PM

Gestão inteligente de parque de estacionamento através de processamento de imagem

PROJETO DE MESTRADO

Mauro Emanuel Ferreira Fernando

MESTRADO EM ENGENHARIA ELETROTÉCNICA - TELECOMUNICAÇÕES

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

maio | 2025

Gestão inteligente de parque de estacionamento através de processamento de imagem

PROJETO DE MESTRADO

Mauro Emanuel Ferreira Fernando

MESTRADO EM ENGENHARIA ELETROTÉCNICA - TELECOMUNICAÇÕES

ORIENTAÇÃO

Luís Armando de Aguiar Oliveira Gomes

COORIENTAÇÃO

João Dionísio Simões Barros

UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e da Engenharia

Mestrado em Engenharia Electrotécnica-Telecomunicações

Gestão inteligente de parque de estacionamento através de processamento de imagem

Mauro Emanuel Ferreira Fernando - 2132121

Orientador: Prof. Luís Armando de Aguiar Oliveira Gomes

Co-orientador: Prof. João Dionísio Simões Barros

Maio de 2025

“Os políticos e as fraldas devem ser mudados frequentemente e pela mesma razão.”

Eça de Queiroz

Resumo

Nesta dissertação desenvolveu-se um sistema inteligente de gestão de parques de estacionamento através de processamento de imagem. O algoritmo usado permite a monitorização de lugares de estacionamento utilizando visão computacional para detetar veículos em áreas definidas dentro de um vídeo capturado por um uma câmera.

Este projeto foi implementado com base na arquitetura de rede neuronal convolucional YOLOv8, utilizada para deteção de objetos em imagens. A utilização do algoritmo baseado na YOLOv8 permite a deteção de veículos em lugares de estacionamento, sendo que o algoritmo utiliza a teoria da região de interesse para o mapeamento dos lugares em parques de estacionamento. Dentro do algoritmo utilizam-se dois tipos de regiões de interesse: região de deteção de veículo e região da vaga marcada, sendo que a sobreposição dessas regiões dita a ocupação da vaga de estacionamento. Para complementar o algoritmo foi criada uma página *web* para dar informação em tempo-real da disponibilidade dos lugares de estacionamento aos automobilistas.

Para implementação do hardware foi utilizado um *Raspberry Pi* 4 e um 5, afim de executar a deteção veículos e verificar a disponibilidade de lugares de estacionamentos. O processo de construção do sistema começou na recolha de imagens de veículos para o treino da máquina (responsável pela deteção), e posteriormente a marcação dos lugares a serem monitorizados. Em seguida passou-se à verificação das sobreposições das regiões (se o veículo detetado estava dentro da vaga) e por fim ao envio em tempo-real para a página *web* do estado de disponibilidade dos lugares.

Os resultados obtidos tanto na simulação quanto na implementação prática de hardware foram satisfatórios com uma percentagem de exatidão na deteção de veículos de 89.92% em condições climáticas e de luminosidade adversas, considerando-se por isso o sistema eficiente na deteção de lugares de estacionamento.

Palavras-chave: Processamento de imagem, deteção de lugares de estacionamento, algoritmo para reconhecimento de veículos, *Raspberry Pi.*

Abstract

In this dissertation an intelligent parking management system was developed through image processing. The algorithm used creates a parking space monitoring system that uses computer vision to detect vehicles in defined areas within a video captured by a camera.

This project was implemented based on the YOLOv8 convolutional neuronal network architecture, used for object detection in images. The use of an algorithm based on YOLOv8 allows the detection of vehicles in parking spaces, and the algorithm uses the region of interest theory for the mapping of parking spaces. Within the algorithm there are two types of regions of interest: a vehicle detection region and a marked space region), and the overlap of these regions dictates the occupancy of the parking space. To complement the algorithm, a web page was created to provide real-time information on the availability of parking spaces to motorists.

For the hardware implementation, a Raspberry Pi 4 and a 5 were used in order to perform vehicle detection and to check the availability of parking spaces. In general, the process of building the system began with the collection of vehicle images for machine training (responsible for detection), and then the marking of the spaces to be monitored. This was followed by the verification of the overlaps of the regions (if the detected vehicle is within the space) and finally sending, in real-time, to the web page the availability status of the spaces.

The results obtained both in simulation and in practical implementation were satisfactory, with an accuracy percentage for vehicle detection of 89.92%, in adverse weather and light conditions, thus creating an efficient system in the detection of free parking spaces.

Keywords: Image processing, parking space detection, algorithm for vehicle recognition, Raspberry Pi.

Agradecimentos

Agradeço a Deus pela vida, à minha tia Engrácia da Luz dos Santos Soito João por estar comigo na saúde, doença, alegria e tristeza, por ter-me dado todo o tipo de apoio.

Agradeço à minha mãe, Joana Maria Fragoso Ferreira pelas orações nos momentos mais difíceis da minha vida, desde o início da minha formação académica até ao seu término.

Agradeço ao meu orientador Prof. Luís Armando de Aguiar Oliveira Gomes e co-orientador Prof. João Dionísio Simões Barros, por todo o apoio, orientação e paciência ao longo do desenvolvimento desta dissertação. Suas sugestões e conhecimentos foram essenciais para o meu crescimento acadêmico e profissional, contribuindo significativamente para a realização deste trabalho.

Agradeço por terem dedicado o seu tempo e por acreditarem no meu potencial, fornecendo um ambiente de aprendizagem e incentivo ao longo deste percurso, sem as suas orientações e conselhos valiosos, este projeto não teria atingido o mesmo nível de qualidade e profundidade.

Agradeço ao meu amigo Filipe Santos pela amizade, ajuda e incentivo desde o início da formação.

Lista de acrónimos e abreviações

Adam - Adaptive Moment Estimation

CNN - Convolutional Neuronal Network

COCO - *Common Objets in Context*

ELU - Unidade linear exponencial

FPN - Feature Pyramid Network

ILSVRC - ImageNet Large Scale Visual Recongnition Challenge

IA - Inteligência Artificial

IOU - Intersecção sobre União

mAP - mean Average Precision

OpenCV - Open Source Computer Vision

PAN - Path Agregation Network

RCNN - *Region-base CNN*

RealVNC - Real Virtual Network Computing

Relu - Unidade linear retificada

ROI - Região de Interesse

SSD - *Single Shot Multibox Detector*

YOLO - *You Only Look Once*

Símbolos

$\mathbf{1}_i^{obj}$ – Indicador que vale 1 se a j-ésima caixa delimitadora da célula i contém um objeto, e 0 em caso contrário

$\mathbf{1}_i^{noobj}$ – Indicador que vale 1 se a j-ésima caixa delimitadora da célula i não contém um objeto, e 0 em caso contrário

$\boldsymbol{B}$ – Número de caixas delimitadoras previstas por células

$\boldsymbol{C_i}$ – Confiança prevista de que a caixa delimitadora contém um objeto

$\widehat{\boldsymbol{C}_i}$ – Confiança da verdade de terreno

$\boldsymbol{g_t}$ – Gradiente atual

$\boldsymbol{m_t}$ – Média móvel exponencial dos gradientes

$\boldsymbol{p_i(c)}$ – Probabilidade prevista para a classe c

$\widehat{\boldsymbol{p}_i}(\boldsymbol{c})$ – Probabilidade verdadeira para a classe c

$\boldsymbol{S^2}$ – Número de células na matriz em que a imagem é dividida

$\boldsymbol{v_t}$ – Média móvel exponencial dos quadrados dos gradientes

$\boldsymbol{x_i}$ $\boldsymbol{e}$ $\boldsymbol{y_i}$ – Coordenadas do centro da caixa delimitadora prevista

$\widehat{\boldsymbol{x}_i}$, $\widehat{\boldsymbol{y}_i}$, $\widehat{\boldsymbol{w}_i}$, $\widehat{\boldsymbol{h}_i}$ – Parâmetros do modelo

$\boldsymbol{w_i}$ $\boldsymbol{e}$ $\boldsymbol{h_i}$ – Coordenadas do centro da caixa delimitadora prevista

$\boldsymbol{\lambda_{coord}}$ – Peso dado à perda de localização

$\boldsymbol{\lambda_{noobj}}$ – Peso dado à perda de confiança quando não há objeto na caixa

$\boldsymbol{\theta_t}$ – Parâmetros do modelo

η – Taxa de aprendizagem

$\in$ – Termo de estabilização numérica

Índice

Índice de figuras

Índice de tabelas

1. Introdução

Este Projeto de Mestrado em Engenharia Eletrotécnica-Telecomunicações apresenta uma solução para a gestão e monitorização em tempo real de lugares de estacionamentos ao ar-livre, com o objetivo de reduzir o congestionamento na procura de lugares para o estacionamento de veículos por parte dos automobilistas.

Existem atualmente várias cidades que disponibilizam às populações soluções inteligentes para o seu dia-a-dia, como é o caso de Amesterdão e Barcelona, possuindo vários projetos promissores na área de mobilidade, energia, educação, resíduos, entre outros [1].

Neste contexto surgiu a ideia de estudar a gestão de estacionamento e testar num dos estacionamentos da Universidade da Madeira, tendo em vista definir uma solução que permita resolver problemas de engarrafamentos, e reduzir a poluição e gastos de combustível e de tempo dos automobilistas, para encontrar um lugar de estacionamento da forma mais cómoda possível.

1.1 Motivação

A partir da revolução industrial, as pessoas tiveram a obrigação e necessidade de se deslocarem para áreas específicas onde era possível terem uma melhor qualidade de vida, quer seja a nível profissional, a nível de educação ou a nível de cuidados de saúde, surgindo assim o conceito de cidade, um local onde as pessoas se aglomeram.

Um dos problemas da maioria das cidades é a elevada densidade populacional, e com este aumento surgem outros problemas, como por exemplo, ambientais, socioculturais e governamentais. Desta forma é imprescindível o uso das tecnologias com o objetivo da modernização dos serviços e da gestão dos recursos, garantindo um desenvolvimento mais sustentável das cidades, criando-se assim o conceito de *Smart Cities*, que tem como objetivo tornar uma cidade mais auto--sustentável, melhorando desta forma a qualidade de vida da sua população.

Sendo que atualmente, num mundo cada vez mais acessível em termos de tecnologia, existe um aumento do número de cidades inteligentes em desenvolvimento, como por exemplo na Noruega, Áustria, Suíça, países baixos, entre outros [1].

Com o crescimento das cidades começaram a surgir novos problemas nas grandes cidades, isto é, a nível de trânsito, a nível de lugares de estacionamento, poluição do ar, poluição sonora e poluição visual. O número de lugares de estacionamento não tem acompanhado o aumento do número de automóveis em circulação.

Com isto, hoje em dia, encontrar um lugar de estacionamento nas áreas metropolitanas pode ser um grande desafio para os automobilistas, podendo a procura tornar-se caótica se realizada em hora de ponta. Sendo que o tipo de procura mais utilizado é designado de “procura às cegas”, que consiste em fazer uma procura sequencial por um lugar na zona perto do destino do automobilista [1].

Este tipo de procura é muito cansativo para o automobilista pois, para além deste perder muito tempo, gasta muito combustível, contribuindo assim para o congestionamento automóvel na área em questão e para o aumento da poluição que este gerará. De forma a mitigar este tipo de problema surgiu uma solução: Criar um sistema de deteção de lugares estacionamento, com uma página *web* que permita informar aos automobilistas da disponibilidade de lugares.

1.1.1 Identificação do problema

Neste estudo, o projeto do sistema de informação da disponibilidade de lugares de estacionamento baseado em processamento de imagem é um trabalho complexo que envolve o uso da uma máquina que deve ser treinada para detetar quaisquer veículos validados a partir da perspetiva da câmera, sendo proposto o uso do algoritmo de subtração de fundo para rastrear qualquer deteção dos veículos.

Naturalmente é necessário avaliar quão preciso esse algoritmo seria num programa em tempo real. Existem alguns algoritmos que envolvem o programa de processamento de imagem, como o W4 [2], o modelo Gaussiano único [3] e o algoritmo LOTs [4], que é um dos mais precisos na deteção de veículos.

Outro problema do programa de processamento de imagem está no entorno do estacionamento, isto é, a precisão com que o programa consegue detetar os veículos é afetada pela localização da câmera. Os ambientes que estão expostos a qualquer clima imprevisível (se o dispositivo de captura estiver localizado na parte externa do edifício), como chuva ou vento, têm influência no reconhecimento do número de lugares vagos do estacionamento.

Em resumo, os dois principais problemas de identificação do projeto eram:

- Quão preciso será o modelo treinado para detetar os veículos?
- A precisão seria mantida em diferentes condições ambientais?

Este estudo aborda como implementar um sistema que pode rastrear a disponibilidade de lugares de estacionamento utilizando técnicas de processamento de imagem. Projetar o sistema de lugares de estacionamento requer técnicas de processamento de imagens que envolvem a implementação de aprendizagem de máquina (*Machine Learning*).

A aprendizagem de máquina é a maneira mais simples de detetar veículos por meio de processamento de imagem, como o nome indica, é a deteção automática de padrões razoáveis em termos de dados [5].

Para criar esses dados é possível usar o modelo de rede neuronal YOLOv8 que contém um arquivo vetorial de processamento. Em seguida, o resultado dos dados do arquivo em YOLO será transferido para o hardware que neste projeto foi um *Raspberry Pi*. Ao utilizar a placa de câmera do *Raspberry Pi*, o processamento de imagem pode ser feito e implementado num único dispositivo. A partir daí, o resultado é calculado e verificado a precisão das deteções de veículos e da disponibilidade de estacionamentos.

1.2 Objetivos

O objetivo principal deste projeto de mestrado foi estudar e desenvolver um sistema inteligente de deteção de lugares de estacionamento baseado em processamento de imagem, tendo em conta os seguintes aspetos: o número de lugares vagos numa dada área e a disponibilização de informação, em tempo real, sobre os estados dos lugares para os automobilistas.

