempenho do modelo de rede neural, e também ajuda a evitar *overfitting* ([ABUSALIM et al., 2022](#)). A técnica é realizada através de transformações geométricas nas imagens da base de dados, tais como rotação, espelhamento e outras técnicas ([AN et al., 2022](#)). A [Figura 12](#) ilustra um exemplo de *data augmentation*, onde foi utilizada uma mescla de técnicas de autocontraste, equalização e solarização.

Figura 12 – Técnica de *data augmentation*



Fonte: An et al. (2022)

2.1.3.2 Normalização

A técnica de normalização é utilizada para padronizar os dados em uma mesma escala, uma vez que muitos modelos de *machine learning* não funcionam com dados em escalas diferentes (IZONIN et al., 2022).

Uma das técnicas de normalização mais comuns é o *Standard Scaler*, que assume que os dados possuem uma distribuição normal, sendo assim, sua média é igual a 0 e o desvio padrão é igual a 1. Ao aplicar essa técnica, a rede neural não precisa trabalhar com valores altos, já que os dados estarão entre 0 e 1, e todos os dados estarão na mesma escala, fazendo com que a rede tenha uma melhora em seu desempenho (SUBRAMANIAN, 2018). A fórmula é dada por:

$$X' = \frac{X_i - \mu}{\sigma} \quad (2.7)$$

μ é a média dos dados, σ o desvio padrão desses dados, e X_i o i-ésimo valor dos dados.

2.1.3.3 Non Maximum Suppression (NMS)

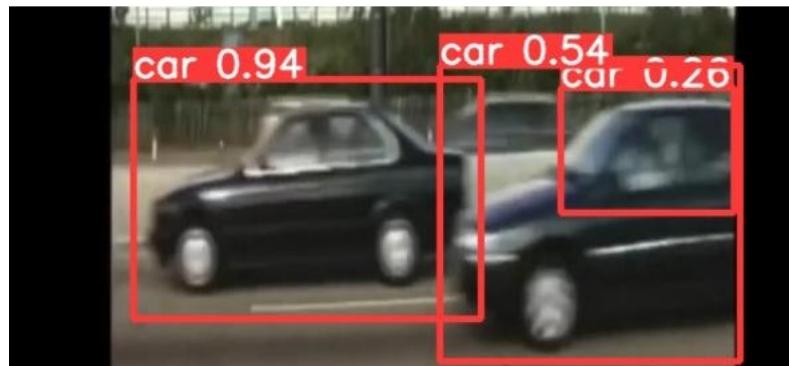
A técnica de *Non Maximum Supression* é uma técnica utilizada para evitar que um mesmo objeto seja detectado e classificado várias vezes em uma imagem. A técnica verifica as *bounding boxes* geradas pelo modelo e seleciona apenas as que possuem maior probabilidade de conter um objeto. Se duas ou mais *bounding boxes* tiverem um IoU maior do que um valor pré-definido, a técnica manterá apenas a *bounding box* com o maior valor de confiança (WANG et al., 2021).

Na Figura 13, é possível observar um exemplo em que a utilização da técnica de NMS seria útil, já que um carro está sendo detectado duas vezes, gerando uma redundância na detecção. A técnica de NMS, manteria apenas a *bounding box* com maior valor de confiança, garantindo uma detecção mais confiável. O valor de confiança é considerado o valor de certeza que o modelo tem da classificação daquele objeto.

2.1.3.4 Fine-tuning

A técnica de *fine-tuning* é utilizada em aprendizado de máquina para otimizar o tempo e os recursos necessários para treinar uma nova rede neural. Essa técnica consiste em utilizar um modelo de aprendizado profundo já treinado para treinar outro modelo. Em vez de treinar a rede neural do zero, a técnica transfere os pesos e as informações aprendidas da rede já treinada para

Figura 13 – Técnica de NMS



Fonte: <<https://www.kaggle.com/datasets/shikamaru/car-object-detection>>

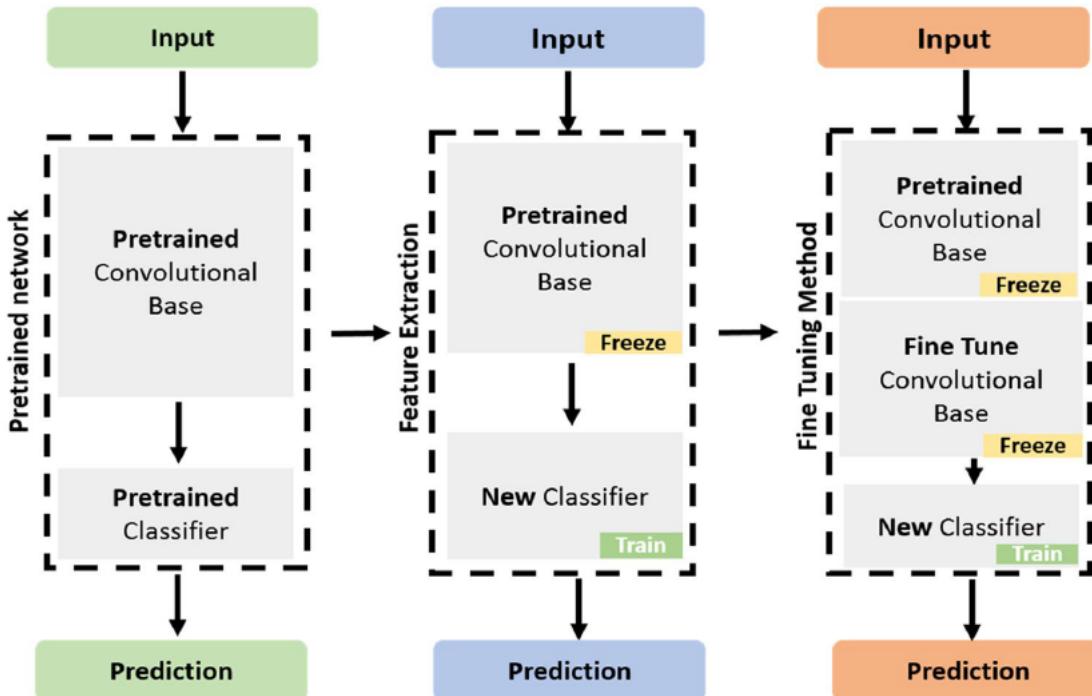
o novo modelo (ZAAOURI; EZZEDINE, 2018). Assim, ela só precisará treinar as camadas que forem diferentes do modelo treinado, reduzindo significativamente o tempo e esforço necessários.

Por exemplo, é possível utilizar um modelo pré-treinado para detectar diversos tipos de animais em um modelo que será utilizado para detectar apenas cachorros. Nesse caso, as camadas de classificação e regressão do modelo precisariam ser alteradas para detectar apenas um tipo de animal. Além disso, pode-se "congelar" os pesos das camadas iniciais do modelo, já que são responsáveis apenas por extrair as características dos objetos. Isso diminui o tempo de treinamento do modelo, já que serão menos camadas para treinar.

A técnica de *Fine-tuning* é bastante utilizada na área de Computação Visual, tanto em problemáticas de detecção de objetos, classificação de imagens ou segmentação de imagens, pois a utilização de *fine-tuning* com modelos pré-treinados alcança um desempenho melhor que a utilização de modelos sem treinamento(XU et al., 2021).

A Figura 14 ilustra o processo de *fine-tuning* em um modelo genérico. O primeiro passo é encontrar um modelo pré-treinado que seja compatível com a base de dados que será utilizado. Esse modelo genérico já foi treinado em uma base de dados semelhante à que será utilizada, já que para a utilização de um modelo pré-treinado ser eficiente, é necessário que a base de dados possua similaridade com o objetivo do modelo, por exemplo, não é possível utilizar um modelo pré-treinado para detectar animais em um problema de detecção de veículos, mas é possível utilizá-lo para detectar cachorros, já que eles também são animais.

Após encontrar um modelo pré-treinado, as camadas responsáveis por extrair as características são "congeladas", pois já foram treinadas para extrair as informações relevantes para o objetivo do modelo e, portanto, são utilizados como extratoras de atributos (ASIF et al., 2022). No entanto, as camadas responsáveis pela classificação são substituídas por novas camadas, já que o modelo precisará classificar elementos específicos e não mais uma grande variedade de elementos.

Figura 14 – Técnica de *fine-tuning*

Fonte: [Asif et al. \(2022\)](#)

No modelo da direita da Figura 14, é adicionada uma nova camada ou conjunto de camadas. Essas camadas podem ser ajustes realizados nesse modelo para extrair características de outro tipo de elemento, como utilizar camadas de um modelo pré-treinado que detecta cachorros e outras camadas de um modelo pré-treinado que detecta gatos. Ou então, pode-se utilizar camadas treinadas para extrair características do mesmo elemento, mas com uma profundidade diferente. Por exemplo, para um modelo de detecção de gatos, uma camada pode extrair características como a cor dos olhos e outra camada pode extrair características como a silhueta do animal.

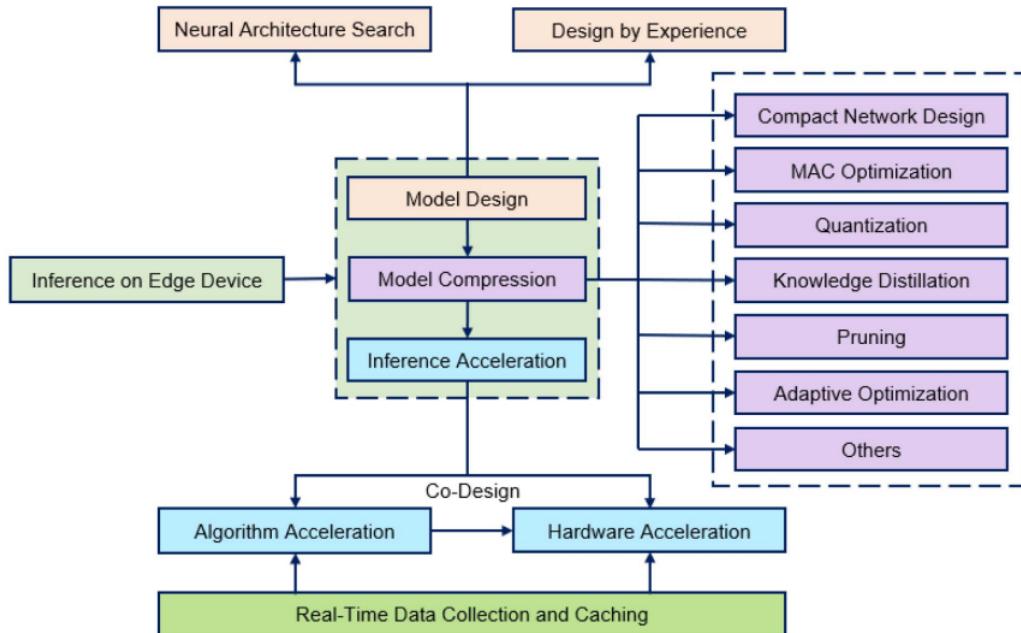
Essa técnica de *fine-tuning* também pode ser utilizada nos modelos que foram comprimidos, como os que passaram por técnicas de poda, com o objetivo de recuperar a acurácia perdida após a compressão.

2.1.3.5 Compressão de Modelo

A compressão de modelo é o nome dado para diversas técnicas que são utilizadas para reduzir o tamanho do modelo em termos de parâmetros e custo computacional (SHUVO et al., 2023). Essas técnicas são frequentemente empregadas em modelos projetados para serem executadas em arquitetura com restrição no poder computacional, como dispositivos de borda. A Figura 15 ilustra as etapas geralmente utilizadas na construção desses modelos.

A primeira etapa é a da concepção do modelo, que é criado com base em modelos de

Figura 15 – Etapas de construção de modelo para dispositivos de borda



Fonte: [Shuvo et al. \(2023\)](#)

estado da arte e treinado com uma base de dados para o propósito do usuário. Quando o modelo está funcionando e tem bom desempenho, é realizada a compressão do modelo para reduzir seu tamanho, tornando-o apto a ser executado em dispositivos com limitações computacionais. Entre as técnicas de compressão de modelos estão *quantization*, *pruning* e *knowledge distillation*. A próxima etapa é a aceleração do modelo, que pode ser realizada por meio de hardware, como GPUs robustas, ou por, código, o que é mais comum para dispositivos de borda.

Existem diversas técnicas de compressão de modelos, entre elas:

Pruning ou poda: É utilizada para remover os parâmetros redundantes de uma rede neural, incluindo pesos, *kernels*, neurônios ou camadas desnecessárias ([SHUVO et al., 2023](#)). Essa técnica pode ser estruturada ou não-estruturada.

Na poda não-estruturada, os pesos são removidos, mas os neurônios são mantidos, desde que possuam pelo menos uma conexão. Embora essa técnica resulte em uma redução no tamanho do modelo, ela também pode levar a computação irregular, o que pode limitar o paralelismo dos hardwares utilizados ([WU et al., 2021a](#)). Já na poda estruturada, os pesos são removidos juntamente com camadas inteiras, resultando em uma redução mais significativa no tamanho do modelo e aceleração do mesmo, sem ser limitado pelas condições de hardware.

Quantization: É uma técnica utilizada para reduzir a quantidade de bits necessários para

armazenar os pesos ou valores de ativação de um modelo (WANG; MA; YANG, 2022). Embora a maioria dos modelos utilize 32 bits como padrão, muitas vezes essa quantidade de bits é desnecessária para armazenar os valores dos pesos, assim podendo ser reduzida para economizar espaço e melhorar a eficiência computacional.

Existem diferentes modos de quantização, como o binário, que altera o valor de um peso para 1 se ele for positivo ou para 0 se ele for negativo. Outro modo é a quantização dinâmica, em que cada camada pode ter uma quantidade de bits de armazenamento diferente, dependendo da sua necessidade.

Os benefícios de utilizar a quantização incluem a redução do tamanho do modelo, a simplificação dos cálculos realizados nas camadas e a melhoria na eficiência energética (SHUVO et al., 2023).

Knowledge Distillation: É uma técnica utilizada para treinar um modelo menor e mais rápido (conhecido como modelo estudante), utilizando o conhecimento obtido por um modelo maior e mais complexo (conhecido como modelo professor) (WANG et al., 2018a).

Após o modelo professor ser treinado e ter um bom desempenho na generalização dos dados, o conhecimento adquirido é transferido para o modelo estudante. Isso pode ser feito de várias formas, mas uma abordagem comum é treinar a camada anterior à última camada *softmax* do modelo estudante para imitar a camada correspondente do modelo professor. Outro modo seria utilizar camadas intermediárias do modelo professor no modelo estudante.

Uma variação do *knowledge distillation* é o *multi-knowledge distillation*, onde o modelo estudante é treinado para imitar a saída de vários professores. Um dos modos seria a imitação da média do valor de *softmax* dos modelos professores. No entanto, poderia não ser tão eficiente, já que alguns modelos professores podem ter desempenho superior a outros (WANG; LU, 2021), nesses casos seria necessário criar um peso de importância para cada modelo professor.

Low-rank factorization: É utilizada para reduzir a complexidade computacional e espaço do modelo. Essa técnica é aplicada por meio da decomposição de matrizes, que remove parâmetros redundantes e mantém apenas aqueles que são mais relevantes para o modelo (SAHRAEIAN; COMPERNOLLE, 2017).

low-rank factorization pode ser utilizada tanto em camadas densas quanto em camadas convolucionais. Quando aplicadas em camadas convolucionais, reduz o tempo de inferência dos dados, enquanto nas camadas densas, pode diminuir o gasto de armazenamento.

2.1.4 Aplicações na Visão Computacional

As CNNs são amplamente empregadas na área de visão computacional devido à sua capacidade de preservar as informações espaciais das imagens, o que as torna uma escolha popular

em diversas aplicações ([SUBRAMANIAN, 2018](#)). Por exemplo, na classificação de imagens, [G et al. \(2022\)](#) utilizou uma rede neural convolucional para classificar números manuscritos com alta precisão. Em relação à segmentação de imagens, [Yin et al. \(2020\)](#) aplicou uma técnica de CNN para segmentar discos ópticos e detectar tumores ou anomalias. Além disso, na detecção de objetos, [Zaatouri e Ezzedine \(2018\)](#) desenvolveu um sistema alternativo para o monitoramento de tráfego baseado em CNNs. Esses exemplos destacam a eficácia das CNNs em diversos problemas de visão computacional.

2.1.4.1 Classificação de Imagens

Na tarefa de classificação de imagens, o objetivo é fazer com que o modelo analise a imagem e determine a qual classe ela pertence. Uma base de dados muito utilizada na área é a *Modified National Institute of Standard and Technology* (MNIST), que consiste em 60 mil imagens de números manuscritos de 0 a 9 ([G et al., 2022](#)). A [Figura 16](#) ilustra um conjunto de exemplos dessa base de dados.

Figura 16 – Estrutura de uma CNN

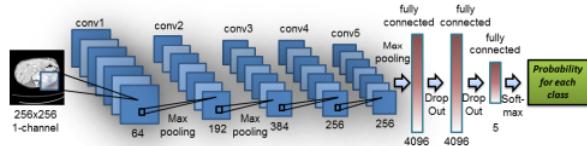


Fonte: [G et al. \(2022\)](#)

Para classificação de imagens, é comum a utilização de uma camada densa no final do modelo para realizar a classificação ([SUBRAMANIAN, 2018](#)). A [Figura 17](#) exibe um exemplo de uma CNN utilizada para classificar anomalias em cinco locais do corpo humano: pernas, pulmões, fígados, pescoço e região pélvica. Esse modelo foi desenvolvido com o objetivo de auxiliar na detecção precoce de anomalias em exames médicos ([ROTH et al., 2015](#)).

A camada densa é responsável por realizar a classificação final após a extração das características da imagem pelas camadas convolucionais e de *pooling* da rede neural. Entretanto, como os valores de saída das camadas de *pooling* e convolucionais são matrizes, e o valor de entrada de uma camada densa precisa ser um vetor, é comum a utilização de uma camada que faça essa transformação. Nesse caso, a camada *flatten* foi utilizada, ela transforma a matriz de saída da camada anterior em um vetor unidimensional, como é demonstrado na [Figura 17](#).

Figura 17 – Base de dados MNIST



Fonte: Roth et al. (2015)

2.1.4.2 Segmentação de Imagens

A tarefa de segmentação de imagens tem como objetivo identificar regiões ou objetos em uma imagem e separá-los em segmentos distintos com base em suas características (GAUTAM; MATHURIA; MEENA, 2022). Esses segmentos podem ser definidos como regiões de pixels com propriedades semelhantes, como cor, textura, entre outras. Dessa forma, a segmentação permite uma análise mais detalhada das imagens.

O trabalho de Gautam, Mathuria e Meena (2022) utiliza a segmentação de imagens para identificar objetos em uma autoestrada. Para isso, os segmentos semelhantes são alterados para ter a mesma cor e textura, tornando mais fácil para o carro identificar objetos em seu campo de visão. Por exemplo, segmentando todos os seres humanos em uma determinada cor, assim fazendo com que o carro evite as áreas com essa cor. A Figura 18 e a Figura 19 demonstra um exemplo de segmentação para a ideia de carros automáticos.

Figura 18 – Autoestrada sem segmentação



Fonte: Gautam, Mathuria e Meena (2022)

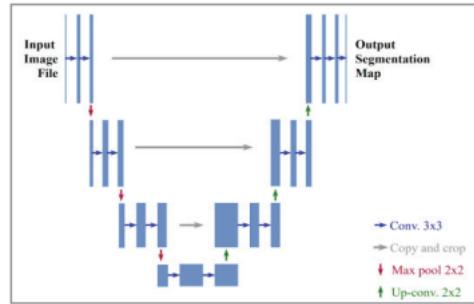
Figura 19 – Autoestrada segmentada



Fonte: Gautam, Mathuria e Meena (2022)

O modelo de rede neural U-net é um dos modelos mais populares para segmentação de imagens e foi desenvolvido por Olaf Ronnerberg (GAUTAM; MATHURIA; MEENA, 2022). Yin et al. (2020) também utilizou o U-net como base para estrutura de sua rede neural. O modelo U-net utiliza uma estrutura de *encoder-decoder*. A Figura 20 ilustra a estrutura do U-net.

Figura 20 – Modelo U-net



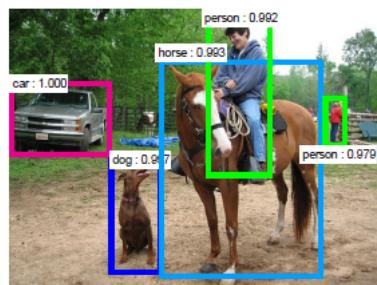
Fonte: [Gautam, Mathuria e Meena \(2022\)](#)

2.1.4.3 Detecção de objetos

A tarefa de detecção de objetos consiste em encontrar e classificar os objetos presentes em imagens ou vídeos, esses objetos são delimitados por uma *bounding box*.

Os modelos de detecção de objetos atuais são classificados em duas abordagens: abordagem em um estágio e abordagem em dois estágios ([XU et al., 2023](#)). Os modelos de um estágio geralmente são mais eficientes em termos de velocidade de processamento, mas possuem menor acurácia em comparação aos modelos de dois estágios, que costumam ser mais precisos, mas também mais lentos. O *You Only Look Once* (YOLO) e o *Single Shot Detector* (SSD) são exemplos de modelos de abordagem em um estágio, já a família R-CNN é um exemplo de abordagem em dois estágios. A [Figura 21](#) ilustra um exemplo de detecção de objetos.

Figura 21 – Exemplo de detecção de objetos



Fonte: [Fan, Brown e Smith \(2016\)](#)

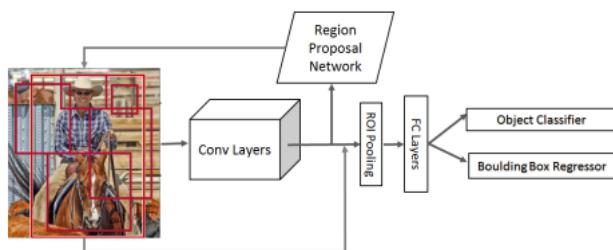
2.1.4.3.1 Faster R-CNN

O modelo Faster R-CNN é uma evolução do Fast R-CNN, que por sua vez já era uma melhoria em relação ao R-CNN. A diferença do Faster-R-CNN para o Fast R-CNN, está na adição da camada de *Region Proposal Network*, responsável pela seleção das *bounding boxes* do modelo.

O Faster R-CNN alcança a performance do estado da arte em relação à detecção de objetos, sendo um dos 3 modelos mais utilizados nessa área, juntamente com YOLO e SSD (WANG; BOCHKOVSKIY; LIAO, 2022). Geralmente alcança uma precisão maior que os demais modelos, porém é mais lento, devido ao seu processo de detecção.

