MASTER'S DEGREE IN WEB TECHNOLOGY AND SYSTEMS ENGINEERING

AI-Powered Vehicle Image Recognition for Smart Urban Planning and Traffic Management

Paulo André Guimarães

Supervisor: Firmino Oliveira da Silva

Vila Nova de Gaia

Academic Year 2024-2025

Acknowledgements

First, I thank Teacher Firmino Silva for guiding me in this master course from the very beginning. Deep thanks to my family for supporting me in this process of pursuing changes and self-realization. And, to Engineer Afonso Costa and F. Vaal, just to let you know that you've done a lot to this very soul responsible for this document.

INSTITUTO POLITÉCNICO DE GESTÃO E TECNOLOGIA

AI-Powered Vehicle Image Recognition for Smart Urban Planning and Traffic Management

Paulo André Guimarães

Aprovado em 16/12/2025

Composição do Júri

Presidente
Prof. Doutor Jorge Pereira Duque

Arguente
Prof.ª Doutora Célia Talma Gonçalves

Orientador
Prof. Doutor Firmino Oliveira da Silva

Vila Nova de Gaia
2025

Projeto de Mestrado realizado sob a orientação do Professor Doutor Firmino Oliveira da Silva, apresentado no ISLA - Instituto Politécnico de Gestão e Tecnologia de Vila Nova de Gaia para obtenção do grau de Mestre em Engenharia de Tecnologias e Sistemas Web, conforme o Despacho nº 9371/2020.

Abstract

Modern cities face complex challenges in traffic management and road maintenance, which impact both quality of life and infrastructure efficiency. AI-powered vehicle image recognition offers a promising solution by transforming visual data into actionable insights for optimized transportation planning and predictive infrastructure maintenance.

Artificial Intelligence enables complex, data-driven tasks through machine learning, with computer vision processing visual data to extract actionable insights in urban environments. By leveraging vehicle identification and object detection, AI enhances urban planning and transportation systems, optimizing traffic flow, reducing congestion, and improving safety, while also enabling proactive infrastructure maintenance through real-time analysis. Recent advances in deep learning and convolutional neural networks have introduced robust, real-time image recognition capabilities that offer practical solutions for the challenges of urban mobility and infrastructure management.

This work focuses on the development of an AI-driven image recognition application for vehicle identification, aimed at supporting integrated urban planning, transportation systems optimization, and infrastructure monitoring. The research begins with an overview of artificial intelligence, machine learning, and deep learning principles, with particular emphasis on the architecture and effectiveness of CNNs in object detection tasks. A structured methodology is presented, detailing the proposed architectural system, selection of relevant datasets, data annotation processes, and experimental setup. Special attention is given to the implementation of state-of-the-art object detection models, such as YOLO (You Only Look Once), trained and evaluated using the mixed COCO+BDD100k dataset within the PyTorch framework, and optimized through GPU acceleration to achieve high-speed inference and detection accuracy. Based on the Design Science Research methodology, this work developed a real-time vehicle tracking system using a YOLOv11n, achieving a detection precision of approximately 78% and a Multi-Object Tracking Accuracy (MOTA) of 67.5%, successfully demonstrating capabilities for vehicle counting, speed estimation, and ESAL calculation to support urban planning and predictive maintenance.

Furthermore, the study discusses the system's potential to integrate with urban monitoring platforms, offering real-time data streams for city planners and traffic authorities. The findings underscore the transformative potential of AI in advancing urban mobility, safety, and infrastructure resilience, while also identifying avenues for future research, including the integration of multi-source data, scalability challenges, and adaptive learning mechanisms for evolving urban environments.

KEYWORDS: Artificial Intelligence (AI); Vehicle Image Recognition; Urban Planning; Transportation Management; Deep Learning; Smart Cities

Resumo

As cidades modernas enfrentam desafios complexos na gestão do tráfego e manutenção de estradas, o que impacta tanto a qualidade de vida quanto a eficiência da infraestrutura. O reconhecimento de imagens de veículos com IA oferece uma solução promissora ao transformar dados visuais em informações acionáveis para um planeamento de transporte otimizado e uma manutenção preditiva da infraestrutura.

A Inteligência Artificial permite tarefas complexas e orientadas por dados através da aprendizagem máquina, sendo que a visão computacional processa dados visuais para extrair informações acionáveis em ambientes urbanos. Ao alavancar a identificação de veículos e a deteção de objetos, a IA melhora o planeamento urbano e os sistemas de transporte, otimizando o fluxo de tráfego, reduzindo congestionamentos e melhorando a segurança, além de permitir uma manutenção proativa da infraestrutura através de análise em tempo real. Avanços recentes em aprendizagem profunda e redes neuronais convolucionais introduziram capacidades robustas de reconhecimento de imagem em tempo real que oferecem soluções práticas para os desafios da mobilidade urbana e gestão de infraestruturas.

Este trabalho foca-se no desenvolvimento de uma aplicação de reconhecimento de imagem orientada por IA para identificação de veículos, com o objetivo de apoiar o planeamento urbano integrado, a otimização dos sistemas de transporte e a monitorização de infraestruturas. A investigação começa com uma visão geral dos princípios de inteligência artificial, aprendizado de máquina e aprendizagem profunda, com ênfase particular na arquitetura e eficácia das CNNs em tarefas de deteção de objetos. É apresentada uma metodologia estruturada, detalhando o sistema arquitetónico proposto, a seleção de conjuntos de dados relevantes, os processos de anotação de dados e o enquadramento experimental. É dada atenção especial à implementação de modelos de deteção de objetos de última geração, como o YOLO, treinados e avaliados usando o conjunto de dados do COCO+BDD100k no ambiente PyTorch, e otimizados através de aceleração por GPU para alcançar alta velocidade de inferência e precisão de deteção. Com base na metodologia Design Science Research, este trabalho desenvolveu um sistema de rastreamento de veículos em tempo real usando um YOLOv11n, alcançando uma precisão de deteção de aproximadamente 78% e uma Multi-Object Tracking Accuracy (MOTA) de 67,5%, demonstrando com sucesso capacidades para

contagem de veículos, estimativa de velocidade e cálculo de ESAL para apoiar o planeamento urbano e a manutenção preditiva.

Além disso, o estudo discute o potencial do sistema para se integrar a plataformas de monitorização urbana, oferecendo fluxos de dados em tempo real para planeamento urbano e autoridades de trânsito. As conclusões reforçam o potencial transformador da IA no avanço da mobilidade urbana, segurança e resiliência das infraestruturas, enquanto identifica direções para pesquisas futuras, incluindo a integração de dados de múltiplas fontes, desafios de escalabilidade e mecanismos de aprendizagem adaptativa para ambientes urbanos em evolução.

PALAVRAS-CHAVE: Inteligência Artificial (IA); Reconhecimento de Imagens de Veículos; Planeamento Urbano; Gestão dos Transportes; Deep Learning; Cidades Inteligentes

Summary

List Of Figures

List Of Tables

Acronyms

AAT – Average Annual Traffic
ADT – Average Daily Traffic
AI - Artificial intelligence
BDD100k – Berkeley Deep Drive 100K
CNN – Convolutional Neural Network
COCO – Common Objects in Context
CQRS – Command Query Responsibility Segregation
CPU – Computer Processing Unit
CV – Computer Vision
DDD – Applying Domain-Driven Design
DM – Deep Learning
DSR – Design Science Reasearch
ESAL – Equivalent Single Axle Load
FPS – Frame Per Second
GPU – Graphic Processing Unit
IDF1 – Identity F1 score
ILS – Image Labeler Suite
IoT – Internet of Things
IoU – Intersection over Union
ITMS – Information Traffic Management System
ITS – Information Traffic System
LHR – AAT
mAP – Mean Average Precision
ML – Machine Learning
MOTA – Multi Oblect Tracking Accuracy
NLP – Neural Language Processing
OSE – Onthological Search Engine
ReLU – Rectified Linear Unit
REST – Representational State Transfer
RCNN – Recursive CNN
SPP – Spatial Pyramid Pooling-Net
SSD – Single Shot MultiBox Detector
UI – User Interface
VDF – Vehicle Damage Factor
YOLO – You Look Only Once

1. Introduction

1.1 Context

Modern urban environments face increasing challenges related to traffic congestion, road safety, and the timely maintenance of infrastructure. As vehicle density rises and cities grow more complex, traditional traffic monitoring systems often fail to provide real-time, accurate data necessary for efficient urban management. To address these gaps, advanced sensor technologies are being adopted such as those used by the U.S. Department of Transportation, (2024) to monitor cracks and structural weaknesses in bridges—enabling early detection, timely repairs, and the prevention of catastrophic failures while improving safety, reducing costs, and extending infrastructure lifespan. In parallel, integrating AI-powered image recognition for vehicle identification offers a powerful solution, by enabling real-time traffic flow analysis, early detection of road degradation, and more effective urban planning (Di Grande et al., 2024)

Artificial Intelligence (AI) has evolved to revolutionize industries and societies worldwide, particularly through the advent of machine learning and deep learning. Computer Vision (CV), an integral component of AI, endows computers with the capability to analyse and extract information from visual data, such as images or videos, thereby opening new frontiers for image processing and analysis across many disciplines (Marasinghe et al., 2024). The primary objective of this work is to develop and evaluate an AI-driven image recognition system capable of accurate vehicle detection, classification, and tracking. This system is designed not only to optimize traffic management but also to support predictive infrastructure maintenance by calculating traffic-induced road degradation through Equivalent Single Axle Load (ESAL) metrics. By leveraging a YOLO-based model trained on the mixed dataset (COCO+BDD100k), this research aims to demonstrate a practical, scalable prototype that transforms visual data into actionable insights for smarter urban ecosystems.

As new concepts are being embraced, like smart cities, which are AI driven systems presented in form of smart traffic lights, noise or air quality prediction, and foot traffic as well as car traffic prediction faculties, the integration of AI technology with urban planning practices presents an opportunity for urban planners to enhance their capabilities to analyse

large urban datasets, recognise patterns and trends, and make informed predictions through modelling and simulation (Marasinghe et al., 2024). Urban space as a dynamic system, composed of human and commercial activity, flows of energy and matter, and their interactions, can no longer be analysed as a static space built of structures and roads. In the rapidly evolving landscape of our modern digital society accompanied by AI opportunities, an intelligent city is a beacon for a transformative endeavour that modern smart cities all over the world are set to embark upon (Kourtit et al., 2024). Exploring AI techniques to detect, classify and identify these dynamics is particularly important.

1.2 Motivation

In the realm of urban planning and traffic management, accurate car identification can revolutionize how we plan cities, handle traffic flow, detect violations, manage congestion, and anticipate infrastructure maintenance. According to Liao, (2022), the constant improvement of the country's road infrastructure, the road surface is influenced by environmental factors, including temperature, traffic load, weathering, which gradually reduce the pavement structure's strength, eventually leading to various disease characteristics (such as cracks, rutting, potholes, etc.). With the development of computer vision and deep learning, image classification, object detection, and segmentation techniques have been widely employed in the detection of road pavement damages (Ren et al., 2024). Urban planners can leverage this technology to analyse traffic patterns and vehicle usage, leading to better infrastructure development and resource allocation.

The potential for improving operational efficiency, safety, and planning underscores the importance of advancing AI-based car identification systems. Moreover, the dynamic nature of urban environments necessitates robust and adaptable AI models capable of functioning under diverse conditions. From varying lighting and weather conditions to different vehicle angles and occlusions, the need for resilient AI solutions is quite clear. Clearly, in the modern era, as we recognize the complexities of urban life, the pursuit of enhancing the quality of life in cities and their neighborhoods has taken center stage (Kourtit et al., 2024). Developing such solutions requires not only sophisticated algorithms but also extensive and diverse datasets to train and validate these models. The motivation for this study is driven by the

transformative potential of AI in image recognition, specifically car identification. By addressing the inherent challenges and leveraging advanced AI techniques, this research aims to contribute significantly to the fields of urban planning and traffic management and enhance the accuracy, efficiency, and applicability of car identification systems, thereby driving innovation and improving societal outcomes. With this, the research question comes as: **How can AI-powered vehicle image recognition enhance urban planning strategies for traffic management in cities?** So, this study aspires to reach the forefront of this transformative journey, providing insights and advancements that will help shape the future of urban management and transportation safety.

1.3 Purpose

To respond the question from previous section, the primary objective of this dissertation is set to explore and enhance the application of Artificial Intelligence (AI) in image recognition, aiming to implement a robust AI model capable of accurate and efficient car identification under diverse conditions, resorting to Convolutional Neural Network (CNN) architecture and the most recent technologies tailored for object identification, also discuss concrete cases on early detection of infrastructure degradation. The global and detailed objectives go as follows:

Global Objective

The general objective of this dissertation is to explore, implement, and evaluate the application of Artificial Intelligence (AI) in image recognition, with a specific focus on real-time car detection, classification, and tracking. The goal is to train a robust AI-based system that operates efficiently under diverse environmental conditions, contributing to advancements in traffic management and urban planning.

Detailed Objectives

1. *Model training and Enhancement:*

    a. Implement an AI model based on Convolutional Neural Network (CNN) architecture and the latest object detection technologies.

b. Address challenges in object identification, including variability in car appearance (colour, modifications).

2. *Performance Optimization:*

a. Explore techniques to mitigate environmental factors such as low lighting, occlusions, and adverse weather conditions.

b. Improve the model's ability to differentiate between visually similar objects.

3. *Evaluation and Validation:*

a. Conduct thorough evaluation using up-to-date datasets, such as Cityscapes, Waymo or BDD100k , to ensure accuracy, precision, and robustness in real-world scenarios.

b. Measure performance using relevant metrics, such as mAP (mean Average Precision), MOTA (Multi-Object Tracking Accuracy), and FPS (Frames Per Second).

4. *Practical Application:*

a. Investigate the practical implications of AI-based car detection and classification systems in enhancing urban planning, aiding traffic management, and anticipate road degradation.

b. Analyse integration strategies for deploying the system in real-world environments, ensuring scalability, reliability, and ease of adoption.

5. *Real-Time System Implementation:*

a. Develop a web-based interface for real-time visualization of car detection and tracking, enabling live monitoring of traffic flow and vehicle categorization.

1.4 Method

To achieve the objectives outlined in this dissertation, a systematic and comprehensive approach will be employed. The method encompasses several stages (Figure 1.1), from data collection and preparation to model training, evaluation, and application analysis. Each stage is critical to ensure the robustness and effectiveness of the AI model for car detection.

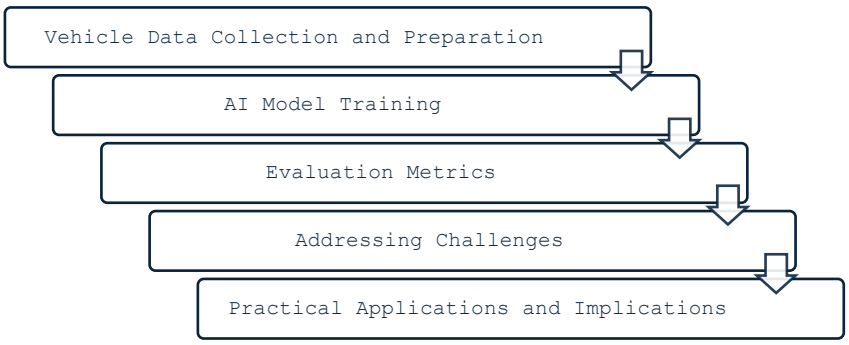


Figure 1.1: Method for Applying AI in vehicle classification

By employing these methodologies, this dissertation aims to implement a robust and effective AI model for car identification, addressing key challenges and demonstrating practical applications that can enhance urban planning in terms of transportation and traffic management.

1.5 Structure of the document

This work is organised into seven chapters. The first chapter presents the purpose of the dissertation, its context, motivation and the process to achieve the desired results. The second chapter focuses on the state of the art, presenting the background of AI, the basic concepts and explanation on how it works, a brief history and classification of artificial technologies considering the capabilities and their components regarding human dissimulation. The second part of the chapter presents the core of AI, approaching machine learning in general, then diving into deep learning with the concepts, theoretical foundation and evolution to neural networks, convolution and recurrent neural networks. The same chapter delves into image recognition, explaining the key features of this work, image detection and classification, from the concepts, process, to the technologies that support this essential part of AI known as computer vision. The last section of the chapter is presented in resume the

most relevant related works from articles, opinions, technologies blogs and tendencies presented by the giants on AI.

The third chapter of the work explains the general methodology applied to the goals here proposed, presenting the requirements of the system, technologies and tools. As for the fourth chapter, focuses on the experimental setup, from training the model to be used in the prototype and the evaluation through performance metrics observation.

In the last chapters, it is presented the discussion, practical application and the conclusion, analysing the inherent performance metrics and the limitations, considering the objectives of the work. Moreover, and an approach on the practical application and impact analysis of AI-Powered Vehicle Image Recognition for Smart Urban Planning and Traffic Management. Then, the conclusion of the work and suggestions for future works.

2. Artificial Intelligence in Urban Planning and traffic management

In this chapter, a brief context is presented on Artificial Intelligence, from historical reference to the theoretical foundation to properly understanding how AI supports urban planning through image recognition.

2.1 Artificial Intelligence and Its Role in Urban Planning

Urban planners are increasingly using artificial intelligence (AI) to optimise the design and management of cities, improving decision-making in urban planning (Ponce et al., 2023). These optimisations and management that resorts to the use of AI, bring new concepts in our way of life; smart cities, which englobes land use optimisation planning, population growth prediction, transportation planning, traffic management, environment sustainability, and infrastructure disaster response and prevention. Central to the development of these smart cities are Big Data and Artificial Intelligence (AI), two transformative technologies that offer new ways of managing and analysing urban environments (Ejaz et al., 2025).

Current AI development focuses on five main areas of human dissimilation: Human learning processing, represented by machine learning (ML); Human thinking processing, represented by data mining (DM), Human vision, represent by computer vision (CV), Human language and conversation, represented by Natural Language Processing (NLP), and Human knowledge – represented by Ontological-based Search Engine (OSE) (Lee, 2020). However, AI applications in urban planning rely on Machine Learning, Computer Vision, Natural Language Processing, Predictive Analytics, and additionally Automation and Optimization.

In cities, ML models are used to predict traffic flow, forecast energy usage, or identify areas at risk of crime. As more data is collected, AI models continuously improve their accuracy and efficiency (Ejaz et al., 2025). AI-driven predictive analytics helps mitigate climate change impacts and urban inequalities by forecasting disasters, infrastructure risks, and socio-economic trends for proactive planning. Moreover, researchers emphasizes its effectiveness in forecasting congestion, optimizing the movement of vehicles, and promoting more flexible transportation networks (Igorevich Rozhdestvenskiy & Poornima, 2024).

Computer vision (CV), an integral component of AI, can be defined as a technological field that endows computers with the capability to analyse and extract information from visual data, such as images or videos, thereby opening new frontiers for image processing and analysis across many disciplines (Marasinghe et al., 2024). It helps with the extraction of useful information from image and video data, for better comprehension of our environment. Cutting-edge urban research has employed modern tools including social platforms, mobile devices, sensor networks, and street-level imagery to gather more extensive datasets and study city dynamics. CV applications in urban planning rely on various types of data sources, such as satellite imagery, street view images, photographs, social media images, video data, and so on, used to identify and understand urban patterns, dynamics, character, growth, land use change, and socioeconomic challenges (Marasinghe et al., 2024). Automation and Optimization: AI can automate routine urban tasks, such as traffic signal control or waste management, by adjusting systems based on real-time data (Ejaz et al., 2025).

In addition, according (Ejaz et al.,2025) effective infrastructure management is crucial for ensuring cities function efficiently. By implementing predictive maintenance through real-time monitoring of infrastructure such as roads, bridges, and others distribution systems.

2.2 Artificial Intelligence for traffic management and Smart Cities

The transportation sector is one of the major sectors of the smart city, and over the past several decades, there have been widespread traffic-related issues due to the fast population growth and the corresponding rise in the number of vehicles (Saini & Sharma, 2025). Modern cities explore the capabilities of AI for enhancing traffic and transportation systems. From predictive algorithms to smart traffic lights, AI systems offer the potential to optimize traffic flow, reduce delays, and enhance commuter experiences (Francisco et al., 2024). According (Ogunkan & Ogunkan, 2025), Singapore and New York City have implemented AI-driven systems for Real-time traffic optimization , and have achieved good results, reducing congestion and improving mobility.