De uma forma mais específica, pretendeu-se:

- Estudar conceitos de *machine learning* e modelos de treino de máquina para deteção de veículos;
- Treinar um modelo para a deteção de veículos e combinar com um algoritmo de tomada de decisão sobre a disponibilidade de lugares;
- Desenvolver uma página *web* que irá informar em tempo real sobre a disponibilidade dos lugares de estacionamento;

- Projetar esquemas analíticos para a implementação do sistema de gestão inteligente de estacionamento;
- Analisar o sistema a partir de simulações;
- Montar um protótipo e retirar resultados experimentais.

1.3 Organização do projeto

Esta secção apresenta a visão geral do conteúdo deste relatório.

No primeiro capítulo, Introdução, foi apresentada e explicada a motivação para a realização deste projeto, definidos os objetivos e descrita a organização do projeto.

O segundo capítulo, revisão da bibliografia, abordará o conhecimento geral do projeto, o que envolve o conhecimento comum do algoritmo para deteção dos veículos e como é possível produzir o referido algoritmo que será implementado no hardware.

No terceiro capítulo, metodologia, será descrita a metodologia e os procedimentos de desenvolvimento da gestão de lugares de estacionamento baseado em processamento de imagem, tanto na parte de simulação quanto na implementação em hardware.

No quarto capítulo, análise e resultados, serão apresentados o desenvolvimento do protótipo e os respetivos resultados, tanto da simulação e implementação no hardware.

No quinto capítulo, conclusões, o resumo dos resultados será correlacionado com os objetivos do projeto, para se verificar se foram cumpridas as metas traçadas.

As referências e anexos, são a parte final do relatório, e incluem as referências que foram utilizadas nesta pesquisa e os diversos itens que podem ser considerados como parte complementar do projeto.

2. Revisão da bibliografia

Na revisão da bibliografia começa-se por fazer uma introdução aos sistemas de gestão de lugares de estacionamento e descrevem-se os principais defeitos. Numa segunda fase apresentam-se conceitos relacionados com *machine learning,* redes neuronais e a especificação de uma ferramenta de treino para a deteção de veículos. Na última fase descreve-se como o algoritmo vai determinar os estados dos lugares (livres ou ocupadas) através da teoria das regiões de interesse e como é feita a marcação dos lugares de estacionamento.

2.1 Tecnologias de deteção

Um sistema automatizado de gestão de disponibilidade de estacionamento é aquele que deteta qualquer lugar de estacionamento disponível dentro de uma determinada área de interesse. A automação da tecnologia de disponibilidade de estacionamentos pode ser feita por dois tipos de sistema [6]: método baseado em sensores e método baseado em imagem.

A comparação entre estes dois tipos de tecnologias de deteção pode ser vista na tabela 1.

Tabela 1- Comparação entre as tecnologias de deteção de disponibilidade de estacionamentos [6].

Tipo de deteção tecnológica	Técnica de tecnologia	Dispositivo usado
Método baseado em sensores	Usa emissor de ondas ultrassónicas para a área de estacionamento e recebe a onda de volta após ser refletida.	Sensores ultrassónicos
Método baseado em visão	Usa câmeras como método de captura para visualizar a localização da área de estacionamento. O sistema rastreia os dados de imagem e deteta a disponibilidade de lugares de estacionamento.	Câmera

O método baseado em sensores é amplamente utilizado em sistemas de gestão de estacionamentos, mas existem desvantagens em usar a tecnologia baseada em sensores. Um dos problemas é que a implementação desta tecnologia requer muitos dispositivos para funcionar, uma vez que o dispositivo só pode rastrear um lugar de estacionamento.

Isto significa que se houver 50 lugares de estacionamento, numa única área de cobertura, isso exige que 50 dispositivos ultrassónicos sejam colocados, um para cada lugar de estacionamento individual. Isso envolve custos elevados para adquirir e instalar esses sensores ultrassónicos, com base no número de lugares de estacionamento.

Outra condicionante é que cada dispositivo deve ser colocado no topo da área de estacionamento para que possa operar com um desempenho ideal, o que é difícil para uma área de estacionamento em espaço aberto. Outro problema prende-se com o custo e a complexidade da instalação de cablagem para cada sensor ultrassónico.

O método baseado em visão resolve o problema existente na tecnologia baseada em sensores. Como o nome indica, esta tecnologia utiliza técnicas de processamento de imagem para digitalizar vídeos ao vivo, capturados pelas câmeras, e rastrear quaisquer veículos detetados dentro da cobertura da câmera.

Esta tecnologia utiliza uma única câmera que pode cobrir toda a área do estacionamento. Neste trabalho a implementação da tecnologia de deteção baseada em visão computacional será estudada, para se analisar o método mais adequado para detetar a disponibilidade de lugares de estacionamento com precisão.

2.2 Perspetiva histórica

Embora o olho humano seja capaz de identificar, instantaneamente e precisamente, um determinado objeto, incluindo o seu conteúdo, localização e recursos visuais próximos, os sistemas habilitados para visão computacional são relativamente baixos em precisão e velocidade.

São por isso muito importantes quaisquer avanços que levem a melhorias na criação de sistemas mais inteligentes, muito parecidos com os humanos, como por exemplo, dirigir um veículo equipado com uma tecnologia de condução assistida habilitada por visão computacional, poderia prever e tentar impedir um acidente de trânsito antes do incidente, mesmo que o motorista não esteja consciente de suas ações.

Por isso a deteção de objetos em tempo real tornou-se um assunto extremamente necessário no progresso da automação. A visão computacional e a deteção de objetos são campos de grande aplicação na aprendizagem de máquina e espera-se que ajudem a desbloquear os potenciais dos sistemas robóticos de resposta geral.

Dentro desta área vale a pena destacar a invenção do primeiro algoritmo de Rede Neuronal Convolucional (*Convolutional Neural Network* - CNN) na década de 1990 e avanços significativos como a AlexNet [7], que venceu o *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) em 2012, na área de classificação de imagens (mais tarde referido como ImageNet). Os algoritmos CNN têm sido capazes de fornecer soluções eficazes para a deteção de objetos em várias abordagens.

2.3 Rede neuronal convolucional

As redes neuronais convolucionais (CNN) podem ser consideradas uma subcategoria das Redes Neurais Profundas, especificamente inventadas para processamento de imagens e deteção de objetos. Os algoritmos CNN podem ser utilizados sem exigirem uma enorme quantidade de parâmetros substanciais predefinidos para a imagem fornecida. Essa facilidade em treinar um modelo e a grande quantidade de informações disponíveis na internet tornaram possíveis a utilização dos algoritmos das CNN [8].

Os seres humanos possuem uma notável capacidade de reconhecer cenários e identificar objetos com facilidade, quer ao observar uma imagem, quer ao analisar o mundo ao seu redor. Esse processo ocorre de forma intuitiva e inconsciente, permitindo-nos reconhecer padrões, generalizar informações e adaptar-nos a diferentes contextos sem esforços significativos. No entanto, ao contrário dos humanos, as máquinas não possuem essa aptidão natural [9].

Quando um computador recebe uma imagem como entrada, esta é interpretada apenas como um conjunto de valores numéricos que representam os pixéis. A máquina organiza esses valores numa matriz, onde cada número, variando entre 0 e 255, indica a intensidade da cor correspondente a cada pixel, tal como se pode observar na figura 1mbora para os humanos esses números possam não ter um

significado imediato, são a única forma pela qual as máquinas conseguem "visualizar" e processar imagens [9].

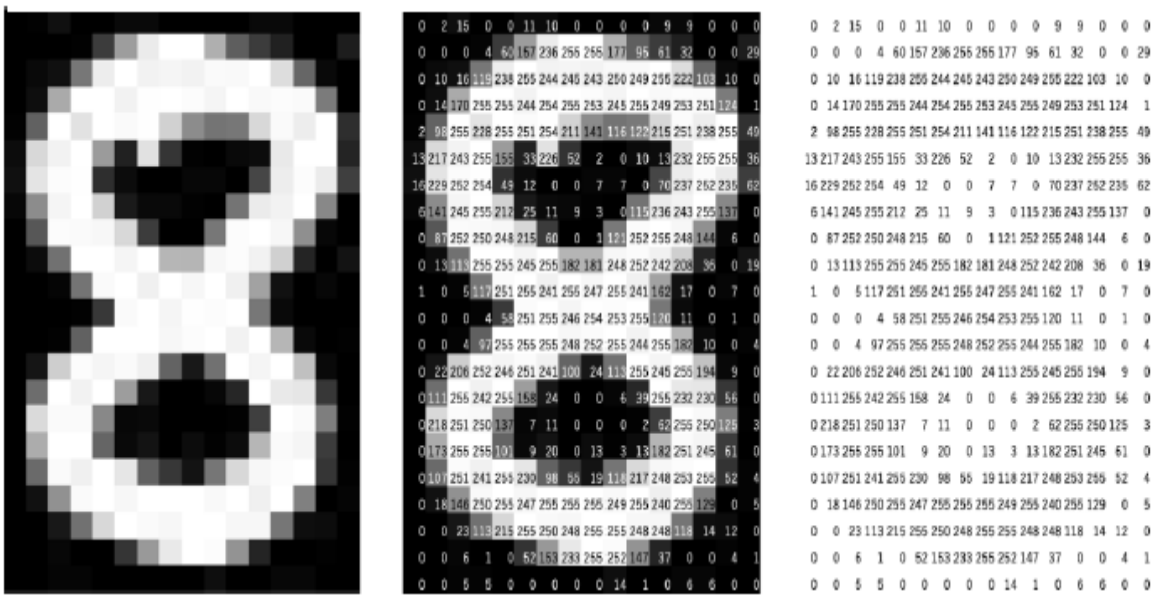


Figura 1- Imagens como matriz de pixéis [10].

2.3.1 Classificação e segmentação semântica de imagens

O objetivo é que as máquinas sejam capazes de distinguir, entre um conjunto de imagens, as características únicas que definem, por exemplo, um cão como um cão e um gato como um gato. De uma forma geral, as redes neuronais convolucionais analisam padrões específicos de baixo nível presentes na imagem, como contornos, arestas e curvaturas. À medida que essas características são processadas por várias camadas com diferentes funções, a rede consegue determinar a classe que melhor descreve a imagem, atribuindo uma distribuição de probabilidades de cada possível categoria (figura 2) [11].

Figura 2- Classificação de imagem de gato pela *AlexNet (PASCAL VOC)* [12].

Para além da tarefa de classificação de imagens, na qual é atribuída uma única descrição de alto nível a toda imagem, estudos recentes demonstram que, com algumas modificações na sua estrutura, as redes neuronais convolucionais podem

ser aplicadas a outras tarefas avançadas, como deteção de objetos, segmentação semântica de imagens e segmentação de instâncias (figura 3). Estas técnicas permitem extrair informações mais detalhadas e localizar elementos específicos dentro do ambiente, tornando a análise mais precisa e contextualizada [13].

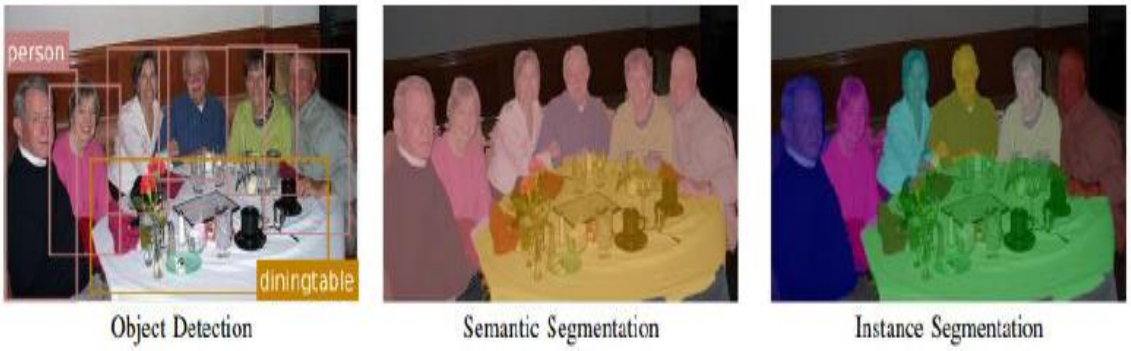


Figura 3- Deteção, segmentação semântica e por instâncias [13].

No caso da segmentação semântica, o objetivo é proporcionar uma compreensão mais detalhada do ambiente, atribuindo a cada pixel da imagem uma categoria específica. Esta abordagem permite que as máquinas extraiam informações de diversos cenários do mundo real, possibilitando a realização de várias tarefas [13].

Entre as principais aplicações da segmentação semântica de imagens, destacam-se [13]:

- A utilização em veículos autónomos, onde é essencial um reconhecimento preciso do ambiente;
- O diagnóstico médico, facilitando a segmentação de células, tecidos e órgãos de interesse;
- O desenvolvimento de robôs capazes de navegar e manipular objetos no ambiente;
- A edição de imagens e vídeos e a criação de “óculos inteligentes” concebidos para descrever o ambiente e pessoas com deficiência visual.

2.4 Componentes básicos das redes de convolução

Uma imagem percorre uma série de camadas dentro de uma rede CNN rede antes de gerar uma saída, que pode consistir numa distribuição de probabilidades. A forma como estas camadas e componentes são organizados desempenha um papel crucial na definição das diferentes arquiteturas de redes neuronais, cada uma com as suas próprias vantagens e limitações [14].

De forma geral, a arquitetura básica de uma CNN é composta por camadas alternadas de convolução e *pooling*, seguidas por uma ou mais camadas totalmente conectadas (figura 4).

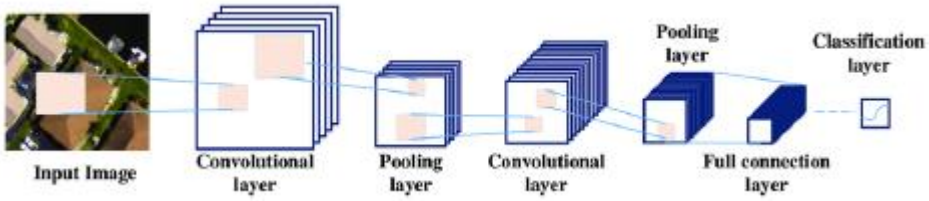


Figura 4- Estrutura típica de uma CNN com sete camadas [15].