O processo de detecção do Faster R-CNN é dividido em 4 etapas, demonstradas na Figura 22, e descritas posteriormente:

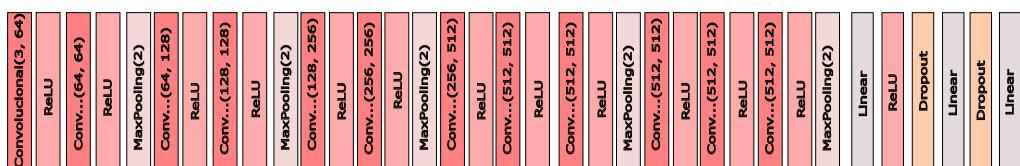
Figura 22 – Camadas do Faster R-CNN



Fonte: Fan, Brown e Smith (2016)

Conv Layers: É composta por um conjunto de camadas de aprendizado profundo, que são responsáveis por extrair características da imagem. No método Faster R-CNN, é utilizado um modelo de redes neurais convolucionais moderno para essa etapa. ([REN et al., 2015](#)) utilizou o modelo VGG16. Esse modelo possui a seguinte estrutura:

Figura 23 – Estrutura do modelo VGG16



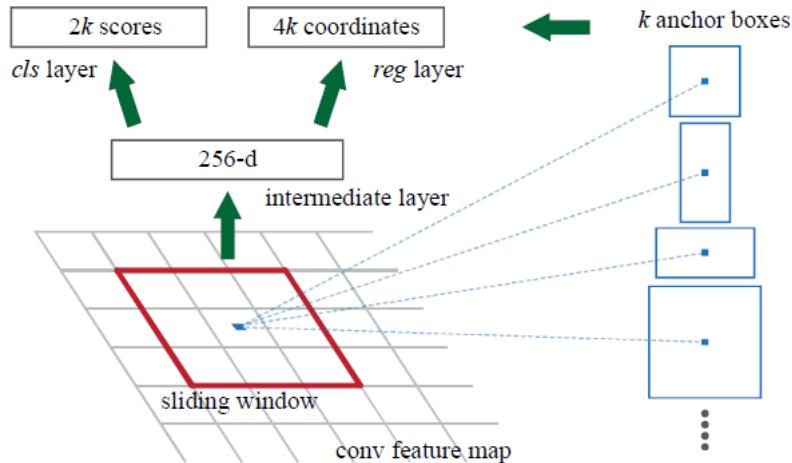
Fonte: Produzida pelo autor

O resultado extraído dessa estrutura é conhecido como mapa de características. Esses mapas de características são enviados para a etapa de RPN. A RPN é responsável por selecionar possíveis locais na imagem que possam conter um objeto, demarcando esses locais com *bounding boxes*.

Region Proposal Network (RPN): É responsável por gerar propostas de regiões que provavelmente contêm objetos na imagem. Ela recebe os mapas de características da camada de convolução e gera, para cada posição na imagem, múltiplas regiões candidatas à possuir um objeto. Cada região é representada por uma *anchor*, que é a região que delimita o objeto, e

é representada pelo ponto central da região. As *anchors* possuem 5 valores, a coordenada x, coordenada y, altura da região, largura da região e a classificação do objeto(FAN; BROWN; SMITH, 2016). A Figura 24 ilustra um exemplo de *anchor* em um mapa de características.

Figura 24 – Region Proposal Network

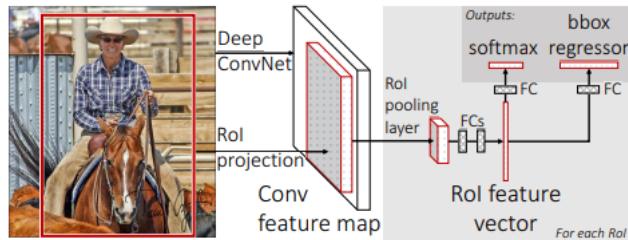


Fonte: Fan, Brown e Smith (2016)

Durante o treinamento da RPN, as propostas geradas são comparadas com as anotações dos objetos e são atribuídos rótulos para cada *anchor*. Esses rótulos servem para calcular o erro da RPN e ajustar os pesos do modelo. Para avaliar as propostas das possíveis regiões que contém objeto, é utilizada a métrica *Intersection over Union* (IoU). As propostas que possuem IoU maior que o limite pré-definido ((FAN; BROWN; SMITH, 2016) utilizou 0.7), são consideradas que possuem um objeto, e as demais são descartadas. Após o processo de verificação, as propostas de regiões que não foram deletadas são enviadas para a camada de *RoI Pooling*.

RoI Pooling: É responsável por extrair regiões dos mapas de características das camadas anteriores e convertê-las em tamanhos fixos e menores. A *RoI Pooling* utiliza de *pooling* para realizar a redução de dimensões e, em seguida, transforma os dados em um vetor unidimensional para serem processados pela camada *FC Layers*. A *RoI Pooling* geram duas saídas, uma para a classificação dos objetos e outra para a regressão das *bounding boxes* (GIRSHICK, 2015). A Figura 25 ilustra o funcionamento da camada de *RoI Pooling*.

Fully Connected(FC) Layers: É responsável por receber os mapas de características das camadas anteriores, e realizar a classificação dos objetos detectados, além da regressão das *bounding boxes*. O resultado final é a combinação dos resultados das duas saídas da *FC Layers*.

Figura 25 – *RoI Pooling*

Fonte: [Girshick \(2015\)](#)

2.1.4.3.2 *You Only Look Once* (YOLO)

O YOLO é um modelo de aprendizado profundo de abordagem de 1 estágio, que pode ser utilizado para classificação, segmentação de imagens e detecção de objetos com alta precisão ([REDMON et al., 2016](#)). Este modelo é capaz de generalizar objetos com eficiência, por isso é bastante utilizado em tarefas de visão computacional.

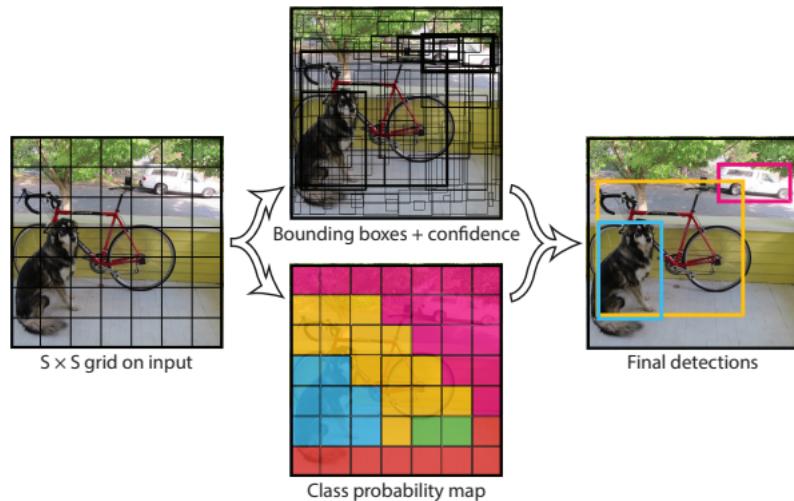
O YOLO divide a imagem em N grades e cada grade é associada a M *bounding boxes*, em que é calculado um *score* que representa a confiança de que um objeto está presente naquela grade. Para que um objeto seja considerado presente em uma grade, o centro do objeto deve estar dentro dessa grade. O YOLO utiliza uma técnica chamada de *free anchor*, que prevê o objeto a partir de seu centro, sem precisar da região que delimita o objeto. Cada anchor é composto por cinco valores: coordenadas x e y, largura e altura do objeto, e um *score*, semelhante ao Faster R-CNN descrito na [subseção 2.1.4.3.1](#). A [Figura 26](#) ilustra as etapas realizadas pelo YOLO em uma imagem. Com isso, é possível ter uma visão do processo de detecção de objetos realizado pelo modelo.

A estrutura do YOLO é dividida em três etapas: *backbone*, *neck* e *head*, que são descritas abaixo:

Backbone: É responsável por extraír as características dos dados de entrada. Essa camada é formada por uma CNN, alguns desses modelos são: DenseNet e ResNet. No caso do YOLOv7, essa camada é formada pelo E-ELAN, que é uma variação do *EfficientNet-LAN* (ELAN) ([WANG; BOCHKOVSKIY; LIAO, 2022](#)).

Neck: É responsável por receber os mapas de características da camada de *backbone* e gerar predições dos possíveis locais de objetos na imagem, delimitando esses locais com *bounding boxes*, ou seja, o objetivo é reduzir o número de regiões propostas que precisam ser processadas na etapa seguinte, permitindo que o modelo foque nas regiões que provavelmente possuem objetos. Essa camada pode ser comparada à camada *Region Proposal Network* do modelo Faster R-CNN.

Figura 26 – Etapas de uma imagem no YOLO



Fonte: Redmon et al. (2016)

Essa camada ajuda a reduzir o processamento desnecessário de regiões irrelevantes, assim, aumenta a eficiência do modelo. Para isso, a camada *Neck* utiliza técnicas de agrupamento de pixels e combinação de informações das camadas anteriores para identificar as regiões com probabilidade de conter um objeto.

Head: É responsável por realizar a classificação dos objetos e a regressão das *bounding boxes* que delimitam esses objetos. Essa camada é comparável à camada *Fc Layers* do modelo Faster R-CNN.

Ao receber as propostas de regiões geradas pela camada *Neck*, a camada *Head* utiliza as informações de características dessas regiões para determinar se há um objeto, caso exista, ela irá classificar o objeto em uma das classes pré-definidas na base de dados. Além disso, essa camada ajusta as coordenadas das *bounding boxes* para se ajustar melhor aos objetos. O resultado final dessa camada é uma lista com as classes, *bounding boxes* e *scores* dos objetos detectados.

A versão mais recente do YOLO é o YOLOv8, desenvolvido pela Ultralytics. Sua documentação² possui informações detalhadas sobre seu funcionamento e configuração.

Durante o treinamento, o YOLOv8 utiliza técnicas de *data augmentation*, incluindo a técnica de *mosaic augmentation*. Nessa técnica, o modelo concatena quatro imagens diferentes para treinar a rede, ou seja, gera imagens com diferentes perspectivas e posições de objetos. Isso ajuda a aumentar os dados de treinamento e torna mais difícil para o modelo memorizar as posições exatas dos objetos, o que reduz o risco de *overfitting* e, ajuda a generalização do modelo.

² Disponível em: <<https://docs.ultralytics.com>>. Acesso em março de 2023.

O YOLOv8 possui cinco variações, que são: nano, small, medium, large e extra-large. Cada variação possui tamanho e número de parâmetros diferentes, o que afeta o desempenho e a velocidade de inferência do modelo.

O [Quadro 1](#) apresenta informações sobre cada variação pré-treinada do YOLOv8 para detecção de objetos, com a base de dados *Common Object in Context*³ (COCO), entre elas, o número de parâmetros, a taxa de detecção e a velocidade de processamento.

Quadro 1 – Dados dos modelos do Yolov8 pré-treinado

Model	Size (pixels)	mAP	Speed CPU(ms)	params (M)	flops(B)
YOLOv8n	640	37.3	80.4	3.2	8.7
YOLOv8s	640	44.9	128.4	11.2	28.6
YOLOv8m	640	50.2	234.7	25.9	78.9
YOLOv8l	640	52.9	375.2	43.7	165.2
YOLOv8x	640	53.9	479.1	68.2	257.8

Fonte: [Jocher \(2023\)](#)

A escolha de qual variação do YOLO utilizar, vai depender do propósito para qual será empregada e do equipamento no qual será executada. Como o [Quadro 1](#) demonstra, a quantidade de parâmetros está relacionado ao desempenho do modelo, ou seja, quanto mais parâmetros, maior a taxa de detecção (mAP), mas menor a velocidade de inferência. Por exemplo, o modelo YOLOv8x, não seria adequado para sistemas embarcados devido à sua quantidade de parâmetros, que exigiria mais recursos computacionais e aumentaria o tempo de execução em comparação com outras variações.

O YOLOv8n seria o mais adequado para dispositivos com recursos limitados, que seria o caso de um sistema embarcado. Para problemas que necessitem de maior taxa de detecção, e não tem limitação com os recursos, seria recomendado a utilização do YOLOv8x, mesmo necessitando de um tempo de processamento maior.

2.2 Sistema de Estimativa de Velocidade Veicular

A estimativa de velocidade, como o próprio nome sugere, é uma técnica utilizada para medir a velocidade de um veículo, contribuindo para o gerenciamento do tráfego em rodovias. Através da utilização de dispositivos de medição de velocidade, os proprietários de veículos podem ser notificados e responsabilizados caso excedam o limite de velocidade, o que contribui para reduzir o número de infratores nas vias. Como resultado, essa abordagem ajuda a diminuir a ocorrência de acidentes de trânsito e também possibilita investigar se a velocidade excessiva esteve envolvida em algum acidente.

³ Disponível em: <<https://cocodataset.org/#home>> Acesso em junho de 2023

Dentre as ferramentas utilizadas para estimativa de velocidade, destacam-se os sistemas de radares, que se baseiam no efeito Doppler, a detecção por meio de lasers (LiDAR), por sensores embaixo do pavimento e, mais recentemente, o uso de *deep learning* para o reconhecimento das velocidades dos veículos (CVIJETIĆ; DJUKANOVIĆ; PERUNIĆIĆ, 2023).

A estimativa de velocidade por meio do sistema de radar é realizada através de um aparelho que emite um sinal em direção ao veículo. Ao receber o sinal refletido pelo veículo, é possível calcular sua velocidade (WERNECK, 2009). Essa medição é feita considerando que a distância entre o observador e a fonte de onda é alterada à medida que o veículo se movimenta, e a diferença na frequência do sinal nesse intervalo de tempo está relacionada à velocidade do automóvel.

Na detecção por sensores de pavimento, geralmente existem dois sensores posicionados sob o solo. Quando o veículo passa por um dos sensores, este é acionado, e ao passar pelo outro, é realizado o cálculo da velocidade com base na distância percorrida e no tempo gasto para passar pelos sensores.

Já no sistema LiDAR, os objetos são projetados em dimensões 3D, permitindo que a distância seja calculada através do uso de luz (PRASETYA et al., 2021). Ou seja, o veículo é projetado em um ponto A, e ao ser projetado novamente no ponto B, a distância percorrida e o tempo decorrido são utilizados para calcular a velocidade do veículo. A Figura 27 faz a comparação entre um ambiente real e um ambiente desenvolvido pelo sistemas LiDAR

Figura 27 – Comparaçāo entre imagem real e imagem desenvolvida por câmeras LiDAR



Fonte: Jeon e Ko (2018)

Existem diversas técnicas para realizar a estimativa de velocidade utilizando *deep learning*, como demonstrado no trabalho de Wu et al. (2021b), que desenvolveu um modelo capaz de mensurar a velocidade do veículo com base na distância do veículo em relação a uma referência na estrada, considerando diferentes quadros. De maneira geral, para estimar a velocidade usando aprendizado profundo, é necessário um modelo de detecção de objetos para identificar o veículo. Em seguida, a abordagem mais comum é calcular a distância percorrida pelo veículo entre o

primeiro quadro em que ele foi detectado e o i -ésimo quadro, por exemplo, utilizando a distância euclidiana, que é representada pela fórmula:

$$d(v_i, v_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \quad (2.8)$$

Sendo v_i a posição do veículo no i -ésimo quadro, v_j a posição do veículo no j -ésimo quadro. A posição do veículo é representada por um ponto (x, y) , geralmente esse ponto é o ponto central da *bounding box* do veículo detectado. Após isso, utiliza a distância entre os pontos, e o tempo gasto para o veículo sair do i -ésimo ponto até o j -ésimo ponto para calcular a velocidade estimada do veículo.

Uma das questões a serem enfrentadas nessa abordagem é o impacto da posição da câmera na estimativa da velocidade, exigindo a aplicação de técnicas para reduzir essa influência. Uma forma de abordar esse problema é calcular a relação geométrica entre o ponto na imagem e o ponto geográfico do veículo (HUANG et al., 2015). Dessa forma, é possível considerar as distorções causadas pela perspectiva da câmera e obter estimativas mais precisas da velocidade do veículo em diferentes posições de captura.

Uma abordagem eficaz para mitigar a influência da câmera é a utilização da matriz de homografia. Essa matriz contém informações intrínsecas e extrínsecas da câmera (CAMPOS, 2009), permitindo estabelecer a relação entre objetos na imagem e em outras imagens ou até mesmo no espaço geográfico. Com a matriz de homografia, é possível mapear informações da imagem para o plano real. A Figura 28 ilustra a fórmula empregada no desenvolvimento da matriz de homografia.

Figura 28 – Matriz de Homografia

$$\begin{aligned} x &= \frac{u}{s} \\ y &= \frac{v}{s} \end{aligned} \quad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -f_x & 0 & a_x \\ 0 & -f_y & a_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\tilde{s}\tilde{\mathbf{m}} = \mathbf{A}[\mathbf{R} \quad \mathbf{t}]\tilde{\mathbf{M}}$$

Fonte: Campos (2009)

A equação $[u, v, 1]$ representa as coordenadas homogêneas do objeto na imagem, enquanto $[X, Y, Z]$ representa as coordenadas homogêneas do objeto no plano real. A matriz A é responsável por conter informações intrínsecas da câmera, como o ponto óptico e o comprimento focal, enquanto a matriz $[\mathbf{R} \mathbf{t}]$ armazena as informações extrínsecas, como direção e sentido do objeto (CAMPOS, 2009). A matriz $[\mathbf{R} \mathbf{t}]$ é formada por duas transformações matriciais: uma rotação (\mathbf{r})

e uma translação (t). A translação é responsável por alterar a posição do objeto; por exemplo, se um objeto está originalmente na posição do ponto $P(x, y)$, após uma translação t , o objeto será deslocado para a posição do ponto $Q(X+t_x, y+t_y)$. Já a rotação é responsável por alterar o sistema de coordenadas do objeto. Essas transformações são comumente aplicadas em computação gráfica.

Vários objetos podem ser imaginados, cada um com seu próprio sistema de coordenadas, também conhecido como sistema local. Quando esses objetos compõem uma cena, é necessário aplicar transformações de rotação para alinhar seus sistemas de coordenadas com um sistema global, que representa a cena como um todo. Dessa forma, os objetos são corretamente posicionados e orientados dentro do contexto da cena, facilitando a estimativa de velocidade dos veículos e outras aplicações de visão computacional. Essas transformações são amplamente utilizadas em computação gráfica para criar cenas tridimensionais e permitir a interação entre os objetos em um ambiente virtual.

Uma abordagem para calcular a matriz de homografia é utilizar 4 ou mais pontos correspondentes entre o objeto na imagem e o objeto no plano de referência (CAMPOS, 2009). Essa técnica torna o cálculo mais simples, pois não é necessário obter informações específicas, como o ponto óptico da câmera ou o comprimento focal. Ao utilizar pontos correspondentes entre as duas coordenadas, é possível estabelecer relações de transformação que permitem calcular a matriz de homografia de forma direta.

Com a matriz de homografia e o cálculo da distância euclidiana entre as posições do veículo, torna-se possível realizar a estimativa de velocidade do veículo, seguindo os seguintes passos: primeiro, o veículo é detectado e suas posições são registradas nos pontos A e B; em seguida, o tempo decorrido para percorrer a distância entre esses pontos é calculado. Posteriormente, os pontos A e B são convertidos para o sistema do espaço geográfico através da matriz de homografia; assim, a distância entre esses pontos no espaço geográfico é calculada. Por fim, a velocidade do veículo é estimada a partir da relação entre a distância percorrida e o tempo gasto nesse trajeto. A [Figura 29](#) demonstra um exemplo de estimativa da velocidade de veículos através de *deep learning*, e utilizando matriz de homografia para fazer a conversão do ponto no sistema da imagem para o sistema espacial.

A velocidade do veículo é recalculada a cada n quadros, sendo n a quantidade de quadros capturados pela câmera por segundo. Neste exemplo específico, a câmera grava a 50 quadros por segundo (50fps), portanto, a velocidade é recalculada a cada 50 quadros. Esse intervalo de atualização frequente permite uma estimativa precisa e contínua da velocidade do veículo ao longo do tempo, possibilitando uma análise detalhada de sua trajetória e comportamento no ambiente.

Além disso, o experimento conduzido por Mejia et al. (2021) também empregou uma técnica de suavização da velocidade. Essa estratégia envolve a atribuição de pesos à velocidade calculada do veículo em momentos anteriores e à velocidade atual, a fim de minimizar mudanças

Figura 29 – Estimativa de Velocidade dos veículos



Fonte: [Mejia et al. \(2021\)](#)

abruptas na estimativa. Através da ponderação adequada de ambos os valores, geralmente dentro da faixa entre 0 e 1, onde α representa o peso da velocidade atual e $(1 - \alpha)$ o peso da velocidade anterior, quanto mais próximo α estiver de 1, maior será a relevância da velocidade atual, enquanto mais próximo de 0, menor será a sua importância. O peso ideal pode ser determinado experimentalmente, adaptando-se a diferentes situações. A suavização proporciona uma variação mais gradual e controlada na estimativa de velocidade, tornando-a mais estável e menos suscetível a flutuações abruptas. Essa abordagem contribui de forma significativa para uma estimativa de velocidade mais consistente e confiável, especialmente em cenários onde podem ocorrer ruídos ou variações momentâneas nos dados capturados.

3

Trabalhos Relacionados

Este capítulo tem como objetivo apresentar alguns trabalhos relevantes e/ou semelhantes ao propósito do projeto, que consiste em desenvolver um modelo de estimativa de velocidade de veículos utilizando um modelo Faster R-CNN ou YOLO para tarefa de detecção de objetos. Para isso, realizou-se uma breve revisão sistemática com o intuito de investigar abordagens e mecanismos utilizados para estimativa de velocidade de veículos.

A fim de selecionar os trabalhos a serem apresentados, foram considerados aqueles que seguem os critérios abaixo:

- Artigos publicados e disponíveis integralmente em base de dados científicas;
- Devem ser relacionados ao tema de estimativa de velocidades de veículos, demonstrando uma abordagem para calcular essa velocidade;
- Trabalhos publicados entre 2018 e 2023;
- Escritos em inglês.

Para a realização da pesquisa, foram utilizadas as bibliotecas digitais IEEE Xplore¹, ScienceDirect², ACM Digital Library³ e Scopus⁴.

As *strings* de busca foram aplicadas no título, *abstract* e palavras-chave, sendo:

- (“vehicle speed estimation” OR “vehicle speed measurement” OR “vehicular speed inference”) AND (“Faster R-CNN” OR “YOLO”)

¹ Disponível em: <<https://ieeexplore.ieee.org/Xplore/home.jsp>>. Acesso em março de 2023.