Intelligent traffic management is applied in transportation to regulate and maintain the flow of vehicles and people, to avoid congestion, accidents and other inconveniences in transportation. Intelligent Transportation Systems (ITS) have started to incorporate AI for

better traffic signal optimization, improving vehicular flow at intersections, where studies have demonstrated a 25% reduction in congestion through reinforcement learning-based adaptive traffic lights (Francisco et al., 2024). The main domains of these systems are as follows:

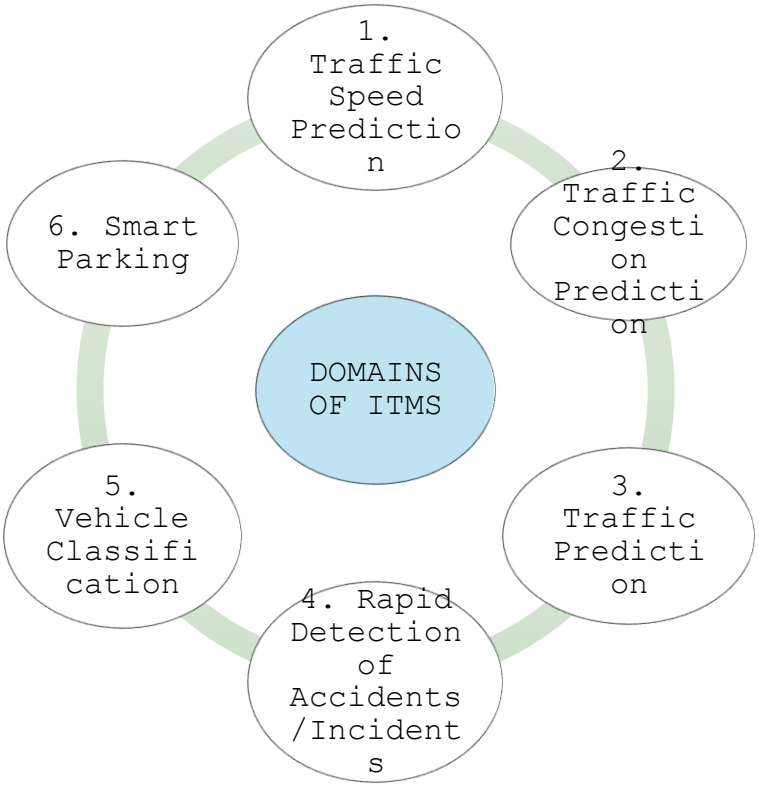


Figure 2.1: Components of Intelligent Traffic management System, adapted from (Saini & Sharma, 2025)

Smart traffic management systems leveraging AI and IoT are transforming urban mobility by addressing key challenges like congestion, accidents, and inefficient parking. In Saini & Sharma (2025), is highlight several implementations of ITMS as presented on Figure 2.1:

- Traffic speed prediction, where AI algorithms, such as those in Singapore's Smart Mobility 2030 program, analyze real-time data to optimize traffic flow, reducing delays during peak hours.

- Traffic congestion prediction systems, like Los Angeles' ATSAC, use IoT sensors and machine learning to anticipate and mitigate bottlenecks, cutting peak-time delays by up to 13%.

- Incident detection and classification, where AI-powered systems, such as New York City's Connected Vehicle Pilot, quickly identify accidents or disruptions, improving emergency response times and minimizing road hazards.

- Smart parking solutions, like Barcelona's IoT-enabled app, guide drivers to available spots using real-time data, reducing unnecessary circling and lowering emissions by 30%.

- Traffic prediction models, such as those in Beijing and Amsterdam, forecast vehicle flow and adjust signal timings dynamically, shortening travel times and easing congestion.

- Vehicle classification technologies, including automated license plate recognition (e.g., NYC's toll system) and AI-powered cameras (e.g., Shenzhen's traffic monitoring), help enforce regulations and streamline toll collection, enhancing efficiency.

AI systems help build accurate data by monitoring the volume of traffic, vehicle flow density and the environment and infrastructural impact of vehicles on the roads. AI analyzes the necessary data to predict when maintenance is required, supporting studies like (Wubuli et al., 2025), on determining of preventive highway maintenance, (Faqih Seknun et al., 2025) on assessment of road maintenance to reduce potential environmental damage, and more.

2.3 Machine learning and Computer Vision

The field of computer vision has experienced significant growth due to the proliferation of machine learning technologies (Zhu & Shen, 2025). Computer Vision is about how computers deal with images, using the most advanced of machine learning features like deep learning, to perform tasks such as image processing, image classification, object detection, object segmentation, image colouring, image reconstruction, and image synthesis. Computer Vision techniques are widely applied across urban research, with methods tailored to specific study goals, as we can see in Figure 2.2.

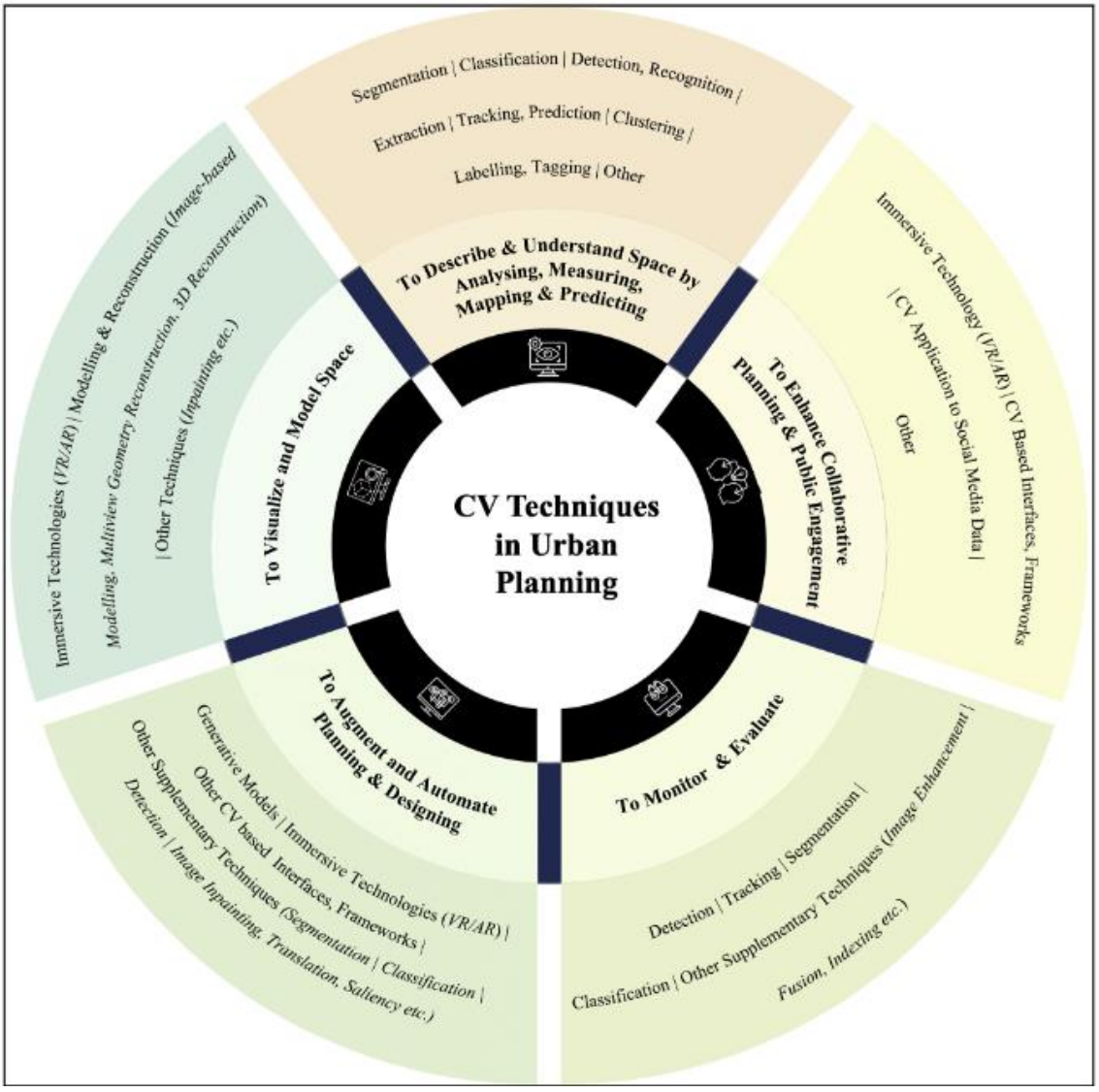


Figure 2.2: Application of CV techniques for different urban planning tasks.

Among these, in monitoring and evaluating, we have the essential task for this work:

- Image classification and detection algorithms for issue identification and data analysis.
- Object tracking for monitoring in implementation/evaluation phases.
- Scene classification and feature extraction for diverse analytical purposes, enable robust extraction of spatial and behavioral insights from visual data, supporting various stages of urban planning.

According to (Cernadas, 2024) computer vision applications involve the integration of elements such as in the table 2.1:

Table 2.1: Elements of Computer Vision.

Support for data recording	Type of input data	Machine vision-related aim of the application	Type of processing	Experimental testing
Microscopes; UAVs; satellites; robots; MRI, X-ray, and CT devices; and others	2D images, videos, radar, LIDAR	Detection or recognition, image segmentation, image classification, 3D modeling or reconstruction, object tracking, defect detection, object counting or measurements from images, and visual inspection, among others.	nonlearning-based methods, learning-based methods, and hybrid methods	Datasets

Machine learning encompasses three primary approaches: supervised, unsupervised, and reinforcement learning. Supervised learning uses labeled data to make predictions, with techniques like linear regression modeling straightforward relationships, and nonlinear regression handling more complex patterns. Unsupervised learning, in contrast, works with unlabeled data to uncover hidden structures through clustering (grouping similar data points), dimensionality reduction (simplifying data while preserving key features), and generative models (creating new, similar data). Finally, reinforcement learning operates on a trial-and-error basis, where an agent learns optimal actions by interacting with the environment and receiving feedback in the form of rewards, making it ideal for applications like game AI and robotics. Together, these methods provide powerful tools for extracting insights and building intelligent systems across diverse domains.

2.4 Deep Learning

Deep Learning(DL) is a branch of Machine Learning that focuses on artificial neural networks with multiple layers of interconnected neurons (Krauss, 2024), and the depth is defined by the numbers of layers. As in the brain, the neuron is also the fundamental processing unit in many areas of AI (Krauss, 2024). In recent years, deep learning (DL)

models have yielded a new generation of computer vision methods, such as convolutional neural networks (CNN) and transformers. CNNs are employed to analyze traffic image feeds, detecting congestion by recognizing patterns such as vehicle density, movement, and speed, making them effective for spatial pattern recognition in traffic data (D et al., 2025), and have become the standard DL-based approaches for many recognition tasks.

2.4.1 Convolutional Neural Networks

Convolutional neural network (CNN) uses weight sharing strategy to explore similar structures that occur in different locations in an image. Through sharing the convolutional weights locally for an entire image, this drastically reduces the amount of parameters that need to be learned and render the network equivalent with respect to translations of the input (i.e., the number of weights no longer depends on the size of input image) (Jiang et al., 2019). In CNN, convolutional layers work by gathering the input data, then filtering to detect specific features like edges, corners, or textures. Then a complete check on the data for similarities with the filters, the convolution process, producing a matching table or feature map. Then, the results are passed through an activation function that decides which patterns to keep.

$$X_k^{l+1} = \sigma(W_k^l \times X^l + b_k^l)$$

Formula 2.1

The formula says: take the input X^l, apply the convolution W_k^l, add the bias b_k^l, and pass it through the activation function σ to get the next layer's output X_k^{l+1}. As we can see summarised in Figure 2.3.

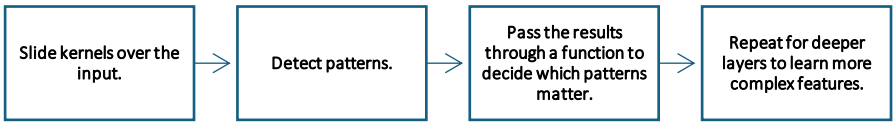


Figure 2.3: Process of a convolutional neural network.

The basic architecture of a convolutional neural network is shown in the figure below.

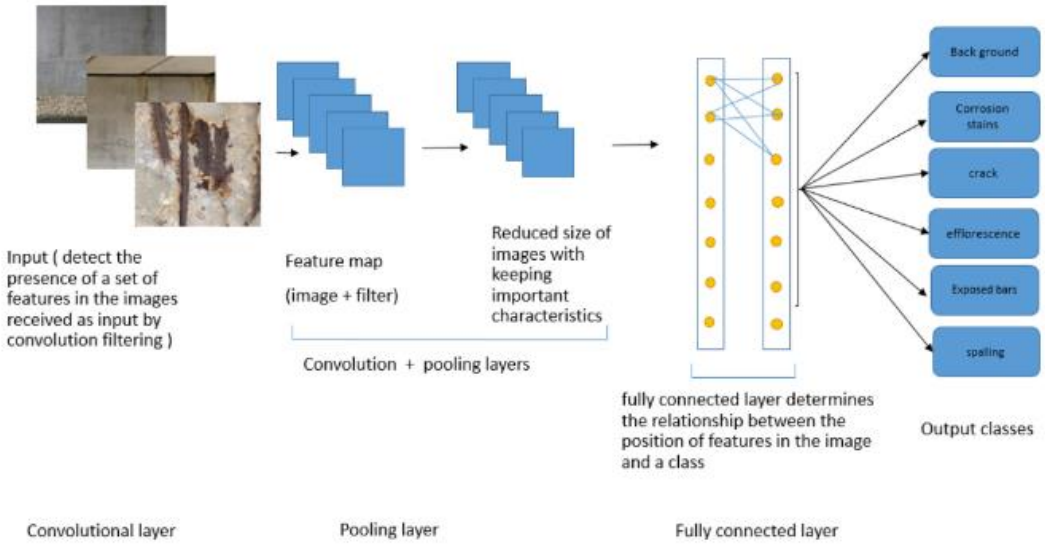


Figure 2.4: Convolutional Neural Network (CNN) architecture (Abubakr et al., 2024) .

The core component of CNNs is the convolutional layer, which is always at least their initial layer (Abubakr et al., 2024). As presents the figure 3, CNNs start with the convolution layer, applying a filter (kernel) to the input image. This kernel strides over the image, block by block, where each block is a collection of pixel cells. During this process, it performs matrix multiplication, which results in a lower resolution image. Typically, a CNN is structured in two main sections, feature extraction and the classification process. A basic CNN for classification task is made up by a convolution layer, Pooling layer, Activation function, Batch normalisation, Dropout, Fully connected layer.

Pooling layer: In short, the pooling procedure, like the convolution process, can be thought of as a pooling function without weights, in which the input feature mapping group is divided into many regions and each area is pooled to yield a value as a generalisation of this region (Zhao et al., 2024).

Activation function: An activation function called a rectified linear unit (ReLU) is one of the most popular DL activation functions that addresses the problem of vanishing gradients and adds the property of nonlinearity to a DL model (Abubakr et al., 2024)**,** it is a mathematical operation applied to the output of a filter. It serves a crucial role in neural networks by enhancing their representational power and learning ability. In a neural network, each layer's input and output involve a linear summation process, meaning the output of one layer is essentially a linear transformation of its input. The activation function's primary goal

is to provide the model with the nonlinearity property (Abubakr et al., 2024). This enables the neural network to approximate complex nonlinear functions, expanding its applicability to a broader range of nonlinear problems.

Batch normalisation: The whole idea of gradient descent is to minimise the objective function by iteratively updating the parameters in the opposite direction of the gradient of the objective function (Zhao et al., 2024). Gradient descent is an optimization technique that minimizes an objective function by iteratively adjusting parameters in the *opposite* direction of its gradient (since the gradient points in the direction of steepest ascent). The algorithm works by randomly initialized parameter value, then compute the gradient of the objective function at that point. Update the parameters by moving in the *negative* gradient direction, and repeat this process until the function value converges (changes negligibly) or a predefined iteration limit is reached.

Dropout: it is a regularisation technique that improves generalisation by randomly deactivating network units or connections with a fixed probability during training. This process creates multiple "thinned" network variants, and the resulting trained network, with its optimized weights, serves as an effective approximation of the ensemble of these variants (Figure 2.5b).

Fully connected layer: A fully connected layer is a global operation, unlike convolution and pooling, and is usually used at the end of a network for classification. Each neuron in the fully connected layer connects to all neurons in the previous layers (Figure 2.5a). After convolution and pooling extract sufficient image features, the fully connected layer handles classification.

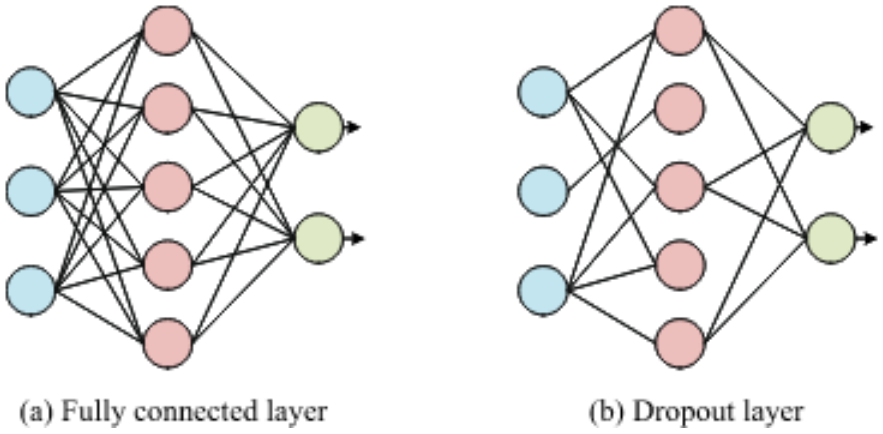


Figure 2.5: Distinction between a fully connected layer and dropout layer (Zhao et al., 2024)

Typically, CNNs flatten the final feature maps into a vector, which is then passed to a fully connected layer and output layer for classification. For instance, in a three-class image problem, the output layer would have three neurons. The fully connected layer also combines local, class-specific features from earlier convolution or pooling layers

In summary, CNNs process data through five key layers; pooling summarizes feature map regions, activation functions introduce nonlinearity for complex pattern learning, batch normalization stabilizes training by optimizing gradient descent, dropout prevents overfitting through random neuron deactivation, and fully connected layers integrate features for final classification; all working together to enable efficient extraction, transformation, and classification of hierarchical features from input data."

2.4.2 Image classification

Image classification is an algorithm that predicts a class label given an input image (Bird & Lotfi, 2024). CNNs represent one of the most powerful deep learning approaches for image classification. The main process of image classification includes preprocessing the original

image, extracting image features, and classifying the image using a pre-trained classifier, in which the extraction of image features plays a pivotal role (Zhao et al., 2024).

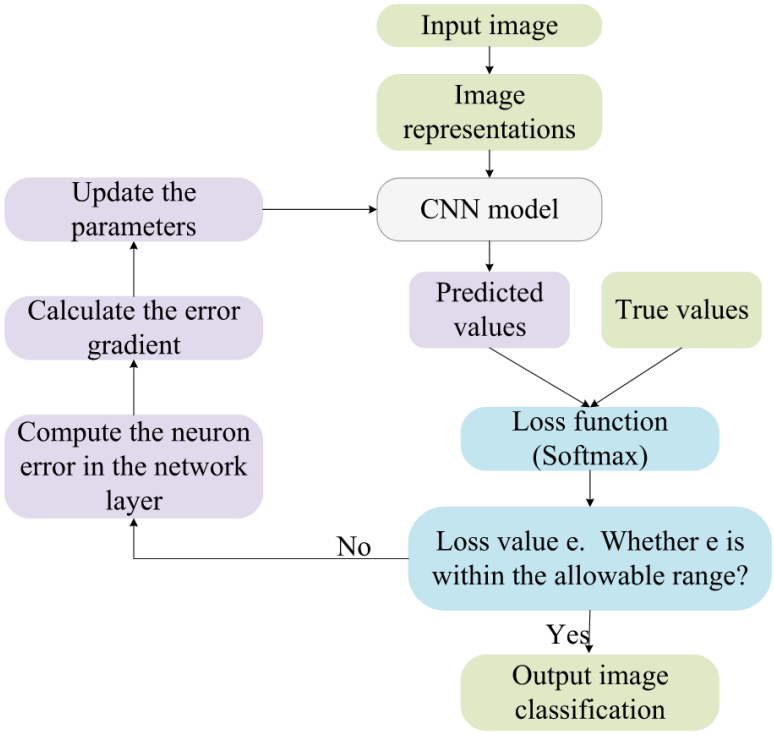


Figure 2.6: Data flow diagram for image classification (Zhao et al., 2024).