2.4.1 Camada de convolução

A convolução é uma operação matemática entre duas funções que gera uma terceira, representando a forma como a primeira é modificada pela segunda [16]. No contexto das redes neuronais convolucionais, o primeiro argumento desta operação corresponde à imagem de entrada (*input*), enquanto o segundo, conhecido como filtro (ou *kernel*, no termo mais comum em inglês), atua sobre a imagem. O resultado desta operação é denominado mapa de ativação (*feature map*) [17]. Compreender o conceito e a matemática subjacente à convolução é fundamental, uma vez que este processo ocorre repetidamente ao longo das diferentes camadas da rede.

Para ilustrar o funcionamento desta operação, pode-se imaginar um feixe de luz que incide sobre uma pequena região da imagem. Partindo do canto superior esquerdo, esse feixe desliza progressivamente sobre toda a imagem. Suponha-se ainda que esse feixe transporta um conjunto de pesos, e que, ao sobrepor-se a cada nova região da imagem, calcula o produto entre os seus pesos e valores dos pixéis dessa área. Nesta analogia, o feixe de luz representa o filtro, enquanto a área que este cobre é designada como campo recetivo. Os produtos gerados são armazenados nos chamados mapas de ativação [18].

Um exemplo deste processo pode ser observado na figura 5. Para cada campo recetivo, calcula-se a soma dos produtos obtidos, resultando num único valor. Esse valor indica o grau de ativação daquela região em relação ao filtro aplicado.

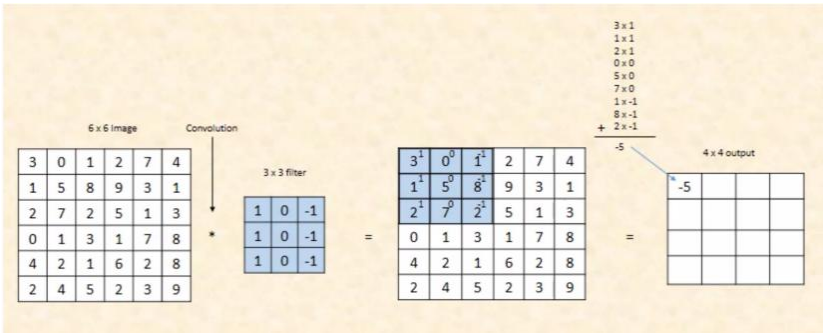


Figura 5- Convolução entre a imagem de entrada 6x6 e o filtro 3x3, *stride* 1 [19].

Esse processo repete-se até que toda a imagem tenha sido percorrida, dando a origem a um mapa de ativação (figura 6).

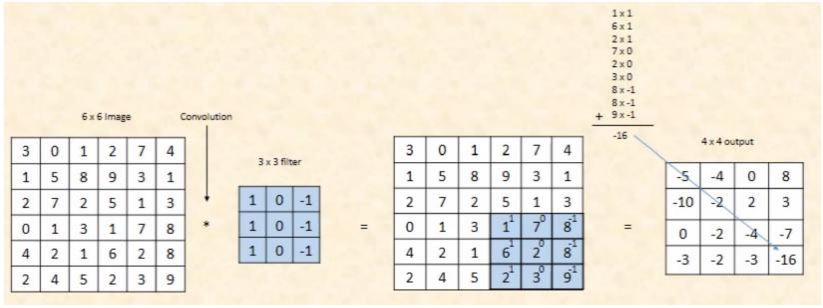


Figura 6- Um mapa de ativação com dimensão 4x4 [19].

No exemplo apresentado, uma imagem de entrada com dimensão 6x6, ao ser processada por um filtro de 3x3 gerou um mapa de ativação com tamanho 4x4. Em certas situações, essa redução nas dimensões pode não ser desejável. Para evitar esse efeito, adicionam-se um conjunto de zeros em torno da imagem original, uma técnica conhecida como zero *padding* (figura 7) [20].

A distância percorrida pelo filtro entre cada campo recetivo é denominada por *stride.* No exemplo apresentado, o *stride* é igual a 1, uma vez que o filtro avança 1 pixel em cada iteração. A manipulação desta e de outras variáveis permite controlar o tamanho do mapa de ativação gerado [15].

Para cada filtro utilizado numa camada de convolução, é gerado um mapa de ativação, que destaca as regiões onde as características identificadas pelo filtro estão presentes [16].

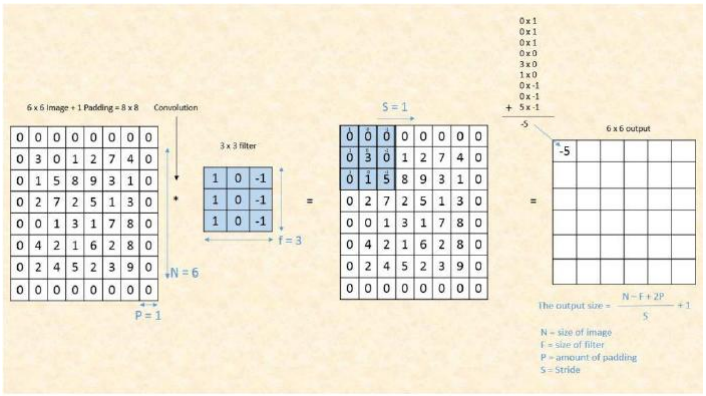


Figura 7- Zero *padding* evita a redução das dimensões na convolução [20].

Na figura 8 são apresentados quatro filtros distintos, concebidos para detetar contornos na imagem. A aplicação desses filtros resultou na geração de quatro mapas de ativação, evidenciando diferentes características das bordas da imagem original.

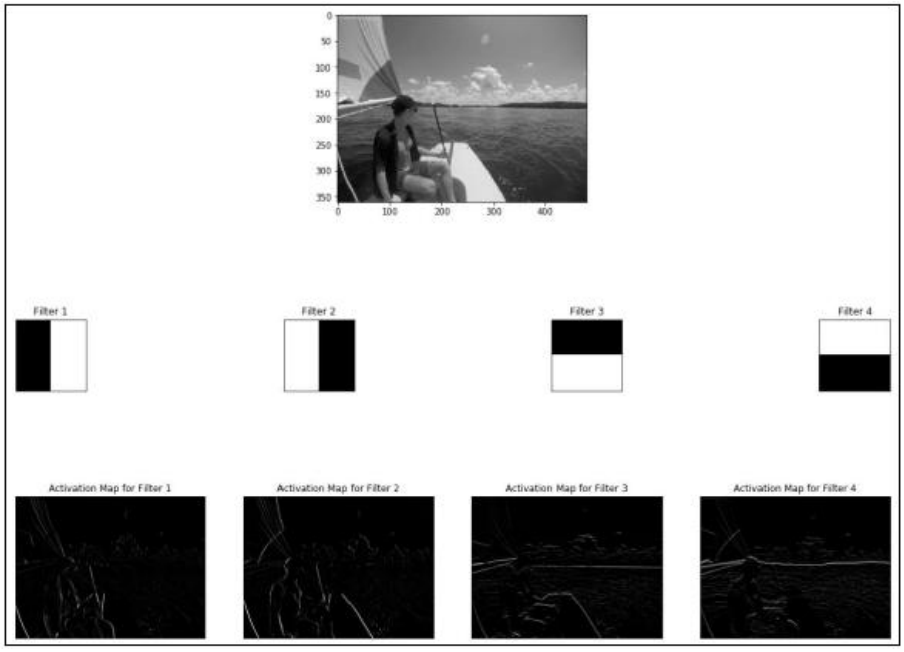


Figura 8- Mapas de ativação gerados usando quatro filtros [23].

No caso de imagens a cores, a principal diferença está no facto de a entrada possuir três dimensões. Como ilustrado no exemplo da figura 9, a imagem de entrada tem dimensões de 6x6x3, sendo que o último valor corresponde ao número de canais, que representam as intensidades de vermelho, verde e azul (RGB) de cada pixel. É desta forma que as imagens a cores são representadas.

A operação matemática permanece essencialmente a mesma, com a diferença de que o filtro também deve possuir três canais. Para cada campo recetivo, realiza-se

o produto entre os valores dos pixéis e os respetivos canais do filtro. Em seguida, os três valores obtidos são somados, resultando, mais uma vez, num único número (figura 9).

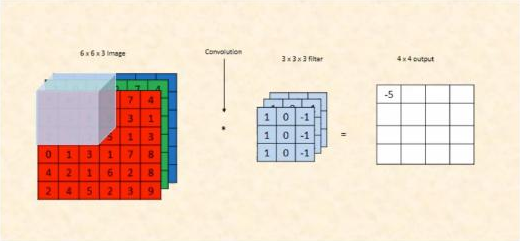


Figura 9- Canais de uma imagem colorida [24].

É importante salientar que a largura e a altura do filtro podem ser ajustadas conforme necessário. No entanto, para que a operação matemática seja válida, a profundidade do filtro deve corresponder à profundidade da entrada, ou seja, deve ter o mesmo número de canais [25].

2.4.2 Camada não-linear

Após as camadas de convolução, é convencional aplicar imediatamente uma camada não linear, também conhecida como camada de ativação. O objetivo desta camada é introduzir não linearidade num sistema que, até esse ponto, realizou operações lineares, como multiplicações e somas, na camada de convolução [25].

A introdução de não linearidade no modelo permite estabelecer relações mais complexas entre as entradas e saídas da rede, o que é essencial para a aprendizagem de padrões em dados complexos, como imagens [25].

No passado, funções como a tangente hiperbólica e a sigmóide eram frequentemente utilizadas como funções de ativação. No entanto, estudos mais recentes demonstram que a função *ReLU* (*Rectified Linear Unit*) proporciona melhores resultados, pois permite que a rede seja treinada de forma significativamente mais rápida, devido à sua eficiência computacional, sem comprometer a precisão do modelo [26]. A camada *ReLU* aplica a função f(x) = max(0,x) a todos os valores do volume de entrada (figura 10).

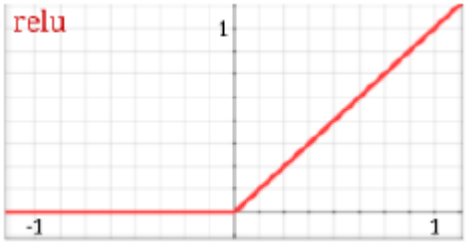


Figura 10- Função ReLU [27].

Em termos simples, essa operação substitui todas as ativações negativas geradas pela convolução por zero (figura 11). Dessa forma, a ReLU aumenta a capacidade não linear do modelo sem alterar as ativações positivas provenientes da camada de convolução [27].

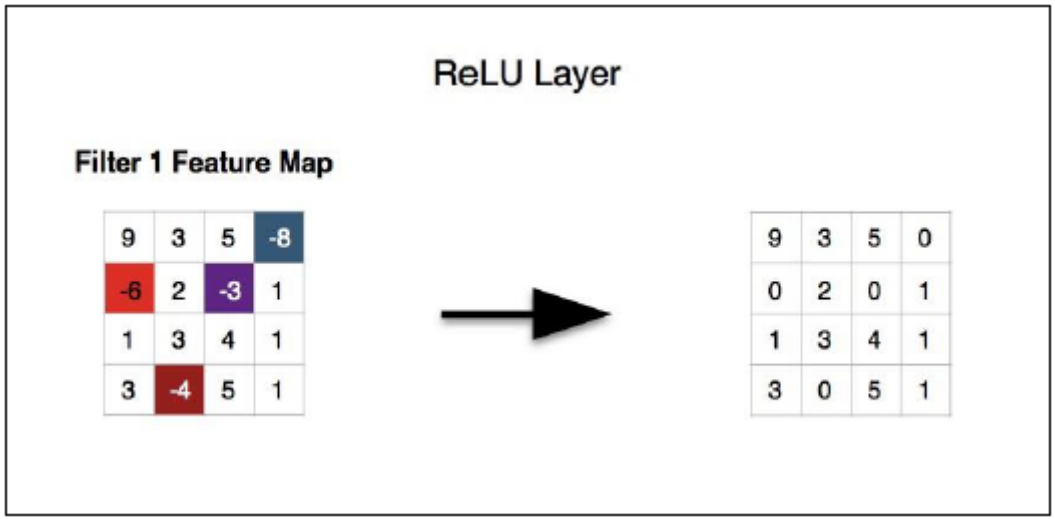


Figura 11- Aplicação da camada ReLU a um mapa de ativação [28].

2.4.3 Camada de *pooling*

Para aumentar a profundidade da rede, é necessário reduzir a dimensão espacial da representação da imagem. A camada de *pooling* tem precisamente essa função, ao condensar a informação e diminuir significativamente o número de parâmetros da rede. Esse processo, também conhecido como *downsampling*, contribui para a otimização do tempo de processamento e acelera os cálculos computacionais da rede [29].

Além da eficiência computacional, a camada de *pooling* também melhora a invariância espacial da rede, tornando-a mais robusta a variações como rotações, alongamentos, contrações, ou deslocamentos das características identificadas [29].

O funcionamento desta camada é relativamente simples: de forma semelhante à convolução, define-se uma região retangular e um *stride*. No entanto, em vez de extrair novas informações, o objetivo é comprimir e consolidar os dados previamente extraídos. A seguir, são apresentados os dois tipos de *pooling* mais utilizados.

Na primeira abordagem, para cada região percorrida, é selecionado e retornado o valor máximo dentro do retângulo definido (figura 12). Este método, conhecido como *max pooling*, é o mais utilizado entre os diferentes tipos de *pooling* [30].