² Disponível em: <<https://www.sciencedirect.com/>> Acesso em março de 2023.

³ Disponível em: <<https://dl.acm.org/>>. Acesso em março de 2023.

⁴ Disponível em: <<https://www.scopus.com/home.uri>> Acesso em agosto de 2023.

- (“vehicle speed estimation” OR “vehicle speed measurement” OR “vehicular speed inference”) AND (“camera calibration” OR “deep learning”).

A Tabela 2 apresenta a quantidade de resultados encontrados em cada biblioteca digital para ambas *strings* de buscas.

Tabela 2 – Resultados da pesquisa dos trabalhos relacionados

Biblioteca	Resultados (<i>String 1</i>)	Resultados (<i>String 2</i>)
IEE	4	18
ScienceDirect	9	20
ACM	2	6
Scopus	9	34
Total	4	78

Fonte: Autor

O processo de seleção dos estudos foi realizado por meio da pesquisa de artigos usando as *strings* de buscas. Em seguida, os resumos dos artigos foram filtrados, e aqueles que não possuíam um resumo relacionado ao tema foram excluídos. Os demais artigos foram lidos para análise. Na Tabela 3, são apresentados a quantidade de trabalhos rejeitados devido à falta de conformidade com os critérios de aprovação.

Tabela 3 – Resultados de exclusão dos trabalhos relacionados

Biblioteca	Resultados (<i>String 1</i>)	Resultados (<i>String 2</i>)
IEE	0	9
ScienceDirect	6	14
ACM	0	1
Scopus	0	13
Total	7	27

Fonte: Autor

3.1 Análise de Publicações Selecionadas

Esta seção consiste em uma análise detalhada dos trabalhos escolhidos, que foram selecionados devido à sua abordagem diferenciada em relação à maioria dos estudos conduzidos anteriormente. Aqui, será abordada uma perspectiva mais profunda sobre os trabalhos que despertaram maior interesse, discutindo suas características distintivas.

3.1.1 The Vehicle Speed Measuring and Precision Analysis of Video Shot by Moved Camera Based on 2D DLT Algorithm

O trabalho de [Huang et al. \(2015\)](#) tem como objetivo desenvolver um modelo de estimativa de velocidade de veículos, com a finalidade de investigar as velocidades dos veículos envolvidos em acidentes de trânsito em estradas. Esta abordagem visa auxiliar na compreensão das causas dos acidentes e na identificação de possíveis responsáveis.

No desenvolvimento do projeto, foi utilizado o algoritmo *Direct Linear Transformation* (DLT), em português, Transformação Linear Direta. Esse algoritmo é responsável por estabelecer a relação entre os pontos do objeto no espaço tridimensional e sua representação na imagem, permitindo a conversão das posições entre o espaço tridimensional e a imagem. Essa relação é estabelecida por meio da criação da matriz de homografia.

Para realizar a criação da matriz de homografia, [Huang et al. \(2015\)](#) utilizaram como referência 6 pontos na imagem com suas coordenadas espaciais conhecidas. A [Figura 30](#) apresenta esses pontos na imagem, os quais são utilizados para estabelecer a relação entre a imagem e o plano 3D.

Figura 30 – Cena de teste do 2D DLT



Fonte: [Huang et al. \(2015\)](#)

Para estimar a velocidade, é considerado o momento em que o veículo passa de um ponto P para um ponto Q . O cálculo da velocidade é determinado pela fórmula:

$$v = \frac{s \times f}{n} \quad (3.1)$$

onde s é a distância entre os pontos P e Q , f é a taxa de quadros por segundo (FPS) da câmera e n é o número de quadros em que o veículo passou do ponto P para o ponto Q .

3.1.2 Vehicle Speed Estimation Using Computer Vision And Evolutionary Camera Calibration

O modelo desenvolvido por Mejia et al. (2021) realiza o rastreamento e a estimativa de velocidade de veículos, utilizando a matriz de homografia para converter as coordenadas de pontos na imagem em coordenadas correspondentes no plano real. Além disso, os autores introduzem um algoritmo denominado *Estimation of Density Evolutionary Algorithm* (EDA), um algoritmo evolucionário de estimativa de densidade, para o cálculo da matriz de homografia. Eles realizaram uma comparação com algoritmos amplamente utilizados para esse propósito, como o *Direct Linear Transformation* (DLT) e o *Random Sample Consensus* (RANSAC).

O algoritmo DLT, dado as coordenadas dos pontos na imagem e no plano, calcula a matriz de homografia. O RANSAC, por outro lado, usa mais de 4 pontos, utilizando alguns para criar a matriz de homografia e os restantes para avaliar a taxa de erro dessa matriz, retornando assim a matriz com a menor taxa de erro. O método EDA, desenvolvido pelos autores, envolve alterações nos valores de latitude e longitude dos pontos para criar a matriz de homografia, buscando os valores de latitude e longitude que resultem na menor taxa de erro para essa matriz.

Uma vez criada a matriz de homografia, o modelo utiliza o YOLOv4 para detecção de veículos, juntamente com uma técnica de rastreamento para identificar e relacionar o mesmo veículo em diferentes cenas. A cada n quadros, o ponto de localização do veículo é extraído e a distância entre o ponto atual e o ponto anterior em que o veículo foi localizado é calculada. Essa distância é determinada utilizando a fórmula de Haversine, um cálculo que estima a distância entre dois pontos considerando a latitude e longitude. A fórmula para a estimativa de velocidade é dada por:

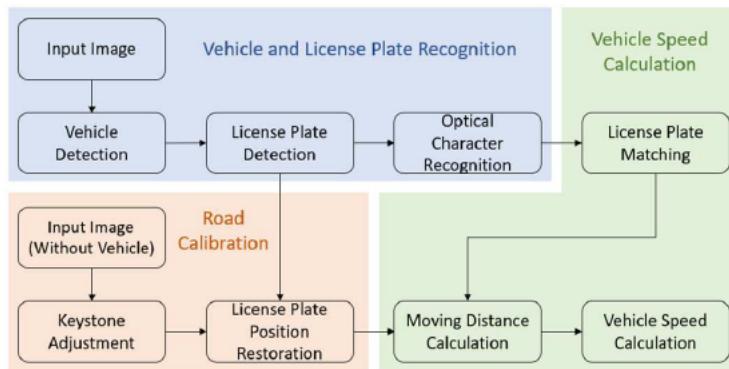
$$v = \frac{d(p, q)}{t} = \frac{d(p, q)}{f_s^{-1}} \times 3600 \quad (3.2)$$

Nessa fórmula, $d(p, q)$ representa a distância em quilômetros entre o ponto p e o ponto q , t é o tempo que o veículo leva para percorrer essa distância, e f_s^{-1} é o tempo gasto para cada estimativa de velocidade. Adicionalmente, uma técnica de suavização da velocidade é empregada para evitar mudanças abruptas na velocidade estimada. Nesse sentido, para determinar a velocidade do veículo, são atribuídos pesos tanto à velocidade atual do veículo quanto à última velocidade calculada para o veículo. Essa abordagem de suavização contribui para um resultado mais estável e coerente, minimizando variações bruscas na velocidade estimada.

3.1.3 Design and Implementation of Vehicle Speed Estimation Using Road Marking-based Perspective Transformation

O objetivo do trabalho de Wu et al. (2021b) é desenvolver um modelo de estimativa de velocidade de veículos que não dependa de informações específicas da câmera, como distância focal ou ponto óptico, nem necessite das coordenadas dos pontos na imagem e no plano 3D para a criação da matriz de transformação. Para atingir esse objetivo, eles propuseram um modelo que utiliza as características da estrada como ponto de referência para a calibração da câmera. O processo para alcançar esse objetivo consiste em várias etapas, conforme ilustrado na Figura 31.

Figura 31 – Arquitetura do modelo de estimativa de velocidade desenvolvido por Wu et al. (2021b)



Fonte: Wu et al. (2021b)

A primeira etapa é a detecção do veículo na imagem, seguida pela detecção da placa do veículo, a qual desempenha um papel crucial no processo de calibração da câmera. A placa é utilizada para identificar o veículo nos cálculos de velocidade e também para a extração dos caracteres que compõem a identificação do veículo. Posteriormente, o algoritmo valida a associação entre a placa e o veículo, determinando qual placa pertence a qual veículo. Com essa informação, é calculada a distância do veículo e, por fim, a velocidade é estimada.

No processo de calibração, a imagem da estrada passa por diversas transformações, como filtragem mediana, visando facilitar a localização das faixas de trânsito. A maior faixa de trânsito é selecionada como referência para a criação da matriz de transformação, desde que seja maior que um milésimo do tamanho da imagem. Além disso, é realizada uma transformação da perspectiva da imagem de modo que tanto a faixa de trânsito quanto o veículo sejam apresentados em perspectiva horizontal, o que facilita a avaliação da distância.

A equação para calcular a velocidade (km/h) é a seguinte:

$$speed = \frac{d \cdot fps \cdot 60 \cdot 60}{100 \cdot 1000} \quad (3.3)$$

onde d é a distância entre o ponto central da placa do veículo no quadro i e o ponto central da placa do mesmo veículo no quadro j , e fps é a taxa de quadros por segundo com a qual a câmera está gravando. Os resultados obtidos pelos autores estão apresentados no [Quadro 2](#).

Quadro 2 – Resultados obtidos no trabalho de [Wu et al. \(2021b\)](#)

Velocidade	Quantidade de detecções	Erro absoluto (AE)	Erro relativo (RE)
20 km/h	29	1,54km/h	7,72 %
30 km/h	25	2,40km/h	7,99 %
40 km/h	18	5,66km/h	14,11 %

Fonte: [Wu et al. \(2021b\)](#)

3.1.4 Deep learning-based vehicle speed estimation using the YOLO detector and 1D-CNN

O modelo proposto por [Cvijetić, Djukanović e Peruničić \(2023\)](#) utiliza o YOLOv5 para a detecção de veículos, juntamente com um modelo de *deep learning* que eles denominaram de 1D-CNN, devido à sua característica principal de empregar camadas de convolução 1D. Essa abordagem elimina a necessidade de calibração da câmera, tornando o modelo aplicável em diversos cenários. A estimativa de velocidade é conduzida pelo modelo 1D-CNN.

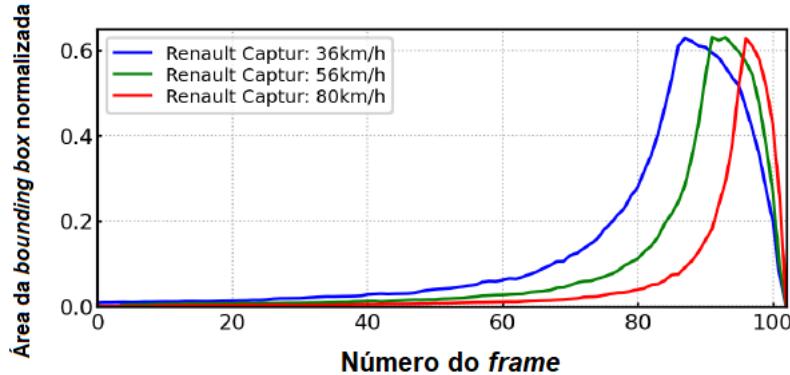
Para realizar a estimativa de velocidade usando a 1D-CNN, é essencial obter a área das *bounding boxes* do veículo durante os quadros em que ele é detectado. [Cvijetić, Djukanović e Peruničić \(2023\)](#) identificaram que a velocidade afeta o tamanho dessas *bounding boxes*, pois quanto maior a velocidade, mais acentuada é a curva das áreas das *bounding boxes*. Eles denominaram essa variação de "change in bounding boxes area"(CBBA), ou seja, a alteração na área das caixas delimitadoras. A [Figura 32](#) ilustra a influência da velocidade na variação do CBBA.

Para cada veículo, as áreas das *bounding boxes* são registradas e normalizadas em relação à maior área de *bounding box* entre todos os veículos, ou seja, as áreas são divididas pela maior área encontrada. Em seguida, essas áreas normalizadas são usadas como entrada para o modelo 1D-CNN, que retorna o valor estimado da velocidade para aquele veículo.

O modelo 1D-CNN desenvolvido pelos autores é constituído por duas camadas de convolução 1D, cada uma composta por 64 filtros e um tamanho de *kernel* igual a 10. Além disso, o processo de treinamento foi executado por 100 épocas, utilizando um tamanho de lote igual a 32.

Para conduzir o experimento, os autores desenvolveram um conjunto de dados denominado vs13. Essa base de dados contém diversos vídeos de 13 carros diferentes em velocidades variadas, sendo que as velocidades foram registradas por meio de um sensor LiDAR. Os resultados obtidos pelos autores estão apresentados na [Tabela 4](#).

Figura 32 – CBBA da velocidade de 3 veículos



Fonte: Cvijetić, Djukanović e Peruničić (2023)

Tabela 4 – Resultados obtidos no trabalho de Cvijetić, Djukanović e Peruničić (2023)

Veículo	RMSE (km/h)
Citroen C4 Picasso	3,09
Kia Sportage	5,57
Mazda 3 Skyactive	2,53
Mercedes AMG 550	2,87
Mercedes GLA	1,31
Nissan Qashqai	2,46
Opel Insignia	2,12
Peugeot 208	3,02
Peugeot 3008	3,32
Peugeot 307	3,86
Renault Captur	2,42
Renault Scenic	1,85
VW Passat B7	1,43
média	2,76

Fonte: Cvijetić, Djukanović e Peruničić (2023)

3.1.5 Vehicle Speed Estimation Based on 3D ConvNets and Non-Local Blocks

O trabalho de Dong, Wen e Yang (2019) introduziu um modelo inovador para estimar a velocidade dos veículos, eliminando a necessidade de calibragem da câmera. Eles adotaram uma abordagem de *deep learning*, empregando camadas convolucionais 3D. Enquanto as CNNs 2D são amplamente utilizadas para extrair informações espaciais, as 3D vão além, incorporando informações temporais. Adicionalmente, foram empregados blocos "non-local" para permitir que as camadas convolucionais capturem informações globais, expandindo sua capacidade além das

limitações dos campos receptivos locais.

A utilização dos blocos "non-local" permitiu avaliar a relevância de pontos individuais da imagem em relação aos demais pontos na mesma imagem. Isso contribuiu para a obtenção de informações globais, enriquecendo o modelo com detalhes espaço-temporais essenciais sobre o comportamento dos veículos.

Os autores também empregaram outra técnica importante, a convolução multi-escalar, reconhecendo uma particularidade nos vídeos: o tamanho dos veículos não é diretamente proporcional ao tamanho real, devido à variação causada pela proximidade com a câmera. Sendo assim, [Dong, Wen e Yang \(2019\)](#) incorporaram múltiplas camadas convolucionais como entrada para o vídeo, cada uma delas com um tamanho de *kernel* diferente. Essa abordagem permite que o modelo capture informações em diversas escalas. Os valores resultantes das saídas dessas camadas convolucionais são concatenados, o que habilita o modelo a lidar com variações de escala nos objetos em análise. Essa estratégia é fundamental para garantir que o modelo seja capaz de abordar cenários que envolvem diferentes tamanhos de veículos de maneira eficaz.

Com o intuito de melhorar a eficiência do modelo em termos de velocidade de inferência, os autores optaram por transformar as camadas de CNN 3D em camadas de CNN (2 + 1)D, onde as dimensões de tempo e espaço são tratadas separadamente. Por exemplo, uma camada 3D $t \times k \times k$ foi substituída por uma camada $t \times 1 \times 1$ juntamente com uma camada $1 \times k \times k$, onde t representa a dimensão temporal e k a dimensão espacial (largura e altura). Essa estratégia permitiu preservar a qualidade do modelo, ao mesmo tempo que otimizava sua eficiência. Os resultados obtidos pelos autores e também pelos métodos utilizados para comparação do seu modelo estão descritos na [Tabela 5](#).

Tabela 5 – Resultados obtidos no trabalho de [Dong, Wen e Yang \(2019\)](#)

Métodos	MAE	MSE	Automático	Detecção-Rastreio
GPS	1,64	-	✓	✗
RADAR	1,07	-	✓	✗
FULLACC	8,59	104,98	✓	✓
OPtScale	1,71	9,66	✗	✓
OPtScaleVP2	15,66	411,67	✗	✓
OPtCalib	1,43	8,71	✗	✓
OPtCalibVP2	2,43	15,63	✗	✓
Proposto	2,73	14,62	✓	✗

Fonte: [Dong, Wen e Yang \(2019\)](#)

3.1.6 Vision-based Vehicle Speed Estimation Using the YOLO Detector and RNN

O estudo conduzido por [Peruničić, Djukanović e Cvijetić \(2023\)](#) apresenta um modelo de estimativa de velocidade que utiliza a estrutura YOLO para a detecção dos veículos, combinada com um modelo de Rede Neural Recorrente (RNN). Nessa abordagem a RNN emprega a posição do veículo e as dimensões da *bounding box* para realizar a estimativa de velocidade.

O YOLO foi selecionado devido à sua natureza de 1-estágio, que permite detecções em uma única passagem pela rede, resultando em eficiência no processo. Dentro das variantes do YOLO, a escolha foi do YOLOv5, reconhecido por sua precisão e velocidade de detecção. Para o modelo RNN, a escolha dos autores foi a Long Short-Time Memory (LSTM), uma vez que essa estrutura é capaz de reter informações processadas em etapas anteriores, graças ao seu estado oculto, que recebe a entrada da rede no instante t e a saída do estado oculto do instante $t - 1$.

O processo começa com a extração dos pontos superior-esquerdo e inferior-direito da *bounding box* para cada veículo em cada quadro onde ele é detectado. Esses pontos são então usados para calcular a área da *bounding box* em cada quadro. Para evitar lacunas nas áreas, onde um veículo é detectado em um quadro e não nos seguintes (resultando em dados ausentes), os autores empregaram a interpolação linear para preencher esses espaços.

As áreas calculadas são então utilizadas como entradas para o modelo LSTM, que efetua as previsões de velocidade. Cada conjunto de áreas de um veículo constitui uma entrada, e para uniformizar os tamanhos, [Peruničić, Djukanović e Cvijetić \(2023\)](#) padronizaram todas as entradas com o tamanho do menor conjunto de áreas. Esta abordagem alcançou em um erro médio quadrático (RMSE) de 4,08 km/h, com uma característica interessante da independência de informações específicas da câmera ou das coordenadas geográficas do local.

Os resultados obtidos no trabalho estão apresentados na [Tabela 6](#), os quais indicam que o modelo possui um grande potencial, mas ainda requer novos experimentos para alcançar um erro comparável ao dos equipamentos de medição de velocidade, como as pistolas utilizadas pelos policiais, para fins de aferição.

Este trabalho demonstra que as Redes Neurais Recorrentes (RNNs), embora amplamente utilizadas no processamento de linguagem natural (PLN), também têm aplicações viáveis em outras áreas, como a visão computacional. As RNNs aproveitam a capacidade de reter informações temporais, permitindo a modelagem da localização do veículo em diferentes momentos, o que facilita a previsão da velocidade do veículo.

Tabela 6 – Resultados obtidos no trabalho utilizando RNN de Cvijetić, Djukanović e Peruničić (2023)

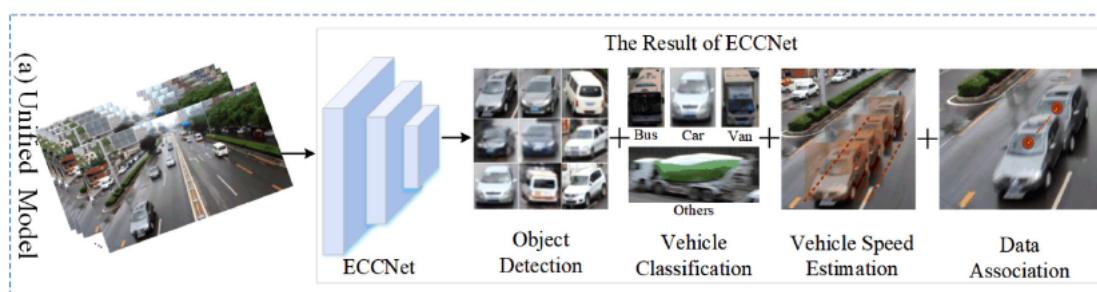
Veículo	$\text{RMSE}_{\text{areaPosition}}$	$\text{RMSE}_{\text{area}}$
Citroen C4 Picasso	5,04	2,38
Kia Sportage	3,25	4,90
Mazda 3 Skyactive	5,10	5,10
Mercedes AMG 550	4,58	4,51
Mercedes GLA	4,06	3,65
Nissan Qashqai	2,93	3,14
Opel Insignia	9,40	3,77
Peugeot 208	3,99	3,28
Peugeot 3008	3,27	3,15
Peugeot 307	5,35	4,99
Renault Captur	4,60	4,90
Renault Scenic	3,84	3,92
VW Passat B7	3,27	3,77
média	4,70	4,08

Fonte: Cvijetić, Djukanović e Peruničić (2023)

3.1.7 ECCNet: Efficient chained centre network for real-time multi-category vehicle tracking and vehicle speed estimation

O modelo proposto por Yu et al. (2022), diferentemente das abordagens anteriores, dispensa o uso de diferentes modelos para diferentes etapas do processo de estimativa de velocidade. Em vez disso, um único modelo é empregado para realizar a detecção do veículo, o rastreio e a estimativa da velocidade. Esse modelo específico, criado pelos autores, foi denominado de Efficient Chained Centre Network (ECCNet). Uma visão ilustrativa das etapas do modelo ECCNet pode ser observada na Figura 33.

Figura 33 – Estrutura do ECCNet



Fonte: Yu et al. (2022)

Uma característica do ECCNet é a sua capacidade de utilizar quadros adjacentes para

aprimorar a estimativa de velocidade quando um veículo é detectado em um determinado quadro i . Isso é possível devido à interconexão dos ramos do modelo, que permite a transferência de informações entre esses ramos. Dessa forma, os quadros $i - 1$ e $i + 1$ são explorados para melhorar a precisão da estimativa.

A fim de realizar a estimativa de velocidade, os autores incorporaram uma camada de convolução 3D, permitindo assim que o modelo extraísse não apenas informações espaciais, mas também temporais. Sendo assim, alcançando uma capacidade de predição de velocidade sem a necessidade de calibração da câmera. Além disso, para evitar o retrabalho na extração de mapas de características, os autores optaram por reutilizar os mapas gerados durante as fases de detecção e rastreio do veículo. Essa estratégia reduz a complexidade geral do modelo, uma vez que os mapas já extraídos não precisam ser recalculados.