In Figure 2.6, the input image is processed by the CNN model which extracts features and generates predicted values, these are compared to the true values using a loss function (Softmax) to calculate the error, if the error exceeds the allowable range the model updates its parameters by computing the error gradient and adjusting neuron weights, repeating this process until the error falls within range, at which point the final image classification is output. For this, it is important for the data (image) to be effectively annotated (labelled). Where, image labelling consists in mapping visual features to semantic and spatial labels effectively describing image content, with "label" and "annotation" often used interchangeably in the literature (Sager et al., 2021). It comprises five steps:

1. *Data Collection*
2. *Labelling(annotation)*
3. *Postprocessing*
4. *Quality assessment*

5. *Data exportation*

The first step of image labelling is Data collection, which is gathering the images or videos, depending on the data, and describing the image with sentences, keywords, taxonomies, ontology, and others. This process can be done manually or automated by software (Image Labelling Software - ILS), depending on the goals. The figure below illustrates the concept of labeled data.

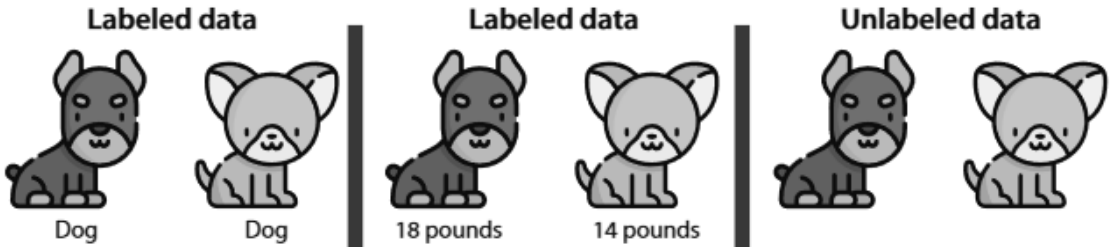


Figure 2.7: Example of labelled and unlabelled data. From (Serrano, 2021)

Let us understand that annotated data is data that comes with a tag or label, and the label can be a type or a number. As for unannotated or unlabelled data, it is the data that comes with no tag. Assessing the quality of the labelling is important for the performance of any supervised model, by interpreting errors and similarities to deal with bias. For this assessment, ILS like labelme, Roboflow, and others can be used.

2.4.3 Object detection

Object detection serves as a foundational computer vision task, enabling solutions for more advanced applications like image segmentation, object tracking, and activity recognition (Zhao et al., 2024). In recent years, researchers have concentrated on devising CNN-based object detectors to achieve real-time detection (A. Wang et al., 2024a). The process goes through training a classifier to distinguish the desired object and non-desired object in fixed-size image windows, assigning high scores to desired object and low scores to non-desired object.

There are two classes of deep learning object detection, the two-stages methods and one-stage methods, as we can see on the figure:

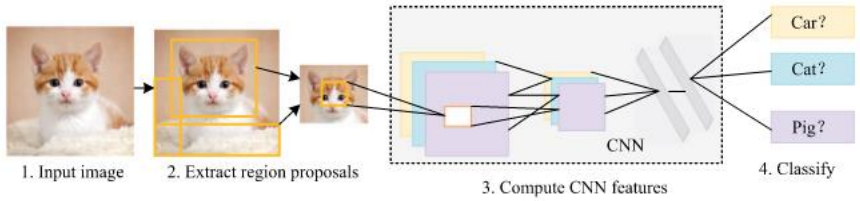


Figure 2.8: Two-stages detectors (Zhao et al., 2024).

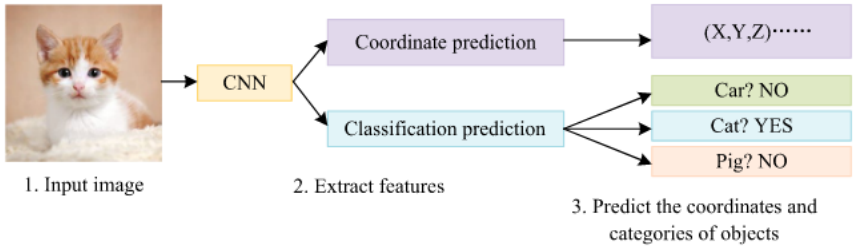


Figure 2.9: One-stage detector, from (Zhao et al., 2024).

Two-stages detectors

In Figure 2.8, we have the basic workflow of two-stage object detectors where first an input image is processed, then region proposals are extracted to identify potential object locations, after which CNN features are computed for each proposed region, and finally these features are classified to determine the object categories, demonstrating the sequential localization-then-classification approach characteristic of architectures like R-CNN, as shows the Figure 2.10; Faster Region-based Convolutional Neural Network (Faster R-CNN) which is an evolution of Fast Region-based Convolutional Neural Network (Fast R-CNN) as we can see on figure 2.11; Mask R-CNN and Spatial Pyramid Pooling-Net (SPP-Net).

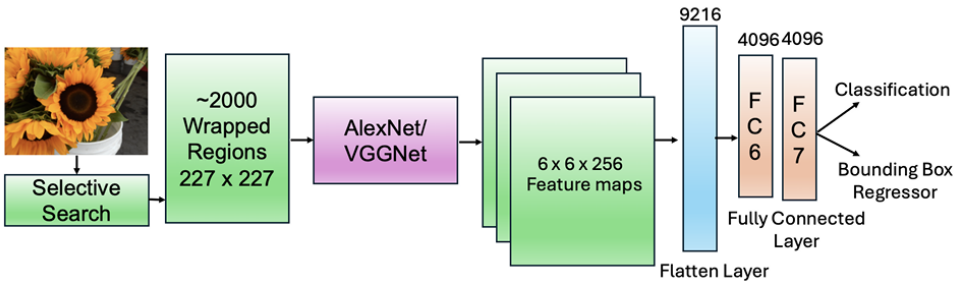


Figure 2.10: R-CNN Architecture(Neha et al., 2024)..

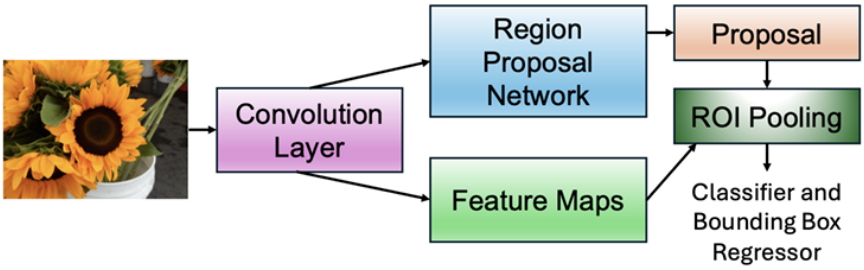


Figure 2.11: Fast R-CNN architecture (Neha et al., 2024).

One-stage detector

The one-stage detector, Figure 2.9, begins with an input image which is processed through feature extraction to generate hierarchical representations, then directly predicts both the bounding box coordinates for object locations and category probabilities for object classification in a single unified step, increasing speed by bypassing region proposals (Neha et al., 2024). Models like Single Shot MultiBox Detector (SSD) and You Look Only Once (YOLO) were developed achieving a high inference speed, as we can see in Table 2.2. However, compared to two-stage detectors, the detection accuracy is less accurate (Zhao et al., 2024).

For comparison purposes, the table below shows the percentage of the mean Average Precision(mAP) for different detectors.

Table 2.2: Quantitative Performance Comparison of Object Detection Models on different Dataset (Zhao et al., 2024).

Model	**Type**	**Pascal VOC (mAP)**	**COCO (mAP)**	**ImageNet (mAP)**	**Open Images (mAP)**	**Inference Speed (FPS)**	**Model Size (MB)**
RCNN	2-stage	66%	54%	60%	55%	~5	200
Fast RCNN	2-stage	70%	59%	63%	58%	~7	150
Faster RCNN	2-stage	75%	65%	68%	63%	~10	180
Mask RCNN	2-stage	76%	66%	69%	64%	~8	230
YOLO	1-stage	72.5%	58.5%	61.5%	57.5%	~45–60	145
SSD	1-stage	75%	63.5%	66.5%	61.5%	~19–46	145

Architectures like YOLO and SSD that uses one-stage detectors, it is prioritized speed as for they are often used in real-time applications. YOLO (You Only Look Once) has emerged as a key player in real-time object detection, and it exceeds other models in inference speed.

It is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy (Ultralytics, 2025).

2.5 Traffic Volume Impact on the Roads

Average daily traffic (ADT) and average annual traffic (AAT or LRH) are two types of traffic data important in transportation planning (Putri et al., 2024). Where, according to Putri et al. (2024), ADT refers to the number of vehicles that pass an observation point for 24 hours, while LHRT is the number of vehicles that pass an observation point for 24 hours calculated throughout the year. The capacity of road pavement construction is in terms of the number of repetitions (trajectories) of the load of the axis of the traffic wheel in a standard axle load unit known as the ESAL (Equivalent Single Axle Load) unit (Solahudin & Susanto, 2025). Where, to measure the damage that truck axles cause to roads, experts use a standard unit. This unit represents the damage from a single axle carrying 18,000 pounds (which is about 8 tons) (Putri et al., 2024; Solahudin & Susanto, 2025), and it is called "damage value of 1.", or Vehicle Damage Factor (VDF), essential for determining pavement thickness.

The AASHTO 1993 design method counts all the heavy vehicles that will use a road over its lifetime (W18)). Since traffic isn't spread evenly, it uses simple rules to focus only on the trucks in the busiest lane, which is the one that determines how thick the road needs to be. Table 2 analyzes how different types of trucks contribute to road damage over a year, where car is categorized by classes. Table 2.3 also considers truck load as critical factor; A single loaded truck like a 7a does thousands of times more damage than an empty one of the same class. This is quantified using "VDF" (Vehicle Damage Factors) and summed up into a final "ESAL" number, which represents the total wear and tear. The key takeaway is that a small number of overloaded heavy trucks (contributing to a total of 10,922 ESALs) are responsible for the overwhelming share of the pavement damage, which is equivalent to over 626,000 passes of a standard 18,000-pound axle.

Table 2.3: Overload ESAL and W18 values calculation for 2021, from (Putri et al., 2024)

Vehicle Class	Vehicle Axle	Fill/Empty	LHR 2021 Standard	LHR 2021 Overload	VDF Standard	VDF Overload	ESAL 2021
2	1.1	Standard	3125	0	0.0005	0.0005	1.56

3	1.1	Standard	783	0	0.0007	0.0007	0.55
4	1.1	Standard	595	0	0.0286	0.0286	17.01
5a	1.2	Standard	4	0	2.6	2.6	10.63
5b	1.2	Standard	7	0	2.6	2.6	19.24
6a	1.2 L	Fill	976	58	0.3	0.64	375.32
6a	1.2 L	Empty	309	0	0.1	0.004	30.86
6b	1.2 H	Fill	438	55	1	15.26	1317.99
6b	1.2 H	Empty	153	0	0.7	0.04	107.26
7a	1.22	Fill	416	55	10.1	11.74	4838.61
7a	1.22	Empty	165	0	2.7	0.02	444.73
7b	1.2 + 22	Fill	36	5	2.2	8.04	118.83
7b	1.2 + 22	Empty	16	0	1.4	0.01	22.62
7c	1.2 – 22	Fill	243	32	8.5	25.59	2885.36
7c	1.2 – 22	Empty	38	0	5.2	0.08	196.41
7c	1.2 – 222	Fill	64	8	3.3	22.66	399.95
7c	1.2 – 222	Empty	13	0	2.5	0.11	31.64
7c	1.22 - 222	Fill	10	1	4.7	29.97	90.78
7c	1.22 - 222	Empty	4	0	3.2	0.18	13.08
TOTAL ESAL Overload							**10922.38**
W_{18} Overload 2021							**626066.63**

To determine the percentage of traffic growth (i) during the service life of a road plan using the AASHTO (1993) method (Putri et al., 2024), we can use the following formula:

$$i = (\frac{ADT_n}{ADT_0})^{\frac{1}{n}} - 1$$

Formula 2.2

The cumulative ESAL can be computed as:

$$ESAL = \sum_{C} (ADT_c * VDF_c * Y)$$

Formula 2.3

Where ADT_c is the Annual Daily Traffic for vehicle class c, VDF_c is the Vehicle Damage Factor (based on axle type and weight), and Y is the number of design years.

The remaining pavement life can be estimated as:

$$Remaining\ Life(\%) = 100(1 - \frac{Actual\ ESAL}{Design\ ESAL})$$

Formula 2.4

2.5.1 An AI & Computer Vision Approach for Vehicle Counting and Classification

AI and computer vision system, using real-time object detection models, it identifies each vehicle and assigns it a unique ID. Sophisticated tracking algorithms then follow each vehicle's movement frame-by-frame, ensuring the same vehicle is never counted twice as it passes through the monitored area.

The system classifies each vehicle into predefined categories (car, truck, bus, motorcycle) and tracks its movement to determine traffic direction and count. Based on this classification and the established Vehicle Damage Factors (VDFs) for each class, the system automatically calculates the Equivalent Single Axle Load (ESAL), providing a direct metric for assessing the pavement impact of the observed traffic flow. Additionally, the system calculates the average speed of vehicle according to the direction of the vehicles.

Finally, all this analyzed information is automatically saved into CSV file or database. This directly generates the traffic data needed for urban planning and traffic analyses.

2.6 Related Works

Significant research such as Zhao et al. (2024) and Neha et al. (2024), have explored the integration of artificial intelligence (AI) and computer vision (CV) in car detection, classification, and tracking. Studies by Yigitcanlar et al. (2020) and Abubakr et al. (2024) demonstrate the utility of such systems in traffic flow analysis, congestion management, and infrastructure planning. Kamrowska-Załuska (2021) emphasized the importance of big data mining and AI in studying dynamic urban systems, highlighting the role of image recognition in mapping traffic patterns and enabling smart city innovations. The ANST model, developed by Nadarajan & Sivanraj, (2022), enhances traffic forecasting by merging LSTM networks with attention mechanisms, effectively incorporating spatiotemporal relationships and environmental conditions for superior predictive performance. By integrating street view

images and urban networks, Yap et al (2023), assessed active mobility, leveraging deep learning to examine the impact of traffic environment factors on subjective choices. To contend traffic congestion on urban networks, a recent DQL framework by H. Wang et al (2023), using partial detector inputs showed 3.9-22% improvements over conventional methods in real-world validation. Latest study by D et al (2025) AI-based traffic systems combine real-time data and machine learning for accurate congestion detection (94.89% accuracy) and adaptive signal control, significantly improving traffic flow over traditional methods.

Recent advances in vehicle recognition have been driven by YOLO (You Only Look Once) architectures, which enable real-time object detection critical for urban traffic management. A work by Valdovinos-Chacón et al2025) presented a YOLO-based system that achieves 96% vehicle detection accuracy for adaptive traffic light control, demonstrating potential for Latin American cities, combining real-time object detection with IoT coordination to optimize intersection timing. Tracking algorithms like ByteTrack, Botsort and DeepSORT have shown promise. ByteTrack's innovative association of low-confidence detections demonstrates significant improvements (up to +10 IDF1) for urban traffic monitoring, achieving real-time performance (30 FPS) with 80.3 MOTA accuracy (Zhang et al., 2022), which is particularly valuable for smart city applications.

Table 2.4: Related work summary.

Authors	Year	Contribution
Zhang et al.	2022	ByteTrack's innovative association of low-confidence for object tracking.
Nadarajan & Sivanraj	2022	Enhancement of traffic forecasting by merging LSTM networks with attention mechanisms.
Abubakr et al.	2024	Utility of AI and Computer vision in traffic flow analysis, congestion management, and infrastructure planning.
Zhao et al.	2024	Integration of artificial intelligence (AI) and computer vision (CV) in car detection, classification, and tracking.
Valdovinos-Chacón et al.	2025	Presented a YOLO-based system that achieves 96% vehicle detection accuracy for adaptive traffic light control.
D et al.	2025	Congestion detection with AI-based traffic systems that combine real-time data and machine learning.

Table 2.4 summarizes the latest relevant contributions on vehicle recognition with AI, however, most existing work focuses on either traffic analysis or infrastructure monitoring in isolation. Few studies comprehensive address how vehicle recognition can directly inform urban planning decisions for predictive roads maintenance, a gap this research aims to bridge by developing an integrated framework that connects real-time vehicle analytics with long-term urban development strategies.

3. System Design and Methodology

3.1 Research Methodology and Development Approach

This work employs the Design Science Research (DSR) methodology, based on De Sordi, (2021), to design and evaluate a vehicle tracking system framework. DSR is chosen for its focus on developing IT artifacts that solve practical organizational issues while maintaining scientific rigor.

The implementation of DSR followed a structured three-phase development approach, as illustrated in Figure 3.1. First, a computer vision model was trained on an annotated dataset to establish the core detection capability. Second, this model was implemented and optimized within a functional software prototype. Third, the system was extended to extract, analyse, and persist useful traffic information.

Figure 3.1: Development approach

This process ensured the creation of an artifact that fulfils the need for real-time monitoring of vehicles and road usage. The practical application of this work shows its relevance on smart cities planning and intelligent transportation, considering that vehicle detection, tracking and counting can help urban planning and predictive maintenance of infrastructures by analysing the volume of traffic, improving the flow of vehicle and the impact on the road degradation.

3.2 System requirements

To guide the development of the system, a comprehensive set of requirements was established covering functional capabilities, quality attributes, and domain-specific constraints. These requirements ensure the system meets both technical objectives and practical urban planning needs.

- Functional requirements

Table 3.1: Functional requirements of the system

Code	Requirements	Description
FR01	Vehicle detection	The system must detect accurately vehicles on video frames
FR02	Object tracking	The system needs to rigorously maintain identity of vehicles across frames
FR03	Counting vehicles	The system will define a line to count the vehicles that cross it.
FR05	Different data source	It needs to be capable of processing data from different sources, stream and recorded data.
FR06	Calculate Esal	The system needs to calculate daily ESAL variable
FR07	Save reports	It needs to create a database for recording the daily traffic volume data
FR07	Web Interface	The system must provide a web interface that displays the processed video with the detections and counts updated in real time.

- Non-functional requirements

Table 3.2: Non-functional requirements.

Code	Non-Functional Requirement	Description
NFR01	Performance:	The system must process a minimum of 15 frames per second (FPS) on an Intel i5 CPU and NVIDIA RTX 1070 GPU.
NFR02	Reliability:	The system must maintain availability greater than 99% during operation, with error handling for unstable video sources.
NFR03	Maintainability	The system must have high cohesion and low coupling, with a maintainability index greater than 70 (measured by tools such as SonarQube).
NFR04	Scalability	The architecture must support multiple concurrent tracing sessions, with resource isolation.
NFR05	Low Latency	End-to-end latency (capture from frame to display on the interface) must be less than 500 milliseconds.

- Domain Requirements

Table 3.3: Domain requirements

Code	Domain-Requirements	Description
DR01	Vehicle Classes	Recognize the classes: car, truck, bus and motorcycle.
DR02	Directional traffic analysis	Up/down for horizontal lines, left/right for vertical lines.

DR03	Pavement impact estimation	ESAL per class with domain-specific Vehicle Damage Factors (VDFs).
DR04	Speed estimation	Speed estimation: approximate scene-based conversion (pixels→meters) for indicative averages.

These requirements collectively ensure the development of a technically robust system specifically designed for urban intelligence applications. They establish the foundation for delivering accurate, real-time traffic analytics to support data-driven urban planning and predictive maintenance, directly informing the architectural design that follows.

3.3 Proposed System Architecture

This section outlines the architectural structure principles and patterns adopted for the vehicle detection and tracking prototype, ensuring a robust, maintainable, and scalable system. The architecture is designed to align with the domain requirements, emphasizing modularity, testability, and performance optimization.

3.3.1 Architectural Principles and Patterns

Clean Architecture

The system is designed according to Clean Architecture principles (Lano & Yassipour Tehrani, 2023), ensuring that business rules remain independent of frameworks, databases, and external systems. This separation improves maintainability and testability by isolating the core logic from infrastructure dependencies. Applying Domain-Driven Design (DDD)(Junker & Lazzaretti, 2025; Kapferer & Zimmermann, 2020), enables the creation of a rich domain model for vehicle detection and counting, supported by a shared ubiquitous language between developers and domain experts. The use of Command Query Responsibility Segregation (CQRS) further enhances performance by separating read operations (e.g., metric queries) from write operations (e.g., frame processing) («Implementing Command Query Responsibility Segregation (CQRS) in Large-Scale Systems», 2024), allowing each to be independently optimized and scaled for real-time processing.