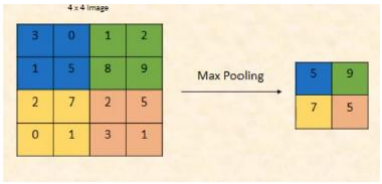


Figura 12- *Max pooling* para um retângulo 2x2 e *stride* 2 [30].

Na segunda abordagem, para cada região percorrida, é calculada e retornada a média dos valores dentro do retângulo definido (figura 13). Este método, conhecido como *average pooling*, é uma alternativa ao *max pooling* e pode ser utilizado dependendo da aplicação desejada [30].

Figura 13- *Average pooling* para um retângulo 2x2 e *stride* [30].

2.4.4 Camada totalmente conectada

Esta camada tem a função de produzir a decisão final sobre a classificação da imagem. Esta camada faz parte da estrutura das redes de classificação e depende da análise e extração de características realizadas pelas camadas anteriores, nomeadamente as de convolução e *pooling* [30].

Nesta fase, todas as características e atributos extraídos da imagem são interligados através de uma estrutura neuronal tradicional, composta por camadas ocultas, onde todos os neurónios estão totalmente conectados entre si (figura 14) [30]. Essa interligação permite que a rede combine as informações aprendidas nas camadas anteriores para produzir a classificação final da imagem.

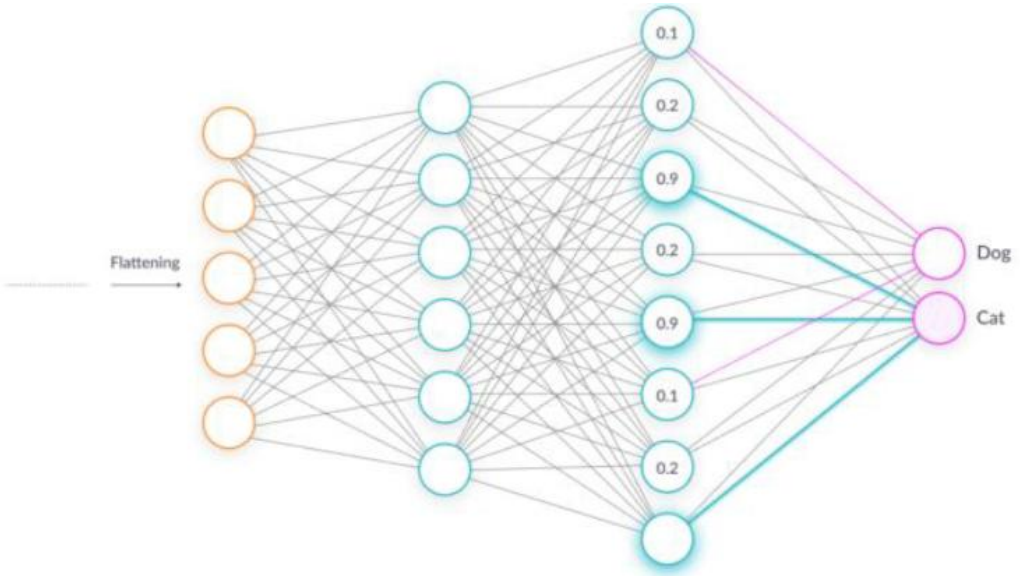


Figura 14- Exemplo de camadas totalmente conectadas de uma CNN [30].

Antes de entrar na camada totalmente conectada, a saída da camada anterior deve passar por um processo de achatamento (*flattening*). Neste processo, os mapas de ativação, que até então eram representados como matrizes, são transformados num único vetor, que servirá de entrada para a camada totalmente conectada [30].

A camada totalmente conectada analisa as saídas da camada anterior, que representam mapas de ativação com características de alto nível de abstração, e determina quais classes estão mais correlacionadas com essas características. Por exemplo, se a rede classifica uma imagem como sendo um cão, isso indica que os mapas de ativação associados a características como patas ou a presença de quatro pernas obtiveram valores elevados. De forma semelhante, se a rede prevê que a imagem pertence à classe “pássaro”, é porque os mapas de ativação associados a bicos e asas apresentaram valores altos [30].

A rede estabelece essas associações através de um processo de treino, ajustando os seus parâmetros de acordo com os exemplos apresentados. Para converter os valores das previsões em probabilidades normalizadas, pode ser utilizada uma função *softmax*, que atribui a cada classe uma percentagem correspondente à sua probabilidade de ser a correta [30].

2.5 Tipos de algoritmos de deteção de objetos

Os algoritmos disponíveis para a deteção de objetos podem ser divididos em duas categorias: algoritmos baseados em classificação e algoritmos baseados em regressão. Ambos são utilizados para prever um valor de saída com base num conjunto de dados de entrada, mas distinguem-se pelo tipo de variável que pretendem prever [31].

- **Classificação**: O objetivo é atribuir uma categoria ou um rótulo discreto a uma determinada amostra. Exemplos incluem a deteção de e-mails de *spam* ("*spam*" ou "não *spam*") ou a classificação de imagens entre "gato", "cão" e "pássaro";
- **Regressão:** O objetivo é prever um valor contínuo. Exemplos incluem a previsão do preço de um imóvel com base em características como localização, área e número de divisões.

A tabela 2 resume as diferenças entre algoritmos de Classificação e Regressão:

Tabela 2- Diferenças entre algoritmos de Classificação e Regressão [31].

Critério	Classificação	Regressão
Saída	Variável categórica (ex.: "positivo" ou "negativo", "classe A" ou classe B").	Variável contínua (ex.: temperatura, preço de um produto).
Objetivo	Associar um rótulo a uma amostra com base em características.	Estimar um valor numérico contínuo com base em padrões.
Exemplo	Diagnóstico médico (se um paciente tem uma determinada doença).	Previsão do consumo de energia numa habitação.
Métricas comuns	Exatidão, Precisão, *Recall*, F1-Score.	Erro Médio Quadrático (MSE), Coeficiente de Determinação (R^2).
Algoritmos comuns	Árvore de decisão, KNN, SVM, *Random Forest,* Redes Neuronais.	Regressão Linear, Regressão Polinomial, SVR, *Random Forest Regressor.*

2.5.1 Algoritmos baseados em classificação

Num algoritmo baseado em classificação, a primeira etapa é a seleção da região de interesse (RoI) na imagem, tal como se pode ver na figura 15. Em seguida, essa região é classificada com o uso de uma rede neuronal convolucional. Esta abordagem de realizar um estágio antes do outro pode ser lenta devido à necessidade de executar os algoritmos de predição em cada região selecionada no primeiro estágio.

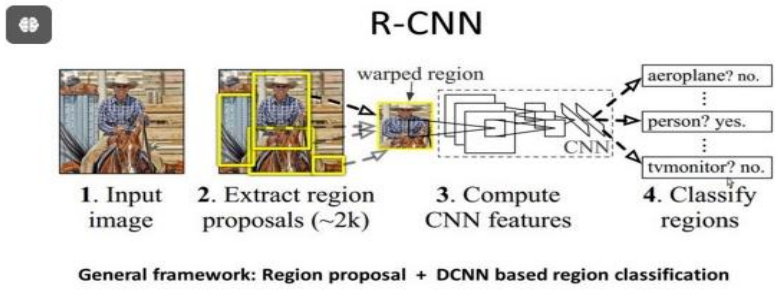


Figura 15- Estrutura típica do algoritmo R-CNN [33].

Alguns exemplos comuns para este tipo de algoritmo são o Retina Net, *Region base* CNN (RCNN), o *Fast-RCNN*, *Faster R-CNN* e *Mask-RCNN* [32].

Na figura 15, a primeira camada (1) recebe uma imagem de entrada, na segunda camada (2) é realizada a extração de 2000 propostas de regiões utilizando um método de segmentação ascendente (*bottom-up*), a terceira camada (3) calcula as características para cada região proposta utilizando uma rede CNN de grande escala e, em seguida, a última camada (4) classifica cada região utilizando máquinas de vetores de suporte lineares (*SVM*) específicas para cada classe.

2.5.2 Algoritmos baseados em regressão

Os algoritmos baseados em regressão são implementados de forma a prever simultaneamente as classes dos objetos e as respetivas caixas delimitadoras em toda a imagem, sem a necessidade de destacar regiões de interesse previamente. Como a deteção de objetos pode ser formulada como um problema de regressão, esses algoritmos eliminam a necessidade de um *pipeline* (sequências de etapas complexas, ou seja, uma rede neuronal complexa), tornando o processo mais eficiente e rápido [34].

Exemplos desse tipo de algoritmo são os algoritmos *Single Shot Multibox Detector* (SSD) e YOLO (*You Only Look Once*). Devido à simultaneidade da deteção e à sua natureza de alta velocidade, eles são comummente usados para a deteção de objetos em tempo real.

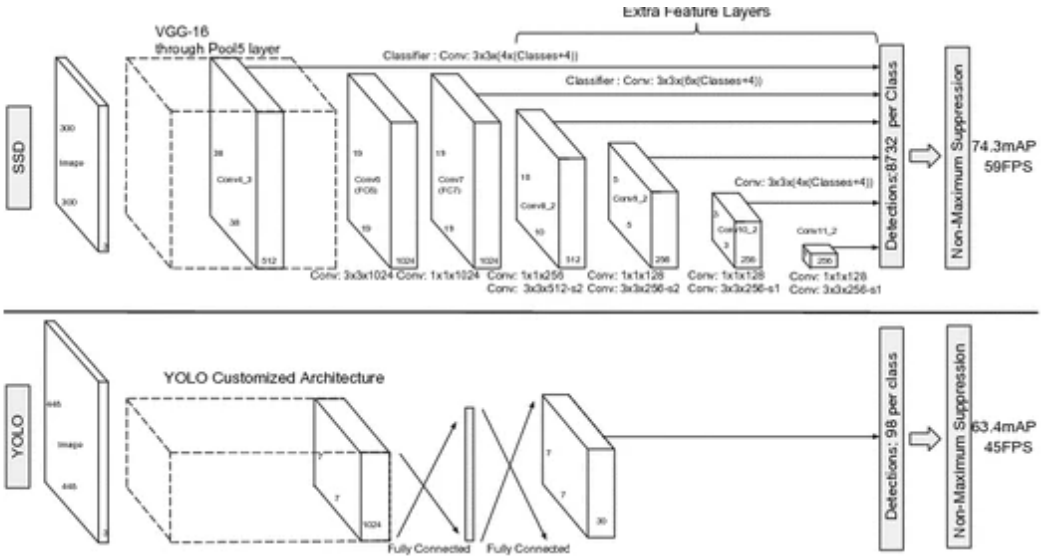


Figura 16- Comparação entre o modelo SSD e YOLO [34].

A figura 16 mostra a comparação entre esses dois modelos de deteção de tiro único. O SSD adiciona várias camadas de características ao final de uma rede base, que preveem os deslocamentos em relação às caixas predefinidas de diferentes escalas e proporções, bem como as suas taxas de confiança associadas. O SSD, com um tamanho de entrada de 300x300, supera claramente o seu equivalente YOLO 448x448 em precisão e velocidade [34].

A deteção e compreensão dos algoritmos YOLO mais populares requerem uma configuração inicial do que será previsto antes dos modelos serem usados. A previsão resulta numa caixa delimitadora (especificando a localização do objeto) junto com uma classe que possui a maior probabilidade entre o conjunto estabelecido de classes.

2.6 *You Only Look Once*

YOLO é uma família de modelos de deteção de objetos em tempo real. Diferente dos métodos tradicionais que utilizam múltiplas etapas, o YOLO transforma a deteção de objetos num único problema de regressão, onde uma única rede neuronal prevê as caixas delimitadoras (*bounding boxes*) e as classes dos objetos

simultaneamente numa única passagem pela imagem [34]. O YOLO divide o processo de deteção em três componentes principais:

Primeiro a divisão da imagem em matriz S×S (exemplo: 7×7 para YOLOv1). Cada célula da matriz é responsável por prever N caixas delimitadoras (*bounding boxes*). Depois atribui uma pontuação de confiança a cada caixa, e classifica os objetos contidos na célula.

A segunda componente é a regressão direta das caixas delimitadoras. O modelo prevê diretamente as coordenadas das caixas delimitadoras em relação à imagem, incluindo as coordenadas normalizadas do centro da caixa dentro da célula (x,y), a largura e altura da caixa normalizadas pela imagem (w,h), e a confiança na deteção do objeto (C).

Finalmente a terceira componente é a classificação das classes dos objetos. Cada caixa recebe uma pontuação baseada na probabilidade de conter um objeto e na classe do objeto identificado.

Note-se que desde a sua introdução, o YOLO passou por várias versões, cada uma trazendo melhorias significativas, tal como apresentado na tabela 3:

Tabela 3- Evolução do YOLO [35].

Versão	**Ano**	**Principais Melhorias**
YOLOv1	2015	Introdução do modelo YOLO; processamento rápido, mas baixa precisão para objetos pequenos.
YOLOv2	2016	Melhorias na arquitetura, introdução de ancoragem (*anchor-boxes*) e melhor compromisso entre rapidez e precisão.
YOLOv3	2018	Uso de redes mais profundas (Darknet-53), deteção multi-escala e melhor classificação.
YOLOv4	2020	Uso de CSPDarknet53, melhorias no processamento paralelo e aprimoramentos de pré-processamento.
YOLOv5	2020	Implementação otimizada em *PyTorch*, tornando-o mais acessível e eficiente.
YOLOv6	2022	Melhorias em eficiência, especialmente para aplicações industriais.
YOLOv7	2022	Melhor desempenho em tempo real, introdução de métodos otimizados para inferência.
YOLOv8	2023	Uso de técnicas sem âncoras (*anchor-free*), melhor precisão e flexibilidade para classificação, deteção e segmentação.

2.6.1 Funcionamento do YOLO

O algoritmo YOLO é um algoritmo baseado em regressão, que prevê as probabilidades de classe do objeto e as caixas delimitadoras que especificam a sua localização em toda a imagem [38]. As caixas delimitadoras do objeto são descritas por “bx” e “by”, onde as coordenadas “x” e “y” representam o centro da caixa em relação aos limites da célula da matriz. Os parâmetros “bw” e “bh” representam a largura e a altura da caixa, em relação ao tamanho total da imagem. O valor “c” representa a classe do objeto [36].