Os experimentos foram realizados utilizando a base de dados UA-DETRAC, que engloba 10 horas de vídeos em 24 localizações distintas de Beijing e Tiajin, na China. Para avaliar a eficácia do modelo, o erro médio quadrado (RMSE) foi empregado como métrica, além do erro médio absoluto (MAE) e também o F1-score, recall e precisão para validação da classificação do modelo. Os resultados são demonstrados na [Tabela 7](#)

Tabela 7 – Resultados obtidos no trabalho de [Yu et al. \(2022\)](#)

Cenários	Precisão	Recall	F1	MAE	RMSE
ensolarado	0,87	0,88	0,87	2,85	4,00
nublado	0,76	0,80	0,78	2,77	3,51
chuvisco	0,67	0,69	0,67	3,18	3,90
noturno	0,79	0,73	0,76	3,59	4,48

Fonte: [Yu et al. \(2022\)](#)

3.1.8 A Real-Time Vehicle Counting, Speed Estimation, and Classification System Based on Virtual Detection Zone and YOLO

O estudo conduzido por [Lin, Jeng e Lioa \(2021\)](#) propõe um modelo abrangendo estimativa de velocidade, contagem de veículos e classificação. Para alcançar esses objetivos, eles adotaram um sistema baseado em zonas de detecção virtual, combinado com a tecnologia YOLO.

Para efetuar a contagem de veículos, empregaram um modelo de mistura gaussiana (GMM), identificando as regiões de movimento dos objetos. Em cada vídeo, uma área de detecção virtual é delimitada, e o incremento na contagem ocorre sempre que um veículo cruza essa fronteira. A estimativa da velocidade é executada quando um carro é identificado dentro da zona de detecção virtual. O cálculo é feito através da distância percorrida pelo veículo entre o ponto de detecção inicial e o ponto final. Para determinar o tempo que o veículo leva nesse

trajeto, é aplicada a fórmula:

$$t = \frac{P}{fps} \quad (3.4)$$

Nesta fórmula, P representa o número de quadros que o veículo leva para transitar entre os pontos, e fps é a frequência de quadros capturados pela câmera por segundo. Com base nesse intervalo de tempo, aplicam a fórmula da velocidade média para obter a estimativa da velocidade do veículo.

A validação do modelo foi realizada utilizando os conjuntos de dado: Montevideo Audio and Video Dataset (MAVD) e o GARM Road-Traffic Monitoring (GRAMRTM). Para avaliar a eficácia, empregaram a métrica do erro percentual absoluto. Os resultados dos testes estão descritos na Tabela 8

Tabela 8 – Resultados obtidos no trabalho de [Lin, Jeng e Lioa \(2021\)](#)

ID Veículo	Velocidade real (km/h)	velocidade estimada (km/h)	Diferença	Erro (%)
1	60	63	3	5
2	70	75	5	7
3	72	63	-9	12,5
4	99	100	1	1
5	84	85	1	1
6	67	60	-7	10
7	73	71	-2	2,7
8	67	64	-3	4,4
9	37	43	6	16
10	73	77	4	5
11	55	50	-5	9
12	48	54	6	12,5
13	111	127	16	14,4
14	79	75	-4	5
15	69	71	2	2,8
16	82	75	-7	8,5
17	83	73	-10	12
Erro médio:	7,6			

Fonte: [Lin, Jeng e Lioa \(2021\)](#)

3.1.9 Vehicle Speed Estimation From Audio Signals Using 1D Convolutional Neural Networks

O estudo proposto por [Čavor e Djukanović \(2023\)](#) apresenta um modelo para a estimativa de velocidade com base em informações de áudio do veículo durante seu trajeto em uma estrada específica.

Para realizar a predição de velocidade, os autores utilizaram uma Rede Neural Convolucional Unidimensional (1D CNN), que recebe como entrada o áudio do veículo nos 3 segundos anteriores e 3 segundos posteriores ao momento em que o veículo passa pela câmera. O áudio é amostrado a uma taxa de 44.100Hz e é organizado em um vetor de dimensão 246.600, que é então alimentado na 1D CNN.

A eficácia dessa abordagem foi avaliada por meio de uma base de dados desenvolvida pelos próprios autores, denominada vs13, que está disponível digitalmente. Essa base de dados inclui informações de 13 veículos diferentes, com gravações de 400 amostras de velocidade variando entre 30-110 km/h.

Para realizar os testes, [Čavor e Djukanović \(2023\)](#) executaram o treinamento do modelo 13 vezes, reservando um veículo diferente para teste em cada rodada. Após os 13 experimentos, um para cada veículo de teste, eles calcularam a média do erro médio quadrático (RMSE) como métrica de eficácia. Os resultados dos testes são demonstrados na [Tabela 9](#)

Tabela 9 – Resultados obtidos no trabalho utilizando áudio dos veículos

Veículo	RMSE₁ (km/h)	RMSE₂ (km/h)
Citroen C4 Picasso	7,83	7,15
Kia Sportage	11,38	7,93
Mazda 3 Skyactive	9,22	8,48
Mercedes AMG 550	10,44	8,5
Mercedes GLA	11,32	8,24
Nissan Qashqai	10,17	5,53
Opel Insignia	8,00	9,75
Peugeot 208	5,81	4,71
Peugeot 3008	10,70	10,25
Peugeot 307	9,49	8,09
Renault Captur	7,94	8,23
Renault Scenic	15,38	19,96
VW Passat B7	7,85	8,62
média	9,65	8,88

Fonte: [Cvijetić, Djukanović e Peruničić \(2023\)](#)

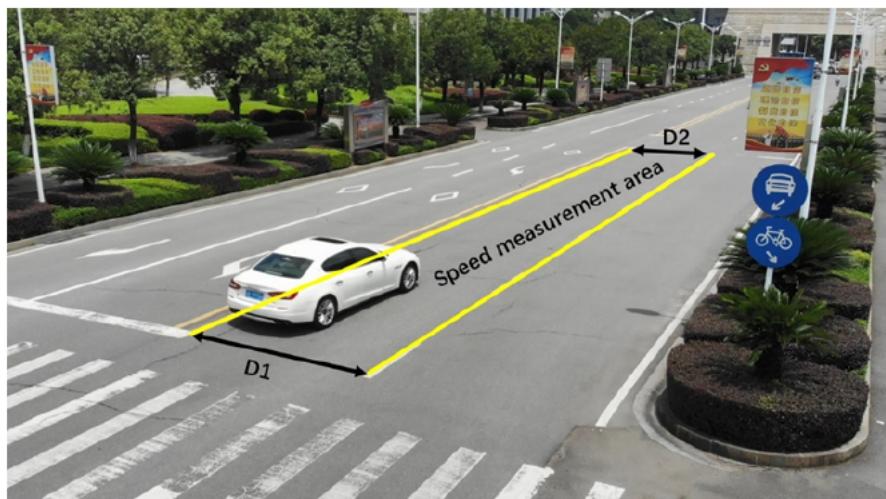
3.1.10 Intelligent Highway Speed Monitoring UAV System Based on Deep Learning

O estudo conduzido por [Yi, Guan e Li \(2021\)](#) apresenta um modelo que combina estimativa de velocidade e reconhecimento automático de placas veiculares. Os autores justificam essa abordagem pelo alto custo e a inviabilidade do método convencional de estimativa de velocidade em rodovias, que é difícil de ser deslocado entre diferentes locais. Em resposta a esse

desafio, eles desenvolveram um modelo especialmente adaptado para o monitoramento por meio de drones.

Para efetuar a estimativa de velocidade, o sistema verifica se um veículo está dentro de uma faixa da rodovia, como é ilustrado na [Figura 34](#). Caso afirmativo, o modelo realiza uma predição da velocidade, considerando o segmento específico da rodovia que é visível no vídeo. A relação entre as distâncias na imagem e no mundo real (espaço geográfico) é estabelecida por meio da aplicação da matriz de homografia. Para o reconhecimento das placas automotivas, os autores empregam o modelo Connectionist Temporal Classification (CTC).

Figura 34 – Esquema de medição de linha da via



Fonte: [Yi, Guan e Li \(2021\)](#)

A fase de experimentação foi realizada utilizando o hardware do Kit NVidia Jetson TX2. Para avaliar a eficácia do modelo, utilizaram as métricas do erro médio quadrático (RSE) e o erro médio absoluto (MAE), resultando em valores de 0,420 e 0,567, respectivamente.

Os resultados alcançados pelos autores estão apresentados na [Tabela 10](#). Os testes envolveram sete veículos distintos, cujas placas e velocidades estimadas em km/h são exibidas nos resultados para quatro diferentes métodos: detecção via radar rodoviário, uso de uma matriz de homografia como princípio, emprego da linha da via como referência para calcular a distância percorrida pelo veículo e, por último, um modelo que utiliza aprendizado de conjunto.

A abordagem de incorporar drones nesse modelo torna a proposta inovadora e interessante. Através dessa implementação, o modelo pode cobrir uma extensa variedade de rodovias, contribuindo para a análise de uma ampla área de tráfego. Em muitos cenários, placas que estabelecem limites de velocidade nem sempre são respeitadas, a menos que haja uma fiscalização eletrônica. Entretanto, ao empregar drones para monitorar essas rodovias, os motoristas podem sentir uma maior restrição em exceder os limites de velocidade, uma vez que a possibilidade de detecção se torna mais ampla e abrangente.

Tabela 10 – Resultados obtidos no trabalho de Yi, Guan e Li (2021)

Placar veicular	XAXY189	XA4880G	XAC33X6	XA2201A	MD8514G	XA639CX	XA85YY9
Velocidade radar (km/h)	81,5	85,4	83,6	85,2	83,4	101,8	100,7
Velocidade matriz de homografia (km/h)	82,96	86,14	84,21	87,29	84,64	102,56	101,58
Velocidade extração de linha da via (km/h)	83,16	87,24	85,62	87,21	85,24	103,65	102,57
Velocidade aprendizado de conjunto (km/h)	80,86	84,58	82,8	84,67	82,78	100,92	99,89

Fonte: Yi, Guan e Li (2021)

3.2 Análise Comparativa

A Tabela 11 demonstra um resumo dos trabalhos selecionados durante a revisão sistemática. Esses estudos representam algumas das abordagens e metodologias utilizadas para a detecção de veículos em diversos cenários. Ao examinar os autores, modelos, bases de dados, hardware utilizado, taxas de quadros alcançadas e métricas avaliadas, é possível obter uma visão panorâmica das estratégias empregadas na área. A análise desses trabalhos contribui para a compreensão das tendências atuais no campo da detecção de veículos, sendo como base para as futuras pesquisas e desenvolvimento.

Tabela 11 – Análise Comparativa dos trabalhos da Revisão Sistemática

Capítulo 3	Autor	Modelo	Base de Dados	Hardware	Taxa de quadros	Métrica	Resultado
							PAE
Linguo et al. (2022)	Haar Cascade 2D DLT	Própria	Cloud Cloud	30fps	-	-	1,22%
Huang et al. (2015)	OD: YOLOv2 e OCR OD: YOLOv4 / SE: EDA	Própria Câmeras públicas de Seattle vs 1.3 Dataset	Computador Pessoal	30fps	-	AE e RE	-
Wu et al. (2021b)	OD: YOLOv5m SE: 1D-CNN 3D ConvNet	-	30fps	-	-	RMSE	1,22 e 9,94%
Mejia et al. (2021)	OD: YOLOv5m SE: 1D-CNN BrnoCompSpeed	-	50fps	MSE e MAE	RE	-	2,76km/h
Cvijetić, Djukanović e Peruničić (2023)	OD: YOLOv3 OD: YOLOv5	Própria	-	-	-	-	1,62 e 2,73
Dong, Wen e Yang (2019)	OD: Faster R-CNN SE: AutoCalib OD: CTC SE: lane extraction	Câmeras públicas em Seattle	NVIDIA JETSON TX2	-	-	PAE	-
Tomas et al. (2022)	OD: Faster R-CNN SE: AutoCalib OD: CTC SE: lane extraction	Própria	NVidia AI City	30fps	MSE e MAE	8,98%	-
Prajwal et al. (2022)	OD: Faster R-CNN SE: AutoCalib OD: YOLOv5	Própria	Computador Pessoal	50fps	ER	0,42 e 0,567	0,42 e 0,567
Bhardwaj et al. (2018)	OD: Faster R-CNN SE: AutoCalib OD: CTC SE: lane extraction	Diversas rodovias	Computador Pessoal	30fps	RMSE	5,61%	5,61%
Yi, Guan e Li (2021)	OD: Faster R-CNN SE: AutoCalib OD: CTC SE: lane extraction	Própria	Rodovias da Noruega	-	RMSE	27,30km/h	27,30km/h
Song, Guan e He (2018)	OD: Faster R-CNN	Própria	BrnoCompSpeed	30fps	RMSE	12,10km/h	12,10km/h
Giannakeris et al. (2018)	OD: 3D Deformable model DeepConvLSTM	Diversas rodovias	Computador Pessoal	-	RMSE	7,6km/h	7,6km/h
Skjermo et al. (2020)	OD: Mask R-CNN	Própria	Computador Pessoal	50fps	MAE	5km/h	5km/h
Hua, Kapoor e Anastasiu (2018)	1D CNN	Própria	Cloud	30fps	MAE	3,86km/h	3,86km/h
Revaud e Humenberger (2021)	OD: YOLO	Própria	Cloud	-	-	-	-
Abdelgawad et al. (2019)	OD: Mask R-CNN	-	-	-	Mean e Min Error	5,39km/h	5,39km/h
Do et al. (2021)	OD: YOLOv4 SE: Inception-v4 OD: SSD SE: LSTM	BrnoCompSpeed e AIC18 Shinjuku, Hitotsubashi vs 1.3 Dataset	Computador Pessoal	50fps	MSE	16,67km/h	16,67km/h
Carratù et al. (2022)	OD: Mask R-CNN	SE: 1D-CNN	-	-	MAE	1,88km/h	1,88km/h
Yohannes et al. (2023)	OD: YOLOv4 SE: Inception-v4 OD: SSD SE: LSTM	BrnoCompSpeed	Computador Pessoal	30fps	RMSE	8,88km/h	8,88km/h
Wang et al. (2018b)	OD: YOLOv4	vs 1.3 Dataset	-	50fps	MAE, AEP	0,96km/h	0,96km/h
Čavor e Djukanović (2023)	OD: YOLOv5 SE:RNN	BrnoCompSpeed	-	30fps	RMSE	4,08km/h	4,08km/h
Rais e Munir (2021)	OD: HWD-Net	Rodovias urbanas	Computador Pessoal.	30fps	-	-	-
Peruničić, Djukanović e Cvjetić (2023)	OD: YOLO	-	-	-	Average Error	2,87km/h	2,87km/h
Ashraf et al. (2023)	Própria	-	-	-	RE	0,70%	63,0,70%
Trivedi, Mandalapu e Dave (2022)	-	-	-	-	-	-	-
Zhao, Tang e Wang (2023)	-	-	-	-	-	-	-

Yu et al. (2022)	ECCNet	UA-DETRAC	Computador Pessoal.	-	RMSE e MAE	3,10km/h
Rodríguez-Rangel et al. (2022)	OD: YOLOv3	Própria	-	60fps	MAE	-
Shahbazi et al. (2020)	OD: YOLOv3	UAV	-	30fps	RMSE	12,10km/h
Bell, Xiao e James (2020)	OD: YOLOv2	Rodovias britânicas	-	25fps	RMSE, MAE	-
Liu, Wang e Song (2020)	-	estrada de Xi'an	Computador Pessoal	25fps	Estimation Error	2km/h
Li et al. (2019b)	OD: YOLOv3	UAV dataset	Computador Pessoal	17fps	Average Error	-
Lin, Jeng e Lioa (2021)	OD: YOLOv4	(MAVD e (GRAMRTM))	7,60%	-	30fps	MAPE

Fonte: Autor

4

Experimentos de Comparação do Faster R-CNN e YOLOv8

Foi realizado um experimento para comparar a eficiência das versões do YOLOv8 e do Faster R-CNN na detecção de objetos. As versões do YOLOv8 utilizadas foram a *nano*, *small* e *medium*, disponibilizadas pela Ultralytics¹. Já para o Faster R-CNN, foi utilizada uma versão disponibilizada na biblioteca torchvision do Pytorch², que utiliza o modelo ResNet50 para a extração dos *feature maps*. A base de dados utilizada foi a *Car Object Detection*³, que contém 1.001 imagens de treino e 175 de teste para detecção de carros. Para uma melhor compreensão do experimento, este será dividido em subseções, representando cada etapa do experimento. O código do experimento encontra-se no apêndice A.

O experimento foi conduzido no ambiente de desenvolvimento do Google Colab, utilizando a linguagem de programação Python.

4.1 Preparação dos dados

A base de treino foi dividida em dois conjuntos: treino e validação. A técnica de *hold-out* foi empregada, dividindo a base em 80% para treino e 20% para validação.

Para utilizar os dados no modelo YOLOv8, alguns passos específicos foram necessários. Por exemplo, foi necessário utilizar a função *normalize* para normalizar as informações das *bounding boxes* pela altura e largura da imagem, já que o YOLOv8 requer esse formato de entrada. Além disso, o modelo Faster R-CNN utiliza as coordenadas de *x* e *y* mínima e máxima da *bounding box*, enquanto o YOLOv8 utiliza a coordenada do ponto central e a largura e altura da *bounding box*.

Outro passo importante para treinar o modelo YOLOv8 é criar um arquivo chamado

¹ Disponível em: <<https://www.ultralytics.com/>>. Acesso em março de 2023

² Disponível em: <<https://pytorch.org/>>. Acesso em março de 2023

³ Disponível em: <<https://www.kaggle.com/datasets/sshikamaru/car-object-detection>>. Acesso em março de 2023

data.yaml, que contém informações sobre os dados de treinamento, validação e teste, além de possuir a quantidade e os nomes das classes dos objetos.

4.2 Treinamento do modelo

O modelo do Faster R-CNN utilizado neste experimento tem a rede neural resnet50 como *backbone*. A utilização do parâmetro *weights* faz com que o modelo receba os pesos de um modelo já pré-treinado com a base de dados que passar nesse parâmetro, nesse caso a base de dados do Coco versão 1.

A versão do modelo Faster R-CNN utilizada possui 41.755.286 parâmetros, tamanho de 160 megabytes e quantidade de operações de 134,38GFLOPs, já o modelo YOLOv8 nano utilizado possui 3.157.200 parâmetros; 225 camadas; quantidade de operações de 8,9 GFLOPs e tamanho de 6,23 megabytes.

Para verificar a eficiência do modelo, foi utilizado a métrica mAP, já a função de otimização utilizada nesse experimento é a SGD.

Após o treinamento de cada época, o modelo é validado com a base de dados de validação, e seu mAP é calculado. A mAP é então comparado com a melhor mAP obtida até o momento, e caso seja melhor, os pesos do modelo, e da função de otimização são salvos.

4.3 Inferência

A função *inference* é utilizada para realizar a detecção de objetos das imagens da base de teste. Ao executar a função, ela retorna 3 listas contendo as *bounding boxes*, *labels* e *scores* de cada imagem processada, outra com as *bounding boxes* e *labels* da base de dados e uma terceira com as mAPs de cada imagem. Além disso, a função *inference* salva as imagens com as *bounding boxes* resultantes, um arquivo .txt com as labels e outro arquivo .txt com as mAPs de cada imagem. Dessa forma, é possível visualizar e avaliar os resultados obtidos pelo modelo.

4.4 Resultados

Após as etapas de preparação dos dados, treinamento do modelo e inferência dos dados, foi possível comparar os resultados obtidos de cada modelo. Sendo testado, o modelo de Faster R-CNN, e o YOLOv8 com as variações *nano*, *small* e *medium*.

A métrica utilizada para comparação dos modelos foi a média da precisão média, ou *Mean Average Precision* (mAP). Para cada modelo foi realizado a média das mAPs encontradas para cada imagem. Os valores resultantes são demonstrados no [Quadro 3](#)

Quadro 3 – Resultados obtidos no experimento de comparação do Faster R-CNN e YOLOv8

Modelo	mAP	Inferência(ms)
Faster R-CNN	0.43	90,91
YOLOv8n	0.5	10,48
YOLOv8s	0.45	17,87
YOLOv8m	0.35	37,47

Fonte: Autor

Em geral, o Faster R-CNN apresenta uma precisão superior ao YOLO, enquanto o YOLOv8m supera o YOLOv8s e o YOLOv8n em termos de precisão. No entanto, neste experimento, observou-se um padrão inverso, no qual a precisão dos modelos foi diferente do esperado. Essa inversão de padrão pode estar relacionada a diversos fatores, que são descritos a seguir.

Os valores baixos das mAPs podem ser atribuídos à escolha do valor de IoU utilizado para avaliar a qualidade das *bounding boxes*, que foi o map@0.5:0.05:0.95, ou seja, o cálculo foi realizado para IoUs variando de 0.5 a 0.95 em intervalos de 0.05. Como o valor de IoU utilizado foi alto, as *bounding boxes* precisaram ser quase idênticas às *bounding boxes* verdadeiras para serem consideradas corretas, o que pode ter afetado negativamente as mAPs dos modelos analisados.

Outro fator que influenciou no resultado baixo das mAPs foi a detecção de um mesmo carro mais de uma vez, como pode ser observado na Figura 35 e na Figura 36. Para resolver ou reduzir esse problema, pode-se utilizar a técnica de *Non Max Supression*, que é discutida na subseção 2.1.3.3. Foi observado que os modelos YOLOv8m e YOLOv8s tiveram mais problemas com a detecção redundante de objetos do que os demais.