The architecture also follows the Ports and Adapters (Hexagonal) pattern, which decouples the application's core from external systems such as hardware interfaces, databases, or APIs, thereby facilitating component substitution without impacting the core logic. In addition, the system adopts an Event-Driven Architecture, where domain events propagate significant occurrences, such as vehicle crossings, across components. This promotes loose coupling, extensibility, and integration with external systems, including traffic management or analytics platforms. Collectively, these architectural patterns ensure a modular, scalable, and maintainable system for vehicle detection and tracking, while providing flexibility for future enhancements.

3.3.2 Structural Diagrams

Figure 3.2 shows the main components and data/control flows of the proposed system, where the browser UI communicates with the FastAPI backend over WebSocket for real-time frames and metrics and via REST for configuration and exports, while the vision pipeline (FrameReader → YOLO → BYTETrack → VehicleCounter) processes frames from webcams, uploaded files, or YouTube via yt-dlp to produce counts, directional speeds, and ESAL summaries.

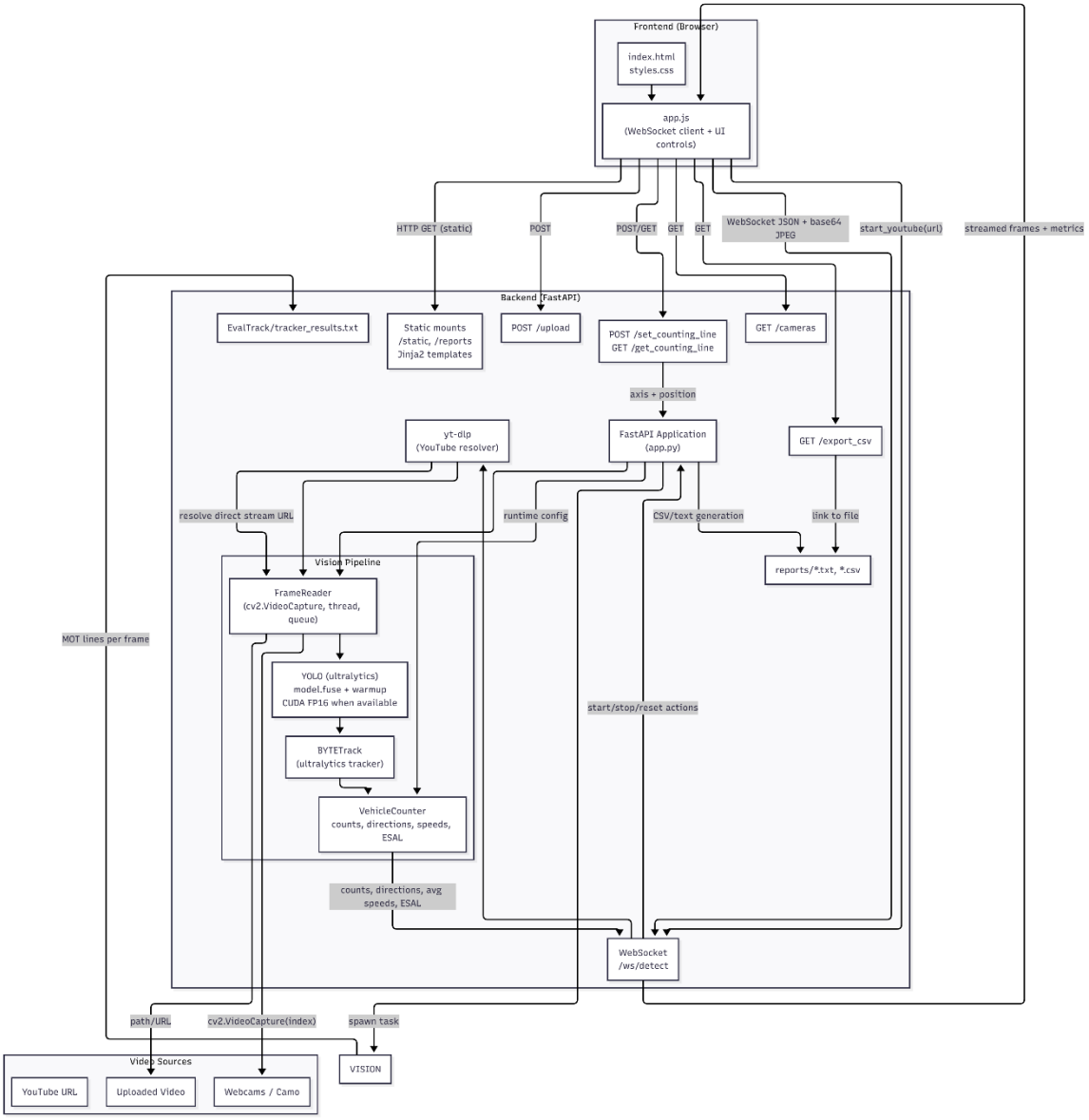


Figure 3.2: Component diagram of the real-time vehicle tracking system (Frontend, FastAPI backend, vision pipeline, sources, and outputs)

The component diagram illustrates a streaming-first design: frames flow from the selected source into the vision pipeline where YOLO performs detection, BYTETrack maintains identities, and VehicleCounter computes per-class and per-direction counts, speeds, and ESAL. The backend serves both static assets and dynamic reports (CSV/TXT), and logs MOT-style outputs for evaluation. The counting line can be set to auto or manual mode via REST, and changes propagate at runtime to maintain coherent direction metrics.

The sequence traces user-initiated actions (Start/Stop, configuration updates) through the backend's streaming pipeline and back to the browser via WebSocket, showing the per-frame loop and report generation, as seen in Figure 3.3.

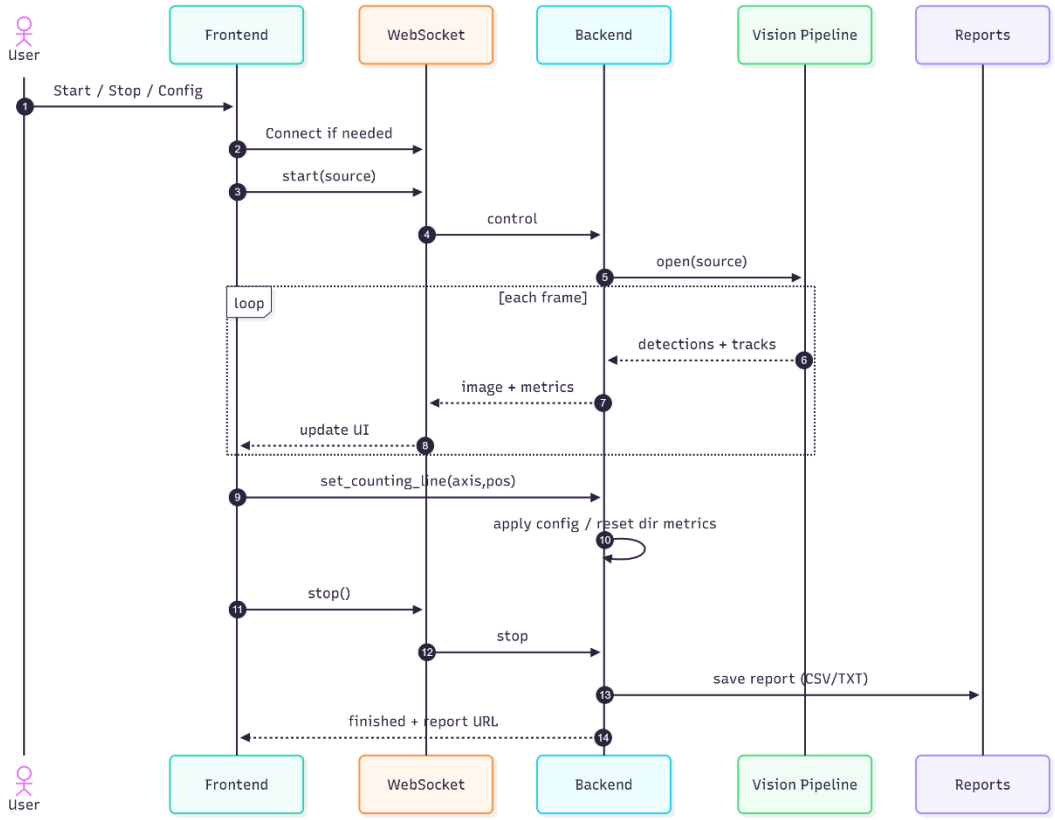


Figure 3.3: Main sequence of operations from user action to real-time processing and reporting.

- User initiates processing (Start via uploaded file, webcam, or YouTube) and may later Stop or change the counting line.

- Frontend (app.js) ensures a WebSocket connection and sends simple control messages (start/stop); REST endpoints handle configuration and exports.

- Backend (FastAPI) spawns a processing task that opens the source and runs the vision pipeline.

- Vision pipeline: FrameReader acquires frames; YOLO detects vehicles; BYTETrack assigns track IDs; VehicleCounter updates counts, directions, speed samples, and ESAL.

- Backend overlays, packages a JPEG frame plus metrics, and pushes them over WebSocket; the UI renders the image and updates the merged metrics table.

- When the counting axis/position changes, backend applies the config and resets directional metrics to preserve semantic correctness.

- On Stop or stream end, backend saves a TXT report and supports CSV export from /export_csv, returning a downloadable URL.

3.3.3 Core Domain Models

This section describes the main domain objects that encapsulate the business logic of traffic counting, directional analysis, and pavement impact estimation.

1. VehicleCounter (backend/app.py):

- Purpose: Central domain service that turns tracked object motion into domain metrics: per-class totals, direction splits (up/down or left/right), average speeds, and ESAL.
- Core state:
    - previous_positions, previous_times: last known center and timestamp per track_id.
    - counted_ids: track_ids already counted to prevent double counts.
    - vehicle_counts: totals per class (car, truck, bus, motorcycle).
    - up_down_counts, left_right_counts, vehicle_direction_counts: direction-split counts per class.
    - counted_speeds_* (up/down/left/right/all): speeds recorded at the moment of crossing for accurate averaging.
    - motion_dx_sum, motion_dy_sum, motion_samples: motion statistics to infer dominant axis.
    - last_counting_axis: last effective axis used for counting in this session.
- Behavior:
    - update(track_id, center_x, center_y, class_name, counting_axis, counting_line_pos, timestamp, frame_dim)

- Estimates direction along active axis; detects single crossing per track and updates all aggregates.
- Records “counted-at-crossing” speeds and contributes to motion statistics (dx/dy).

▪ get_total_counts()

- Returns a merged metrics view: overall totals, ESAL by class and total, per-direction counts and ESAL, and average speeds (overall and by direction).

▪ get_direction_counts(), reset_directional_metrics(), get_dominant_axis(min_samples)

- Direction-aware views, safe axis switching (resets directional aggregates), and automatic axis selection based on observed motion.

- Contract (inputs/outputs):
  - ▪ Input: track_id (int), object center (x, y), class_name, counting_axis (‘x’|‘y’), line position (px), timestamp, frame_dim.
  - ▪ Output: optional {track_id, class, direction} upon a confirmed crossing; totals via get_total_counts().

2. Counting Configuration (runtime model):

- counting_config = { axis: 'x'|'y'|null, pos_frac: 0.0–1.0 }
- Semantics: null axis = auto; pos_frac is normalized position. Changing the effective axis triggers reset_directional_metrics() to keep direction semantics consistent.
- Companion: counter.last_counting_axis captures the runtime-effective axis used in the current processing loop.

3. Detection/Tracking Entities (conceptual):

- Detection: {box (x1,y1,x2,y2), class_name, conf}
- Track: detection + stable track_id assigned by the tracker, used by VehicleCounter to compute motion and crossings.

4. ESAL Model (calculate_esal):

- Vehicle Damage Factors (VDF): car=0.0005, motorcycle=0.0001, bus=0.15, truck=2.0.

- ESAL per class = count × VDF; totals computed both overall and per direction for maintenance planning.

3.3.4 Runtime Components

This subsection summarizes the concrete components that execute the pipeline and expose the system at runtime.

1. Frontend UI (frontend/)

- templates/index.html: Controls (Start/Stop/Reset, Upload, YouTube, Counting mode/position) and merged metrics table (counts, ESAL, average speeds by direction).
- static/js/app.js: WebSocket client to receive frames/metrics; sends actions (start, start_youtube, start_camo, stop, reset_counts); REST for configuration (/set_counting_line, /get_counting_line) and exports (/export_csv, /cameras).
- static/css/styles.css: Visual layout and readability.

2. FastAPI Application (backend/app.py)

- Endpoints:
    - GET /: render UI; GET /cameras: quick camera listing.
    - POST /upload: persist file; POST /set_counting_line, GET /get_counting_line: runtime counting config.
    - GET /export_csv: build CSV and return download URL via /reports.
- WebSocket /ws/detect:
    - Receives control actions: start (uploaded), start_youtube (yt-dlp), start_camo (webcam), stop, reset_counts.
    - Streams base64-encoded JPEG frames and metrics (counts, direction metrics, ESAL, average speeds, counting axis, last count direction).

3. Vision Pipeline (app.py):

- FrameReader: threaded cv2.VideoCapture with a bounded queue; supports path/URL/camera index.
- YOLO (Ultralytics): loads yolo11n.pt; fuses and warms up; uses CUDA FP16 when available; constrained by INFER_SIZE and MAX_DET.
- BYTETrack (via Ultralytics tracker): maintains stable track IDs for counting.
- VehicleCounter: translates tracks into counts, direction metrics, average speeds, and ESAL.

4. Sources and Resolvers

- Uploaded videos (backend/uploaded_videos), webcams (list_cameras/find_camo_camera), YouTube (yt_dlp to direct stream URL).

5. Reporting and Evaluation

- make_report_text() and make_report_csv(): save TXT/CSV under backend/reports, mounted at /reports.
- MOT-style lines written to EvalTrack/tracker_results.txt for later evaluation.

6. Performance and Environment

- Knobs: INFER_SIZE, JPEG_QUALITY, SEND_EVERY, MAX_DET; torch.backends.cudnn.benchmark = True.
- NumPy compatibility guard (1.26.x); CUDA used if available with CPU fallback.

These models form the foundation of the system, supporting extensibility for multiple detection algorithms and counting strategies.

3.3.5 Summary table: Core Domain Models and Runtime Components

Table 3.4: Core Domain Models and Runtime Components summary.

Component	Type	Responsibilities	Key methods/APIs	Core data/state
VehicleCounter	Domain	Track per-ID motion, decide counting axis, count by direction, aggregate speeds/ESAL inputs	update; get_total_counts; reset_directional_metrics; get_dominant_axis	vehicle_counts; up/down/left/right splits; counted_speeds_*; speeds; last_counting_axis; previous_positions/times
ESAL calculator	Domain/utility	Compute ESAL totals by class and by direction	calculate_esal	VDF weights; esal_by_class;

				esal_by_direction; totals
Speed averaging	Domain/utility	Robust average speeds overall and per direction	calculate_direction_speeds; calculate_average_speed	counted_speeds_*; speeds; window, max_kmh
Counting config	Domain/config	Select axis and line position (auto or override)	GET/POST /get_counting_line, /set_counting_line	counting_config: axis, pos_frac
FastAPI service	Runtime	Serve UI, REST, and WebSocket	FastAPI app; routes: /, /upload, /cameras, /export_csv	reports_dir; uploaded_video_path; last_session_start/finish
WebSocket control loop	Runtime	Handle start/stop/youtube/camera actions; push frames/metrics	/ws/detect; websocket_endpoint	processing_task; stop_event
Video processing loop	Runtime	Read frames, run YOLO+BYTETrack, update counts, draw overlays, stream JPEG	process_video_stream	INFER_SIZE, JPEG_QUALITY, SEND_EVERY, MAX_DET; counting_line_pos; fps; payloads
Detection+Tracking	Runtime (ML)	Class-filter detections, tracking IDs, per-frame MOT export	model.track(..., tracker="bytetrack.yaml", classes=[2,3,5,7])	model/device (FP16 on CUDA); MOT result file
FrameReader	Runtime/helper	Non-blocking frame ingestion (optional pattern)	start; get; release	background thread; queue; cap
Rendering/Encoding	Runtime/helper	Draw boxes/labels/line/arrow; JPEG encode	draw_detections_on_frame; cv2.imencode	CLASS_COLORS; counting line overlay
Reporting/Export	Runtime/helper	Generate TXT and CSV reports	make_report_text; make_report_csv; GET /export_csv	files under /reports; report_url

3.4 System Implementation

3.4.1 Technologies and Tools

The implementation of the system employed a carefully selected set of technologies and tools, based on the criteria of maturity, performance, community support and adequacy to the established architectural requirements. The selection followed the guidelines of («A Comprehensive Guide to AI Tech Stack», 2025) for choosing a technological stack in computer vision projects.

Table 3.5: Technologies and tools.

Layer	Technology/ Tool	Version	Justification	Role in Architecture
Presentation	FastAPI	0.104+	Superior performance, native WebSocket and async/await support	Framework Web and API
	WebSocket	Padrão	Real-time two-way communication for video and data transmission.	Real-time protocol
	HTML5, CSS3, JavaScript	ES6+	Responsive Interface web and cross-platform	Frontend interface
Application	Python	3.10+	Broad support for computer vision and ML, clear syntax, vast library ecosystem.	Main Language
	asyncio	3.10+	For concurrency and non-blocking I/O operations.	Asynchronous Processing
	Pydantic	2.0+	Data validation with Python types	DTOs and validation
Domain	Pytorch	2.1+	ML framework with GPU support and widespread adoption in detection models.	Machine learning
	Ultralytics	8.0+	YOLO implementation, simplified APIs	Object detection
	Bytrack		Robust multi-object tracking algorithm to occlusions and low FPS.	Multi-object Tracking
Infrastructure	OpenCV	4.8+	Optimized image/video processing	Computer Vision
	NumPy	1.26+	Efficient numerical computation	Manipulating arrays
	CUDA	11.8+	NVIDIA GPU Acceleration	Hardware acceleration
	yt-dlp	2023+	Extracting YouTube streams	External video sources

FastAPI was chosen for its high performance, automatic OpenAPI documentation, native dependency injection, and active community. PyTorch was selected over TensorFlow for its intuitive interface, better debugging, strong research ecosystem, rapid prototyping capabilities, and compatibility with Ultralytics YOLO.

Critical performance settings:

```
TORCH_BACKENDS = {
    'cudnn.benchmark': True,      # Optimizes convolutions for fixed sizes
    'cudnn.deterministic': False, # Allows for non-deterministic
optimizations
    'matmul.allow_tf32': True,    # Mixed Accuracy for Operations
}

INFERENCE_CONFIG = {
    'imgsz': 640,                 # Precision-speed balancing
```

```
    'fp16': True,                # Mixed Precision for Modern GPU
    'max_det': 100,              # Limit detections per frame
    'conf': 0.45,                # Optimized Confidence Threshold
}
```

Moreover, dependency and environment management rely on Poetry for streamlined dependency handling and virtual environments, Git for version control using Conventional Commits, and pre-commit hooks for automatic code validation.