O YOLO recebe a imagem como entrada e divide-a numa matriz de S x S células (por exemplo, 3 x 3). Em seguida, são aplicadas técnicas de classificação de imagem e localização de objetos a cada célula da matriz, sendo atribuída uma etiqueta a cada uma delas. O algoritmo YOLO verifica cada célula da matriz para determinar se contém um objeto e, caso positivo, identifica a sua etiqueta e as respetivas caixas delimitadoras. Caso uma célula da matriz não contenha um objeto, a sua etiqueta é definida como zero [36].

No caso das células etiquetadas da matriz cada uma é representada por um conjunto de valores: A probabilidade da presença de um objeto na célula (pc), as coordenadas do centro da caixa delimitadora, relativas à célula da matriz (bx, by), a largura e altura da caixa delimitadora, normalizadas em relação ao tamanho total da imagem (bw, bh), e finalmente as probabilidades condicionais das classes do objeto (c1, c2 e c3,…,cn - exemplo: veículo, pessoa, bicicleta, etc.).

Assim, a predição de cada célula é expressa matematicamente como:

$$\hat{\mathbf{y}} = (\mathbf{pc}, \mathbf{bx}, \mathbf{by}, \mathbf{bw}, \mathbf{bh}, \mathbf{c1}, \mathbf{c2}, \ldots \mathbf{cn}) \quad (1)$$

Sendo que a probabilidade da classe do objeto na célula é dada por:

$$\mathbf{P}(\mathbf{classe}|\mathbf{objeto}) = \frac{e^{\mathbf{c_i}}}{\sum_{\mathbf{j=1}}^{\mathbf{n}} e^{\mathbf{c_j}}} \quad (2)$$

onde:

c_i - Representa o *logit* da classe prevista para a célula;

c_j - É o *logit* da classe j, ou seja, o valor numérico que a rede neuronal atribuiu a essa classe antes da normalização;

n - É o número total de classes.

O parâmetro “pc” indica se a célula contém um objeto ou não. Se houver um objeto, o valor de “pc” é 1; caso contrário, é 0. Os parâmetros “bx”, “by”, “bw” e “bh” definem a caixa delimitadora de uma célula e só são atribuídos se houver um objeto presente nessa célula. Os valores “c1”, “c2” e “c3” representam as classes do objeto. Por exemplo, se o objeto for um veículo (classe c2), os valores de “c1”, “c2” e “c3” serão 0, 1 e 0, respetivamente [36].

A matriz é definida como S x S x 8, onde S x S representa o tamanho total da matriz em que a imagem é dividida, e o valor 8 indica o número total de parâmetros: pc, bx, by, bw, bh, c1, c2 e c3. As caixas delimitadoras variam para cada célula da matriz, dependendo da posição dos objetos na respetiva célula.

No exemplo que se segue, a imagem foi dividida em 3x3 células, e foram consideradas 3 classes (pessoa, veículo, bicicleta). Como se pode ver na figura 17, não é possível identificar um objeto válido na primeira célula.

Figura 17- Exemplo de imagem com matriz 3x3 [36].

Por isso, o valor de “pc” é 0 (zero) e os parâmetros da caixa delimitadora não precisam de ser atribuídos, uma vez que não existe um objeto definido. Além disso, a probabilidade de classe não pode ser identificada, pois não há um objeto válido presente (tabela 4).

Tabela 4- Caixa delimitadora e valores de classe para a célula 1 [36].

	y=	pc	bx	by	bh	bw	c1	c2	c3
		0	?	?	?	?	?	?	?

Por outro lado, a 6ª célula contém um objeto válido, pelo que o valor de “pc” é definido como 1, e as caixas delimitadoras do objeto são representadas por “bx”, “by”, “bw” e “bh”. Como o objeto identificado é um veículo, as classes atribuídas à célula são 0, 1 e 0 (tabela 5) [36].

Tabela 5- Caixa delimitadora e valores de classe para a célula 6 [36].

	y=	pc	bx	by	bh	bw	c1	c2	c3
		1	bx	by	bh	bw	0	1	0

Se um mesmo objeto ocupar mais do que uma célula da matriz, a célula que contém o centro do objeto é utilizada para a sua deteção. Para uma identificação precisa do objeto, podem ser aplicados sucessivamente dois métodos*: Intersection over Union* (IOU) e *Non-Max Suppression* (NMS) [36].

A métrica Interseção sobre União (IoU) é amplamente utilizada na avaliação do desempenho de modelos de deteção de objetos. O seu principal objetivo é quantificar a sobreposição entre a caixa delimitadora prevista e a caixa delimitadora real (*ground truth*) [37].

Para compreender como a métrica IoU é calculada, considere-se o seguinte exemplo: Suponha-se que existem três modelos de deteção de objetos, Modelo A, Modelo B e Modelo C, treinados para identificar pássaros. Quando uma imagem contendo um pássaro é tratada por esses modelos, obtêm-se diferentes predições.

Como referência, já se conhece a verdade do terreno (*ground truth*), que está representada a vermelho na figura 18 [37].

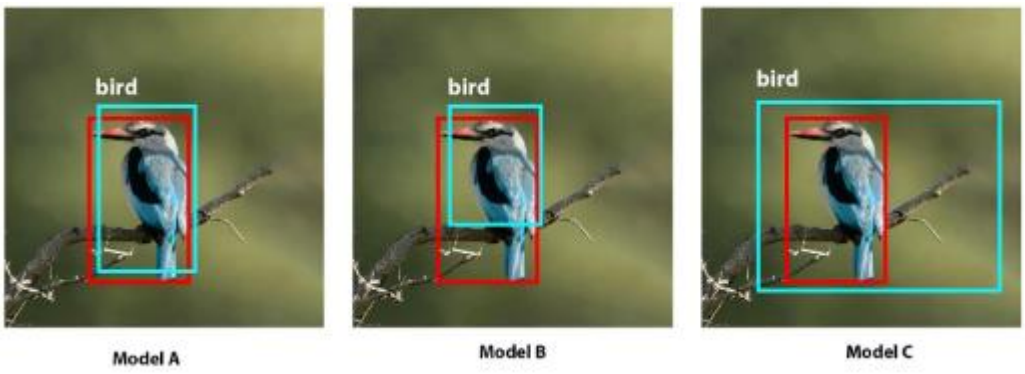


Figura 18- cálculo da métrica IoU [37].

Ao analisar as predições dos diferentes modelos, observa-se que a caixa delimitadora prevista pelo Modelo A apresenta uma sobreposição maior com a realidade (*Ground Truth*) em comparação com o Modelo B.

No entanto, o Modelo C tem uma sobreposição ainda maior com a verdade do terreno, mas também cobre uma grande parte do fundo da imagem. Isto levanta um problema: se a métrica de avaliação considerar apenas a sobreposição, poderá não ser totalmente justa, pois não leva em conta a precisão da localização [37].

Portanto, para avaliar corretamente um modelo de deteção de objetos, não basta apenas medir a sobreposição com a *Ground Truth*, mas é também necessário avaliar o quão bem a predição se ajusta à área real do objeto.

Deste modo, a métrica utilizada deve penalizar os seguintes cenários:

- Quando a predição não cobre toda a área do objeto real;
- Quando a predição ultrapassa os limites do objeto real, cobrindo partes do fundo da imagem;

Tendo isso em consideração, a métrica de Interseção sobre União (IoU) foi concebida para capturar esses aspetos. A métrica IoU é calculada através da seguinte equação [37]:

$$\mathbf{IoU} = \frac{\mathbf{Área\ de\ Interseção}}{\mathbf{Área\ da\ União}} \tag{3}$$

ou seja:

$$\mathbf{IoU} = \frac{|\boldsymbol{B_{pred}} \cap \boldsymbol{B_{real}}|}{|\boldsymbol{B_{pred}} \cup \boldsymbol{B_{real}}|} \quad (4)$$

onde:

Área da interseção - Corresponde à região comum entre a caixa delimitadora real e a caixa delimitadora prevista pelo modelo;

Área da união - Soma das áreas das duas caixas delimitadoras, subtraindo a área de interseção para evitar duplicações.

Se a predição não cobrir toda a área da verdade (*Ground Truth*), a área de interseção será menor, reduzindo assim o valor da IoU. Se a caixa prevista for demasiado grande, o denominador da fração aumenta, reduzindo também o valor do IoU.

Os valores da métrica IoU variam de 0 a 1, onde:

IoU = 0 - Significa que não há qualquer sobreposição entre a predição e a verdade do terreno;

IoU = 1 - Indica uma sobreposição perfeita, ou seja, a predição e a verdade do terreno coincidem exatamente.

Se IoU > 0,5, considera-se que a deteção foi bem-sucedida. Naturalmente é preferível que o valor de IoU calculado seja superior a um determinado limiar (um valor assumido para aumentar a precisão da deteção do objeto), e geralmente é assumido 0,5 [37].

Com a ajuda do limiar de IoU, pode-se determinar se uma previsão é um Verdadeiro Positivo (*True Positive* - TP), Falso Positivo (*False Positive* - FP) ou Falso Negativo (*False Negative* - FN). A figura 19 ilustra diferentes previsões utilizando um limiar de IoU de 0,5.

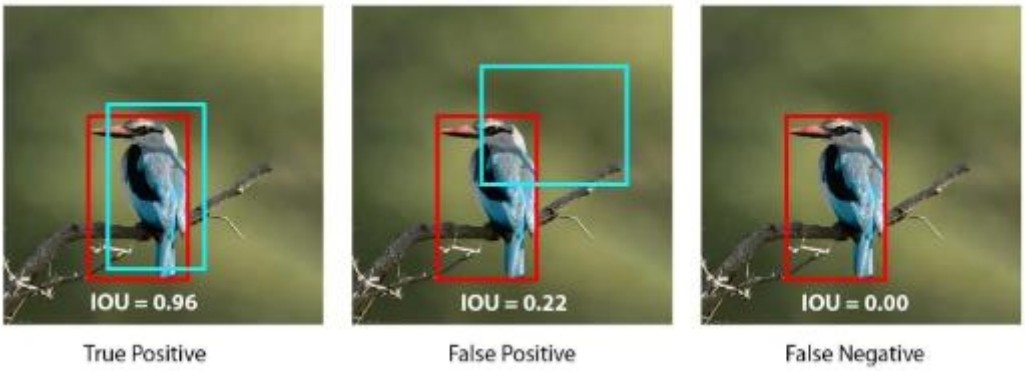


Figura 19- Análise qualitativa das previsões [37].

A decisão de classificar uma deteção como Verdadeiro Positivo ou Falso Positivo depende inteiramente dos requisitos definidos para a tarefa.

Na primeira previsão, o resultado é considerado Verdadeiro Positivo, pois o limiar de IoU está definido como 0.5. No entanto, se aumentarmos esse limiar para 0,97, a previsão passaria a ser um Falso Positivo [37].

Da mesma forma, a segunda previsão apresentada é classificada como Falso Positivo, devido ao limiar estabelecido. No entanto, poderia ser um Verdadeiro Positivo se o limiar fosse reduzido para 0,20 [37]. Teoricamente, até a terceira previsão poderia ser considerada um Verdadeiro Positivo, caso o limiar fosse reduzido para 0.

No contexto da deteção de objetos, a IoU é utilizada como uma métrica auxiliar. No entanto, na segmentação da imagem, a IoU é a métrica principal para avaliar a precisão do modelo.

Na segmentação de imagem, as áreas de interesse não são necessariamente retangulares, podendo apresentar formas regulares ou irregulares. Isto significa que as previsões do modelo são máscaras de segmentação, e não apenas caixas delimitadoras (*bounding boxes*), assim, a avaliação é feita pixel a pixel [37].

Além disso, a definição de Verdadeiro Positivo (TP), Falso Positivo (FP) e Falso Negativo (FN) diferem ligeiramente, pois não depende de um limiar predefinido.

Verdadeiro Positivo (TP): representa a área de interseção entre a *Ground Truth* (GT) e a máscara de segmentação (S) prevista pelo modelo. Matematicamente, isso pode ser representado como a operação lógica E (AND) entre GT e S [37]:

$$\mathbf{TP} = \mathbf{GT} \times \mathbf{S} \tag{5}$$

Falso positivo (FP): Corresponde à área prevista pelo modelo que está fora da *Ground Truth*. Pode ser calculado como a operação lógica OU (OR) entre GT e S, subtraindo a parte correta da previsão (GT):

$$\mathbf{FP} = (\mathbf{GT} + \mathbf{S}) - \mathbf{GT} \tag{6}$$

Falso negativo (FN): Representa o número de pixéis dentro da *Ground Truth* que o modelo não conseguiu prever. Pode ser obtido através da operação lógica OU (OR) entre GT e S, subtraindo a previsão do modelo (S):

$$\mathbf{FN} = (\mathbf{GT} + \mathbf{S}) - \mathbf{S} \tag{7}$$

Sabe-se que no contexto da deteção de objetos, a IoU é definida como a razão entre a área de interseção e a área total combinada da previsão e da *Ground Truth*.

No contexto da segmentação de imagem, como os valores de TP, FP e FN representam áreas ou quantidades de pixéis, a métrica da IoU pode ser expressa como:

$$\mathbf{IoU} = \frac{\mathbf{TP}}{\mathbf{TP+FP+FN}} \tag{8}$$

Este cálculo fornece uma medida quantitativa de sobreposição entre a previsão do modelo e a verdade do terreno, sendo amplamente utilizado para avaliar modelos de segmentação de imagem.

A figura 20 mostra o processo de avaliação do desempenho de modelos de segmentação de imagem, comparando a segmentação predita pelo modelo com a segmentação de referência (*ground truth*).

Figura 20- Segmentação de imagem pela IoU [37].