Figura 35 – Detecção de objetos com YOLOv8m

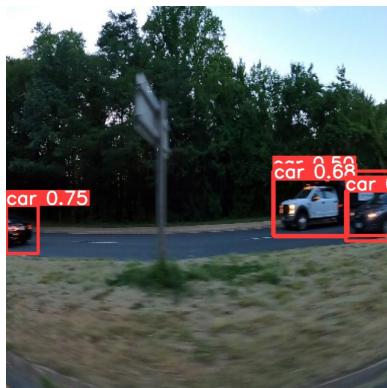
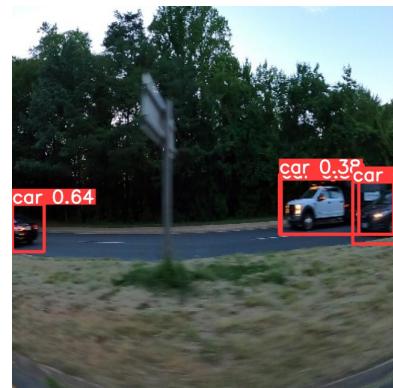
Fonte: [sshikamaru \(2023\)](#)

Figura 36 – Detecção de objetos com YOLOv8s

Fonte: [sshikamaru \(2023\)](#)

Durante os testes realizados, também foi possível observar que o modelo Faster R-CNN teve dificuldades em detectar objetos pequenos, enquanto o YOLO não apresentou o mesmo

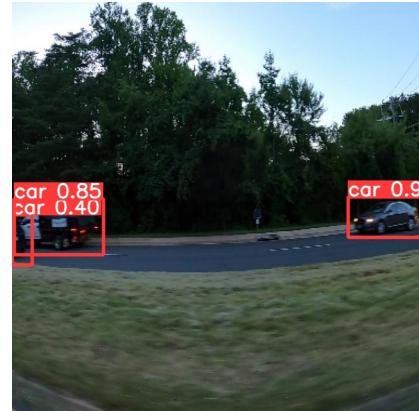
problema. Isso pode ser considerado um dos motivos para o desempenho inferior do Faster R-CNN em relação ao YOLOv8n. Essa dificuldade pode ser visualizada na [Figura 37](#) e na [Figura 38](#).

Figura 37 – Detecção de objetos com Faster R-CNN



Fonte: [sshikamaru \(2023\)](#)

Figura 38 – Detecção de objetos com YOLOv8s



Fonte: [sshikamaru \(2023\)](#)

Com base nos resultados do experimento, observou-se que o modelo YOLOv8n apresentou um excelente desempenho em termos de precisão em comparação aos demais, tornando-se uma escolha viável para aplicações em sistemas embarcados. Em comparação com os outros modelos avaliados, o YOLOv8n é mais compacto, possui menos parâmetros e tem uma velocidade de processamento superior, mantendo ao mesmo tempo uma alta precisão. Portanto, o YOLOv8n é uma opção interessante para aplicações em que os recursos computacionais são limitados.

5

Construção da Base de Dados

A base de dados desempenha um papel crucial na concepção de modelos de aprendizado profundo, já que o treinamento e a validação de um modelo são impossíveis sem uma base sólida. Portanto, uma quantidade significativa de tempo é dedicada à preparação desses dados antes de sua utilização no treinamento de um modelo. O processo abrange diversas etapas, como a criação da base, que envolve a aquisição de imagens, vídeos, textos ou qualquer outro tipo de dados relevantes para o contexto da base; a rotulação; o processamento dos dados; a engenharia de atributos; e outras etapas consideradas necessárias.

Com a ascensão da era da *big data*, que se caracteriza não apenas por volumes massivos de dados, mas também pela veracidade, velocidade e variedade dos mesmos (PARK; NGUYEN; WON, 2015), surge um conjunto de informações crucial, autêntico e gerado a uma alta taxa. Embora a disponibilidade de uma enorme quantidade de dados seja uma realidade, a armazenagem e o processamento desses dados para torná-los prontos para uso representam um desafio complexo, uma vez que essa etapa consome grande parte do tempo dedicado à construção do modelo.

No contexto do desenvolvimento de modelos de estimativa de velocidade por meio de aprendizado profundo, algumas bases de dados estão disponíveis digitalmente. Na Seção 5.1, serão mencionadas algumas dessas bases de dados, enquanto a Seção 5.2 abordará a criação da base de dados própria.

Embora existam outras bases de dados, conforme destacado na Seção 5.1, optou-se pelo desenvolvimento de uma base de dados própria, devido à inadequação das bases existentes em relação aos seguintes critérios:

- Diversidade de veículos: A base inclui amostras de diversas categorias de veículos, como carros, motos, ônibus e caminhões.
- Padrão de placas brasileiras: Os veículos apresentam placas no mesmo padrão das placas brasileiras, tornando a base adequada para treinamento futuro de sistemas de

Reconhecimento Automático de Placas de Veículos (ALPR).

- Contexto local: A base de dados abrange situações que refletem os desafios enfrentados nas vias locais, especificamente em Sergipe. Isso inclui veículos menos comuns em outras regiões, como micro-ônibus que percorrem a capital do estado até áreas mais remotas.

5.1 Outras bases de dados para estimativa de velocidade

5.1.1 Vs13 dataset

A base de dados denominada vs13 foi desenvolvida por Djukanović, Bulatović e Čavor (2022). Essa base contém registros de 400 áudio-vídeos de 13 veículos distintos circulando a diversas velocidades, variando de 30 km/h a 110 km/h. Entre os modelos de veículos presentes na base, estão o Citroen C4 Picasso, Kia Sportage, Mazda 3 Skayactive, Mercedes AMG 550, Mecedes GLA 200D, Nissan Qashqai, Opel Insignia, Peugeot 208, Peugeot 3008, Peugeot 307, Renault Captur, Renault Scenic e VW Passat B7.

As gravações dos vídeos foram capturadas utilizando uma câmera GoPro Hero5 com resolução full HD e taxa de quadros de 30 fps. Os registros foram feitos em Montenegro, nas proximidades da rodovia que conecta Podgorica e Petrovac. A localização exata das gravações na base de dados é ilustrada na Figura 39. Para identificar a velocidade dos veículos, essa informação é incluída no título de cada vídeo. Por exemplo, o nome do arquivo "CitroenC4Picasso_45.mp4" indica que a gravação corresponde ao veículo CitroenC4Picasso circulando a 45 km/h. Importante destacar que as gravações foram realizadas com os veículos mantendo uma velocidade constante, e essa velocidade foi registrada por meio de um sensor LiDAR.

Figura 39 – Amostra da base de dados Vs13



Fonte: Djukanović, Bulatović e Čavor (2022)

5.1.2 UTFPR Dataset

O conjunto de dados da UTFPR, desenvolvido por docentes da Universidade Tecnológica Federal do Paraná, é uma base brasileira amplamente reconhecida. Para a obtenção dos registros,

[Luvizon, Nassu e Minetto \(2016\)](#) posicionaram uma câmera a uma altura de 5,5 metros em relação ao solo, de modo a capturar os veículos em trânsito na rodovia. Com um limite de velocidade de 60 km/h estabelecido para a rodovia, a amostragem das velocidades variou entre 10 km/h e 70 km/h, devido a alguns veículos excederem esse limite.

As gravações foram efetuadas por uma câmera com resolução de 5 megapixels, resultando em imagens de 1920 x 1080 pixels e uma taxa de quadros de 30 fps. O conjunto de dados inclui um total de 20 vídeos, divididos em cinco conjuntos. O primeiro conjunto contém 1.146 amostras de veículos, o segundo possui 4.829 amostras, o terceiro contém 960 amostras, o quarto inclui 1.045 amostras e o quinto apresenta 869 amostras de veículos. Uma amostra da base de dados da UTFPR é ilustrado na [Figura 40](#).

Figura 40 – Amostra da base de dados UTFPR



Fonte: [Luvizon, Nassu e Minetto \(2016\)](#)

5.1.3 UA-DETRAC

A base de dados UA-DETRAC é composta por um conjunto de vídeos com duração total de 10 horas, capturados em 24 localizações distintas em Beijing e Tianjin, na China. A gravação foi realizada por meio de uma câmera Canon EOS 550D, apresentando uma resolução de 960x540 pixels e uma taxa de quadros de 25 fps. Esta base de dados foi desenvolvida no ano de 2015 por [Wen et al. \(2020\)](#).

O conjunto de dados contém aproximadamente 8.250 registros de veículos. Esses veículos foram classificados em quatro categorias: carro, ônibus, van e outras categorias. Além disso, a base de dados foi particionada em dois conjuntos: o conjunto de treinamento e o conjunto de teste. No conjunto de treinamento, aproximadamente 62,75% dos registros são de carros, 1,28% são de ônibus, 7,39% são de vans e 0,52% pertencem a categoria outras. A UA-DETRAC também foi categorizada com base em outras informações relevantes, como as condições climáticas. Por exemplo, 19% das amostras de treinamento foram capturadas em condições nubladas, 10% em condições chuvosas, 15% em condições ensolaradas e 16% durante a noite. Para visualizar melhor essa base de dados, a [Figura 41](#) apresenta uma amostra ilustrativa.

Figura 41 – Amostra da base de dados UA-DETRAC

Fonte: [Wen et al. \(2020\)](#)

5.2 Base de Dados VSE

Essa base de dados é de autoria própria, sendo desenvolvida em conjunto com os colaboradores Victor Gabriel Santos Santana e Víctor Kleverton Lima Barreto. As gravações foram feitas usando uma câmera Canon 1200d e um smartphone Motorola G30, com resoluções de 1280x720 e 640x360 pixels, e taxas de quadros de 50fps e 30fps, respectivamente. As localizações das gravações estão detalhadas no [Quadro 4](#).

Quadro 4 – Localização da gravação da base de dados

Referência	Latitude	Longitude
Orla da Atalia	-10.9890520	-37.0489450
Hospital Nestor Piva	-10.904604	-37.067967
173 Br-235	-10.905867559371748	-37.112554748123046

Fonte: Autor

A criação da base de dados foi desenvolvida com base em diversos requisitos fundamentais. Um dos principais objetivos foi adicionar uma ampla diversidade de veículos nos registros. Esse critério visou evitar que o modelo elaborado para a estimativa de velocidade fosse restrito a apenas um único tipo de veículo, como, por exemplo, somente carros. Além disso, a base de dados foi construída com a intenção de incorporar exemplos de veículos ausentes em algumas bases de dados, como micro-ônibus que transportam pessoas do interior de Sergipe à capital Aracaju.

Outro ponto relevante foi a utilização de mais de uma câmera e da diversidade dos ângulos de filmagem em relação aos veículos. Essa abordagem estratégica resultou em um modelo mais genérico, com a capacidade de se adaptar a múltiplas configurações de câmeras e ângulos em relação aos veículos. Essa flexibilidade contribuiu para aprimorar a aplicabilidade e a eficácia do modelo, independentemente das variações nas configurações de captura. A altura e a distância da câmera até a rodovia onde as gravações foram realizadas são reveladas no [Quadro 5](#).

Quadro 5 – Parâmetros da gravação

Local	Data da gravação	Altura(m)	distância(cm)
Orla da Atalia	07/08/20/23	0,81	62
Orla da Atalia	10/08/2023	1,31	110
Hospital Nestor Piva	07/08/2023	0,83	48
Hospital Nestor Piva (lado Hospital)	10/08/2023	1,28	123
Hosp. Nestor Piva (lado Escola Acrísio)	10/08/2023	1,24	32
173 Br-235	10/08/2023	1,30	77

Fonte: Autor

Para determinar a velocidade dos veículos, utilizou-se de sensores de pavimento localizados em áreas estrategicamente selecionadas. Essa abordagem estratégica permitiu ao modelo enfrentar desafios do mundo real, incluindo variações na velocidade devido à necessidade de frear para evitar colisões ou para permitir a passagem de pedestres. Portanto, mesmo diante dessas flutuações na velocidade, o modelo é capaz de prever a velocidade em que o veículo passou pelo trecho, ou seja, a velocidade final ao atravessar o ponto de estimativa de velocidade, de maneira análoga aos sensores convencionais. Essas situações não seriam capturadas em uma base de dados composta apenas por veículos trafegando em estradas vazias e mantendo velocidades constantes.

A velocidade máxima permitida nas áreas monitoradas é de 40 km/h. Consequentemente, a base de dados inclui velocidades entre 10 e 44 km/h, uma vez que alguns veículos ultrapassaram o limite de 40 km/h. Essa variação representa cenários reais de tráfego e contribui para a riqueza de informações na base de dados.

Para registrar as velocidades dos veículos, utilizou-se a gravação do visor de um dispositivo de fiscalização eletrônica, capturando as imagens com um smartphone Samsung A2 Core. Para simplificar a associação entre as velocidades e os veículos, realizou-se uma sincronização entre os vídeos dos veículos e o vídeo do visor. Isso foi feito com o auxílio do software de edição Adobe Premiere Pro¹, que permitiu sincronizar os vídeos e efetuar as edições necessárias, como a separação dos trechos de cada veículo. Além disso, o vídeo do visor exibia claramente o veículo em questão, confirmado assim tanto o veículo quanto sua velocidade correspondente. A Figura 42 ilustra um exemplo de como as velocidades foram extraídas dos vídeos.

A Figura Figura 42 ilustra um veículo da marca Jeep, registrado pelo sistema de fiscalização eletrônica, viajando a uma velocidade de 30 km/h. Para isolar cada veículo nas gravações, foi realizada uma edição cuidadosa que envolveu cortes e a aplicação de efeitos para remover os veículos para os quais as velocidades não foram registradas. Um desses efeitos, conhecido como redução de opacidade, consistiu em tornar os quadros dos outros veículos menos visíveis, destacando apenas o veículo de interesse.

¹ Disponível em: <<https://www.adobe.com/br/products/premiere.html>>. Acesso em agosto de 2023

Figura 42 – Visor dos sensores da fiscalização eletrônica



Fonte: Autor

Para alcançar isso, reduziu-se gradualmente a intensidade dos quadros dos outros veículos, criando um efeito de transparência. Isso foi feito por meio da manipulação da opção de opacidade no Adobe Premiere Pro. Além disso, adicionou-se uma camada atrás da seção da via onde não havia veículos para destacar ainda mais o veículo de interesse.

Esse processo meticoloso foi aplicado a todas as gravações e fez uso da função de rastreamento do Adobe Premiere Pro, que replicou a máscara criada para o veículo em questão em todos os quadros do vídeo onde o veículo estava presente.

Figura 43 – Trecho do vídeo sem efeito



Fonte: Autor

Figura 44 – Trecho do vídeo com efeito



Fonte: Autor

A fim de associar a velocidade específica a cada veículo, essa informação foi incorporada ao título de cada vídeo. Cada título inclui o tipo de veículo em inglês, seguido pela velocidade em km/h. Por exemplo, *car2_32* indica que se trata de um carro a 32 km/h, onde o número 2 indica a ordem de edição do vídeo, facilitando a identificação dos veículos. Outro exemplo na base de dados é *motorcycle23_17*, indicando a 23^a moto editada a 17 km/h.

Os veículos foram categorizados em *car*, *motorcycle*, *bus* e *truck*, correspondendo a carro, motocicleta, ônibus e caminhão, respectivamente. Algumas localizações foram alvo de gravações por mais de um dia. Para simplificar a rotulagem da base, as pastas que contêm os vídeos foram

denominadas com o nome do local e o dia da gravação. A [Tabela 14](#) apresenta a contagem de veículos para cada gravação.

Tabela 14 – Quantidade de veículos por gravação

Gravação	Carros	Motos	Ônibus	Caminhões	Total
orla_atalaia_1_10-08-23	24	1	0	1	26
orla_atalaia_2_10-08-23	9	0	0	0	9
orla_atalaia_3_10-08-23	25	6	1	0	32
orla_atalaia_07-08-23	67	6	2	5	80
nestor_piva_1_10-08-23	16	7	2	0	25
nestor_piva_2_10-08-23	16	9	0	0	25
nestor_piva_3_10-08-23	17	7	1	0	25
nestor_piva_07-08-23	18	6	1	2	27
br235_km173_1_12-08-23	35	8	2	5	50
br235_km173_2_12-08-23	32	13	2	6	53
br235_km173_3_12-08-23	37	4	2	7	50
Total	296	67	13	26	402

Fonte: Autor

O nome da pasta de gravação segue um formato específico: começa com o nome do local, seguido por um número que indica a sequência da gravação (primeira, segunda, terceira, etc.), e finaliza com a data da gravação. Por exemplo, orla_atalaia_1_10-08-23 indica que a gravação foi realizada na Orla da Atalaia no dia 10 de agosto de 2023, sendo a primeira gravação daquele dia. A [Figura 45](#) apresenta alguns exemplos ilustrativos da base de dados.

Figura 45 – Amostra da base de dados VSE



Fonte: Autor

A coleta de dados foi realizada tanto no período matutino quanto no vespertino, abrangendo condições climáticas que variaram entre ensolaradas e nubladas. Essa abordagem visou abranger situações distintas. A velocidade dos 402 veículos registrados na base de dados é apresentada na [Tabela 15](#).

Tabela 15 – Velocidade dos Veículos

Velocidade(km/h)	Quantidade de Veículos
10	1
12	1
13	2
14	1
15	2
16	3
17	3
18	1
19	1
20	5
21	9
22	7
23	5
24	19
25	25
26	14
27	18
28	33
29	34
30	33
31	31
32	41
33	32
34	19
35	19
36	20
37	7
38	5
39	7
40	1
42	1
43	1
44	1

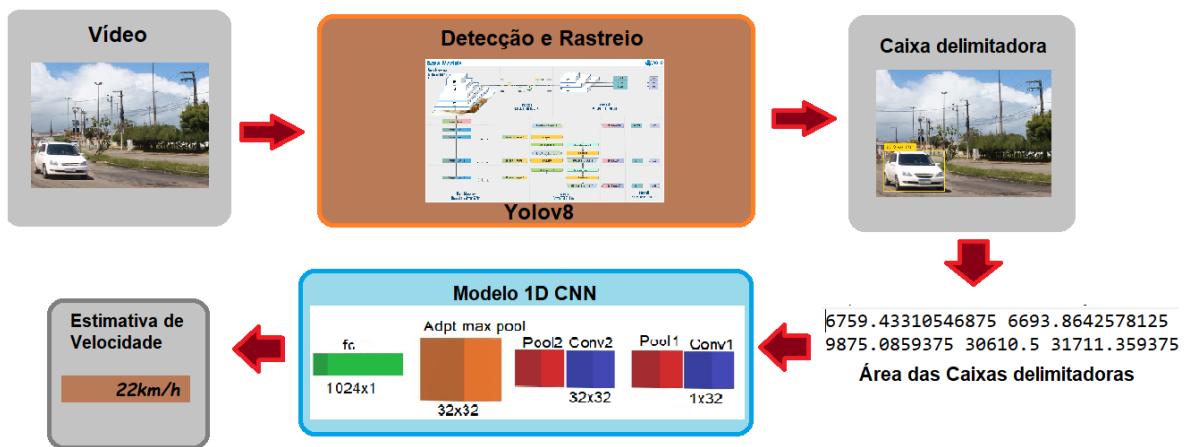
Fonte: Autor

6

Construção do Modelo

Com o intuito de desenvolver um modelo de estimativa de velocidade que possa contribuir para a gestão do tráfego de veículos, seja em rodovias, condomínios ou campos universitários, visando a redução de acidentes, foram seguidas algumas etapas. O processo, detalhado na Figura 46, ilustra o fluxo de trabalho deste projeto, que busca estimar a velocidade de forma eficaz.

Figura 46 – Fluxo de Trabalho do Projeto



Fonte: Autor

Conforme ilustrado na Figura 46, as imagens das cenas veiculares são processadas por um modelo de detecção e rastreamento de veículos, especificamente o YOLOv8, que será detalhadamente abordado na Seção 6.1. Após essa etapa, o YOLOv8 fornece as caixas delimitadoras, conhecidas como *bounding boxes*, que contêm os veículos. Em seguida, é calculada a área de cada *bounding box*. O conjunto de áreas correspondente a um veículo específico é então fornecido ao modelo 1D-CNN. Cada valor neste conjunto representa a área de uma *bounding box*.

do veículo em um determinado quadro, conforme discutido em detalhes na Seção 6.2. O modelo 1D-CNN utiliza essas áreas para estimar a velocidade do veículo em quilômetros por hora (km/h).

6.1 Modelo de Detecção e Rastreio dos Veículos

O experimento realizado, conforme descrito no Capítulo 4, teve como objetivo determinar o modelo mais adequado para a detecção de objetos, uma vez que a revisão da literatura acadêmica destacou que o Faster R-CNN e o YOLO são os modelos de referência nessa área. No projeto, após os experimentos, optou-se pelo uso do YOLOv8.

O YOLOv8 Nano foi escolhido dentre as variantes disponíveis do YOLOv8, devido ao seu tamanho ser menor que os demais, tornando-o mais rápido, e ainda alcançando uma alta precisão. Uma característica que o tornou uma opção ideal foi a adição, pela Ultralytics, da capacidade de rastreamento de objetos a essa versão. Isso teve o benefício de reduzir a necessidade de incluir outro modelo no projeto, uma vez que o rastreamento poderia ser realizado pelo mesmo modelo de detecção.

O rastreamento de objetos é uma tarefa crucial para associar um objeto identificado em cenas diferentes em um vídeo, fornecendo informações sobre a localização contínua desse objeto. Enquanto a detecção se limita a identificar a presença do objeto em quadros individuais, o rastreamento permite estabelecer uma conexão entre esses quadros, facilitando a categorização das *bounding boxes* de cada veículo ao longo de todo o vídeo.

Para uma compreensão mais clara desse processo, a Figura 47 ilustra um exemplo de rastreamento de objetos. Cada veículo é atribuído a um número de identificação exclusivo, que permanece constante em todos os quadros em que o veículo aparece. Por exemplo, um veículo numerado como 1 manterá essa mesma identificação em todos os quadros, o que facilita significativamente a categorização dos veículos ao longo do vídeo.

Figura 47 – Rastreio de Veículos



Fonte: Autor

Cada *bounding box* tem sua área calculada, e essas áreas são inseridas em uma lista específica para cada veículo. Essas listas de áreas são posteriormente utilizadas como entrada para o modelo de estimativa de velocidade. O cálculo das áreas é realizado com base nos pontos

retornados pelo YOLOv8 para cada *bounding box*. O modelo fornece os pontos correspondentes ao canto superior esquerdo e ao canto inferior direito de cada *bounding box*.

6.2 Modelo de Estimativa de Velocidade

Com o objetivo de criar um modelo de estimativa de velocidade que não dependesse de informações da câmera ou de pontos geográficos específicos, como é o caso dos modelos que requerem o uso de matrizes de homografia, optou-se por adotar a abordagem proposta no trabalho de [Cvijetić, Djukanović e Peruničić \(2023\)](#). Nesse trabalho, foi desenvolvido um modelo de *deep learning* denominado 1D-CNN, que estima a velocidade com base nas áreas das *bounding boxes* dos veículos. Como demonstrado na [Figura 32](#). O qual eles demonstraram que quanto maior a velocidade do veículo, mais acentuada é a curva da áreas das *bounding boxes*, fazendo com que o veículo apareça em menos quadros do vídeo.

Para solucionar problemas de detecção de veículos em determinados *frames*, onde o veículo pode ser detectado em alguns *frames*, como nos *frames* 1, 2 e 3, mas não no 4, e depois novamente no *frame* 5, foi empregada a técnica de interpolação linear. Essa técnica preenche as áreas das *bounding boxes* em *frames* ausentes. A fórmula utilizada é a seguinte:

$$\text{area}(x) = \text{area}_z + (\text{area}_y - \text{area}_z) \times \left(\frac{x - z}{y - z} \right) \quad (6.1)$$

Nesta fórmula, $\text{área}(x)$ representa a área desejada para o *frame* x , enquanto y é um *frame* posterior ao qual se deseja encontrar a área. Por exemplo, se x for 2, y pode ser 3 ou posterior. Por outro lado, z é um *frame* anterior a x .