3.4.2 Applied Design Patterns

The implementation of the system incorporated several design patterns (Gamma et al., 1994) to ensure modular, extensible, and easy-to-maintain code. The standards applied are detailed below:

1. **Adapter Pattern** - Integration with external models:

```
class IVehicleDetector(ABC):
    @abstractmethod
    def detect(self, frame: np.ndarray) -> List[VehicleDetection]:
        pass

class YOLOVehicleDetector(IVehicleDetector):
    def __init__(self, model_path: str, config: ModelConfig):
        self.model = YOLO(model_path)  # Adapts YOLO  interface
        self.config = config

    def detect(self, frame: np.ndarray) -> List[VehicleDetection]:
        # Tailors YOLO results for domain
        results = self.model.predict(frame, **self.config)
        return self._parse_detections(results)
```

2. **Strategy Pattern** - Interchangeable algorithms:

```
class ICountingStrategy(ABC):
    @abstractmethod
    def count_vehicles(self, detections: List[VehicleDetection],
                       line_position: float) -> CountingResult:
        pass

class LineCrossingStrategy(ICountingStrategy):
    def count_vehicles(self, detections: List[VehicleDetection],
                       line_position: float) -> CountingResult:
        # Specific implementation of line crossing
        pass

class AreaBasedStrategy(ICountingStrategy):
```

```
    def count_vehicles(self, detections: List[VehicleDetection],
                       area: Polygon) -> CountingResult:
        # Alternative Area-Based Counting Implementation
        pass
```

3. **Factory Pattern** - Flexible object creation:

```
class DetectorFactory:
    @staticmethod
    def create_detector(detector_type: str, config: DetectorConfig) ->
IVehicleDetector:
        if detector_type == "yolo":
            return YOLOVehicleDetector(config.model_path, config)
        elif detector_type == "efficientdet":
            return EfficientDetDetector(config)
        else:
            raise ValueError(f"Unsupported detector: {detector_type}")

class VehicleDetectionFactory:
    @staticmethod
    def from_yolo_result(box, track_id, class_name, frame_id: int) ->
VehicleDetection:
        return VehicleDetection(
            detection_id=uuid.uuid4(),
            track_id=int(track_id),
            vehicle_class=VehicleClass(class_name),
            bounding_box=BoundingBox(*box.xyxy[0].tolist()),
            confidence=float(box.conf),
            timestamp=datetime.now(),
            frame_id=frame_id
        )
```

Applied Behavioral Patterns:

4. **Observer Pattern** - Real-time notifications:

```
class TrackingSubject:
    def __init__(self):
        self._observers: List[TrackingObserver] = []

    def attach(self, observer: TrackingObserver):
        self._observers.append(observer)

    def notify_vehicle_crossed(self, event: VehicleCrossedEvent):
        for observer in self._observers:
            observer.on_vehicle_crossed(event)

class WebSocketObserver(TrackingObserver):
    def __init__(self, websocket: WebSocket):
        self.websocket = websocket
```

```
    async def on_vehicle_crossed(self, event: VehicleCrossedEvent):
        await self.websocket.send_json({
            "type": "vehicle_crossed",
            "vehicle": event.vehicle.to_dict(),
            "timestamp": event.timestamp.isoformat()
        })
```

5. **Template Method Pattern** - Processing pipeline:

```
class VideoProcessingPipeline(ABC):
    def process_frame(self, frame: np.ndarray) -> ProcessingResult:
        # Fixed skeleton, variable steps
        preprocessed = self.preprocess(frame)
        detections = self.detect_vehicles(preprocessed)
        tracked = self.track_vehicles(detections)
        result = self.analyze_results(tracked)
        return result

    @abstractmethod
    def preprocess(self, frame: np.ndarray) -> np.ndarray:
        pass

    @abstractmethod
    def detect_vehicles(self, frame: np.ndarray) -> List[VehicleDetection]:
        pass
```

Additional Creational Standards:

Builder Pattern - Complex session construction:

```
class CountingSessionBuilder:
    def __init__(self):
        self.session = CountingSession()

    def with_video_source(self, source: VideoSource):
        self.session.video_source = source
        return self

    def with_counting_line(self, position: float):
        self.session.counting_line = CountingLine(position)
        return self

    def with_strategy(self, strategy: ICountingStrategy):
        self.session.counting_strategy = strategy
        return self

    def build(self) -> CountingSession:
        self.session.validate()
        return self.session
```

3.5. Core Algorithms

The computer vision pipeline implements state-of-the-art algorithms optimized for the vehicle tracking domain, following the approaches established in recent surveys (Zhu et al., 2024). The processing pipeline consists of the following steps:

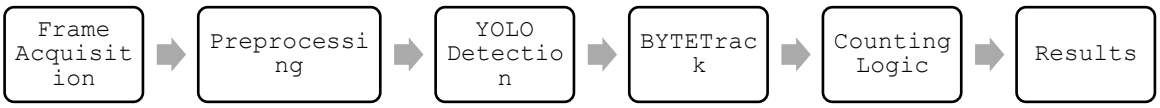


Figure 3.4: Computer vision pipeline for the system.

3.5.1 Vehicle Detection with YOLO

YOLO (You Only Look Once) is an object detection architecture that performs real time detection. The YOLOv11n (nano version) was chosen because it offers a balance between speed and accuracy, essential for real-time applications. Detection is performed on each frame, producing bounding boxes and confidence scores (Neha et al., 2024; Zhao et al., 2024).

Technical Architecture:

```
YOLO_CONFIG = {
    'backbone': 'CSPDarknet',     # Efficient feature extraction
    'neck': 'PAN-FPN',            # Feature Pyramid Networks
    'head': 'Anchor-free',        # Reduced complexity
    'activation': 'SiLU',         # Modern nonlinearity
    'normalization': 'BatchNorm', # Training Stability
}

# Specific optimizations implemented:
class OptimizedYOLOProcessor:
    def __init__(self):
        self.model = self._load_optimized_model()

    def _load_optimized_model(self):
        model = YOLO('yolo11n.pt')
        if torch.cuda.is_available():
            model = model.half()  # FP16 for speed
            model = model.fuse()  # Fusion layers for efficiency
        return model
```

3.5.2 Multi-Object Tracking with BYTETrack

BYTETrack (Zhang et al., 2022) is a multi-object tracking algorithm that associates detections between frames using a strategy of associating high-confidence detections first and then low-confidence detections, reducing identity switches. The tracker uses the bounding boxes and scores provided by YOLO and associates them based on spatial similarity (using IoU - Intersection over Union) and the motion predicted by a Kalman filter. The output is a list of vehicles with consistent unique IDs throughout the video.

Association Algorithm:

```
class BYTETracker:

    def track(self, detections: List[VehicleDetection]) ->
List[VehicleDetection]:
        # Separates detections by trust
        high_conf_dets = [d for d in detections if d.confidence > 0.5]
        low_conf_dets = [d for d in detections if 0.1 < d.confidence <= 0.5]
        # First association: high trust
        tracks_updated = self._associate(high_conf_dets, self.active_tracks)

        # Second association: low confidence with non-associated tracks
        remaining_tracks = [t for t in self.active_tracks if t not in
tracks_updated]
        tracks_updated += self._associate(low_conf_dets, remaining_tracks)

        # Booting new tracks
        new_tracks = self._init_new_tracks(high_conf_dets)
        return tracks_updated + new_tracks

    def _associate(self, detections, tracks) -> List[VehicleTracking]:
        # Uses IoU and motion predicted by Kalman Filter
        cost_matrix = self._compute_iou_cost(detections, tracks)
        matches, unmatched = self._linear_assignment(cost_matrix)
        return self._update_matched_tracks(matches, detections, tracks)
```

Kalman Filter for Motion Prediction:

```
class VehicleKalmanFilter:
    def __init__(self):
        # State: [x, y, w, h, vx, vy, vw, vh]
        self.kf = cv2.KalmanFilter(8, 4)
        self._setup_transition_matrix()

    def predict(self, track: VehicleTracking) -> np.ndarray:
```

```
        self.kf.predict()
        return self.kf.statePost

    def update(self, detection: VehicleDetection):
        # Measurement: [x, y, w, h]
        measurement = np.array([
            detection.bounding_box.center[0],
            detection.bounding_box.center[1],
            detection.bounding_box.width,
            detection.bounding_box.height
        ], dtype=np.float32)
        self.kf.correct(measurement)
```

3.5.3 Virtual Line Counting Algorithm and Speed Average Calculation

The count is performed by means of a virtual line positioned at a fixed x/y-coordinate in the frame (K% of the height/width of the frame). For each vehicle tracked, the central position of the bounding box is calculated, and the virtual line has been crossed by comparing the current position with the previous one. For instance, in case of vertical circulation, the crossing is recorded when the vehicle's previous position is above the line and the current one below (or vice versa, depending on the direction set).

Virtual Line Counting Algorithm:

```
class LineCrossingAlgorithm:
    def __init__(self, line_y: float, direction: str = "downward"):
        self.line_y = line_y
        self.direction = direction
        self.counted_ids = set()
        self.track_history = {}  # {track_id: [y_positions]}
    def check_crossing(self, detection: VehicleDetection) -> bool:
        track_id = detection.track_id
        current_y = detection.bounding_box.center[1]
        if track_id not in self.track_history:
            self.track_history[track_id] = []

        # Maintains limited history
        self.track_history[track_id].append(current_y)
        if len(self.track_history[track_id]) > 5:
            self.track_history[track_id].pop(0)

        # Checks for direction-based crossing
        if self.direction == "downward":
            return self._check_downward_crossing(track_id, current_y)
        else:
```

```
            return self._check_upward_crossing(track_id, current_y)

    def _check_downward_crossing(self, track_id: int, current_y: float) ->
bool:
        if track_id in self.counted_ids:
            return False

        history = self.track_history[track_id]
        if len(history) < 2:
            return False

        # Crossing: Was up, now it's down the line
        previous_y = history[-2]
        return previous_y <= self.line_y < current_y
```

Average speed calculation:

```
def calculate_average_speed(counter):
    """Calculate overall and direction-specific average speeds based on
counting axis."""
    # Define stopped threshold
    STOPPED_THRESHOLD = 1.0  # km/h

    # Calculate speeds based on counting axis
    axis = getattr(counter, 'last_counting_axis', None) or
counting_config.get("axis") or 'y'
    if axis == "y":
        # For horizontal line, only calculate up/down speeds (counted-at-
crossing only)
        speed_up = calculate_direction_speeds(counter.counted_speeds_up,
STOPPED_THRESHOLD)
        speed_down = calculate_direction_speeds(counter.counted_speeds_down,
STOPPED_THRESHOLD)
        relevant_speeds = {'up': speed_up, 'down': speed_down}
    else:
        # For vertical line, only calculate left/right speeds (counted-at-
crossing only)
        speed_left = calculate_direction_speeds(counter.counted_speeds_left,
STOPPED_THRESHOLD)
        speed_right =
calculate_direction_speeds(counter.counted_speeds_right, STOPPED_THRESHOLD)
        relevant_speeds = {'left': speed_left, 'right': speed_right}

    # For overall average, use all counted-at-crossing speeds (fallback to
all speeds if none yet)
    all_counted = counter.counted_speeds_all if hasattr(counter,
'counted_speeds_all') else []
    if not all_counted:
        overall_avg = calculate_direction_speeds(counter.speeds,
STOPPED_THRESHOLD)
    else:
```

```
        overall_avg = calculate_direction_speeds(all_counted,
STOPPED_THRESHOLD)
    relevant_speeds['overall'] = overall_avg

    return relevant_speeds
```

3.5.4 ESAL Calculation for Predictive Maintenance

Vehicle Damage Factors (VDFs), assumed from AASHTO 1993 table values, are applied as follows: car (0.0005), motorcycle (0.0001), bus (0.15), and truck (2.0). Per-class Equivalent Single Axle Loads (ESALs) are computed as ESAL_class = count_class × VDF_class, with total ESAL being the sum across all classes. Directional ESALs are calculated similarly using traffic splits (up/down or left/right) to attribute loads by movement direction. The backend aggregates ESALs by class, direction, and total, then exports CSV and human-readable text reports including timestamps and session metadata.

ESAL calculation:

```
def calculate_esal(counts, split_counts=None):
    VDF = {
        "car": 0.0005,
        "motorcycle": 0.0001,
        "bus": 0.15,
        "truck": 2.0
    }
    esal_by_class = {cls: float(counts.get(cls, 0)) * VDF.get(cls, 0.0) for
cls in VDF}
    esal_total = float(sum(esal_by_class.values()))

    # If split counts provided (e.g. up/down or left/right), calculate those
too
    if split_counts:
        esal_by_direction = {}
        for direction, dir_counts in split_counts.items():
            dir_esal = {}
            dir_total = 0.0
            for cls in VDF:
                val = float(dir_counts.get(cls, 0)) * VDF.get(cls, 0.0)
                dir_esal[cls] = val
                dir_total += val
            esal_by_direction[direction] = {
                "by_class": dir_esal,
                "total": dir_total
            }
        return esal_by_class, esal_total, esal_by_direction
```

```
return esal_by_class, esal_total
```

In summary, BYTETrack is implemented via Ultralytics tracking APIs (enabled by the included bytetrack.yaml in the repository), while speed estimates are approximate and intended solely for comparative directional averages, not legal metrology.

4. Chapter 4: Experimental Setup

4.1 Dataset Selection and Preparation

Table 4.1 illustrates the most common relatable dataset for vehicle detection in urban scenarios, compared for training the model.

Table 4.1: Datasets comparison.

Dataset	Resolution	Annotations	Focus Area	Limitation
Cityscapes	2048×1024	Pixel-wise, object detection	Urban traffic	Limited to European cities
KITTI	1242×375	3D bounding boxes, object detection	Autonomous driving	Small dataset size
COCO	Varies	Bounding boxes, segmentation	General object detection	Not specific to traffic scenes
Waymo Open	1920×1080	3D LiDAR, bounding boxes	Self-driving cars	Requires LiDAR processing
Berkeley Deep Drive (BDD100K)	1280×720	Object detection, segmentation	Diverse driving scenarios	No pixel-wise segmentation

In short, the KITTI dataset is more used for autonomous driving, and it has a smaller dataset size. As for COCO, it is a general object detection dataset and not specialized in traffic scenes. The Waymo Open Dataset is considered beyond the scope of this research. The simplicity in managing the Cityscape dataset with its 5,000 high-resolution images and detailed labelling, specific for autonomous driving, made it a strong candidate. However, Berkeley Deep Drive has more diversity of scenarios and is five times bigger than Cityscape, containing 100,000 high-quality images for vehicle classes. So, Berkeley Deep was the choice. Also, considering that for real-time vehicle detection accuracy, the higher the image resolution, the better for high quality training.

BDD100K is the largest and most diverse open driving video dataset, containing 100,000 high-resolution video clips (over 1,100 hours) collected from more than 50,000 rides across various U.S. regions, weather conditions, times of day, and scene types (city, highway, residential). Released by UC Berkeley BAIR, it includes rich annotations on keyframes—

object bounding boxes, drivable areas, lane markings, and instance segmentation—making it the standard benchmark for multitask perception and autonomous driving research.

Furthermore, BDD100 has the group of classes that are critical for this work: cars, trucks, buses, motorcycles, and bicycles (Table 4.2). In addition, the focus of this dataset is real-world traffic representation, which makes it superior to other datasets considering the mentioned objectives of this work.

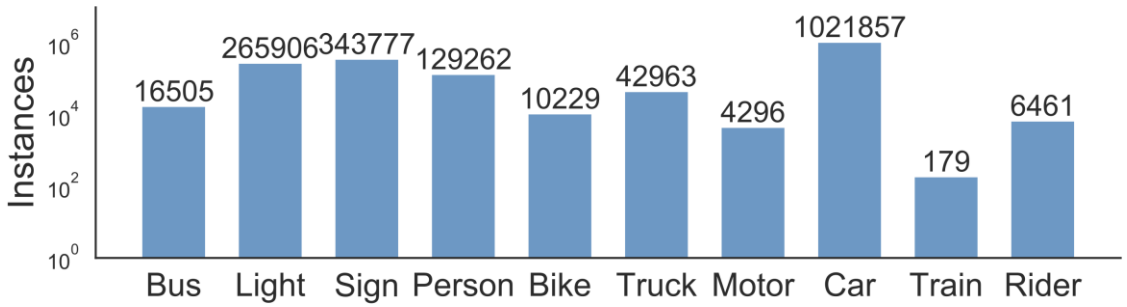


Figure 4.1: Statistics of different types of objects (from BDD100K site).

Statistics of different types of objects.

Figure xx illustrates a balanced class of vehicles in terms of instances. Moreover, this dataset is accessible for research purposes and has no license restrictions; it is open source.

4.2 Model Training (YOLO)

4.2.1 Environment Setup (Hardware/Software)

Experimental Environment

Hardware Specifications

The implementation was conducted on a desktop computer equipped with an Intel(R) Core (TM) i5-8400 CPU, with 16GB of RAM and graphical interface card of NVIDIA (GeForce GTX 1070 GPU) with 16GB VRAM. The system utilized a 1TB SSD for storage, ensuring fast data access during training, as we can see in Table 4.3.

Table 4.2: Hardware speficifications.

CPU	GPU	RAM	Storage
Intel(R) Core (TM) i5-8400 CPU @ 2.80GHz 2.81 GHz	NVIDIA GeForce GTX 1070 8GB VRAM	8.0 GB	1TB SSD

Software Specifications

Regarding the software environment, it was the operative system of Windows 11 pro, Visual Studio Code, Python 3.11 as the programming language. The deep learning framework used was Ultralytics YOLOv11m from the official site, with CUDA 12.1 and cuDNN 8.5.0 for GPU acceleration. Key libraries such as NumPy (1.23.5), OpenCV (4.7.0), and Pandas (1.5.3) were used for data processing and visualization. Table 4.4 shows in resume:

Table 4.3: Software specification.

OS	Python environment	Python version	Support on programming	DL Framework	Key Libraries	CUDA and cuDNN
Windows 11 Pro	Pytorch	Python 3.11	Visual Studio code	Ultralytics YOLOv11n	NumPy, OpenCV, Pandas	CUDA 12.1 cuDNN 8.5.0

Virtual Environment

For the preservation of the project files and to avoid conflicts and misleading on file reading and dependencies, an isolated virtual environment was created, using *pip* in the terminal prompt of VS Code to manage dependencies. The necessaries libraries were installed such as NumPy, Pandas, OpenCV, Pytorch. Others required packages were installed via "*pip*", ensuring consistency across different systems.

Visual Studio Code

The easy integration capability of VS Code for Python language with its Python extension, helps on identifying errors and efficiently fix them with IntelliSense suggestions. VS Code makes it easy to create and manage the project files. Furthermore, it is simple to implement changes and reverse them by simply turning code into comments, which helps on controlling the different versions and approaches used in the code.

VS Code was used to create and apply changes to the train.py file, the dataset.yaml file, and to convert the annotation from "*json*" to "*txt*" format using a python customized.

Moreover, the built-in terminal also allowed me to run scripts to reorganize the dataset folder on the easy way to process, without leaving the IDE.

4.2.2 Training Configuration and Hyperparameters

To train the model, the relevant libraries were imported, and the desired pretrained model was loaded (previously downloaded from ultralytics site), as we can see in the code below.

Training code:

```
import yaml
import multiprocessing
from ultralytics import YOLO

def main():
    with open('dataset.yaml', 'r') as f:
        data = yaml.safe_load(f)
    model = YOLO('yolo11n.pt')
    """
        results = model.train(
        data='dataset.yaml',  # Path to dataset config
        epochs=50,
        imgsz=640,
        batch=8,
        device=0,
        pretrained=True,
        optimizer='AdamW',
        lr0=0.01,
        patience=25,
        save_period=10,
        project='mixed_COCO_BDD100k',
        name='exp1'
    )
    """

    try:
        metrics = model.val()
    except Exception:
        metrics = None
       print('Training completed. Best model saved in
runs/detect/train/weights/best.pt')

if __name__ == '__main__':
    try:
```

```
    multiprocessing.freeze_support()
except Exception:
    pass
main()
```

Dataset configuration:

```
names:
      ...
- car
- truck
- bus
- train
- motorcycle
- bicycle
      ...
nc: 80
path: G:\Users\guima\Downloads\Coco  # dataset root dir
train: mix_COCO_CSCAPE_BDD100/images  # train images (53,518 images)
val: COCO_vehicles_val/images
```

4.3 Prototype Application Development

The architectural components defined in Chapter 3 were integrated into a functioning real-time application. The validated YOLO model was deployed within the FastAPI backend, connecting the vision pipeline (Figure 3.4) to the frontend interface. The system was then tested using the following scenario to validate the end-to-end workflow illustrated in Figure 4.1.

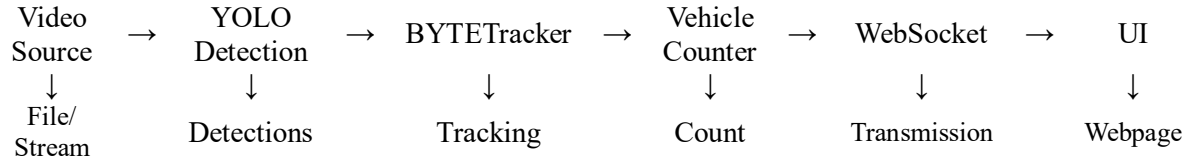


Figure 4.2: How it works.

4.3.1 Backend (FastAPI) and Frontend (Streamlit) Setup)

To test the proposed real-time vehicle detection, tracking and counting system, a prototype python application was created. From Visual Studio Code, it was created a backend code and frontend code, which was hosted on a local virtual server and accessible over the local network using Unicorn for FastAPI as the backend server for hosting the application locally

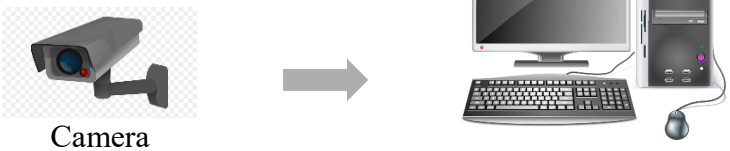
and make it accessible over the local LAN network, and Streamlit as the frontend to build interactive data for user interface (Table 4.5).

Table 4.4: Prototype essential tools.