Neste exemplo, utiliza-se a imagem de um pássaro para demonstrar visualmente como os erros e acertos são identificados:

- Imagem da esquerda: indica a máscara de referência (*Ground Truth Mask*), no qual representa a segmentação real do objeto na imagem, ou seja, as regiões do pássaro identificadas manualmente ou por um especialista. Esta máscara serve de base para a avaliação da predição do modelo;
- Imagem central: indica a máscara Predita (*Predicted Mask*), a qual corresponde à segmentação realizada automaticamente pelo modelo. O objetivo é verificar o grau de correspondência entre esta máscara e a máscara de referência;
- Imagem da direita: indica a comparação entre Máscaras:
- TP (verdadeiros positivos): áreas onde a predição coincide corretamente com a máscara de referência;
- FP (falsos positivos): regiões onde o modelo detetou objeto, mas que não pertencem à máscara real;
- FN (falsos negativos): partes do objeto real que o modelo não conseguiu identificar.

O segundo método para a identificação precisa de objectos é a Supressão de Não-Máximos (*Non-Max Suppression - NMS*) que é usada para remover caixas sobrepostas e manter apenas a mais relevante. O algoritmo NMS funciona da seguinte forma [38]:

- Ordena todas as caixas preditas pela confiança, do maior para o menor;
- Escolhe a caixa com a maior confiança e adiciona-a à lista final de deteções;
- Calcula a IoU entre essa caixa e todas as outras;
- Remove todas as caixas cuja IoU seja maior que um limiar (exemplo: 0,5), pois são redundantes;
- Repete até que todas as caixas sejam processadas.

Além disso, cada célula da matriz prevê 'c' probabilidades condicionais de classe para o objeto contido nessa célula. Essas probabilidades são condicionadas à presença de um objeto na célula da matriz. É importante notar que apenas um conjunto de probabilidades de classe é previsto por célula da matriz,

independentemente do número de caixas delimitadoras associadas a essa célula [38].

A Figura 21 ilustra o processo completo de deteção de objetos pelo algoritmo YOLO, destacando o papel da NMS na fase final do *pipeline*. Inicialmente, a imagem de entrada é dividida numa grelha (por exemplo, 5×5), onde cada célula é responsável por prever as caixas delimitadoras (*bounding boxes*), o grau de confiança e as probabilidades de classe. Estas previsões são feitas para múltiplos objetos possíveis em cada célula.

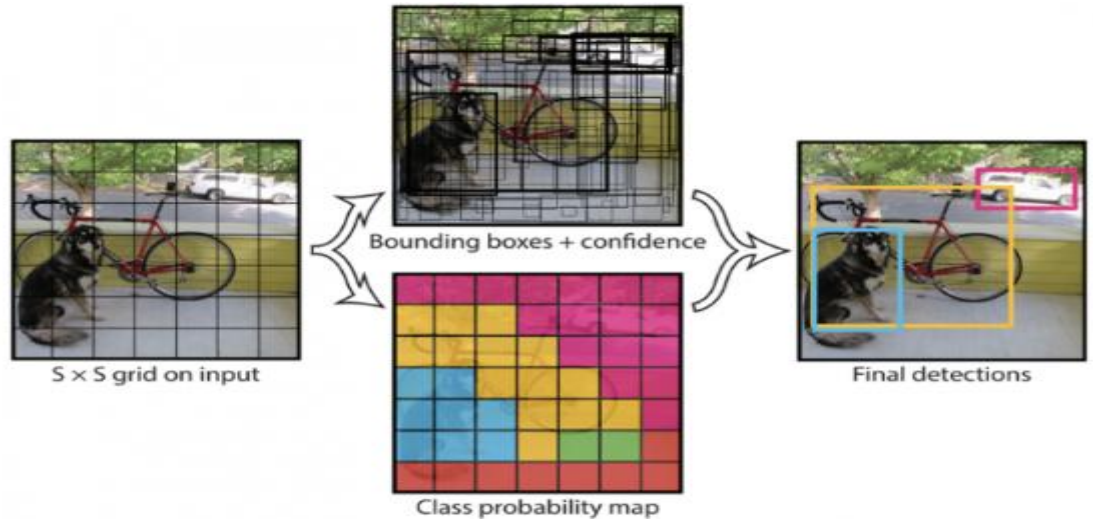


Figura 21- Processo completo de deteção de objeto pelo YOLO [39].

As caixas geradas contêm sobreposições significativas, pelo que é necessário aplicar a NMS para selecionar apenas aquelas com maior grau de confiança. Este processo consiste em manter a caixa com maior *confidence score* e descartar as caixas que têm uma elevada interseção com esta (medida pela métrica *Intersection over Union*, IoU), assumindo que se referem ao mesmo objeto.

Como representado na figura 21, a NMS é aplicada após a geração das *bounding boxes* e das probabilidades de classe, resultando numa deteção final mais limpa e precisa, onde cada objeto é identificado por uma única caixa delimitadora com o respetivo rótulo.

Este mecanismo é essencial para evitar redundâncias e assegurar a fiabilidade dos resultados, contribuindo para o bom desempenho do sistema de deteção de objetos em aplicações em tempo real, como é o caso da deteção de veículos em parques de estacionamento.

2.6.2 Deteção unificada do algoritmo YOLO

O YOLO é apresentado como um algoritmo unificado, onde componentes separados são fundidos numa única rede neuronal, formando o *pipeline* final. Para que cada caixa delimitadora (*bounding box*) seja prevista em paralelo, as características da imagem inteira são analisadas globalmente. O YOLO foi projetado para realizar o seu próprio treino *end-to-end* em tempo real, mantendo uma elevada precisão média [40].

Para alcançar uma deteção unificada, o YOLO divide a imagem de entrada numa matriz de S×S células. Se o centro do objeto estiver dentro de uma célula, essa célula assume a responsabilidade pela deteção do objeto. Assim, cada célula da matriz tenta prever uma caixa delimitadora e as suas pontuações de confiança para todas as classes treinadas.

As pontuações de confiança refletem a certeza do modelo em atribuir um rótulo e uma caixa delimitadora a cada objeto. Formalmente, as pontuações de confiança são definidas como [40]:

$$\mathbf{Confiança} = \mathbf{P}(\mathbf{Objeto}) \times \mathbf{IoU}_{\mathbf{truth,pred}} \tag{9}$$

onde:

$\mathrm{P}(\mathrm{Objeto})$ - Probabilidade da existência de um objeto na célula;

$\mathrm{IoU}_{\mathrm{truth,pred}}$ - *Intersection over Union* (IoU) entre a caixa delimitadora real (*ground truth*) e a caixa predita.

Se um objeto for encontrado dentro da célula, o *score* de confiança será igual ao IoU entre a caixa real e a predita. Caso contrário, a pontuação será zero. A deteção unificada do YOLO gera como saída, para cada objeto, um vetor contendo esses parâmetros:

$$(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}, \mathrm{h}, \mathbf{Confiança}) \tag{10}$$

onde:

(x,y) - Coordenadas do centro da caixa delimitadora em relação aos limites da célula da matriz;

(w,h) - Largura e altura do objeto, normalizadas em relação ao tamanho total da imagem;

Confiança - Probabilidade da presença do objeto multiplicada pelo IoU.

Como mencionado anteriormente, se o centro do objeto não estiver dentro da célula da matriz, essa célula não é responsável pela predição. Todas as coordenadas são normalizadas no intervalo [0,1], e a altura/largura do objeto são estimadas com base na imagem completa. Durante o teste, multiplicam-se as probabilidades condicionais da classe pelas pontuações de confiança da caixa [40]:

$$\mathbf{P}(\boldsymbol{Classe_i}|\boldsymbol{Objeto}) \times \mathbf{P}(\boldsymbol{Objeto}) \times \boldsymbol{IoU_{truth,pred}} = \mathbf{P}(\boldsymbol{Classe_i}) \times \boldsymbol{IoU_{truth,pred}} \quad (11)$$

onde:

$\mathrm{P}(Classe_i|Objeto)$ **-** Probabilidade condicional da classe i, dado que existe um objeto na célula;

$\mathrm{P}(Classe_i)$ - A probabilidade absoluta da classe i estar presente na caixa;

$IoU_{truth,pred}$ **-** O IoU entre a caixa predita e a caixa real (*ground truth*).

Multiplicando essas três quantidades, obtemos a pontuação de confiança específico para a classe i, associado àquela caixa delimitadora. A figura 22 é apresentada a aplicação do algoritmo YOLO para a deteção de objetos, especificamente a identificação de um pássaro em voo.

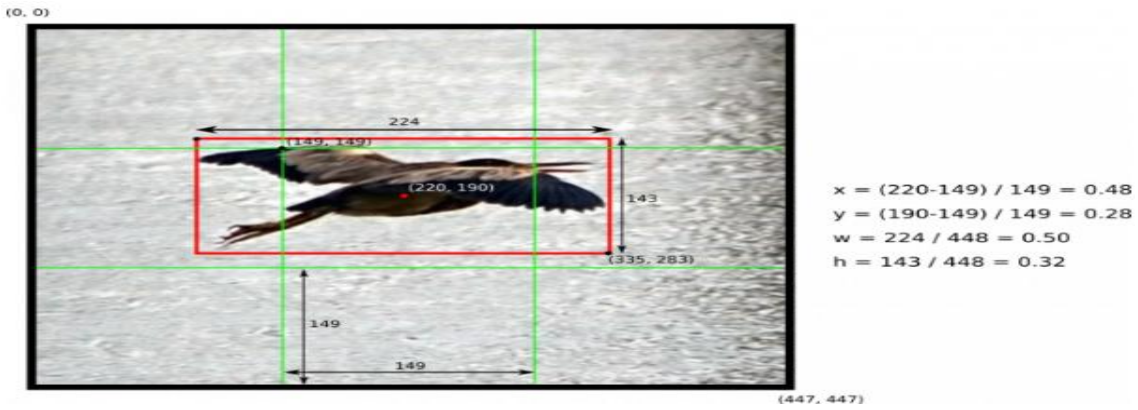


Figura 22- Parâmetros de coordenadas [41].

Começando por explicar a estrutura da imagem, note-se que neste exemplo a imagem de entrada contem um pássaro, identificado por uma caixa delimitadora (*bounding box*) vermelha, e uma matriz verde, que resulta da imagem ser dividida

numa matriz SxS, que ajuda a localizar o objeto. As coordenadas têm origem no canto superior esquerdo, sendo as dimensões da imagem 448x448 pixels.

A *bounding box* está centrada em (220,190), com uma largura de (224,143). O YOLO normaliza os valores no intervalo 0 a 1, facilitando a generalização do modelo. Começando pelo cálculo das coordenadas normalizadas do centro da *bounding box*:

$$\mathbf{x} = \frac{(200-149)}{149} = \mathbf{0,48} \tag{12}$$

$$\mathbf{Y} = \frac{(190-149)}{149} = \mathbf{0,28} \tag{13}$$

Isto significa que a posição do centro da *bounding box* está 48% ao longo do eixo x da célula da grelha e 28% ao longo do eixo y. Em seguida o YOLO realiza a normalização das dimensões da caixa:

$$\mathbf{w} = \frac{224}{448} = \mathbf{0,50} \tag{14}$$

Para w = 0,50 a *bounding box* ocupa 50% da largura da imagem.

$$\mathbf{h} = \frac{143}{448} = \mathbf{0,32} \tag{15}$$

O que significa que a *bounding box* ocupa 32% da altura da imagem.

2.6.3 Rede Neuronal YOLOv1

A rede neuronal YOLOv1 consiste numa rede com 24 camadas convolucionais (CNN), que atuam como extratoras de características, seguidas de 2 camadas totalmente conectadas, responsáveis pela classificação de objetos e pela regressão das caixas delimitadoras. A saída final é um tensor de 7x7x30 (figura 23) [42].

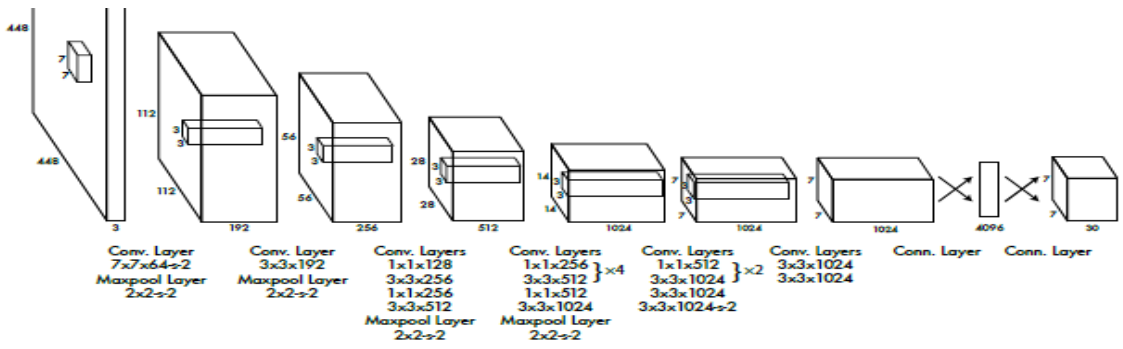


Figura 23- Rede Neuronal YOLOv1 [42].

Cada camada na rede desempenha um papel específico. A imagem de entrada tem um tamanho de 448×448×3, ou seja, tem dimensões espaciais de 448x448, com 3 canais de cor (RGB).

A primeira operação aplicada à imagem é uma camada convolucional, com um filtro de 7×7, com 64 *kernels*, que extrai características básicas (bordas, texturas). Essa camada aplica um *Stride* = 2, que reduz a dimensão da imagem de 448×448 para 224×224. Segue-se uma camada de *Max Pooling* 2×2 com *stride* = 2, que reduz ainda mais a dimensão para 112×112.

A segunda operação aplicada é também uma camada convolucional, com um filtro de 3×3 com 192 *kernels*, que aprende características mais avançadas. Esta camada inclui uma técnica de *Max Pooling* 2×2 com *stride* = 2, que reduz a dimensão para 56×56.