O modelo proposto foi desenvolvido com o intuito de apresentar um alto desempenho em situações do mundo real, como aquelas encontradas em rodovias e vias urbanas com alto tráfego. Isso inclui a capacidade de lidar com a variabilidade na velocidade dos veículos, congestionamentos e outros desafios típicos desse ambiente. Por outro lado, o modelo desenvolvido por [Cvijetić, Djukanović e Peruničić \(2023\)](#) foi testado exclusivamente com a base de dados vs13, a qual eles criaram. Essa base de dados contém veículos com velocidades constantes, o que não oferece informações sobre como o modelo se comportaria diante das flutuações de velocidade comuns em rodovias, por exemplo. Além disso, o modelo desenvolvido foi treinado com uma base de dados diversificada, incluindo uma ampla variedade de carros, motos, ônibus e caminhões. Em comparação, o outro modelo foi treinado e testado usando um conjunto de dados limitado, composto por apenas 12 carros distintos.

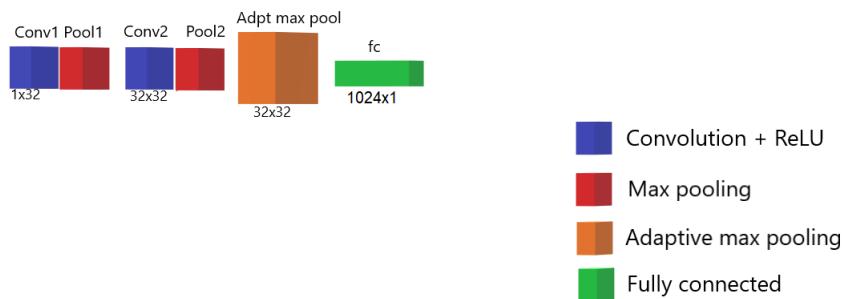
Outras distinções entre os modelos incluem o número de canais e o tamanho do *kernel* das camadas convolucionais. No modelo proposto, o número de canais foi reduzido pela metade, resultando em uma diminuição nos parâmetros do modelo para melhorar a velocidade de inferência e reduzir os requisitos computacionais. Além disso, a redução do tamanho do *kernel* permite ao modelo lidar com menos áreas de interesse para realizar a estimativa de velocidade. Isso significa

que o modelo não precisa analisar tantas regiões da imagem para calcular a velocidade, o que é especialmente benéfico em locais com restrições de espaço, como nas proximidades de sistemas de fiscalização eletrônica em Aracaju. Em tais situações, um veículo normalmente leva cerca de 3 segundos para percorrer a distância do sensor ao visor da fiscalização, conforme demonstrado na base de dados desenvolvida neste projeto.

Por outro lado, o modelo de Cvjetić, Djukanović e Peruničić (2023) não seria adequado para prever velocidades nesses locais, pois o número de áreas disponíveis não é suficiente devido ao tamanho do *kernel*. No entanto, é importante observar que o modelo proposto apresenta uma limitação na variedade de velocidades em que foi testado, uma vez que a base desenvolvida inclui veículos com velocidades de até 40 km/h, enquanto o modelo vs13 abrange velocidades de até 110 km/h.

A Figura 48 exibe a arquitetura do modelo proposto. Sendo duas camadas convolucionais com função de ativação ReLU e seguidas de uma camada de *pooling* para realização da extração de características. Após isso, uma outra camada de *pooling* e por último uma camada densa para realizar a estimativa da velocidade.

Figura 48 – Arquitetura do Modelo 1D-CNN



Fonte: Autor

A primeira camada convolucional é composta por 1 canal de entrada, 32 canais de saída e um *kernel* de tamanho 3. Além disso, foi utilizada a técnica de *padding*¹, já que as informações das bordas são cruciais para acompanhar a variação da área da *bounding box*.

A diferença entre a primeira e a segunda camada convolucional consiste no tamanho do canal de entrada. A primeira camada possui um canal de entrada de tamanho 1, enquanto a segunda possui um canal de entrada de tamanho 32, pois a primeira camada produz 32 mapas de características como saída, que são usados como entrada para a segunda camada.

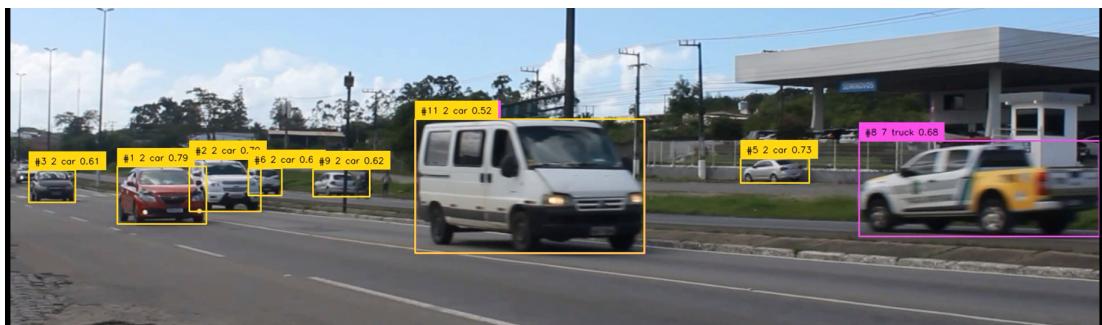
¹ Padding é uma técnica que envolve a adição de valores, geralmente zeros, às bordas dos dados para evitar a redução do tamanho dos dados e a perda de informações nas bordas durante o processo de convolução

Ambas as camadas de *max pooling* têm um tamanho de *kernel* de 2. Por outro lado, a camada de *Adaptive max pooling* é usada para igualar o tamanho dos conjuntos de características dos veículos, uma vez que a camada densa requer que todos os conjuntos de características tenham o mesmo tamanho como entrada. A camada densa recebe como entrada um vetor composto por todos os mapas de características e produz a estimativa de velocidade do veículo.

Durante o treinamento, empregou-se a função de perda *Root Mean Square Error* (RMSE) para calcular o erro entre a estimativa de velocidade e a velocidade real. Esse erro foi usado para atualizar os pesos do modelo. A função de otimização escolhida para realizar essa atualização foi a Adam, reconhecida por sua rápida convergência para mínimos locais e pela sua capacidade de adaptação sem a necessidade de ajustes complexos de hiperparâmetros. Além disso, foi aplicada a técnica de decaimento de peso, uma forma de regularização que ajuda a prevenir o *overfitting* do modelo. Em outras palavras, ela evita que o modelo simplesmente memorize os dados de treinamento em vez de aprender com eles. Essa regularização é implementada adicionando um valor de regularização λ à função de custo.

Este modelo foi desenvolvido para operar de forma eficiente com múltiplos veículos. Isso significa que, quer haja apenas um veículo ou vários na área de visão da câmera, o modelo é capaz de calcular as velocidades de todos eles. Essa capacidade o torna uma escolha eficaz para resolver desafios relacionados à estimativa de velocidade. A [Figura 49](#) apresenta um exemplo de abordagem com vários veículos.

Figura 49 – Detecção de multi-veículos



Fonte: Autor

6.3 Experimentos e Resultados

Os experimentos para desenvolver o modelo foram conduzidos na plataforma Kaggle, um ambiente que permite aos usuários participar de competições de ciência de dados, disponibilizar conjuntos de dados e criar códigos usando recursos oferecidos gratuitamente, com um limite de 30 horas por semana de uso de GPU.

O ambiente de experimentação foi equipado com duas GPUs T4, cada uma com 14,8 GB de RAM, além de uma memória RAM de 13 GB e 73,1 GB de armazenamento disponível. Durante todo o processo, foi utilizada a linguagem de programação Python, juntamente com as ferramentas do PyTorch.

A base de dados, que é detalhada na Seção 5.2, desempenhou um papel essencial no treinamento e na validação do nosso modelo. Para a avaliação do modelo, reservamos os veículos registrados na gravação denominada 'br235_km173_1_12-08-23', com o objetivo de evitar o uso de cenários semelhantes aos já utilizados no treinamento, garantindo assim que as amostras de teste não fossem memorizadas pelo modelo. Enquanto isso, os veículos restantes foram divididos, com 70% alocados para o treinamento e os 30% restantes para a validação.

Durante os experimentos, foram feitas várias modificações na estrutura do modelo de estimativa de velocidade para melhorar o desempenho. O modelo final é detalhado na Seção 6.2. Neste capítulo, serão descritas as alterações feitas até chegar a esse modelo e os resultados de cada teste.

O modelo base utilizado foi proposto por Cvijetić, Djukanović e Peruničić (2023), o qual foi explicado na Subseção 3.1.4 e possui a seguinte estrutura:

- 1^a Camada Convolucional: 1 canal de entrada, 64 canais de saída e um *kernel* de tamanho 10;
- 1^a Camada de Pooling: *kernel* de tamanho 2;
- 2^a Camada Convolucional: 64 canais de entrada, 64 canais de saída e um *kernel* de tamanho 10;
- 2^a Camada de Pooling: *kernel* de tamanho 2;
- Camada de *Adaptive Max Pooling*: Esta camada tem como objetivo uniformizar o tamanho de todos os mapas de características, estabelecendo um tamanho de 32x32.
- Camada Totalmente Conectada: Também conhecida como camada densa, esta camada é responsável por prever a velocidade. Ela recebe uma entrada de 1024, que resulta da camada *Adaptive Max Pooling*. A camada *Adaptive Max Pooling* produz 32 mapas de características, cada um com 32 valores, resultando em uma matriz de dimensões 32x32. Essa matriz é então transformada em um tensor unidimensional de 1024, e a saída da camada é um único valor, que representa a estimativa da velocidade do veículo.

No treinamento do modelo, empregou-se a função de perda MSELoss e a técnica de otimização Adam. Para avaliar o desempenho dos modelos, utilizou-se a métrica *Root Mean Square Error* uma das métricas mais empregadas nos experimentos dos artigos acadêmicos utilizados na revisão sistemática.

Ao aplicar o modelo mencionado no trabalho de Cvjetić, Djukanović e Peruničić (2023) à base de dados desenvolvida neste projeto, não foi obtido sucesso, não foi obtido sucesso, pois o tamanho do *kernel* era muito grande para realizar o processo de convolução nos dados da base. Como resultado, foi necessário realizar modificações no modelo. As alterações feitas durante os testes do modelo incluíram:

1º Experimento: A primeira alteração consistiu na redução do tamanho do *kernel* nas duas camadas de convolução, passando de 10 para 3. Essa adaptação viabilizou a execução do processo de convolução nos dados. A segunda modificação envolveu a adição de uma camada de *Adaptive Max Pooling*, que permitiu lidar com tamanhos variados de dados, eliminando a necessidade de padronizar o tamanho dos dados de entrada. Isso resultou na preservação das informações, contribuindo para o aprimoramento do desempenho do modelo. Anteriormente, ao padronizar os dados, partes dos dados de maior tamanho eram sacrificadas para que coincidissem com o tamanho dos dados menores. Esse experimento foi conduzido ao longo de 150 épocas, e o modelo registrou um RMSE de 3,09 km/h;

2º Experimento: Nesse experimento, a única alteração realizada foi no número de épocas de treinamento, que saiu de 150 para 1.000, uma vez que o modelo tinha potencial para aprender mais com um treinamento mais extenso. Essa modificação teve impacto no resultado do RMSE, que diminuiu para 2,83 km/h;

3º Experimento: Foi adicionado o parâmetro *weight_decay* ao otimizador, com um valor de 0,001. Essa mudança foi implementada porque o modelo estava reduzindo a perda nos dados de teste, mas não estava apresentando melhorias nos dados de validação. O *weight_decay* é um parâmetro de regularização projetado para evitar o *overfitting*, também conhecido como decaimento de peso. Neste teste, o RMSE foi reduzido para 2,63 km/h;

4º Experimento: Foi adicionado o parâmetro de *padding* nas camadas convolucionais, garantindo que os dados de saída tenham o mesmo tamanho que os de entrada. Isso evitou a perda de informações, especialmente nas bordas, que são cruciais para analisar a variação da área das caixas delimitadoras. Neste experimento, o resultado alcançado foi um RMSE de 2,14 km/h;

5º Experimento: No quinto experimento, a quantidade de canais nas camadas convolucionais foi reduzida de 64 para 32, com o objetivo de diminuir a quantidade de parâmetros e tornar o modelo mais leve. A intenção era verificar se, com um número menor de mapas de características, o modelo ainda conseguiria obter resultados comparáveis. Nesse teste, o resultado alcançado foi um RMSE de 2,35 km/h, aproximando-se do resultado anterior de 2,14 km/h;

6º Experimento: A segunda camada convolucional foi removida, e o número de épocas foi aumentado para 3.000 neste experimento. O objetivo era investigar o desempenho do

modelo com apenas uma camada convolucional. No entanto, o resultado mostrou um RMSE de 3,29 km/h, indicando uma redução no desempenho em comparação com o modelo de duas camadas convolucionais, o que era esperado devido à remoção de uma camada;

7º Experimento: Reintroduziu a segunda camada convolucional e adicionou-se uma terceira camada convolucional com o objetivo de avaliar o desempenho comparativo entre modelos com 2 e 3 camadas convolucionais. Manteve-se o mesmo número de épocas do sexto experimento, ou seja, 4.000 épocas. No entanto, durante o treinamento, o modelo enfrentou um problema de dissipação de gradientes, onde os valores dos gradientes diminuíram progressivamente, afetando o desempenho. Isso ocorre porque as atualizações dos pesos das camadas iniciais se tornam cada vez menores. No final, o RMSE obtido neste experimento foi de 3,10 km/h;

8º Experimento: Nesse experimento, a função de otimização foi trocada para o RMSprop, uma vez que essa função incorpora um novo parâmetro projetado para abordar problemas de dissipação de gradiente. O resultado obtido foi um RMSE de 2,03 km/h;

9º Experimento: O modelo empregado corresponde ao descrito na Seção 6.2, com a configuração de duas camadas convolucionais e a função de otimização Adam. O treinamento do modelo foi conduzido ao longo de 4.000 épocas, no entanto, após cerca de 3.000 épocas, não observou-se melhorias significativas no desempenho. Nesse contexto, o RMSE alcançado foi de 1,67 km/h.

Com o objetivo de facilitar a comparação dos resultados, o [Quadro 6](#) apresenta as conclusões obtidas de todos os experimentos delineados, enquanto o [Quadro 7](#) exibe as velocidades reais e previstas pelo modelo final para determinados veículos usados nos testes.

Quadro 6 – Resultados obtidos nos experimentos de estimativa de velocidade

Experimento	RMSE (km/h)
1º	3,09
2º	2,83
3º	2,63
4º	2,14
5º	2,35
6º	3,29
7º	3,10
8º	2,03
9º	1,67

Fonte: Autor

Os resultados obtidos demonstram que o modelo se aproximou dos valores obtidos pelo sensor utilizado para estimativa de velocidade nas rodovias, com um erro médio de 1,67 km/h. É

Quadro 7 – Comparaçāo entre a velocidade real e a estimada pelo modelo desenvolvido

Veículo	Velocidade Real (km/h)	Velocidade Estimada (km/h)	RMSE (km/h)
motorcycle44	30	31	1
truck11	26	26	0
car197	25	30	5
truck8	33	34	1
car206	29	30	1
car199	29	31	2
car183	35	35	0
car195	31	30	1
car192	30	30	0
car189	28	29	1
car207	32	33	1
car175	29	29	0
truck7	31	26	5
car208	30	34	4
car187	33	32	1
car196	43	40	3
car180	33	33	0
motorcycle38	31	31	0
motorcycle37	31	30	1
bus7	29	29	0

Fonte: Autor

relevante observar que, de acordo com a [Tech \(2023\)](#), empresa responsável pelo desenvolvimento das pistolas utilizadas por policiais para estimativa de velocidade, o erro médio dessas pistolas é de 2 km/h. Portanto, o erro médio do modelo desenvolvido neste trabalho está próximo ao erro médio da pistola. No entanto, é importante ressaltar que as pistolas de velocidade têm a capacidade de inferir a velocidade de um veículo até 320 km/h, enquanto o modelo só é eficaz até 40 km/h, considerando as limitações da base de dados utilizada para treinamento.

Além disso, é crucial mencionar que os sensores atuais permitem que os veículos, em zonas de fiscalização eletrônica de 40 km/h, ultrapassem essa velocidade até 47 km/h. Essa margem adicional não se deve ao erro médio dos sensores, mas sim a questões legislativas. Isso ocorre porque, conforme estabelecido pela [Aracaju \(2023\)](#), os veículos só são multados se excederem a velocidade permitida em 10% ou 7 km/h a mais, com o valor mais alto sendo aplicado. Portanto, para a fiscalização de 40 km/h, o limite de multa é de 47 km/h, uma vez que 10% de 40 km/h equivalem a 4 km/h, resultando em um limite superior de 47 km/h.

A inclusão de mais dados, ou seja, mais veículos e suas respectivas velocidades durante o treinamento, poderia reduzir ainda mais esse erro. Com 2/3 da base de dados, o modelo atingiu um RMSE de 5,35 km/h, enquanto com a base completa, conseguiu reduzir para 1,67 km/h.

Além disso, é fundamental destacar algumas desvantagens dos sistemas tradicionais de

estimativa de velocidade utilizados nas vias. Por exemplo, durante as gravações realizadas em um sensor próximo ao Hospital Nestor Piva, tornou-se evidente que esses sensores tradicionais podem apresentar problemas de localização imprecisa. Dependendo de sua instalação, eles podem não cobrir completamente a área que deveriam monitorar. No caso específico do sensor do Nestor Piva, os veículos que transitavam próximos ao meio-fio não tiveram suas velocidades estimadas, e quando vários veículos passavam simultaneamente, o sensor não conseguia estimar a velocidade de ambos.

Outra desvantagem notável é a necessidade de usar dois sensores em uma rodovia, um para cada metade da pista. Enquanto isso, um único sistema de visão computacional baseado em aprendizado profundo, posicionado estrategicamente para abranger toda a via, poderia substituir esses sensores duplicados. Em relação aos custos de implantação, o valor pode variar dependendo do tipo de sensor, da empresa que ganhou a licitação do projeto e outros fatores. No entanto, em comparação com os equipamentos já utilizados, os sistemas atuais já empregam servidores para processamento de informações, incluindo modelos de reconhecimento automático de placas veiculares. Portanto, esses servidores poderiam ser usados para processar o modelo de estimativa de velocidade, ou, caso seja realizada a compressão do modelo, ele pode ser embarcado, reduzindo os custos. Além desses custos, também existe a manutenção da câmera que gravaria o local em tempo real, que também faz parte dos sistemas atuais para a extração de placas dos veículos.

Em relação às desvantagens do modelo em comparação com os sistemas tradicionais de estimativa de velocidade, é fundamental observar que o modelo foi treinado exclusivamente com dados do período matutino e vespertino. Portanto, sua eficácia durante a noite é limitada, exigindo um novo treinamento com dados noturnos para melhorar seu desempenho nesse período. Além disso, o modelo não foi testado em condições de chuva, e sua capacidade de desempenho nessas condições permanece desconhecida. Outra limitação é que o modelo foi treinado apenas com velocidades de até 40 km/h, enquanto os sistemas tradicionais têm a capacidade de lidar com uma variedade maior de velocidades e condições climáticas. Além disso, é importante ressaltar que a câmera utilizada no sistema de visão computacional precisa de manutenção periódica. Por exemplo, se a câmera ficar embaçada, a inferência da velocidade dos veículos pode ser prejudicada.

Comparando o modelo proposto com o modelo desenvolvido por [Cvijetić, Djukanović e Peruničić \(2023\)](#), nota-se diferenças significativas em vários aspectos. O modelo proposto foi treinado e testado usando uma base de dados coletada em condições de tráfego real, ou seja, em vias públicas durante situações normais de tráfego. Essa base de dados abrange uma variedade de tipos de veículos, incluindo carros, motos, ônibus e caminhões. Por outro lado, o modelo de [Cvijetić, Djukanović e Peruničić \(2023\)](#) foi treinado e testado usando uma base de dados que consistia apenas nos veículos dos próprios autores, o que não representa adequadamente situações reais de tráfego e se limita à categoria de carros.

Em termos de desempenho, o modelo proposto atingiu um RMSE de 1,67 km/h, enquanto

o modelo de Cvjetić, Djukanović e Peruničić (2023) obteve um RMSE de 2,76 km/h. No entanto, é importante notar que o modelo de Cvjetić, Djukanović e Peruničić (2023) foi treinado com uma base de dados que incluía velocidades no intervalo de 0 a 100 km/h, abrangendo um espectro mais amplo de velocidades. Em contraste, o modelo proposto foi treinado e validado com uma base de dados contendo amostras de velocidades de até 40 km/h. Portanto, essa diferença nas faixas de velocidade pode influenciar as métricas de desempenho dos modelos.

Para concluir, é importante observar que o modelo desenvolvido não foi implantado e testado em um ambiente real. Todos os experimentos e testes foram conduzidos no ambiente controlado do Kaggle, juntamente com as gravações da base de dados. Embora tenha-se alcançado resultados promissores nos testes realizados, a implantação e teste em um ambiente de tráfego real seriam necessários para validar totalmente o desempenho do modelo em cenários do mundo real.

7

Conclusão

Neste trabalho, foi desenvolvido um modelo de *deep learning* para a estimativa de velocidade de veículos. Para isso, utilizou-se o YOLOv8 para a detecção e rastreamento de veículos, juntamente com um modelo 1D-CNN para a estimativa de velocidade com base na variação das áreas das *bounding boxes*.

Os experimentos tinham como objetivo avaliar o desempenho do modelo proposto, utilizando a métrica RMSE para mensurar a diferença entre os valores reais das velocidades dos veículos e as previsões feitas pela rede. O modelo alcançou um RMSE de 1,63 km/h, o que indica um erro médio de 1,63 km/h entre os valores previstos e as velocidades reais dos veículos. Esses resultados são considerados satisfatórios, uma vez que estão dentro da margem de tolerância comum em tecnologias de medição de velocidade nas rodovias, como as pistolas utilizadas por policiais, que apresentam um erro médio de 2 km/h.