Tool	Purpose	Command
Uvicorn + FastAPI	API backend or web services (fast)	uvicorn app:app ...
Streamlit	Interactive dashboards/UI for data apps	streamlit run webapp.py

4.3.2 Testing Scenario

As presented in Figure 4.3, it used a camera as the source of video streaming from the street road, the video is sent to the server(desktop), the application runs on a webpage in the same device. In this scenario, outputs such as bounding boxes, class labels, and tracking identifiers are rendered via the web interface, thereby providing an end-to-end validation of the integrated detection and tracking pipeline. The counting is presented in real time in the Detection report, at the end it is summarized in a final report at the left part of the page.



Camera

Desktop with GPU used as server and user device

Figure 4.3: System setup.

This setup which is cost-effective, was deployed without dedicated hardware, while maintaining real-time processing compatibility with Ultralytics and FastAPIs, the interface is presented in figure 4.4.

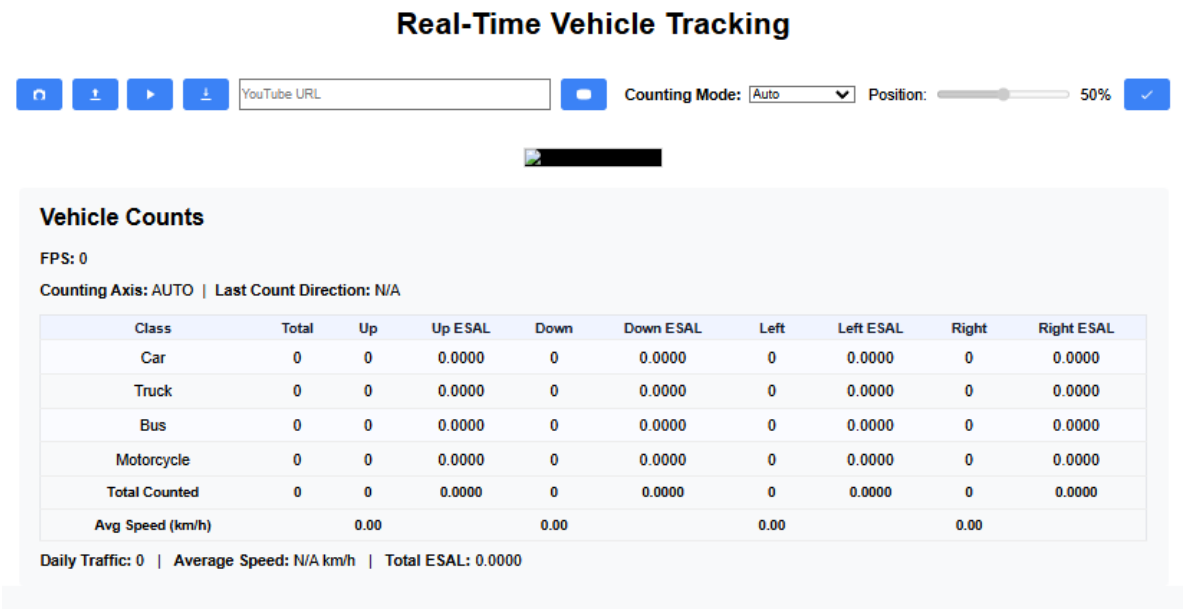


Figure 4.4: User interface

4.4 Evaluation Methodology

4.4.1 Performance Metrics

Real-time object detection and tracking systems must balance speed, accuracy, and efficiency. Below are the key metrics, according (A. Wang et al., 2024b):

1. Speed (Ensuring real-time responsiveness)

   - FPS (Frames Per Second): Measures how many frames the system processes per second. ≥30 FPS (≈33ms/frame) is needed for smooth real-time performance.

   - End-to-End Latency: Total processing time per frame (including detection & tracking). Must stay <33ms to match 30 FPS.

2. Accuracy (Ensuring correct detections & tracking)

   - mAP@0.5 (Mean Average Precision at IoU=0.5): Evaluates detection accuracy by checking if predicted boxes match ground truth (IoU ≥ 0.5). Higher mAP = better detection.

- MOTA (Multiple Object Tracking Accuracy): Measures tracking performance by penalizing false positives, missed detections, and ID switches. >50% is considered acceptable.

3. Efficiency (Optimizing resource usage)

- Hardware Utilization (GPU/CPU Usage): Should stay <80% to prevent overheating and allow multitasking.

- Power Consumption (Watts per Inference): Critical for battery-powered devices (e.g., drones, edge AI). Lower watts = longer runtime.

Supporting Metrics

- IoU (Intersection over Union): Measures overlap between predicted and ground-truth boxes. IoU ≥ 0.5 is a common threshold.

- Precision: % of detected objects that are correct (low false positives).

- Recall: % of actual objects detected (low misses).

These metrics ensure real-time systems are fast, reliable, and efficient in real-world applications.

4.4.2 Experimental Protocol

The experimental evaluation follows a structured protocol designed to ensure reproducibility, comprehensive assessment, and practical relevance to urban planning applications.

Evaluation Datasets and Scenarios

Primary Dataset: COCO + BDD100k

- ~50,000 high-quality urban scene images with fine annotations
- Focus on urban environments across varying conditions
- Vehicle classes: car, truck, bus, motorcycle, bicycle.

Experimental Procedure

Phase 1: Quantitative Model Evaluation

1. Detection Performance Assessment
    a. Evaluate trained YOLO model on validation set
    b. Calculate mAP, precision, recall across all vehicle classes
    c. Generate precision-recall curves and confusion matrices

2. Tracking Performance Validation
    a. Process video sequences through complete detection-tracking pipeline
    b. Compute MOTA, IDF1, and HOTA metrics
    c. Analyse identity preservation across frames

3. Computational Performance Benchmarking
    a. Measure FPS on target hardware (NVIDIA GTX 1070)
    b. Monitor GPU/CPU utilization during continuous operation
    c. Assess memory consumption and thermal characteristics

Phase 2: Qualitative System Evaluation

1. Visual Inspection
    a. Manual review of detection and tracking results
    b. Identification of failure cases and edge conditions
    c. Assessment of bounding box stability and consistency
2. Use Case Validation
    a. Vehicle counting accuracy in simulated traffic scenarios
    b. ESAL calculation reliability compared to manual counts
    c. Integration testing with web interface prototype

Success Criteria

Based on the system requirements established in Section 3.2, the following success thresholds are defined:

Table 4.5: Success criteria

Metric Category	Minimum Acceptance	Target Performance	Excellence Threshold
Detection Accuracy	mAP@50 ≥ 0.50	mAP@50 ≥ 0.65	mAP@50 ≥ 0.75

Tracking Performance	MOTA ≥ 0.50	MOTA ≥ 0.65	MOTA ≥ 0.75
Computational Performance	FPS ≥ 10	FPS ≥ 15	FPS ≥ 25
Counting Accuracy	≥ 85%	≥ 92%	≥ 95%

This comprehensive evaluation methodology ensures rigorous assessment of the proposed system's capabilities while maintaining practical relevance to real-world urban planning applications. The multi-faceted approach addresses both technical performance and operational requirements, providing a solid foundation for validating the research hypotheses and system objectives.

5. Results and Discussion

In this chapter, it will be discussed and analysed the results of the YOLO11n fine-tuning on a mixed COCO + BDD100k dataset. The key performance metrics observed include loss values on training and validation, mean Average Precision (mAP), precision, and recall, to assess the effectiveness of the model.

5.1 Model Training Performance (Loss Curves)

The model was trained for 50 epochs with smooth convergence, as shown in Figure 5.1. Training and validation losses (box) exhibited a rapid initial decline within the first 10 epochs, followed by a steady and gradual decline from epoch 10 to 50; train box loss started at approximately 1.52 and ended around 1.26, while val box loss started near 1.35 and ended around 1.11, indicating a consistent reduction in bounding-box regression error and improved localization accuracy.

Classification loss for both training and validation showed a sharp drop in the first ~10 epochs, followed by a continued steady decrease through the remaining epochs; train cls loss began at ~1.30 and ended near 0.93, whereas val cls loss started at ~1.50 and reached ~1.05 by the final epoch, reflecting strong and ongoing improvement in the model's ability to correctly classify vehicles.

Train and validation Distribution Focal Loss decreased smoothly and almost linearly throughout training, both starting at approximately 1.175–1.18; train DFL ended around 1.04 and val DFL ended near 1.10. The minimal gap between training and validation DFL (and all

other losses) confirms excellent generalization with no signs of overfitting, demonstrating stable and healthy convergence.

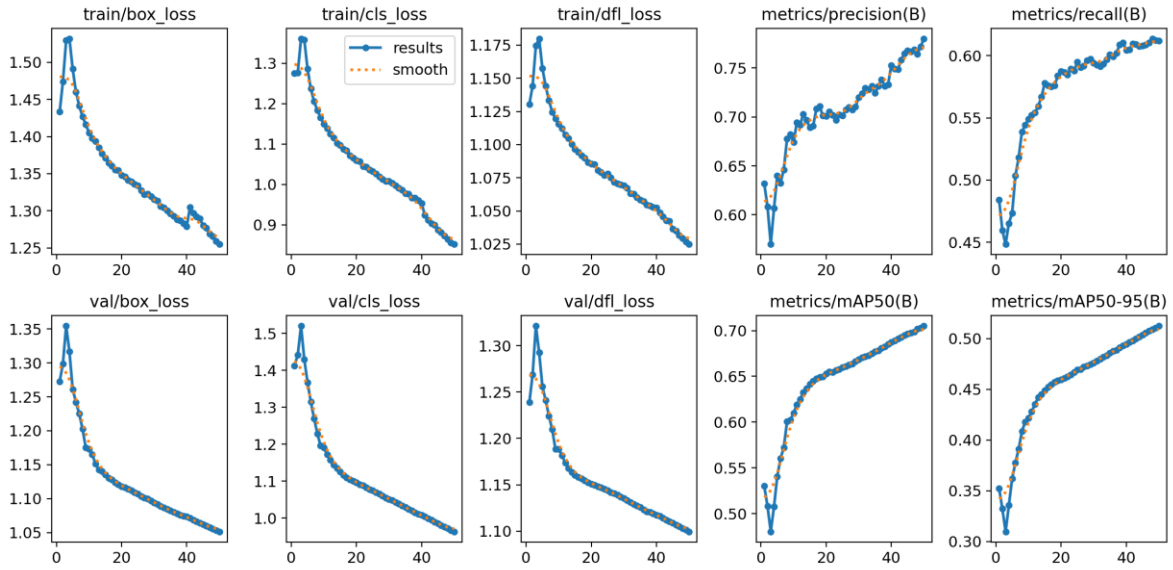


Figure 5.1: Loss curve evaluation graphics and precision evaluation graphics.

Overall, the curves indicate a well-behaved training process with strong final metrics (precision ≈ 0.78, recall ≈ 0.63, mAP@50 ≈ 0.70, mAP@50:95 ≈ 0.55), typical of a robust vehicle-detection model on real-world data.

5.2 Detection and Tracking Performance Metrics

Detection Performance

In figure 5.1, detection performance continued to grow efficiently from the start to the last epoch (the four graphics on the right), achieving a precision and a recall of above 0.78 and 0.61, respectively. A considerable gain is observed in mAP@50, growing from 0.35 to reach a peak of around 0.60 at epoch 50. In parallel, mAP@50:95 reached its peak of 0. 51 at the last epoch, a healthy improvement of the model precision.

Tracking Performance

Tracking evaluation using BYTETrack yielded a MOTA of 0.6753, reflecting robust multi-object tracking performance. Full MOT metrics show IDF1 = 83.1%, IDP = 82.2%, IDR = 84.0%, Recall = 84.9%, and Precision = 83.0%, with 25 mostly tracked (MT) and 11

partially tracked (PT) out of 37 ground truth trajectories. Low identity switches (IDs = 3) and fragmentations (FM = 76) confirm strong association stability, while MOTP = 0.222 indicates good localization accuracy.

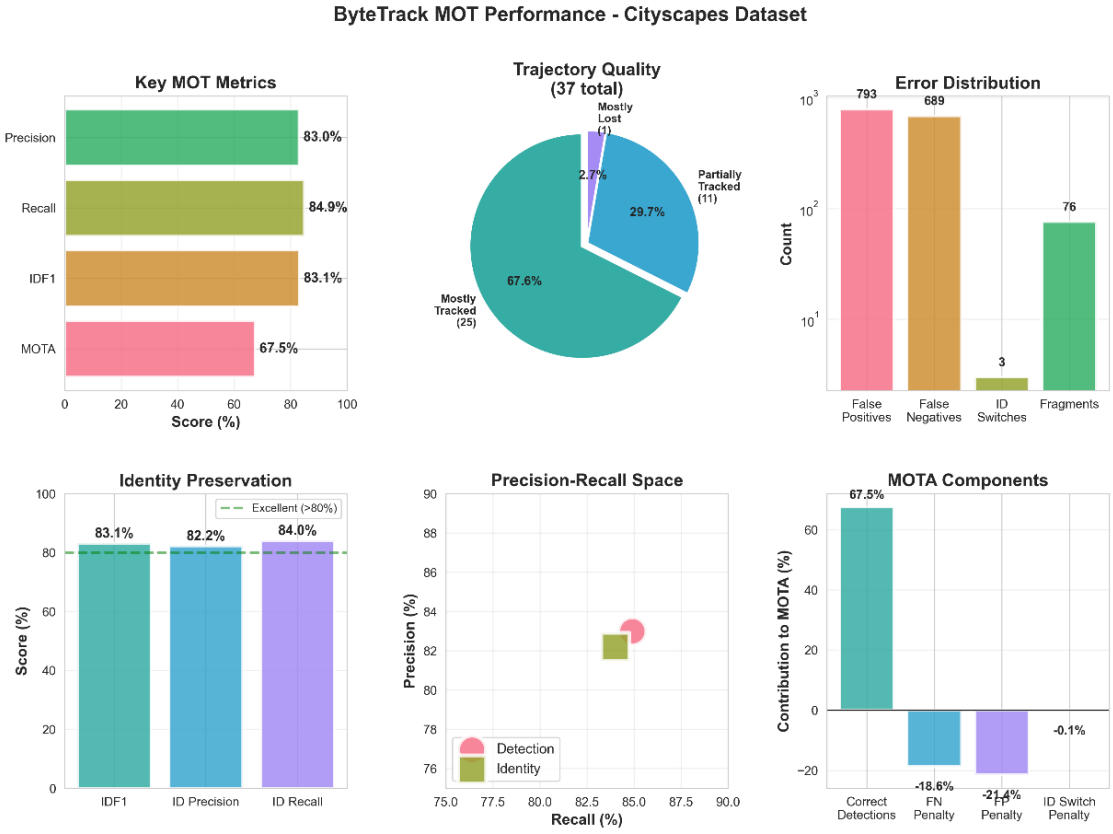


Figure 5.2: ByteTrack performance evaluation.

Overall, the curves on fig 5.1 indicate a well-behaved training process with strong final metrics, typical of a robust vehicle-detection model on a real-world data. And, ByteTrack shows exceptional ID preservation and a high percentage of trajectories tracked. These results validate effective integration of Ultralytics detection with BYTETrack for reliable vehicle tracking across frames.

5.3 Qualitative analysis of the model

The results of comparing the pretrained model and the mixed dataset model show a notable class improvement and stability. Both models were validated on the COCO validation set

with vehicles' classes and compared in the table below; the last column shows the improvements.

Table 5.1:Metrics comparison Yolo1n.pt and the mixed model dataset,

Metric	Pretrained YOLO11n	Mixed 50ep	Improvement
mAP@0.5	0.6453	0.7053	**+9.31% ↑**
mAP@0.5:0.95	0.4595	0.5131	**+11.68% ↑**
Precision	0.7351	0.7800	**+6.11% ↑**
Recall	0.5700	0.6122	**+7.41%**

Below are some random images from the streets of Vila Nova de Gaia for comparison purposes. On the first image (Figure 5.3) we can see that the Yolo11n.pt detects 7 vehicles when there are 4, exactly how the mixed dataset model presents. It is also visible the struggles of Yolo11n.pt when small obstruction is present (see the left side of both images on figure 5.3), however the mixed model behaves consistently. In figure 5.4 and 5.5, the mixed model is slightly more confident than the Yolo11n.pt.

Figure 5.3:Santo Ovidio's metro station (a).

Figure 5.4: Santo Ovidio's metro station (b).

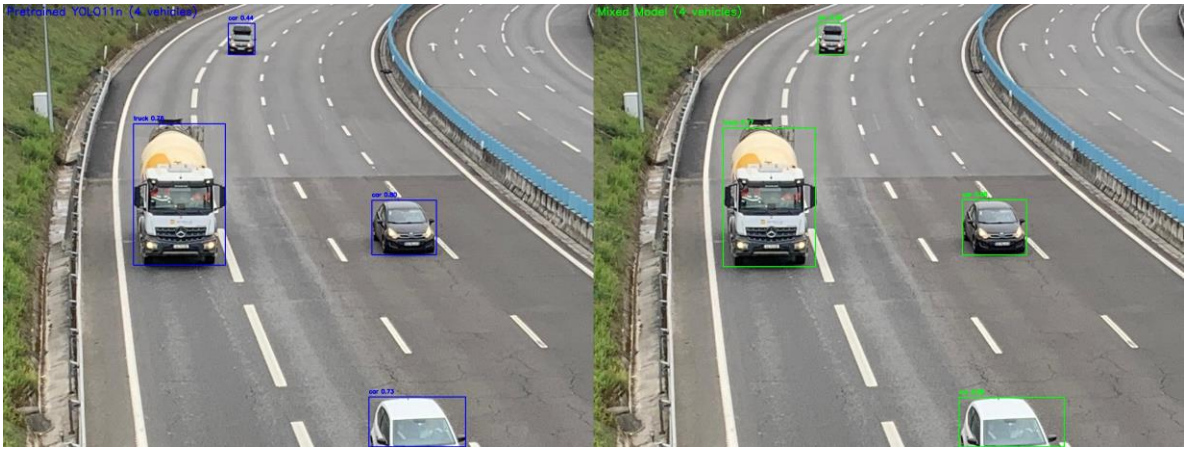


Figure 5.5: Vila Nova de Gaia, Canelas, A29.

The mixed dataset model is more conservative and more optimized for precision over recall, with **strong qualitative performance:** higher confidence, better calibration, and strategic detection filtering. Ideal for applications where false positives are costly (autonomous driving, traffic monitoring).

5.4 Important Fine-tune considerations

The results of this training were considerably positive, considering that the YOLO weights are pretrained on the COCO dataset, with more than 80 classes and around 110 000 instances, compared to the mixed dataset containing 28,518 COCO + 25,000 BDD100k, ~200,000+ vehicle annotations across 8 vehicle classes. Moreover, only few classes were used in training, and 0.7053 of mAP50 and 0.5131 mAP50-90 is sound for YOLO11n.

These results demonstrate the effectiveness of the knowledge transfer process, where the YOLO11n model, previously trained on the COCO dataset with multiple classes, was able to competently specialize in vehicle detection in the urban context of BDD100K dataset, evidenced by the progressive and consistent improvement of all metrics over the 50 finetuning epochs - with emphasis on the significant growth of ~12% in mAP50-95, and accuracy of 0.78, indicating that the model not only learned to identify vehicles with greater accuracy, but also refined its spatial location capacity in complex scenarios, thus validating the strategy of taking advantage of hierarchical characteristics learned in the generic domain for application in specific computer vision tasks.

5.4 Prototype Application Demonstration

The trained model was deployed in a real-time prototype using FastAPI with bytetrack.yaml integration. The system processed video streams at 15 to above 30 FPS on mid-range hardware, depending on video format and data processing, generating per-class ESALs using AASHTO 1993 VDF assumptions (car: 0.0005, motorcycle: 0.0001, bus: 0.15, truck: 2.0). Outputs included directional traffic splits, total ESAL aggregation, and timestamped CSV/text reports, demonstrating practical utility for traffic load monitoring.