De seguida o YOLOv1 usa camadas convolucionais intermédias, para a rede começar a extrair padrões mais complexos. Primeiro aplica-se uma camada 1×1 com 128 e 256 filtros, que reduz a dimensionalidade e melhora a eficiência. Depois seguem-se camadas 3×3 com 256 e 512 filtros, em que o algoritmo aprende representações mais profundas. Finalmente aplica o *Max Pooling* 2×2 com *stride* = 2, que reduz a dimensão da imagem para 28×28.

Refira-se que se usam filtros 1 x 1 para melhorar a eficiência computacional e permite aprender combinações não lineares das características. As camadas convolucionais profundas são responsáveis por reconhecer partes completas de objetos. Para isso utilizam camadas 1×1 com 512 e 1024 filtros e camadas 3×3 com 1024 filtros, com o *Max Pooling* 2×2, que reduz a dimensão para 14×14.

Finalmente as camadas totalmente conectadas transformam as características extraídas em predições finais. Este algoritmo utiliza 4096 neurónios, sendo que a camada final com 30 neurónios corresponde à saída do YOLO. Este valor de 30 canais à saída resulta do YOLO dividir a imagem em 7×7 matrizes, e cada célula prevê B *bounding boxes* (normalmente 2). Cada *bounding box* tem 5 parâmetros (x, y, w, h, confiança) e além disso, há C classes (normalmente 20 no Pascal VOC).

Assim, o número total de saídas (também chamado de tensor de saída) é:

$$S \times S \times (B \times 5 + C) = 7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30$$

Em resumo, o YOLOv1 consiste em três pilares principais: O *backbone* (espinha dorsal) para extração de características, o *neck* (pescoço) focado na agregação de características, e a *head* (cabeça) para gerar as deteções. A figura 24 ilustra esta ideia [42].

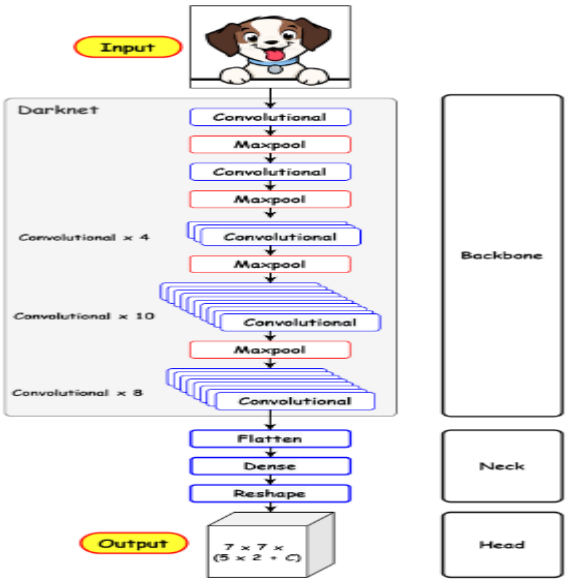


Figura 24- Divisão da Rede neuronal YOLOv1 [42].

2.6.3.1 Espinha dorsal (*backbone*)

A espinha dorsal do YOLO é uma rede neuronal convolucional que processa os pixéis da imagem para extrair características em várias granularidades. Geralmente, a espinha dorsal é pré-treinada com um conjunto de dados de classificação.

Podem-se dividir as características extraídas em níveis de abstração, que vão desde características simples (baixa abstração) até padrões mais complexos (alta abstração).

No caso das características de baixo nível, estas são extraídas nas primeiras camadas convolucionais e representam padrões básicos da imagem, por exemplo, bordas (a rede identifica contornos e diferenças entre áreas claras e escuras), texturas (permite distinguir superfícies rugosas, lisas, padronizadas, etc), gradientes de cor (identifica transições entre cores diferentes) e formas (linhas, círculos, cantos e padrões geométricos). Como exemplo pode-se indicar a deteção da borda de um veículo numa imagem.

No caso das características de médio nível, estas surgem nas camadas intermédias e começam a representar partes específicas dos objetos (a rede pode identificar

olhos, rodas, portas, asas, etc), texturas mais complexas (pele, metal, madeira, etc) e padrões estruturais (elementos que ajudam a diferenciar objetos semelhantes). Como exemplo, pode-se indicar a deteção uma roda como parte de um veículo.

Finalmente, as características de alto nível são extraídas nas camadas mais profundas da rede e representam objetos completos (o *backbone* aprende a associar partes e formar objetos completos), o contexto da imagem (permite perceber a relação entre diferentes objetos na imagem), e as diferenças entre categorias (a rede consegue distinguir entre "veículo" e "motociclo"). Como exemplo, pode-se indicar a capacidade da rede saber que uma roda, portas e vidro fazem parte de um veículo e não de um avião.

2.6.3.2 Pescoço (*neck*)

O pescoço do YOLO (neste caso, a FPN (*Feature Pyramid Network)* selecionada) combina e integra as representações das camadas da CNN antes de as enviar para a cabeça de predição.

A parte "*neck*" do YOLOv1 é, basicamente, a sequência de camadas que conecta o *backbone* às camadas finais responsáveis pelas previsões. Nesta versão do YOLO essa transformação acontece através de uma combinação de camadas *fully connected* (totalmente conectadas).

2.6.3.3 Cabeça (*head*)

A cabeça do YOLO é a parte da rede que faz as previsões das caixas delimitadoras e das classes. Esta é orientada por três funções de perda do YOLO: *class, box e objectness* (confiança de que um objeto está presente).

A cabeça, também conhecida como o detetor de objetos, identifica áreas onde pode haver um objeto, mas não informa qual é o objeto presente nessa área.

Resumindo, a arquitetura YOLOv1 não é complexa, apenas consiste num *backbone* convolucional com duas camadas totalmente conectadas, muito semelhante a uma arquitetura de rede para classificação de imagens. A parte inovadora do YOLO (que o torna um detetor de objetos) reside na interpretação das saídas (*head*) dessas camadas totalmente conectadas [42].

2.6.4 Treino

O YOLOv1 é treinado de forma *end-to-end*, o que é uma grande vantagem. O treino *end-to-end*, especialmente numa arquitetura tão simples, reduz significativamente a probabilidade de erros durante o processo de aprendizagem [42].

Os autores do YOLO recomendam começar por pré-treinar a rede no *ImageNet* (base de dados de imagens). Para isso, utilizam as 20 primeiras camadas convolucionais, seguidas por uma camada de *pooling* média e uma camada totalmente ligada (*fully connected*) com 1000 saídas, correspondentes às 1000 classes do ImageNet [42]. Este é um processo padrão em classificação de imagens e, segundo os autores, conseguem obter uma precisão próxima do estado da arte (para a época) no *ImageNet*. No entanto, este pré-treino é apenas uma etapa preparatória para a tarefa principal.

Após a fase de pré-treino, os autores adaptam a rede para a tarefa de deteção de objetos. Para isso, removem a camada de *pooling* média e a camada totalmente ligada. Em seguida, adicionam mais algumas camadas convolucionais (camadas de convolução 3×3, 1024 filtros) e reconfiguram as camadas totalmente ligadas para se adequarem à tarefa de deteção de objetos [42].

2.6.5 Função de perdas

A parte mais importante para a deteção de objetos é a função de perdas, que é o método utilizado para treinar as saídas da rede de acordo com as necessidades específicas, sendo responsável por treinar a parte da rede que realiza a deteção de objetos [43].

A tarefa de deteção é abordada como um problema de regressão, focado na previsão da área-alvo e da categoria dos objetos. O YOLO prevê múltiplas caixas delimitadoras por célula de grade (matriz). Para calcular a perda para o verdadeiro positivo, que apenas uma dessas caixas seja responsável pelo objeto [43].

Para isso, seleciona a que possui a maior IoU (Interseção sobre união) com a *ground truth*. Essa estratégia leva à especialização entre as predições das caixas, onde cada uma melhora a previsão de certos tamanhos e proporções [43].

O YOLO utiliza o erro quadrático entre as previsões e a *ground truth* para calcular a perda. A função de perda é composta por perdas de classificação, localização e de confiança [43].

Começando pelas perdas de classificação, se um objeto é detetado, a perda de classificação em cada célula é o erro quadrático das probabilidades condicionais de classe para cada categoria [23]:

$$\boldsymbol{L_C = \sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2} \quad (16)$$

em que:

$\boldsymbol{1_i^{obj} = 1}$ - Se um objeto estiver presente na célula i, caso contrário $\boldsymbol{1_i^{obj} = 0}$;

$\boldsymbol{p_i(c)}$ - Representa a probabilidade real da classe ***C*;**

$\boldsymbol{\hat{p}_i(c)}$ - Probabilidade predita pela rede para a classe ***C*** na célula **i**;

$\boldsymbol{(p_i(c) - \hat{p}_i(c))^2}$ - Erro quadrático entre a predição e o valor real.

A perda de localização mede os erros nas localizações e tamanhos das caixas delimitadoras previstas. Considera-se apenas a caixa responsável pela deteção do objeto [23], e calcula-se através de:

$$\boldsymbol{L_L = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]}$$

$$\boldsymbol{+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]} \quad (17)$$

em que:

$\boldsymbol{1_i^{obj} = 1}$ se a caixa delimitadora na célula i for responsável pela deteção do objeto, caso contrário $\boldsymbol{1_i^{obj} = 0}$;

(x_i , y_i) - Coordenadas da *bounding box* predita;

$(\hat{x}_i , \hat{y}_i)$ - Coordenadas reais (*ground truth*);

(w_i , h_i) - Largura e altura preditas;

$(\hat{w}_i , \hat{h}_i)$ - Largura e altura reais;

$(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$ - Erro na posição do centro da *bounding box;*

$\left(\sqrt{w_i} - \sqrt{\widehat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$ - Erro na largura e altura, com raiz quadrada para suavizar grandes variações;

$\boldsymbol{\lambda_{coord}} = \mathbf{5}$ - Peso para o erro de coordenadas da caixa delimitadora. Este fator aumenta o peso da perda nas coordenadas da caixa delimitadora.

Note-se que não se pretende pesar igualmente os erros absolutos em caixas grandes e pequenas. Ou seja, um erro de 2 pixéis numa caixa grande é considerado igual a 1 (um) numa caixa pequena, além disso, para dar mais ênfase à precisão da caixa delimitadora, multiplica-se a perda por um fator igual a 5.

Finalmente se um objeto é detetado na caixa, a perda de confiança (que mede a certeza da presença do objeto) é calculada por [23]:

$$\boldsymbol{L_{conf1}} = \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbf{1}_{\boldsymbol{ij}}^{\boldsymbol{obj}} \left(\boldsymbol{C_i} - \widehat{\boldsymbol{C}}_{\boldsymbol{i}}\right)^{\mathbf{2}} \quad (18)$$

em que:

$\boldsymbol{C_i}$ - IoU (caixa predita, caixa real) – confiança real;

$\widehat{\boldsymbol{C}}_{\boldsymbol{i}}$ - Confiança predita pela rede;

$\left(\boldsymbol{C_i} - \widehat{\boldsymbol{C}}_{\boldsymbol{i}}\right)^{\mathbf{2}}$ - Erro quadrático na predição da confiança.

Se um objeto não é detetado na caixa, a perda de confiança é dada por:

$$\boldsymbol{L_{conf2}} = \boldsymbol{\lambda_{noobj}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbf{1}_{\boldsymbol{ij}}^{\boldsymbol{nobj}} \left(\boldsymbol{C_i} - \widehat{\boldsymbol{C}}_{\boldsymbol{i}}\right)^{\mathbf{2}} \quad (19)$$

onde:

$\mathbf{1}_{\boldsymbol{ij}}^{\boldsymbol{nobj}}$ - Indica que *a bounding box* j da célula i não contém um objeto (1 se não tiver um objeto e 0 se tiver um objeto);

$\boldsymbol{\lambda_{noobj}} = \mathbf{0,5}$ - Peso para o erro de células sem objetos, que reduz o impacto da penalização, pois prever corretamente objetos é mais importante do que prever corretamente o “nada”.

Isto é importante porque a maioria das caixas não contém objetos, o que causa um problema de desequilíbrio de classes, ou seja, treina-se o modelo para detetar o

fundo mais frequentemente do que para detetar objetos. Para remediar isso, pondera-se essa perda com um fator menor (por padrão a 0,5) [43].

A função de perda final é a soma das perdas de localização, confiança e classificação, e é dada da seguinte forma [43]:

$$L = \mathbf{L_l} + \mathbf{L_{conf1}} + \mathbf{L_{conf2}} + \mathbf{L_c}$$

$$\boldsymbol{\lambda_{coord}} \quad \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1_{ij}^{obj}} [(\mathbf{x_i} - \hat{\mathbf{x}}_\mathbf{i})^2 + (\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i})^2]$$

$$+ \boldsymbol{\lambda_{coord}} \quad \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\widehat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\widehat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} \left(C_i - \widehat{C}_i \right)^2 \tag{20}$$

$$+ \boldsymbol{\lambda_{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{nobj} \left(C_i - \widehat{C}_i \right)^2$$

$$+ \sum_{i=0}^{s^2} \mathbf{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \widehat{p}_i(c))^2$$

Os primeiros dois termos da função de perdas representam a perda de localização. Os terceiros e quarto termos da função de perdas representam a perda de confiança: o terceiro termo mede o erro de confiança quando o objeto é detetado na caixa (1_{ij}^{obj}), e o quarto termo mede o erro de confiança quando o objeto não é detetado na caixa (1_{ij}^{nobj}). Como a maioria das caixas está vazia, essa perda é reduzida pelo termo (λ_{noobj}) [43].

O componente final da função de perdas é a perda de classificação, que mede o erro quadrático das probabilidades condicionais da classe para cada classe, apenas se o objeto aparecer na célula (1_{ij}^{obj}) [43].