Durante a condução deste trabalho, foram enfrentados desafios consideráveis na criação da base de dados. As fiscalizações eletrônicas nas vias comumente exibem apenas limites de velocidade de até 40 km/h em seus painéis, e não houve acesso a um sistema LiDAR para a coleta de dados, o que resultou na limitação da base em relação às velocidades. Devido a esses fatores, o modelo foi treinado e testado apenas até a velocidade máxima de 40 km/h.

Além desses desafios, também enfrentou-se dificuldades técnicas. Por exemplo, durante o treinamento do modelo 1D-CNN com um grande número de épocas, a interface do Kaggle frequentemente deixava de responder. Isso resultava na perda do progresso do treinamento, o que complicava ainda mais a realização de experimentos para otimizar os hiperparâmetros do modelo.

Ademais, é importante ressaltar que este trabalho contribui para a área de estimativa de velocidade. Ele apresenta uma abordagem de fácil implementação, não se limitando apenas a vias, mas também podendo ser aplicada em diversas outras situações, como universidades e condomínios, onde o fluxo de veículos frequentemente não respeita os limites de velocidade

estabelecidos. A flexibilidade desse modelo o torna uma ferramenta valiosa para melhorar a segurança viária em uma variedade de cenários, potencialmente reduzindo acidentes e incentivando o cumprimento das velocidades máximas permitidas em diferentes contextos. Isso demonstra o potencial impacto positivo deste trabalho no campo da segurança no trânsito e na aplicação prática do controle de velocidade.

Para futuros trabalhos, algumas áreas de pesquisa podem ser exploradas:

1. **Aprimoramento da base de dados:** Aumentar a quantidade de veículos na base de dados, incluindo aqueles com velocidades superiores a 40 km/h, a fim de avaliar o desempenho do modelo em velocidades mais altas. Além disso, adicionar amostras em diferentes condições climáticas, como chuva, para verificar a eficácia do modelo sob diferentes circunstâncias. O aumento no tamanho da base de dados também pode ajudar a reduzir ainda mais o RMSE;
2. **Criação de um modelo audiovisual:** Além dos dados visuais, como as áreas das *bounding boxes*, é possível considerar o áudio dos veículos durante a estimativa de velocidade, uma vez que o som emitido pelo veículo está relacionado à sua velocidade. A incorporação dessa variável pode contribuir para a redução do erro do modelo e, consequentemente, aumentar sua precisão;
3. **Compressão do modelo:** Explorar técnicas de redução de parâmetros do modelo, como poda, quantização ou destilação de conhecimento, para torná-lo mais leve e, assim, demandar menos recursos computacionais. Isso possibilitaria a execução do modelo em dispositivos com capacidade computacional limitada, como o Jetson Nano;
4. **Integração de um sistema ALPR:** A adição de um sistema de Reconhecimento Automático de Placas de Veículos (ALPR) permite que o modelo identifique as placas dos veículos, o que pode ser usado para identificar e punir os veículos que excedem os limites de velocidade permitidos, garantindo uma fiscalização mais eficaz.

Referências

- ABDELGAWAD, N. E. A. et al. Estimating vehicle speed on highway roads from smartphone sensors using deep learning models. In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2019. p. 979–986. Citado na página 63.
- ABUSALIM, S. et al. Data augmentation on intra-oral images using image manipulation techniques. In: *2022 International Conference on Digital Transformation and Intelligence (ICDI)*. [S.l.: s.n.], 2022. p. 117–120. Citado na página 29.
- AN, F. et al. Group randaugment: Video augmentation for action recognition. In: *2022 5th International Conference on Data Science and Information Technology (DSIT)*. [S.l.: s.n.], 2022. p. 1–5. Citado 2 vezes nas páginas 29 e 30.
- ARACAJU, P. de. *Fiscalização Eletrônica*. 2023. Disponível em: <<https://www.aracaju.se.gov.br/index.php?act=leitura&codigo=30354>>. Acesso em: setembro de 2023. Citado na página 85.
- ASHRAF, M. H. et al. Hvd-net: A hybrid vehicle detection network for vision-based vehicle tracking and speed estimation. *Journal of King Saud University - Computer and Information Sciences*, v. 35, n. 8, p. 101657, 2023. ISSN 1319-1578. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1319157823002112>>. Citado na página 63.
- ASIF, S. et al. A deep learning-based framework for detecting covid-19 patients using chest x-rays. *Multimedia Systems*, Springer, v. 28, n. 4, p. 1495–1513, 2022. Citado 2 vezes nas páginas 31 e 32.
- BELL, D.; XIAO, W.; JAMES, P. Accurate vehicle speed estimation from monocular camera footage. In: . [s.n.], 2020. v. 5, n. 2, p. 419 – 426. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091117109&doi=10.5194%2fisprs-annals-V-2-2020-419-2020&partnerID=40&md5=a88098882851cd782a1b542018a2a5fb>>. Citado na página 64.
- BHARDWAJ, R. et al. Autocalib: Automatic traffic camera calibration at scale. *ACM Trans. Sen. Netw.*, Association for Computing Machinery, New York, NY, USA, v. 14, n. 3–4, nov 2018. ISSN 1550-4859. Disponível em: <<https://doi-org.ez20.periodicos.capes.gov.br/10.1145/3199667>>. Citado na página 63.
- CAMPOS, C. A. T. *Comportamento das componentes de sistemas multi-toque baseados em reflexão interna total confinada*. Tese (Doutorado) — PUC-Rio, 2009. Citado 2 vezes nas páginas 44 e 45.
- CARRATÙ, M. et al. Development of a new speed measurement technique based on deep learning. In: *2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. [S.l.: s.n.], 2022. p. 1–6. Citado na página 63.
- CHEN, S.; LIN, W. Embedded system real-time vehicle detection based on improved yolo network. In: *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. [S.l.: s.n.], 2019. p. 1400–1403. Citado 2 vezes nas páginas 20 e 27.

- CHOI, P.; KWAK, J. A survey on mobile edge computing for deep learning. In: *2023 International Conference on Information Networking (ICOIN)*. [S.l.: s.n.], 2023. p. 652–655. Citado na página 16.
- CVIJETIĆ, A.; DJUKANOVIĆ, S.; PERUNIČIĆ, A. Deep learning-based vehicle speed estimation using the yolo detector and 1d-cnn. In: *2023 27th International Conference on Information Technology (IT)*. [S.l.: s.n.], 2023. p. 1–4. Citado 13 vezes nas páginas 7, 43, 52, 53, 56, 59, 63, 79, 80, 82, 83, 86 e 87.
- DJUKANOVIĆ, S.; BULATOVIĆ, N.; CAVOR, I. A dataset for audio-video based vehicle speed estimation. In: *2022 30th Telecommunications Forum (TELFOR)*. [S.l.: s.n.], 2022. p. 1–4. Citado na página 70.
- DO, T.-H. et al. A novel algorithm for estimating fast-moving vehicle speed in intelligent transport systems. In: *2021 International Conference on Information Networking (ICOIN)*. [S.l.: s.n.], 2021. p. 499–503. Citado na página 63.
- DONG, H.; WEN, M.; YANG, Z. Vehicle speed estimation based on 3d convnets and non-local blocks. *Future Internet*, MDPI, v. 11, n. 6, p. 123, 2019. Citado 4 vezes nas páginas 7, 53, 54 e 63.
- FAN, Q.; BROWN, L.; SMITH, J. A closer look at faster r-cnn for vehicle detection. In: *IEEE. 2016 IEEE intelligent vehicles symposium (IV)*. [S.l.], 2016. p. 124–129. Citado 3 vezes nas páginas 37, 38 e 39.
- G, E. R. et al. Mnist handwritten digit recognition using machine learning. In: *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. [S.l.: s.n.], 2022. p. 768–772. Citado na página 35.
- GAUTAM, S.; MATHURIA, T.; MEENA, S. Image segmentation for self-driving car. In: *2022 2nd International Conference on Intelligent Technologies (CONIT)*. [S.l.: s.n.], 2022. p. 1–6. Citado 2 vezes nas páginas 36 e 37.
- GIANNAKERIS, P. et al. Speed estimation and abnormality detection from surveillance cameras. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. [S.l.: s.n.], 2018. p. 93–936. Citado na página 63.
- GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1440–1448. Citado 2 vezes nas páginas 39 e 40.
- GOYZUETA, C. A. R.; CRUZ, J. E. C. De la; MACHACA, W. A. M. Integration of u-net, resu-net and deeplab architectures with intersection over union metric for cells nuclei image segmentation. In: *2021 IEEE Engineering International Research Conference (EIRCON)*. [S.l.: s.n.], 2021. p. 1–4. Citado na página 28.
- HALAWA, L. J.; WIBOWO, A.; ERNAWAN, F. Face recognition using faster r-cnn with inception-v2 architecture for cctv camera. In: *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*. [S.l.: s.n.], 2019. p. 1–6. Citado na página 23.
- HUA, S.; KAPOOR, M.; ANASTASIU, D. C. Vehicle tracking and speed estimation from traffic videos. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. [S.l.: s.n.], 2018. p. 153–1537. Citado na página 63.

- HUANG, G. et al. The vehicle speed measuring and precision analysis of video shot by moved camera based on 2d dlt algorithm. In: *2015 International Conference on Transportation Information and Safety (ICTIS)*. [S.l.: s.n.], 2015. p. 111–116. Citado 3 vezes nas páginas 44, 49 e 63.
- HUANG, S.-Y. et al. Image classification and adversarial robustness analysis based on hybrid quantum-classical convolutional neural network. *Optics Communications*, v. 533, p. 129287, 2023. ISSN 0030-4018. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0030401823000329>>. Citado na página 21.
- IBGE. *Frota de Veículos*. 2022. Disponível em: <<https://cidades.ibge.gov.br/brasil/pesquisa/22/28120.Acessoem:julhode2023>>. Citado na página 15.
- IZONIN, I. et al. Towards data normalization task for the efficient mining of medical data. In: *2022 12th International Conference on Advanced Computer Information Technologies (ACIT)*. [S.l.: s.n.], 2022. p. 480–484. Citado na página 30.
- JEON, H. H.; KO, Y.-H. Lidar data interpolation algorithm for visual odometry based on 3d-2d motion estimation. In: *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. [S.l.: s.n.], 2018. p. 1–2. Citado na página 43.
- JOCHER, G. *Ultralytics YOLOv8*. 2023. Disponível em: <<https://github.com/ultralytics/ultralytics>>. Acesso em: abril de 2023. Citado na página 42.
- JUSBRASIL. *Excesso de velocidade é infração de trânsito mais cometida pelos motoristas nas rodovias brasileiras*. 2022. Disponível em: <<https://www.jusbrasil.com.br/artigos/excesso-de-velocidade-e-infracao-de-transito-mais-cometida-pelos-motoristas-nas-rodovias-brasileiras/730440849>>. Acesso em: julhode2023. Citado na página 15.
- KARPATY, A. *CS231n Convolutional Neural Networks for Visual Recognition*. [s.n.], 2016. Disponível em: <<https://cs231n.github.io/convolutional-networks/>>. Citado 2 vezes nas páginas 21 e 22.
- KAVARAKUNTLA, T. et al. Performance analysis of distributed deep learning frameworks in a multi-gpu environment. In: *2021 20th International Conference on Ubiquitous Computing and Communications (IUCC/CIT/DSCI/SmartCNS)*. [S.l.: s.n.], 2021. p. 406–413. Citado na página 21.
- LAI, W.; COGHILL, G. Hard-limiting nonlinear functions in artificial neural networks. In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. [S.l.: s.n.], 1991. p. 1747–1752 vol.2. Citado na página 19.
- LI, H. et al. Research on overfitting of deep learning. In: *2019 15th International Conference on Computational Intelligence and Security (CIS)*. [S.l.: s.n.], 2019. p. 78–81. Citado 2 vezes nas páginas 24 e 25.
- LI, J. et al. An adaptive framework for multi-vehicle ground speed estimation in airborne videos. *Remote Sensing*, v. 11, n. 10, 2019. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066730674&doi=10.3390%2frs11101241&partnerID=40&md5=2bf0988f4d65de3de466240668c1dcd3>>. Citado na página 64.

- LIN, C.-J.; JENG, S.-Y.; LIOA, H.-W. A real-time vehicle counting, speed estimation, and classification system based on virtual detection zone and yolo. *Mathematical Problems in Engineering*, v. 2021, 2021. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85122988090&doi=10.1155%2f2021%2f1577614&partnerID=40&md5=87740a15fe46fdbd96e560c2cc3e6e665>>. Citado 4 vezes nas páginas 7, 57, 58 e 64.
- LINGUO, C. et al. Method of multi-lane vehicles speed continuously perceiving based on single roadside camera. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2022. p. 4314–4319. Citado na página 63.
- LU, S.; WANG, Y.; SONG, H. A high accurate vehicle speed estimation method. *Soft Computing*, v. 24, n. 2, p. 1283 – 1291, 2020. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077505820&doi=10.1007%2fs00500-019-03965-w&partnerID=40&md5=a4e5fb031c61b65e89784aa5ae95dc36>>. Citado na página 64.
- LUQUE, A. et al. Visualizing classification results: Confusion star and confusion gear. *IEEE Access*, v. 10, p. 1659–1677, 2022. Citado na página 27.
- LUVIZON, D. C.; NASSU, B. T.; MINETTO, R. A video-based system for vehicle speed measurement in urban roadways. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 18, n. 6, p. 1393–1404, 2016. Citado na página 71.
- MEJIA, H. et al. Vehicle speed estimation using computer vision and evolutionary camera calibration. In: *NeurIPS 2021 Workshop LatinX in AI*. [S.l.: s.n.], 2021. Citado 4 vezes nas páginas 45, 46, 50 e 63.
- NEILY, G. et al. Joint var controller implemented in an artificial neural network environment. In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. [S.l.: s.n.], 1991. p. 1336–1342 vol.2. Citado na página 19.
- ORGANIZATION, W. H. et al. *Road safety in the Eastern Mediterranean Region: facts from the global status report on road safety 2018*. [S.l.], 2020. Citado na página 15.
- PARK, K.; NGUYEN, M. C.; WON, H. Web-based collaborative big data analytics on big data as a service platform. In: *2015 17th International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2015. p. 564–567. Citado na página 69.
- PEI, Z. et al. Iteration time prediction for cnn in multi-gpu platform: Modeling and analysis. *IEEE Access*, v. 7, p. 64788–64797, 2019. Citado na página 19.
- PEREIRA, G.; TRÓI, M. d. *A alta velocidade tem ceifado a vida de 1,35 milhão de pessoas por ano*. 2022. Disponível em: <<https://diplomatique.org.br/a-alta-velocidade-tem-ceifado-a-vida-de-135-milhao-de-pessoas-por-ano/>>. Acesso em: julhode2023>. Citado na página 15.
- PERUNIĆIĆ, A.; DJUKANOVIĆ, S.; CVIJETIĆ, A. Vision-based vehicle speed estimation using the yolo detector and rnn. In: *2023 27th International Conference on Information Technology (IT)*. [S.l.: s.n.], 2023. p. 1–4. Citado 2 vezes nas páginas 55 e 63.
- PIMENTEL, T. R. G. Classificação de padrões temporais de uso do solo e cobertura da terra em séries temporais de índice de vegetação utilizando um sistema neuro-difuso. *Pós-Graduação em Computação Aplicada*. São José dos Campos, INPE, 2014. Citado na página 20.

- PRAJWAL et al. Multi-vehicle tracking and speed estimation model using deep learning. In: *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing*. New York, NY, USA: Association for Computing Machinery, 2022. (IC3-2022), p. 258–262. ISBN 9781450396752. Disponível em: <<https://doi-org.ez20.periodicos.capes.gov.br/10.1145/3549206.3549254>>. Citado na página 63.
- PRASETYA, D. S. A. et al. Experimental analysis of vehicle-to-vehicle communication using light detection and ranging (lidar) for detection and data transmission. In: *2021 International Conference on Artificial Intelligence and Mechatronics Systems (AIMS)*. [S.l.: s.n.], 2021. p. 1–6. Citado na página 43.
- RAIS, A. H.; MUNIR, R. Vehicle speed estimation using yolo, kalman filter, and frame sampling. In: *2021 8th International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*. [S.l.: s.n.], 2021. p. 1–6. Citado na página 63.
- REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado 2 vezes nas páginas 40 e 41.
- REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, v. 28, 2015. Citado na página 38.
- REVAUD, J.; HUMENBERGER, M. Robust automatic monocular vehicle speed estimation for traffic surveillance. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2021. p. 4531–4541. Citado na página 63.
- RODRIGUES, Y. *Pesquisas apontam velocidade como principal causa de mortes no trânsito em todo o mundo*. 2018. Disponível em: <<https://unifor.br/web/osv/pesquisas-apontam-velocidade-como-principal-causa-de-mortes-no-transito-em-todo-o-mundo#:~:text=ConsequAcessoem:julhode2023>>. Citado na página 15.
- RODRÍGUEZ-RANGEL, H. et al. Analysis of statistical and artificial intelligence algorithms for real-time speed estimation based on vehicle detection with yolo. *Applied Sciences (Switzerland)*, v. 12, n. 6, 2022. Cited by: 11; All Open Access, Gold Open Access. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85126803939&doi=10.3390%2fapp12062907&partnerID=40&md5=6d7177300efc86810e851b6e154ea661>>. Citado na página 64.
- ROTH, H. R. et al. Anatomy-specific classification of medical images using deep convolutional nets. In: *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*. [S.l.: s.n.], 2015. p. 101–104. Citado 2 vezes nas páginas 35 e 36.
- S., A.; A, S.; K, G. Classification of agricultural leaf images using hybrid combination of activation functions. In: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. [S.l.: s.n.], 2021. p. 785–791. Citado na página 24.
- SAHRAEIAN, R.; COMPERNOLLE, D. V. Exploiting sequential low-rank factorization for multilingual dnns. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2017. p. 5025–5029. Citado na página 34.
- SHAHBAZI, M. et al. Vehicle tracking and speed estimation from unmanned aerial videos. In: . [s.n.], 2020. v. 43, n. B2, p. 623 – 630. Disponível em: <<https://www.scopus.com/inward/record>>.

- uri?eid=2-s2.0-85091086948&doi=10.5194%2fisprs-archives-XLIII-B2-2020-623-2020&partnerID=40&md5=c2cd62f29df3a580b8eec090b9ddf2bd>. Citado na página 64.
- SHUVO, M. M. H. et al. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, v. 111, n. 1, p. 42–91, 2023. Citado 3 vezes nas páginas 32, 33 e 34.
- SKJERMO, J. et al. Predicting driving conditions at mountain crossings using deep learning. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2020. p. 1–5. Citado na página 63.
- SONG, Y.; GUAN, Z.; HE, W. Vehicle speed measurement based on camera calibration. In: *Proceedings of the 2nd International Conference on Graphics and Signal Processing*. New York, NY, USA: Association for Computing Machinery, 2018. (ICGSP '18), p. 12–15. ISBN 9781450363860. Disponível em: <<https://doi-org.ez20.periodicos.capes.gov.br/10.1145/3282286.3282288>>. Citado na página 63.
- sshikamaru. *Car Object Detection*. 2023. <<https://www.kaggle.com/datasets/sshikamaru/car-object-detection>>. Citado 3 vezes nas páginas 28, 67 e 68.
- SUBRAMANIAN, V. *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. [S.l.]: Packt Publishing Ltd, 2018. Citado 8 vezes nas páginas 21, 22, 23, 24, 25, 26, 30 e 35.
- TECH, L. *Traffic and Speed Enforcement Laser with Video*. 2023. Disponível em: <<https://lasertech.com/product/trucam-ii-speed-enforcement-laser/>>. Acesso em: setembro de 2023. Citado na página 85.
- TOMAS, J. P. Q. et al. Vehicle detection and speed estimation implemented through euclidean algorithm. In: *Proceedings of the 2021 4th International Conference on Computational Intelligence and Intelligent Systems*. New York, NY, USA: Association for Computing Machinery, 2022. (CIIS '21), p. 1–6. ISBN 9781450385930. Disponível em: <<https://doi-org.ez20.periodicos.capes.gov.br/10.1145/3507623.3507624>>. Citado na página 63.
- TRIVEDI, J. D.; MANDALAPU, S. D.; DAVE, D. H. Vision-based real-time vehicle detection and vehicle speed measurement using morphology and binary logical operation. *Journal of Industrial Information Integration*, v. 27, p. 100280, 2022. ISSN 2452-414X. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2452414X21000777>>. Citado na página 63.
- VICERI-SEIDOR. *Entendendo de vez a convolução: base para processamento de imagens*. 2020. Disponível em: <<https://viceri.com.br/insights/entendendo-de-vez-a-convolucao-base-para-processamento-de-imagens/>>. Acesso em: junho de 2023. Citado na página 19.
- WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022. Citado 2 vezes nas páginas 38 e 40.
- WANG, K.; SEIGAL, A. Lower bounds on the rank and symmetric rank of real tensors. *Journal of Symbolic Computation*, v. 118, p. 69–92, 2023. ISSN 0747-7171. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0747717123000044>>. Citado na página 23.

- WANG, L. et al. Improved faster-rcnn algorithm for mask wearing detection. In: *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. [S.l.: s.n.], 2021. v. 4, p. 1119–1124. Citado 2 vezes nas páginas 29 e 30.
- WANG, L.; LU, H. Classification of histopathologic images of breast cancer by multi-teacher small-sample knowledge distillation. In: *2021 2nd International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. [S.l.: s.n.], 2021. p. 642–647. Citado na página 34.
- WANG, M. et al. Discover the effective strategy for face recognition model compression by improved knowledge distillation. In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. [S.l.: s.n.], 2018. p. 2416–2420. Citado na página 34.
- WANG, Q. et al. Critical areas detection and vehicle speed estimation system towards intersection-related driving behavior analysis. In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.: s.n.], 2018. p. 1–6. Citado na página 63.
- WANG, Y.; MA, Z.; YANG, Z. A new quantization deployment method of neural network models integrating lstm layers. In: *2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*. [S.l.: s.n.], 2022. p. 1299–1303. Citado na página 34.
- WEN, L. et al. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 2020. Citado 2 vezes nas páginas 71 e 72.
- WERNECK, M. M. *Como funcionam os radares de trânsito?* 2009. Disponível em: <<https://cienciahoje.org.br/artigo/como-funcionam-os-radares-de-transito/>>. Acesso em: julhode2023>. Citado na página 43.
- WU, C. et al. Improved model compression method based on information entropy. In: *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*. [S.l.: s.n.], 2021. p. 104–108. Citado na página 33.
- WU, W.-P. et al. Design and implementation of vehicle speed estimation using road marking-based perspective transformation. In: *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. [S.l.: s.n.], 2021. p. 1–5. Citado 6 vezes nas páginas 4, 6, 43, 51, 52 e 63.
- XU, D. et al. Mask r-cnn assisted 2.5 d object detection pipeline of 68ga-psma-11 pet/ct-positive metastatic pelvic lymph node after radical prostatectomy from solely ct imaging. *Scientific Reports*, Nature Publishing Group UK London, v. 13, n. 1, p. 1696, 2023. Citado na página 37.
- XU, H. et al. Nasoa: Towards faster task-oriented online fine-tuning with a zoo of models. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2021. p. 5077–5086. Citado na página 31.
- YANG, Z. et al. An artificial neural network based attenuation tomography in free space optical network. In: *2018 International Conference on Networking and Network Applications (NaNA)*. [S.l.: s.n.], 2018. p. 52–57. Citado na página 20.
- YI, J.; GUAN, B.; LI, P. Intelligent highway speed monitoring uav system based on deep learning. In: *Proceedings of the 2021 4th International Conference on Image and Graphics Processing*. New York, NY, USA: Association for Computing Machinery, 2021. (ICIGP '21), p. 73–79. ISBN 9781450389105. Disponível em: <<https://doi-org.ez20.periodicos.capes.gov.br/10.1145/3447587.3447598>>. Citado 5 vezes nas páginas 7, 59, 60, 61 e 63.