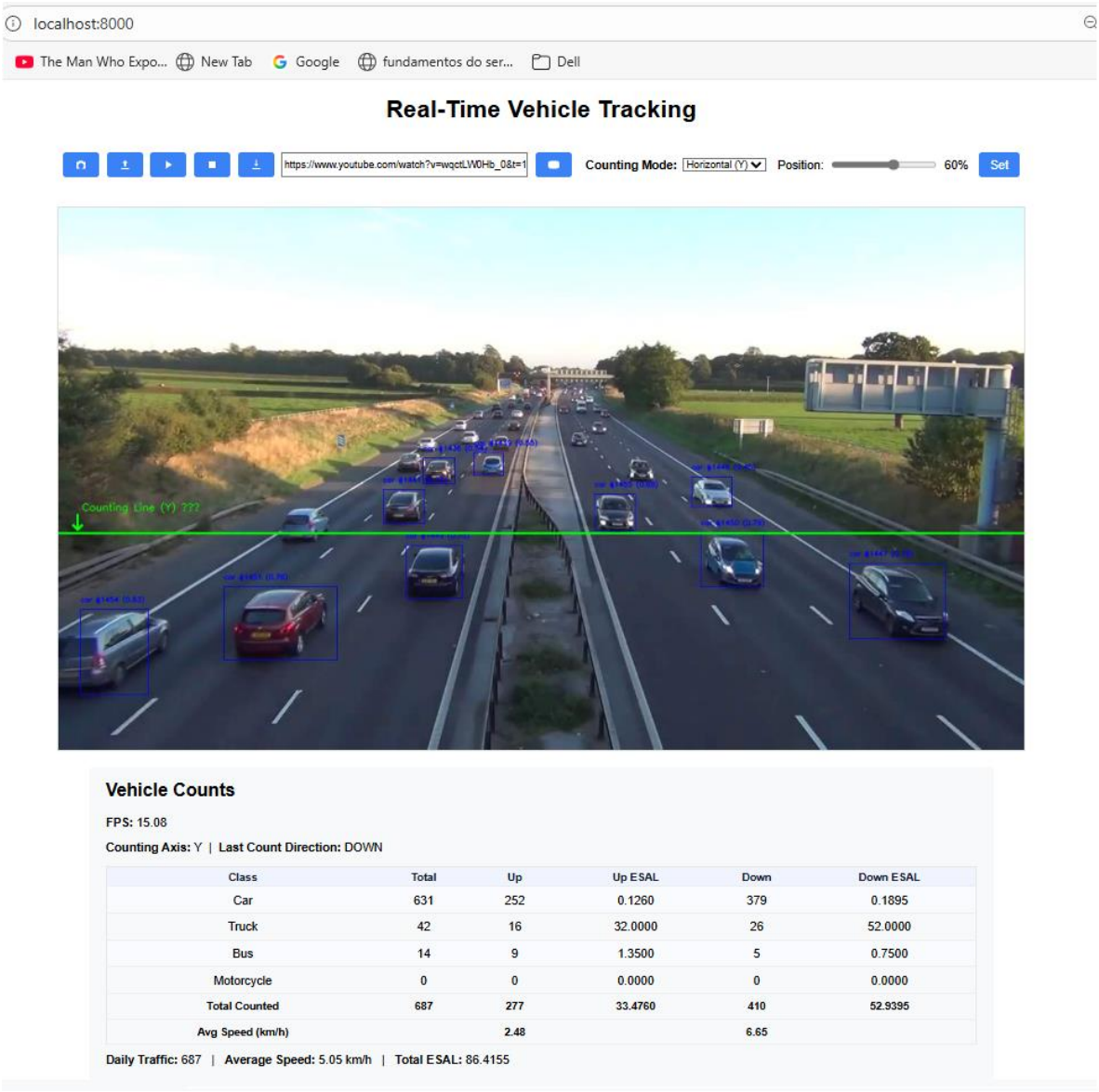


Figure 5.6: Prototype webpage demonstration.

The figure 5.7 shows the application running directly on a video from youtube, as we can see the address in the top part of the image, showing a consistent performance on real-time video.

To assess the counting accuracy of the proposed system, a camera was installed on the balcony capturing the vehicles on the street (figure 5.7), a manual validation was performed by counting 200 vehicles in each traffic direction.

Figure 5.7: Prototype testing.

The model detected 197 vehicles in one direction and 200 in the other. The counting accuracy was therefore calculated using the simple rule of three $\frac{197+200}{200+200} \times 100$ resulting in 99.25%. This high accuracy demonstrates the model's strong reliability in real-world conditions, confirming its suitability for automated traffic monitoring.

5.5 Discussion of Limitations

Despite strong convergence, the model exhibits sensitivity to small objects and heavy occlusion, reflected in the mAP@50:95 gap. Also, knowledge retention requires special attention for producing learning transfer correctly. Speed estimates are approximate and intended for comparative directional analysis only, not legal metrology. VDF values are assumed from AASHTO 1993 tables and may not reflect modern axle configurations. Future work should incorporate load spectra from weight-in-motion data and explore multi-camera fusion for improved 3D tracking and occlusion handling.

Despite these challenges, the research establishes a viable framework for future expansion. Subsequent work should prioritize securing funding for better and updated datasets and computational resources, collaborating with municipal partners for creating datasets with localized traffic data and specific for heavy vehicles, and leveraging distributed computing to improve model performance. These steps would address current gaps while enhancing the technology's practical utility for urban traffic management.

6. Practical Application and Impact Analysis

6.1 The Role of Traffic Volume in Infrastructure Degradation

Traffic volume plays a decisive role in road deterioration, with wear patterns often accelerating beyond what traditional reactive maintenance can address. Heavy vehicles, particularly trucks and buses, impose repeated loads that cause permanent deformation, cracking, and subgrade damage (Ghanizadeh et al., 2025; Septiyani & Indrastuti, 2024).

Two key metrics in traffic analysis are Average Daily Traffic (ADT) — the mean number of vehicles passing a point in 24 hours — and Average Annual Traffic (AAT), which is the daily average over a year. These metrics feed into the Equivalent Standard Axle Load (ESAL) calculation, which converts the combined effect of all vehicle types into the equivalent damage caused by a single standard axle load (Aljaleel et al., 2024).

Recent advancements emphasize integrating traffic metrics into AI-driven Predictive Maintenance (PdM) frameworks, with studies showing that multi-source data fusion significantly enhances deterioration forecasting accuracy (Umair Hassan et al., 2023). These approaches are further refined by hybrid techniques that couple machine learning predictions with dynamic multi-objective optimization to strategically prioritize rehabilitation based on factors like ESAL-derived wear (Alqasi et al., 2024). This data-driven paradigm is already being operationalized; for instance, connected vehicle data is now leveraged to assess road quality at scale via the International Roughness Index (IRI), revealing clear correlations between high traffic volumes and increased roughness to inform targeted PdM investments (Llopis-Castelló et al., 2024). Complementing this, image processing and AI enable proactive, visual distress detection, automatically identifying and prioritizing maintenance

for traffic-induced cracks and potholes to achieve more sustainable infrastructure outcomes (Gopalakrishnan et al., 2022).

6.2 Predictive Maintenance Framework

Predictive maintenance systems can set alerts when remaining life falls below a threshold, allowing timely interventions. Real-time analytics can help urban planners to align maintenance with actual load patterns, turning high-traffic corridors from costly liabilities into efficiently managed assets.

Industry analyses indicate that integrating AI, IoT sensor networks, and predictive-analytics platforms is emerging as a cost-efficient strategy for road-asset management, particularly across European road networks where data quality, interoperability, and organizational change management have been identified as critical enablers of measurable savings (*Europe (virtual) 2024: Harnessing the power of predictive maintenance in roads | McKinsey*, 2024). IoT-enabled pavement-monitoring frameworks further demonstrate that real-time sensing combined with machine-learning models can reduce inspection effort and maintenance expenditures by improving defect detection, prioritization, and intervention timing (Cano-Ortiz et al., 2022; Tamagusko et al., 2024). By incorporating traffic-loading metrics—such as vehicle-type distributions, axle-load spectra, and E SAL factors derived from national datasets like Portugal's TMDA—predictive-maintenance systems can evolve into resilient, cost-effective tools that mitigate the non-linear deterioration associated with heavy-vehicle volume and overloaded axles (Hatoum et al., 2022).

By incorporating vehicle type distributions and axle factors into ESAL calculations from sources like Portugal's TMDA data, PdM can evolve into a resilient, cost-effective paradigm, mitigating the non-linear impacts of traffic volume on global infrastructure.

6.3 Case Study: Traffic Context in Portugal

The necessity for predictive, AI-driven pavement management is starkly illustrated by traffic data from Portugal's National Road Network (RRN), according to the Autoridade da Mobilidade e dos Transportes (AMT, 2023), the overall Annual Average Daily Traffic

(TMDA) across major roads in 2022 was 20,110 vehicles, with intense concentration in metropolitan areas like Lisbon, where the A5 recorded over 144,000 vehicles per day—compared to sparse rural traffic on routes like the A4.

Within the same, A cluster analysis report further segments the network into distinct demand patterns, from high-traffic urban corridors to seasonally spiking tourist routes. These pronounced regional and seasonal variations in volume directly amplify degradation risks, particularly in high-ESAL corridors, underscoring the critical need for the predictive maintenance strategies discussed previously.

6.4 Integration into Smart Urban Ecosystems

The integration of AI-powered vehicle image recognition systems into urban planning and transportation management offers transformative potential, enabling real-time monitoring and analysis of traffic patterns, vehicle density, and infrastructure conditions. These systems provide actionable insights for city planners and traffic authorities while addressing critical challenges caused by heavy traffic loads.

6.4.1 Real-Time Traffic Monitoring and Road Degradation

AI, with computer vision, can analyze live video feeds to:

- Classify vehicles and count traffic volume (ADT)
- Calculate Equivalent Single Axle Loads (ESAL) using predefined Vehicle Damage Factors (VDFs)
- Predict remaining pavement life by comparing cumulative ESAL to design thresholds.

These features can be used by urban planners to monitor infrastructure degradation and actively prevent serious road damages by performing preventive maintenance and adjusting traffic, accordingly, as presents the following subsections.

6.4.2 Predictive Maintenance

With computer vision applied to traffic management, it is possible to:

- Prioritize road repairs: Vehicle overloading is considered one of the most significant causes of accelerating flexible pavement deterioration, reducing the pavement's design life, and affecting the overall sustainability of the pavement system (Hatoum et al., 2022; Putri et al., 2024). By identifying high-ESAL corridors, such as Industrial zones with frequent overloaded trucks, for targeted maintenance, AI can reduce costs of infrastructure repairments.
- Dynamic road resurfacing schedules: Systems integrate real-time ESAL data to adjust maintenance timelines, avoiding premature failures.

These two features can be used by urban planners to automate intervention in advance.

6.4.3 Smart City Integration

Vehicle detection and tracking with AI can be integrated into smart cities in:

- Public transport optimization: Tracking buses and freight vehicles improves routing, while emission-aware policies use vehicle class data to reduce carbon footprints (Singapore, 2018).
- Overload enforcement: Cameras flag overloaded trucks for inspection, mitigating damage quantified by studies such as AASHTO Guide for Design of Pavement Structures.

The integration of AI-powered vehicle detection and tracking moves urban management from a reactive to a predictive and proactive model. As demonstrated, these systems deliver a dual benefit: they optimize real-time operational efficiency, while simultaneously safeguarding long-term public assets through precise, data-driven enforcement against costly wear and tear, as quantified by foundational engineering principles. Ultimately, this technological synergy is not merely about streamlining traffic, it is about building a more sustainable, resilient, and economically viable urban future."

6.4.4 Case Studies

The integration of AI-powered vehicle tracking is revolutionizing urban infrastructure management. Systems like Barcelona's Smart Parking demonstrate how guiding drivers to available spots directly reduces vehicle miles traveled (VMT) and congestion, a benefit

confirmed by recent analyses of IoT-based parking solutions (Bhatnagar et al., 2025). This reduction in circling traffic indirectly lowers the cumulative Equivalent Single Axle Load (ESAL) on pavements, mitigating long-term wear. Furthermore, the manual ESAL-based alert systems pioneered by jurisdictions like Sidoarjo (Indonesia) are now being superseded by predictive AI. Research has progressed to where artificial intelligence can automatically detect pavement damage and forecast deterioration in real-time, effectively automating infrastructure lifespan predictions (Abu Dabous et al., 2025). This synergy of dynamic operational data and long-term structural analytics represents the forefront of building sustainable and resilient urban transport networks.

By merging real-time operational insights like traffic flow with long-term infrastructure analytics (ESAL-based wear models), AI systems modernize urban resilience, as seen in global benchmarks. These innovations, combined with seamless integration into legacy traffic management systems, will pave the way for more resilient, efficient, and universally deployable solutions, ultimately supporting smarter urban ecosystems.

7. Conclusion and Future Work

7.1 Synthesis of Contributions

Resorting to Artificial intelligence to address the challenges of human life has proved to be very successful in response to the rapidly changing events of society, with the structural, demographic, and climate changes imposing volatile and uncertainty situations with a high degree of complexity and ambiguity. In the realm of computer vision, image recognition has been used to bring new solutions and approaches to enhance human life.

In this work it was presented, the base and evolution of machine learning. Also standing as a study to consider as an introduction to computer vision and artificial intelligence. A very simplistic approach on how to apply the available models of computer vision was presented; from gathering the data, training the model, evaluating to implementation in simulated scenario. A methodology to use image recognition with artificial intelligence for improving urban planning and transportation through vehicle identification was proposed and developed

as a prototype, using the recent version of a pre-trained model of YOLO for detecting, tracking and counting the vehicle on the road, can give insights to help on decision making.

Furthermore, the necessary steps to choose the technologies, to prepare the working environment and the requirements were presented and explained in a simple way. Also, the process was presented to select the adequate dataset to enhance the pretrained model according to the goal of this work. Moreover, the evaluation of the model and analysis of the results of the experiment showed that the implemented methodology for training the model was considerably positive and the model learned, although not achieving the peak results.

Key Technical Contributions:

- Implementation of a YOLOv11-based vehicle detection system achieving around 78% precision and 63% recall;
- Development of an integrated tracking and counting pipeline using ByteTrack algorithm;
- Creation of a web-based prototype application with real-time visualization capabilities;
- Application of Clean Architecture and Domain-Driven Design principles to computer vision systems;
- Demonstration of ESAL-based calculation for predictive maintenance framework for urban infrastructure.

7.2 Implications for Urban Planning and Traffic Management

This study provides valuable insights into how AI-driven vehicle recognition can optimize traffic management and urban infrastructure. The integration of AI-powered vehicle recognition systems into traffic monitoring frameworks can enable real-time identification and classification of vehicles, leading to improved traffic flow regulation and generating useful data for predicting road maintenance. It can help to detect congestion patterns automatically, identify high-density traffic zones, and analyze peak-hour trends, which allows urban planners to design more effective road infrastructure, allocate resources efficiently, and implement data-driven traffic control measures. Additionally, accurate

vehicle identification contributes to the development of intelligent transportation systems (ITS), which facilitate the enhancement of public transport services.

Within the concept of smart cities, urban planners can leverage AI-driven vehicle recognition data to implement better solutions by using the insights derived from traffic patterns and vehicle movement analysis for determining optimal locations for roads, pedestrian pathways, and public transport facilities. Furthermore, AI-enabled simulations can model different urban development scenarios, allowing policymakers to make informed decisions on sustainable infrastructure planning.

Specific Implications:

- Real-time Traffic Optimization: AI systems can dynamically adjust traffic signals and routing based on actual vehicle counts and classifications
- Predictive Infrastructure Management: ESAL calculations enable proactive maintenance scheduling based on actual road usage patterns
- Data-Driven Urban Planning: Vehicle classification data informs long-term infrastructure development and public transport planning
- Cost Reduction: Automated monitoring reduces manual inspection costs and enables targeted maintenance interventions

7.3 Challenges and Future Research Directions

Despite its numerous advantages, AI-driven image recognition for vehicle identification faces several challenges, including model accuracy in varying environmental conditions, computational resource requirements, and ethical concerns regarding data privacy. Future work will focus on curating specific datasets for vehicle detection, refining model training strategies, and evaluating performance on additional real-world datasets to improve model robustness against occlusions, adverse weather conditions, and variations in vehicle appearance. Evaluate rigorously the tracking capacity of the trained model and improve the model regarding processing frames per second. Additionally, integrating AI with edge computing solutions can enhance real-time processing capabilities, making these systems more scalable and deployable in urban environments.

Specific Challenges Identified:

- Environmental Robustness: Performance degradation under adverse weather conditions and low-light scenarios.
- Computational Requirements: High GPU power demands limiting deployment to less robust models.
- Dataset Class Imbalance: Underrepresentation of essential vehicle classes affecting detection accuracy.
- Dataset Limitations: Bias in European-centric training data affecting generalizability to other regions.
- Access to road and pavement data to estimate lifespan with consideration to traffic load, environmental conditions, and material quality.

Future Research Directions:

- Multi-sensor Fusion: Integrating LiDAR or thermal imaging with existing camera systems to improve detection accuracy in challenging conditions.
- Edge Computing: Developing lightweight models for local device deployment to reduce latency and bandwidth demands.
- Adaptive Learning: Creating mechanisms for systems to evolve with changing urban environments and vehicle designs.
- Dataset Diversification: Collaborative efforts with cities worldwide to create more representative training datasets.
- Advanced Tracking Algorithms: Implementing more sophisticated multi-object tracking to handle complex urban scenarios.
- Real-time ESAL Integration: Developing live ESAL calculation and alert systems for immediate maintenance prioritization.
- Integration with legacy systems: Explore the possibility of integrating these models with legacy traffic management systems.

Based on experimental results, specific improvements are needed:

- The model achieves a reasonable balance between precision and recall but may benefit from additional training data to reduce missed detections and increase precision.
- Fine-tuning hyperparameters such as the learning rate and batch size to further optimize model performance.
- Diversify and add better quality images and annotations originally in YOLO format to avoid annotation conversion.
- Diversify the dataset with more vehicle classes and models to specialize the model in vehicle detection.

Despite current limitations, this research establishes a viable framework for future expansion and demonstrates the significant potential of AI-powered vehicle recognition systems to transform urban mobility and infrastructure management. The system's modular design provides a scalable foundation for city-wide deployment. Future work will focus on integrating edge computing and multi-camera networks to transform the prototype into a comprehensive, city-scale predictive maintenance platform.

Bibliography

A Comprehensive Guide to AI Tech Stack. (2025, fevereiro 25). *Sparx IT Solutions*. https://www.sparxitsolutions.com/blog/ai-tech-stack/

Abu Dabous, S., Ait Gacem, M., Zeiada, W., Hamad, K., & Al-Ruzouq, R. (2025). Artificial intelligence applications in pavement infrastructure damage detection with automated three-dimensional imaging – A systematic review. *Alexandria Engineering Journal*, *117*, 510–533. https://doi.org/10.1016/j.aej.2024.11.081

Abubakr, M., Rady, M., Badran, K., & Mahfouz, S. Y. (2024). Application of deep learning in damage classification of reinforced concrete bridges. *Ain Shams Engineering Journal*, *15*(1), 102297. https://doi.org/10.1016/j.asej.2023.102297

Aljaleel, Z. M., Ahmed, N. Y., & Atemimi, Y. K. A. (2024). Finite Element Modeling to Predicting Rutting in Flexible Pavements under Overloading. *Salud, Ciencia y Tecnología - Serie de Conferencias*, *3*, 822. https://doi.org/10.56294/sctconf2024822

Alqasi, M. A. Y., Alkelanie, Y. A. M., & Alnagrat, A. J. A. (2024). Intelligent Infrastructure for Urban Transportation: The Role of Artificial Intelligence in Predictive Maintenance. *Brilliance: Research of Artificial Intelligence*, *4*(2), 625–637. https://doi.org/10.47709/brilliance.v4i2.4889

AMT. (2023). *Amt-autoridade.pt*. https://www.amt-autoridade.pt/

Bhatnagar, P., Sahu, P., Dawra, D. S., Sharma, S., & Sharma, S. (2025). *A Review of IOT Based Smart Parking Systems: Advancements, Challenges & Future Directions*. *11*(9).

Bird, J. J., & Lotfi, A. (2024). CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. *IEEE Access*, *12*, 15642–15650. IEEE Access. https://doi.org/10.1109/ACCESS.2024.3356122

Cano-Ortiz, S., Pascual-Muñoz, P., & Castro-Fresno, D. (2022). Machine learning algorithms for monitoring pavement performance. *Automation in Construction*, *139*, 104309. https://doi.org/10.1016/j.autcon.2022.104309

Cernadas, E. (2024). Applications of Computer Vision, 2nd Edition. *Electronics*, *13*(18), Artigo 18. https://doi.org/10.3390/electronics13183779

Cityscapes Dataset – Semantic Understanding of Urban Street Scenes. (2020, outubro 17). https://www.cityscapes-dataset.com/

D, M., Alaswad, F., Aljaddouh, B., Ranganayagi, L., & R, S. (2025). AI-Powered Traffic Management: Improving Congestion Detection and Signal Regulation. *2025 International*

Conference on Multi-Agent Systems for Collaborative Intelligence (ICMSCI), 899–904. https://doi.org/10.1109/ICMSCI62561.2025.10894186

De Sordi, J. O. (2021). Variations of the DSR Approach. *Design Science Research Methodology*, 111–120. https://doi.org/10.1007/978-3-030-82156-2_7

Di Grande, S., Berlotti, M., & Cavalieri, S. (2024). *AI-Powered Urban Mobility Analysis for Advanced Traffic Flow Forecasting*. 57–64. https://doi.org/10.5220/0012625900003714

Ejaz, U., Ramon, W., & Olaoye, G. (sem data). *The Role of Big Data and AI in Smart Cities and Urban Planning*.