- YIN, P. et al. Deep guidance network for biomedical image segmentation. *IEEE Access*, v. 8, p. 116106–116116, 2020. Citado 2 vezes nas páginas 35 e 36.
- YOHANNES, E. et al. An improved speed estimation using deep homography transformation regression network on monocular videos. *IEEE Access*, v. 11, p. 5955–5965, 2023. Citado na página 63.
- YU, C. et al. Eccnet: Efficient chained centre network for real-time multi-category vehicle tracking and vehicle speed estimation. *IET Intelligent Transport Systems*, v. 16, n. 11, p. 1489 – 1503, 2022. Cited by: 0. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85133517369&doi=10.1049%2fitr2.12227&partnerID=40&md5=28d57be9588805f3280f1a5a3522aabe>>. Citado 4 vezes nas páginas 7, 56, 57 e 64.
- ZAAOURI, K.; EZZEDINE, T. A self-adaptive traffic light control system based on yolo. In: *2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*. [S.l.: s.n.], 2018. p. 16–19. Citado 2 vezes nas páginas 31 e 35.
- ZHAO, L.; TANG, Y.; WANG, L. Research on vehicle speed identification method based on time interpolation method and feature point recognition. *International Journal of Advanced Robotic Systems*, v. 20, n. 1, 2023. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85150029115&doi=10.1177%2f17298806231153726&partnerID=40&md5=7329cf0f7622bfa82ec8da59a10e041>>. Citado na página 63.
- ČAVOR, I.; DJUKANOVIĆ, S. Vehicle speed estimation from audio signals using 1d convolutional neural networks. In: *2023 27th International Conference on Information Technology (IT)*. [S.l.: s.n.], 2023. p. 1–4. Citado 3 vezes nas páginas 58, 59 e 63.

Apêndices

APÊNDICE A – Códigos do experimento comparando Faster R-CNN e YOLOv8

Código 1 – Preparação dos dados do experimento com Faster R-CNN

```
1 from google.colab import drive
2 import pandas as pd
3 import torch
4 from torch.utils.data import DataLoader, Dataset
5 import torchvision.transforms as T
6 from PIL import Image
7 import os
8 from sklearn.model_selection import train_test_split
9
10 drive.mount('/content/drive')
11
12 dataset = pd.read_csv('/content/drive/MyDrive/Colab
13 Notebooks/Marcelo/data/train_solution_bounding_boxes (1).csv')
14
15 image_unique = dataset.image.unique()
16
17 class Car_data(Dataset):
18     def __init__(self, data, image_unique, labels):
19         self.unique = image_unique
20         self.data = data
21         self.labels = labels
22     def __len__(self):
23         return len(self.unique)
24     def __getitem__(self, idx):
25         image_name = self.unique[idx]
26         img = Image.open(os.path.join(self.data,
27             image_name)).convert('RGB')
28         boxes = self.labels[(image_name ==
29             self.labels['image'])].values[:, 1: ].astype('float')
30         labels = torch.ones((boxes.shape[0]), dtype = torch.int64)
31         target = {}
32         target['boxes'] = torch.tensor(boxes)
33         target['label'] = labels
34         return T.ToTensor()(img), target
35
36 def custom_collate(data):
37     return data
38
39 x_train, x_val = train_test_split(image_unique, test_size = 0.2,
40     random_state= 42)
41 train_data = Car_data('/content/drive/MyDrive/Colab
42 Notebooks/Marcelo/data/training_images', x_train, dataset)
43 val_data = Car_data('/content/drive/MyDrive/Colab
44 Notebooks/Marcelo/data/training_images', x_val, dataset)
45 train_loader = DataLoader(train_data, batch_size = 8, shuffle =
46     True, collate_fn = custom_collate)
47 val_loader = DataLoader(val_data, batch_size = 8, shuffle = True,
48     collate_fn = custom_collate)
```

Código 2 – Criação do modelo Faster R-CNN

```

1 from torchvision.models.detection import fasterrcnn_resnet50_fpn
2 from torchvision.models.detection.faster_rcnn import
3     FastRCNNPredictor
4 from torchvision.models.detection.faster_rcnn import
5     FastRCNNPredictor
6 data = 'FasterRCNN_ResNet50_FPN_Weights.COCO_V1'
7 model = fasterrcnn_resnet50_fpn(weights=data)
8 num_classes = 2
9 in_features = model.roi_heads.box_predictor.cls_score.in_features
10 model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
11     num_classes)

```

Código 3 – Funções auxiliares que ajudaram no treinamento do modelo Faster R-CNN

```

1 from torchmetrics.detection import mean_ap
2
3 def meanAveragePrecision(predict, target):
4     metric = mean_ap.MeanAveragePrecision()
5     metric.update(predict, target)
6     x = metric.compute()
7     mAp = x['map'].numpy()
8     return mAp
9
10 def validate(model, dataloader):
11     model.eval()
12     mAp = 0
13     with torch.no_grad():
14         for data in dataloader:
15             target = []
16             imgs = []
17             for d in data:
18                 imgs.append(d[0].to(device))
19                 targ = {}
20                 targ['boxes'] = d[1]['boxes'].to(device)
21                 targ['labels'] = d[1]['label'].to(device)
22                 target.append(targ)
23             output = model(imgs)
24             mAp += meanAveragePrecision(output, target)
25     return mAp / len(dataloader)

```

Código 4 – Treinamento do modelo Faster R-CNN

```
1 from tqdm import tqdm
2 from torch import optim
3
4 optimizer = optim.SGD(model.parameters(), lr = 0.001, momentum =
5     0.9, weight_decay = 0.0005)
6 device = 'cuda' if torch.cuda.is_available() else 'cpu'
7 model.to(device)
8
9 def train(model, dataloader, epochs, optimizer, val_loader):
10    losses = []
11    best_loss = 0
12    best_ap = 0
13    mAps = []
14    for epoch in range(epochs):
15        model.train()
16        epoch_loss = 0
17        loop = tqdm(enumerate(dataloader), total = len(dataloader))
18        for idx, data in loop:
19            imgs = []
20            targets = []
21            for d in data:
22                imgs.append(d[0].to(device))
23                targ = {}
24                targ['boxes'] = d[1]['boxes'].to(device)
25                targ['labels'] = d[1]['label'].to(device)
26                targets.append(targ)
27            loss_dict = model(imgs, targets)
28            loss = sum(v for v in loss_dict.values())
29            epoch_loss += loss.cpu().detach().numpy()
30            optimizer.zero_grad()
31            loss.backward()
32            optimizer.step()
33            map = validate(model, val_loader)
34            mAps.append(map)
35            loop.set_description(f"mAp: {map}")
36            loop.set_postfix(loss = epoch_loss)
37            losses.append(epoch_loss)
38            if map > best_ap:
39                best_ap = map
40                torch.save({
41                    "model_state_dict": model.state_dict(),
42                    "optimizer_state_dict": optimizer.state_dict()
43                }, 'best_model.pt')
44    return losses, mAps
45 loss, mAp = train(model, train_loader, 30, optimizer, val_loader)
```

Código 5 – Funções auxiliares que ajudaram na inferência do modelo Faster R-CNN

```

1 def get_bboxes(labels):
2     bboxes = []
3     with open(labels, 'r') as f:
4         lines = f.readlines()
5         for line in lines:
6             values = line.split(' ')
7             if len(values) < 4: break
8             boxes = []
9             boxes.append(float(values[1]))
10            boxes.append(float(values[2]))
11            boxes.append(float(values[3]))
12            boxes.append(float(values[4].split('\n')[0]))
13            bboxes.append(boxes)
14    return bboxes
15 def desnormalize(bboxes, image):
16    image = Image.open(image).convert('RGB')
17    width, height = image.size
18    for bbox in bboxes:
19        xmin = bbox[0] - bbox[2]/2
20        ymin = bbox[1] - bbox[3]/2
21        xmax = bbox[0] + bbox[2]/2
22        ymax = bbox[1] + bbox[3]/2
23        bbox[0] = xmin * width
24        bbox[1] = ymin * height
25        bbox[2] = xmax * width
26        bbox[3] = ymax * height
27    return bboxes
28 def show_image(image, bboxes, save = False, dir = None):
29    if not os.path.isdir(dir):
30        os.mkdir(dir)
31    img = np.array(Image.open(image).convert('RGB')).astype('float')
32    for bboxe in bboxes:
33        img = cv2.rectangle(
34            img,
35            (int(bboxe[0]), int(bboxe[1])),
36            (int(bboxe[2]), int(bboxe[3])),
37            color = (255, 0, 0),
38            thickness = 2
39        )
40    if save:
41        if dir is None:
42            dir = '/content/show_image'
43            if not os.path.isdir(dir):
44                os.mkdir(dir)
45            name = image.split('/')[-1]
46            cv2.imwrite(os.path.join(dir, name), img)
47    else:
48        cv2.imshow(img)

```

Código 6 – Função de inferência com Faster R-CNN

```

1 def inference(model, source, save = False, dir = None):
2     model.eval()
3     files = glob.glob(source + "/*.[jp][pn]g")
4     results, targets, maps = [], [], []
5     for file in files:
6         image = Image.open(file).convert('RGB')
7         image = T.ToTensor()(image)
8         dir_file = file.replace('/images', '/labels').replace('.jpg',
9             '.txt').replace('.png', '.txt')
10        target = {}
11        bboxes = get_bboxes(dir_file)
12        bboxes = desnormalize(bboxes, file)
13        label = torch.ones(len(bboxes), dtype = torch.int64).to(device)
14        target['labels'] = label
15        target['boxes'] = torch.tensor(bboxes).to(device)
16        targets.append(target)
17        with torch.no_grad(): output = model([image.to(device)])
18        result = {}
19        result['boxes'] = output[0]['boxes']
20        result['labels'] = output[0]['labels']
21        result['scores'] = output[0]['scores']
22        result_list, target_list = [], []
23        result_list.append(result)
24        target_list.append(target)
25        mAp = meanAveragePrecision(result_list, target_list)
26        maps.append(mAp)
27        results.append(result)
28        if save:
29            if dir is None:
30                dir = '/content/show_image'
31                if not os.path.isdir(dir): os.mkdir(dir)
32                image_name = file.split('/')[-1]
33                dir_save = os.path.join(dir, image_name)
34                os.mkdir(dir_save)
35                show_image(file, result['boxes'], save = True, dir =
36                    f'{dir_save}/inference')
37                show_image(file, target['boxes'], save = True, dir =
38                    f'{dir_save}/ground_truth')
39                dir_lbl = dir_save + '/' + image_name.replace('.jpg',
40                    '.txt').replace('.png', '.txt')
41                with open(dir_lbl, 'a') as f:
42                    f.writelines('Image: ' + ' ' + image_name + '\n' +
43                    'mAp: ' + ' ' + str(mAp) + '\n'
44                    'Bboxes: {' + '\n')
45                    for box in result['boxes']:
46                        f.writelines(' ' + str(box[0]) + ' ' + str(box[1]) + ' ' +
47                            str(box[2]) + ' ' + str(box[3]) + '\n')
48                    f.writelines('}' + '\n')
49    return results, targets, maps

```

Código 7 – Inferência de dados do experimento com Faster R-CNN

```
1 !wget https://app.roboflow.com/ds/s4gd3l9oss?key=pxE4Cs4MS4
2 !7z x /content/s4gd3l9oss?key=pxE4Cs4MS4
3
4 import glob
5 import cv2
6 from google.colab.patches import cv2_imshow
7 import numpy as np
8
9 checkpoint = torch.load('/content/best_model.pt')
10 model.load_state_dict(checkpoint['model_state_dict'])
11 optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
12
13 source = '/content/valid/images'
14 results, targets, mAps = inference(model, source, save = True)
```

Código 8 – Funções auxiliares que ajudaram na preparação dos dados do modelo YOLOv8

```

1 def normalize(labels, image_dir):
2     for idx in labels.index:
3         name = labels['image'][idx]
4         image = os.path.join(image_dir, name)
5         if os.path.isfile(image):
6             img = Image.open(image)
7             width, height = img.size
8             labels['width'][idx] = labels['width'][idx] / width
9             labels['height'][idx] = labels['height'][idx] / height
10            labels['xc'][idx] = labels['xc'][idx] / width
11            labels['yc'][idx] = labels['yc'][idx] / height
12
13 training_path = '/content/data/train'
14 valid_path = '/content/data/val'
15 os.mkdir(training_path + '/images')
16 os.mkdir(training_path + '/labels')
17 os.mkdir(valid_path)
18 os.mkdir(valid_path + '/images')
19 os.mkdir(valid_path + '/labels')
20
21 def move_images(dir_src, dir_dst, images):
22     for i in images.index:
23         image = images['image'][i]
24         image_path = os.path.join(dir_src, image)
25         os.rename(image_path, os.path.join(dir_dst, f'images/{image}'))
26
27 def add_label(dir, images):
28     for i in images.index:
29         image = os.path.join(dir, images['image'][i])
30         if os.path.isfile(image):
31             label = image.replace('/images', '/labels').replace('.jpg',
32                                         '.txt').replace('.png', '.txt')
33             if os.path.isfile(label):
34                 with open(label, 'a') as f:
35                     f.write('\n' + '0' + ' ' + str(images['xc'][i]) + ' ' +
36                         str(images['yc'][i]) + ' ' + str(images['width'][i]) + ' ' +
37                         str(images['height'][i]))
38             else:
39                 with open(label, 'a') as f:
40                     f.write('0' + ' ' + str(images['xc'][i]) + ' ' +
41                         str(images['yc'][i]) + ' ' + str(images['width'][i]) + ' ' +
42                         str(images['height'][i]))

```

Código 9 – Preparação dos dados do experimento com YOLOv8

```

1 from google.colab import drive
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 import glob
5 import numpy as np
6 drive.mount('/content/drive')
7 !7z x /content/drive/MyDrive/Colab\ Notebooks/archive.zip
8 dataset = pd.read_csv('/content/data/train_solution_bounding_boxes
9     (1).csv')
10 dataset['width'] = dataset.loc[:, 'xmax'] - dataset.loc[:, 'xmin']
11 dataset['height'] = dataset.loc[:, 'ymax'] - dataset.loc[:, 'ymin']
12 dataset['xc'] = dataset.loc[:, 'xmin'] + dataset.loc[:, 'width']/2
13 dataset['yc'] = dataset.loc[:, 'ymin'] + dataset.loc[:, 'height']/2
14 normalize(dataset, '/content/data/training_images')
15
16 os.rename('/content/data/testing_images', '/content/data/test')
17 os.rename('/content/data/training_images', '/content/data/train')
18 file = open('/content/data/data.yaml', 'w')
19 file.writelines(
20     'train: ../train/images' + '\n' +
21     'val: ../val/images' + '\n' +
22     'test: ../test/images' + '\n' +
23     'nc: 1' + '\n' +
24     "names: ['car']" + '\n')
25 file.close()
26 images = dataset.loc[:]
27 images.drop_duplicates('image', keep = 'first', inplace = True)
28 X_train, X_valid, _, _ = train_test_split(images, images['xmin'],
29                                             test_size = 0.2, random_state = 42)
30 move_images(training_path, training_path, X_train)
31 move_images(training_path, valid_path, X_valid)
32 add_label(os.path.join(training_path, 'images'), dataset)
33 add_label(os.path.join(valid_path, 'images'), dataset)
34 files = glob.glob('/content/data/train/vid_*.jpg')
35 num_of_images = len(files)
36 if num_of_images > 0:
37     num = int(num_of_images * 5 / 100)
38     shuffle = np.random.permutation(num_of_images)
39     '''for i in shuffle[: num]:
40         image = files[i].split('/')[-1]
41         os.rename(files[i], f'/content/data/train/images/{image}')
42         label = image.replace('.jpg', '.txt').replace('.png', '.txt')
43         with open(os.path.join('/content/data/train/labels', label),
44                   'w+') as f:
45             f.close()'''
46     for idx in shuffle:
47         os.remove(files[idx])

```

Código 10 – Treinamento do modelo YOLOv8

```
1 !pip install ultralytics
2 from ultralytics import YOLO
3
4 def set_res_dir(train = True):
5     res_dir_count = len(glob.glob('/content/runs/detect/results_*'))
6     if train:
7         res_dir = f'results_{res_dir_count + 1}'
8     else:
9         res_dir = f'results_{res_dir_count}'
10    return res_dir
11
12 def train(model = 'yolov8n.pt', data = 'coco128.yaml', epochs = 25):
13     res_dir = set_res_dir()
14     model = YOLO(model = model)
15     model.train(data = data, epochs = epochs, name = res_dir, imgsz =
16                 640)
17     return f'/content/runs/detect/{res_dir}/results.png',
18           f'/content/runs/detect/{res_dir}/weights/best.pt'
19 models = [
20     'yolov8n.pt',
21     'yolov8s.pt',
22     'yolov8m.pt'
23 ]
24 data = '/content/data/data.yaml'
25 sources = r'/content/data/test/images/*.[jp][pn]g'
26 results = []
27 inferences = []
28 best_models = []
29 for model in models:
30     result, best_model = train(model, data, 30)
31     results.append(result)
32     best_models.append(best_model)
```

Código 11 – Funções auxiliares da inferência dos dados com YOLOv8

```
1 from torchmetrics.detection import mean_ap
2 import torch
3 import cv2
4 from google.colab.patches import cv2_imshow
5 import matplotlib.pyplot as plt
6
7 def meanAveragePrecision(predict, target):
8     metric = mean_ap.MeanAveragePrecision(box_format = 'cxcywh')
9     metric.update(predict, target)
10    result = metric.compute()
11    mAp = result['map']
12    return mAp.numpy()
13
14 def get_bboxes(labels):
15     bboxes = []
16     with open(labels, 'r') as f:
17         lines = f.readlines()
18         for line in lines:
19             values = line.split(' ')
20             if len(values) < 4: break
21             boxes = []
22             boxes.append(float(values[1]))
23             boxes.append(float(values[2]))
24             boxes.append(float(values[3]))
25             boxes.append(float(values[4].split('\n')[0]))
26             bboxes.append(boxes)
27     return bboxes
28
29 def desnormalize(bboxes, image):
30     image = Image.open(image).convert('RGB')
31     width, height = image.size
32     for bbox in bboxes:
33         xmin = bbox[0] - bbox[2]/2
34         ymin = bbox[1] - bbox[3]/2
35         xmax = bbox[0] + bbox[2]/2
36         ymax = bbox[1] + bbox[3]/2
37         bbox[0] = xmin * width
38         bbox[1] = ymin * height
39         bbox[2] = xmax * width
40         bbox[3] = ymax * height
41     return bboxes
```

Código 12 – Função de Inferência do experimento com YOLOv8

```

1 def inference(model_name, source):
2     device = 'cuda' if torch.cuda.is_available() else 'cpu'
3     value = model_name.split('/')[-3].split('_')[-1]
4     files = glob.glob(source + '/*.[jp][pn]g')
5     results = []
6     targets = []
7     mAps = []
8     for file in files:
9         dir = file.replace('/images', '/labels').replace('.jpg',
10                         '.txt').replace('.png', '.txt')
11         target = {}
12         result = {}
13         bboxes = get_bboxes(dir)
14         target['boxes'] = torch.tensor(bboxes).to(device)
15         target['labels'] = torch.zeros(len(target['boxes']), dtype =
16             torch.int64).to(device)
17         targets.append(target)
18         image = file.split('/')[-1]
19         inference_dir = f'inference_{value}/{image}'
20         model = YOLO(model_name)
21         output = model.predict(file, name = inference_dir, save = True,
22             save_txt = True)
23         result['boxes'] = output[0].boxes.xywhn.to(device)
24         result['labels'] = torch.zeros(len(output[0].boxes), dtype =
25             torch.int64).to(device)
26         result['scores'] = output[0].boxes.cls.to(device)
27         results.append(result)
28         img_result = []
29         img_target = []
30         img_result.append(result)
31         img_target.append(target)
32         mAp = meanAveragePrecision(img_result, img_target)
33         map_file = f'/content/runs/detect/{inference_dir}/map.txt'
34         with open(map_file, 'a') as f:
35             f.writelines(
36                 f'Image: {image}\n mAp: {mAp}\n Boxes: ' + '{\n')
37             )
38             for box in target['boxes']:
39                 f.writelines(
40                     ' ' + str(box[0]) + ' ' + str(box[1]) + ' ' +
41                     str(box[2]) + ' ' + str(box[3]) + '\n'
42                 )
43             f.writelines('}')
44             mAps.append(mAp)
45     return results, targets, mAps

```

Código 13 – Inferência dos dados do experimento com YOLOv8

```
1 !wget https://app.roboflow.com/ds/s4gd3l9oss?key=pxE4Cs4MS4
2 !7z x /content/s4gd3l9oss?key=pxE4Cs4MS4
3
4 for img in glob.glob('/content/valid/images/*.[jp][pn]g'):
5     dir_label = img.replace('/images', '/labels').replace('.jpg',
6                             '.txt').replace('.png', '.txt')
7     with open(dir_label, 'r') as f:
8         lines = f.readlines()
9         if len(lines) == 0:
10             os.remove(dir_label)
11             os.remove(img)
12
13 results_list = []
14 targets_list = []
15 mAps_list = []
16 source = '/content/valid/images'
17 for best_model in best_models:
18     results, targets, mAps = inference(best_model, source)
19     value = best_model.split('/')[-3].split('_')[-1]
20     dir_dst = f'/content/runs/detect/inference_{value}'
21     set_ground_truth(source, dir_dst)
22     results_list.append(results)
23     targets_list.append(targets)
24     mAps_list.append(mAps)
```