Europe (virtual) 2024: Harnessing the power of predictive maintenance in roads | McKinsey. (2024). https://www.mckinsey.com/industries/infrastructure/global-infrastructure-initiative/roundtables/europe-2024-harnessing-the-power-of-predictive-maintenance-in-roads?utm_source=chatgpt.com

Faqih Seknun, H., Setyawan, A., & Pungky Pramesti, F. (2025). Assesment of road condition and roads maintenance to reduce potential environmental damage. *IOP Conference Series: Earth and Environmental Science*, *1438*(1), 012085. https://doi.org/10.1088/1755-1315/1438/1/012085

Francisco, K. V., Robles, E. C., & Samson, H. P. (2024). *Artificial Intelligence for Traffic Management: A Comprehensive Review of Advances and Challenges* (SSRN Scholarly Paper No. 5062545). Social Science Research Network. https://doi.org/10.2139/ssrn.5062545

Ghanizadeh, A. R., Fakhri, M., Amlashi, A. T., & Dessouky, S. (2025). Effect of strain waveform modeling and loading frequency on the fatigue life of asphalt concrete. *Construction and Building Materials*, *462*, 139906. https://doi.org/10.1016/j.conbuildmat.2025.139906

Hatoum, A. A., Khatib, J. M., Barraj, F., & Elkordi, A. (2022). Survival Analysis for Asphalt Pavement Performance and Assessment of Various Factors Affecting Fatigue Cracking Based on LTPP Data. *Sustainability*, *14*(19), 12408. https://doi.org/10.3390/su141912408

Igorevich Rozhdestvenskiy, O., & Poornima, E. (2024). Enabling Sustainable Urban Transportation with Predictive Analytics and IoT. *MATEC Web of Conferences*, *392*, 01179. https://doi.org/10.1051/matecconf/202439201179

Implementing Command Query Responsibility Segregation (CQRS) in Large-Scale Systems. (2024). *International Journal of Research in Modern Engineering & Emerging Technology*, *12*(12), 49–73. https://doi.org/10.63345/ijrmeet.org.v12.i12.3

INNOVATIVE TECHNOLOGIES FOR TRAINING AND EDUCATING YOUNG PEOPLE. (2025). International Science Group.

Jiang, X., Hadid, A., Pang, Y., Granger, E., & Feng, X. (Eds.). (2019). *Deep Learning in Object Detection and Recognition*. Springer Singapore. https://doi.org/10.1007/978-981-10-5152-4

Junker, A., & Lazzaretti, F. (2025). API Design Supported by Domain-Driven Design. *Crafting Great APIs with Domain-Driven Design*, 71–119. https://doi.org/10.1007/979-8-8688-1457-0_5

Kamrowska-Załuska, D. (2021). Impact of AI-Based Tools and Urban Big Data Analytics on the Design and Planning of Cities. *Land*, *10*(11), Artigo 11. https://doi.org/10.3390/land10111209

Kapferer, S., & Zimmermann, O. (2020). *Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling*. 299–306. https://doi.org/10.5220/0008910502990306

Kourtit, K., Nijkamp, P., Osth, J., & Turk, U. (2024). Is artificial intelligence a trustworthy route navigation system for smart urban planning? *Eastern Journal of European Studies*, *15*(2), 30–47. https://doi.org/10.47743/ejes-2024-0203

Krauss, P. (2024). *Artificial Intelligence and Brain Research: Neural Networks, Deep Learning and the Future of Cognition*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-68980-6

Lano, K., & Yassipour Tehrani, S. (2023). Introduction to Clean Architecture Concepts. *Undergraduate Topics in Computer Science*, 35–49. https://doi.org/10.1007/978-3-031-44143-1_2

Lee, R. S. T. (2020). *Artificial Intelligence in Daily Life*. Springer Singapore. https://doi.org/10.1007/978-981-15-7695-9

Liao, K. (2022). Road Damage Intelligent Detection with Deep Learning Techniques. *2022 IEEE 5th International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 795–799. https://doi.org/10.1109/ICISCAE55891.2022.9927635

Llopis-Castelló, D., Camacho-Torregrosa, F. J., Romeral-Pérez, F., & Tomás-Martínez, P. (2024). Estimation of Pavement Condition Based on Data from Connected and Autonomous Vehicles. *Infrastructures*, *9*(10), 188. https://doi.org/10.3390/infrastructures9100188

Marasinghe, R., Yigitcanlar, T., Mayere, S., Washington, T., & Limb, M. (2024). Computer vision applications for urban planning: A systematic review of opportunities and

constraints. *Sustainable Cities and Society*, *100*, 105047. https://doi.org/10.1016/j.scs.2023.105047

Nadarajan, J., & Sivanraj, R. (2022, dezembro 13). *Attention-Based Multiscale Spatiotemporal Network for Traffic Forecast with Fusion of External Factors*. https://www.mdpi.com/2220-9964/11/12/619

Neha, F., Bhati, D., Shukla, D. K., & Amiruzzaman, M. (2024). *From classical techniques to convolution-based models: A review of object detection algorithms* (No. arXiv:2412.05252). arXiv. https://doi.org/10.48550/arXiv.2412.05252

Ogunkan, D. V., & Ogunkan, S. K. (2025). Exploring big data applications in sustainable urban infrastructure: A review. *Urban Governance*, *5*(1), 54–68. https://doi.org/10.1016/j.ugj.2025.02.003

Ponce, P., Peffer, T., Mendez Garduno, J. I., Eicker, U., Molina, A., McDaniel, T., Musafiri Mimo, E. D., Parakkal Menon, R., Kaspar, K., & Hussain, S. (2023). *Data and AI Driving Smart Cities* (Vol. 128). Springer International Publishing. https://doi.org/10.1007/978-3-031-32828-2

Putri, S. A., Sholichin, I., & Fatikasari, A. D. (2024). Analysis of The Influence of Vehicle Overloading on The Remaining Life of The Road Plan. *Composite: Journal of Civil Engineering*, *3*(2), 13–24. https://doi.org/10.26905/cjce.v3i2.13274

Ren, M., Zhang, X., Zhi, X., Wei, Y., & Feng, Z. (2024). An annotated street view image dataset for automated road damage detection. *Scientific Data*, *11*(1), 407. https://doi.org/10.1038/s41597-024-03263-7

Sager, C., Janiesch, C., & Zschech, P. (2021). A survey of image labelling for computer vision applications. *Journal of Business Analytics*, *4*(2), 91–110. https://doi.org/10.1080/2573234X.2021.1908861

Saini, K., & Sharma, S. (2025). Smart Road Traffic Monitoring: Unveiling the Synergy of IoT and AI for Enhanced Urban Mobility. *ACM Comput. Surv.*, *57*(11), 276:1-276:45. https://doi.org/10.1145/3729217

Septiyani, Y. N., & Indrastuti. (2024). The Impact of Load Traffic of Road Deterioration in Urban Areas: Case Study Jalan KH Abdul Halim Majalengka. *LEADER: Civil Engineering and Architecture Journal*, *2*(4), 911–919. https://doi.org/10.37253/leader.v2i4.10145

Serrano, L. G. (2021). *Grokking machine learning*. Manning Publications.

Singapore, M. of T. (2018). *LTA | Land Transport Master Plan 2040*. https://www.lta.gov.sg/content/ltagov/en/who_we_are/our_work/land_transport_master_plan_2040.html

Solahudin, N., & Susanto, M. D. P. (2025). Analysis of Road Damage Factors Based on Vehicle Load and Volume on K.H. Zaenal Arifin Road Segment Cikulak – Cibogo. *Devotion : Journal of Research and Community Service*, *6*(4), Artigo 4. https://doi.org/10.59188/devotion.v6i4.25454

Tamagusko, T., Gomes Correia, M., & Ferreira, A. (2024). Machine Learning Applications in Road Pavement Management: A Review, Challenges and Future Directions. *Infrastructures*, *9*(12), 213. https://doi.org/10.3390/infrastructures9120213

Ultralytics. (2025). *Home*. https://docs.ultralytics.com/

Umair Hassan, M., Hagen Steinnes, O.-M., Gribbestad Gustafsson, E., Løken, S., & A. Hameed, I. (2023, março 8). *Predictive Maintenance of Norwegian Road Network Using Deep Learning Models*. https://www.mdpi.com/1424-8220/23/6/2935

U.S. Department of Transportation. (2024, janeiro). *FHWA Bridge Preservation Research Roadmap*.

Valdovinos-Chacón, G., Ríos-Zaldivar, A., Valle-Cruz, D., & Lara, E. R. (2025). Integrating IoT and YOLO-Based AI for Intelligent Traffic Management in Latin American Cities. Em R. Sandoval-Almazán & D. Valle-Cruz (Eds.), *Artificial Intelligence in Government: Latin America Challenges and Expectations* (pp. 227–253). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-87623-3_10

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024a). *YOLOv10: Real-Time End-to-End Object Detection*.

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024b). *YOLOv10: Real-Time End-to-End Object Detection* (No. arXiv:2405.14458). arXiv. https://doi.org/10.48550/arXiv.2405.14458

Wang, H., Yuan ,Yun, Yang ,Xianfeng Terry, Zhao ,Tian, & and Liu, Y. (2023). Deep Q learning-based traffic signal control algorithms: Model development and evaluation with field data. *Journal of Intelligent Transportation Systems*, *27*(3), 314–334. https://doi.org/10.1080/15472450.2021.2023016

Wubuli, A., Li, F., Cao, S., & Zhang, L. (2025). Timing of Preventive Highway Maintenance: A Study from the Whole Life Cycle Perspective. *Sustainability*, *17*(3), Artigo 3. https://doi.org/10.3390/su17031009

Yap, W., Chang, J.-H., & Biljecki, F. (2023). Incorporating networks in semantic understanding of streetscapes: Contextualising active mobility decisions. *Environment and Planning B: Urban Analytics and City Science*, *50*(6), 1416–1437. https://doi.org/10.1177/23998083221138832

Yigitcanlar, T., Kankanamge, N., Regona, M., Ruiz Maldonado, A., Rowan, B., Ryu, A., Desouza, K. C., Corchado, J. M., Mehmood, R., & Li, R. Y. M. (2020). Artificial Intelligence Technologies and Related Urban Planning and Development Concepts: How Are They Perceived and Utilized in Australia? *Journal of Open Innovation: Technology, Market, and Complexity*, *6*(4), Artigo 4. https://doi.org/10.3390/joitmc6040187

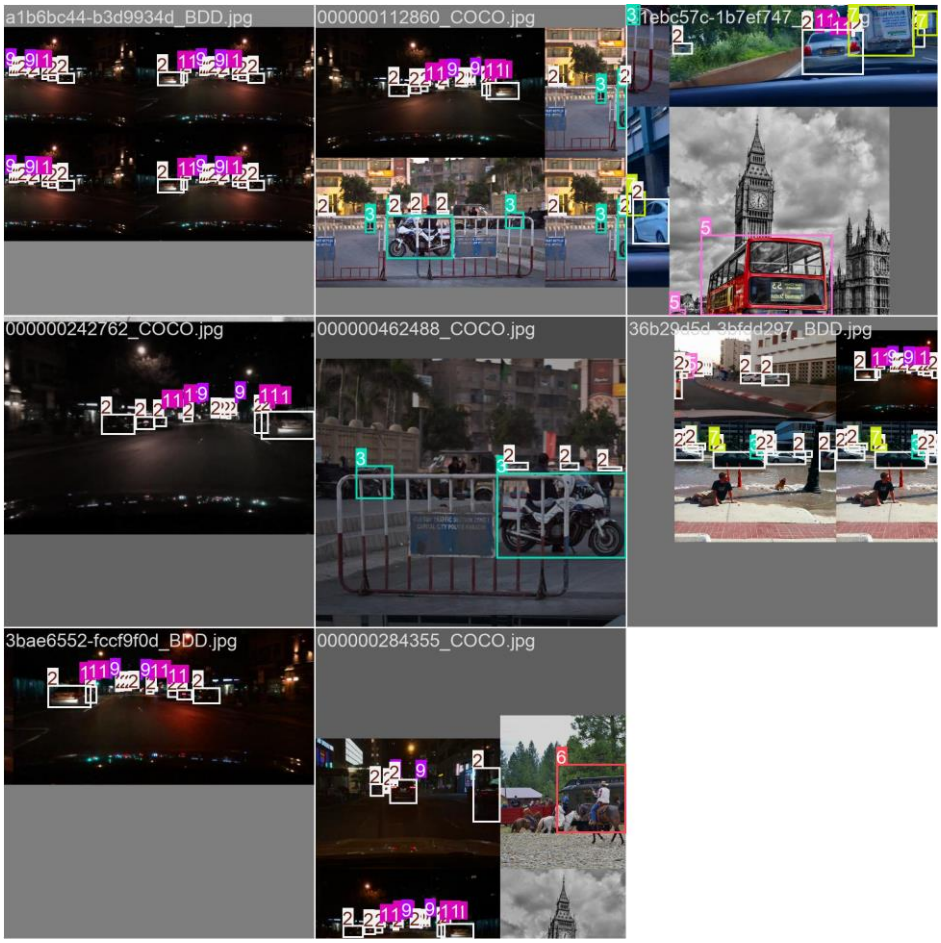
Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., & Wang, X. (2022). ByteTrack: Multi-object Tracking by Associating Every Detection Box. Em S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, & T. Hassner (Eds.), *Computer Vision – ECCV 2022* (Vol. 13682, pp. 1–21). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-20047-2_1

Zhao, X., Wang, L., Zhang, Y., Han, X., Deveci, M., & Parmar, M. (2024). A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, *57*(4), 99. https://doi.org/10.1007/s10462-024-10721-6

Zhu, Y., Wang, Y., An, Y., Yang, H., & Pan, Y. (2024). *Real-Time Vehicle Detection and Urban Traffic Behavior Analysis Based on Unmanned Aerial Vehicle Traffic Videos on Mobile Devices* (SSRN Scholarly Paper No. 4976574). Social Science Research Network. https://doi.org/10.2139/ssrn.4976574

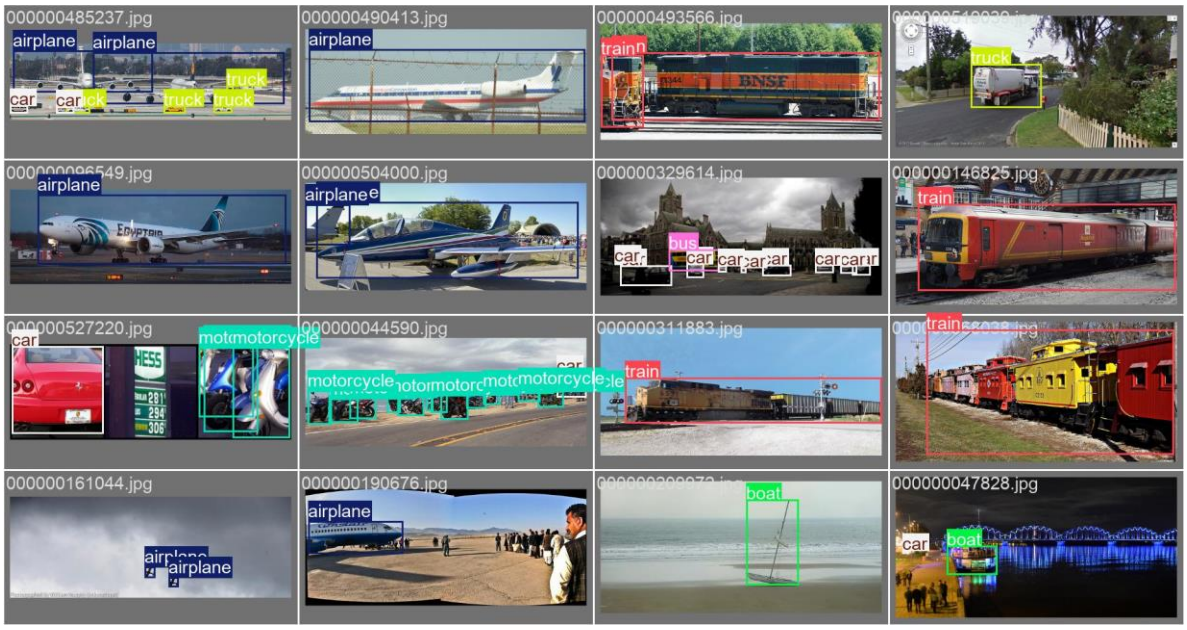
Appendix

Appendix A - Training Batch

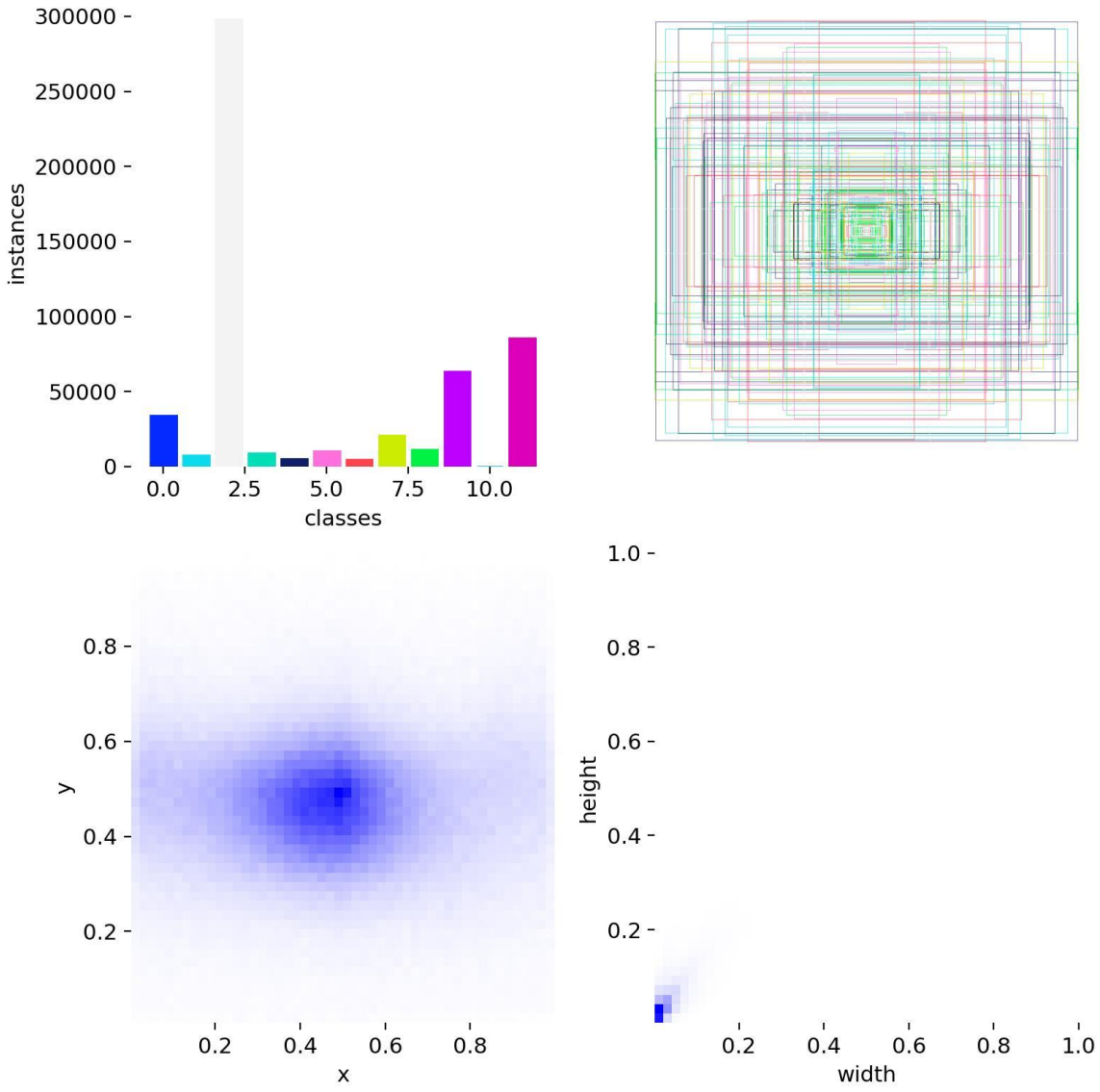


Appendix B

Validation Batch



Appendix C - Class Balance



Appendix D

Confusion Matrix