2.7 Visão geral do YOLOv8

O YOLOv8 é uma das versões mais recentes do modelo de deteção de objetos YOLO. Esta nova versão introduz uma arquitetura redesenhada, como se pode ver na figura 25, diferenciando-se dos seus antecessores ao incluir melhorias significativas em eficiência e precisão. Entre as principais inovações, destacam-se um novo *backbone* otimizado, uma cabeça de deteção aprimorada e a adoção de um modelo *anchor-free*, eliminando a necessidade de caixas de ancoragem [44].

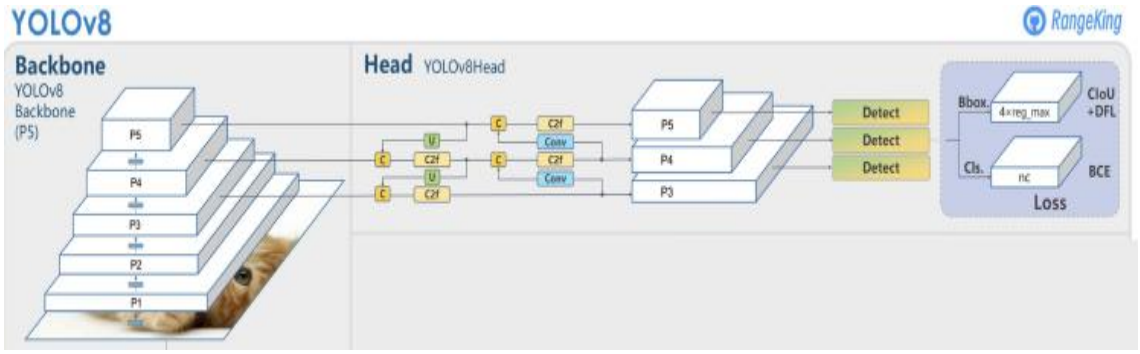


Figura 25- Arquitetura do YOLOv8 [45].

Além disso, o YOLOv8 mantém e aperfeiçoa o uso da *Feature Pyramid Network* (FPN) e da *Path Aggregation Network* (PAN), otimizando a passagem de informações entre diferentes escalas. A FPN reduz gradualmente a resolução espacial da imagem de entrada enquanto aumenta o número de canais de características, permitindo a criação de mapas de características que melhoram a deteção de objetos em diferentes escalas e resoluções [44].

Outra novidade é a introdução de uma nova ferramenta de rotulagem, que simplifica o processo de anotação e melhora a eficiência na preparação dos dados.

A arquitetura PAN, por outro lado, agrega recursos de diferentes níveis da rede por meio de conexões *skip* (ligações diretas entre camadas não consecutivas dentro da rede neuronal). Ao fazer isso, a rede pode capturar melhor recursos em múltiplas escalas e resoluções, o que é crucial para detetar com precisão objetos de diferentes tamanhos e formas [44].

O YOLOv8 segue a abordagem tradicional de redes de deteção de objetos, composta por *Backbone* (extrai características da imagem), *Neck* (refina e combina características de diferentes escalas) e *Head* (faz a deteção final de objetos). Cada bloco da figura 26 representa diferentes camadas e operações.

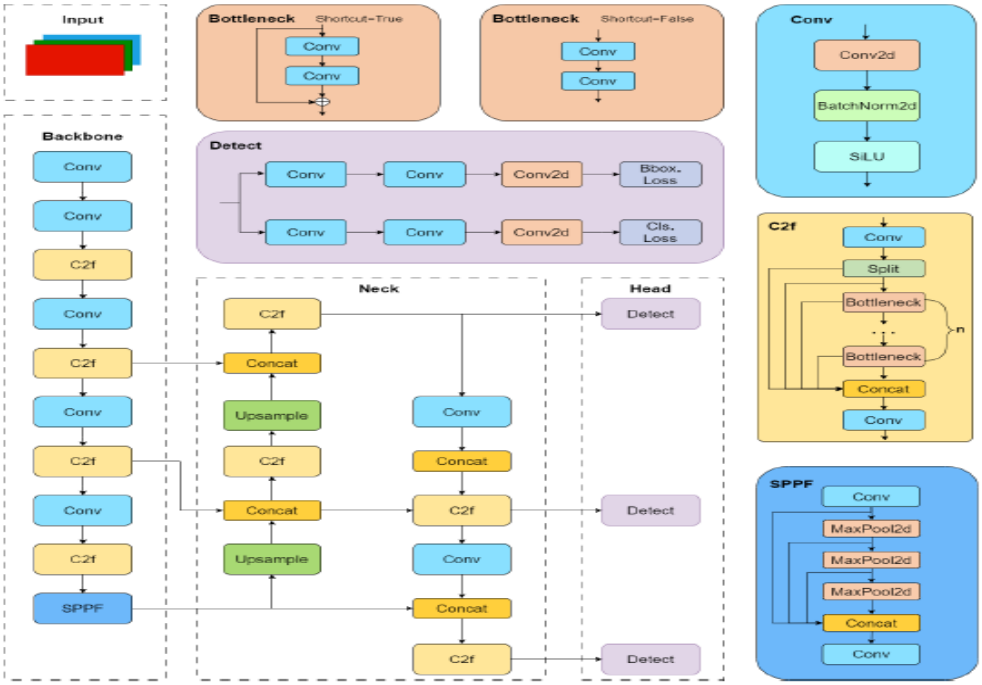


Figura 26- Estrutura detalhada do YOLOv8 [47].

2.7.1 *Backbone* (Extração de características)

O *backbone* é a parte inicial da rede, responsável por extrair as características da imagem de entrada.

O *backbone* é composto por **c**amadas convolucionais para capturar padrões (conv), por um bloco especial que melhora a extração de características (C2f) e por um último bloco designado SPPF (*Spatial Pyramid Pooling Fast*), que reduz o tamanho da imagem sem perder informações espaciais importantes.

O bloco C2f contém múltiplos *bottlenecks* (pequenas redes residuais) e usa a operação *split* para dividir as características e depois as concatenar. Essa estratégia melhora a eficiência computacional e preserva mais detalhes da imagem.

O bloco SPPF contém múltiplas operações de *MaxPool2d* que permitem capturar características de diferentes tamanhos na imagem. Este bloco reduz a dimensão sem comprometer a informação essencial.

2.7.2 *Neck* (Fusão de características multi - escala)

O *neck* pega nas características extraídas no *backbone* e melhora a robustez da rede ao combinar informações de diferentes escalas. Os três tipos de blocos utilizados são o *Upsample* (aumenta a resolução da imagem para melhorar a deteção de objetos pequenos), o *Concat* (junta características de diferentes

camadas para preservar detalhes) e novamente o bloco *C2f*, para refinar as características combinadas.

O uso do *upsample* seguido de concatenação permite que a rede aprenda melhor relações entre objetos de diferentes tamanhos.

2.7.3 *Head* (Saída e deteção de objetos)

O *head* gera as previsões finais de deteção, contendo: camadas convolucionais para refinar as características, com três saídas separadas: *BBox Loss* (calcula a precisão das caixas delimitadoras), *Cls Loss* (calcula a precisão da classificação dos objetos) e *Detect* (faz a predição final da localização e da classe dos objetos).

Essa divisão permite treinar a rede de forma eficiente, otimizando cada componente separadamente.

2.8 Função de perdas do YOLOv8

No YOLOv8, a função de perda (*loss function*) é composta por três componentes principais: perda das caixas delimitadoras, perda de classificação e perda de regressão baseada em distribuição.

Para o cálculo das perdas das caixas delimitadoras o YOLOv8 utiliza a Complete *IoU Loss (CIoU Loss*), que melhora a *IoU Loss* adicionando termos de distância e relação de aspecto [46]:

$$\boldsymbol{L_{box} = 1 - CIoU} \tag{21}$$

A CIoU é definida como:

$$\mathbf{CIoU = IoU - \frac{\rho^2(b,b^g)}{c^2} - \alpha v} \tag{22}$$

onde:

$\rho^2(b, b^g)$ - Distância euclidiana ao quadrado entre os centros da caixa predita *b* e da caixa real b^g;

c - Distância diagonal mínima que cobre ambas as caixas;

v - Termo que mede a diferença na relação de aspecto entre *b* e b^g;

α - Peso ajustável do termo v.

Para o cálculo da perda de classificação o YOLOv8 utiliza *Binary Cross-Entropy* (BCE) *Loss* para a classificação dos objetos [46]:

$$\boldsymbol{L_{cls}} = -\sum_i [\boldsymbol{y_i}\,\mathbf{log}(\boldsymbol{\hat{y}_i}) + (\mathbf{1} - \boldsymbol{y_i})\mathbf{log}\,(\mathbf{1} - \boldsymbol{\hat{y}_i})] \tag{23}$$

onde:

y_i - Rótulo real da classe (1 para classe correta, 0 para errada);

$\hat{y}_i$ - Probabilidade predita pelo modelo.

Finalmente para o cálculo da perda de regressão baseada em distribuição o YOLOv8 utiliza *Distribution Focal Loss* (DFL) para refinar a localização da caixa delimitadora. A DFL ajuda a suavizar as previsões, tornando as coordenadas da caixa mais precisas [46]:

$$\boldsymbol{L_{dfl}} = \sum_i \boldsymbol{p_i}\mathbf{log}\,(\boldsymbol{\hat{p}_i}) \tag{24}$$

onde:

p_i - Distribuição real das coordenadas da caixa;

$\hat{p}_i$ - Distribuição predita pelo modelo.

A função de perda final é a soma das perdas de localização, confiança e classificação, e é dada da seguinte forma [46]:

$$\boldsymbol{L} = \boldsymbol{\lambda_{box}}\boldsymbol{L_{box}} + \boldsymbol{\lambda_{cls}}\boldsymbol{L_{cls}} + \boldsymbol{\lambda_{dfl}}\boldsymbol{L_{dfl}} \tag{25}$$

onde:

L_{box} - Perda das caixas delimitadoras (*bounding box loss*);

L_{cls} - Perda de classificação (*classification loss*);

L_{dfl} - Perda de regressão baseada em distribuição (*distribution focal loss*);

$\lambda_{box}, \lambda_{cls}, \lambda_{dfl}$ - Pesos de cada termo na função de perda.

2.9 Região de interesse

A região de interesse (*Region of Interess* – ROI), é uma região considerada da área da imagem, ignorando a parte exterior da região, chamada de plano de fundo [48]. Conforme proposto em [49], para detetar a disponibilidade de lugares de estacionamento, a região detetada deve estar localizada na mesma região de interesse. Ou seja, quando o modelo YOLO deteta o objeto desejado, neste caso o veículo, e a área do detetor da máquina deve se sobrepor à região de interesse.

Ao utilizar o método condicional de região de sobreposição, o sistema consegue identificar se a região selecionada, neste caso a área do lugar de estacionamento, está sobreposta a outra região ativa, neste caso o próprio veículo. A figura 27 mostra como a região de interesse é implementada.

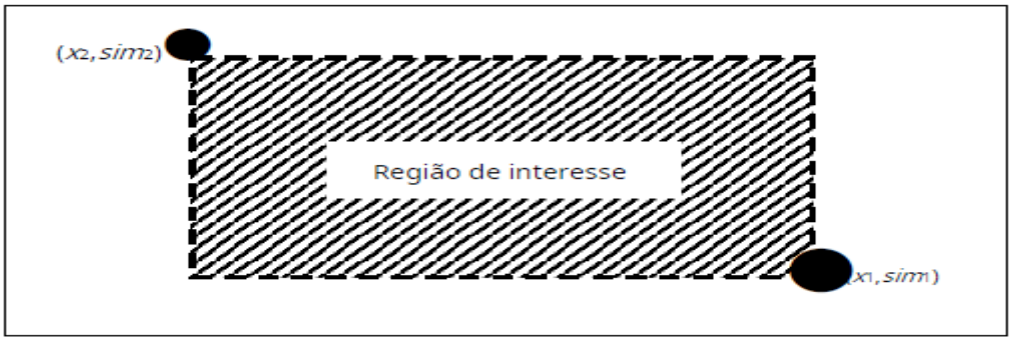


Figura 27- Região de interesse (ROI) [50].

No desenvolvimento desta abordagem para a implementação de hardware, o processo de deteção da disponibilidade de estacionamento torna-se mais simples em comparação com outros métodos, como o apresentado em [50], devido à menor necessidade de processamento, uso de sensores mais eficientes, redução de ruído na imagem, etc.

A figura 28 mostra os critérios de sobreposição das ROI que podem ser consideradas ROI sobrepostas. Estes critérios são importantes para garantir que o programa possa identificar a deteção de disponibilidade de estacionamento e possa identificar a disponibilidades de lugares.

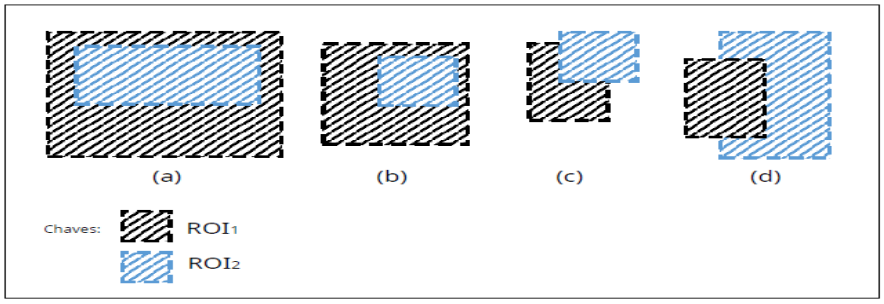


Figura 28- Regiões de interesse sobrepostas [51].

A figura 28 representa diferentes configurações de Regiões de Interesse (ROI) utilizadas para a deteção da disponibilidade de estacionamento, onde ROI_1 e ROI_2 representam duas regiões de interesse.

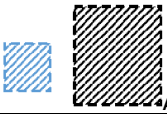
São mostradas quatro configurações distintas de sobreposição entre ROI_1 e ROI_2, rotuladas como (a), (b), (c) e (d):

a) ROI_2 está contida dentro de ROI_1 - A segunda região de interesse está completamente dentro da primeira;
b) ROI_2 é menor e também está dentro de ROI_1 - Uma variação da primeira configuração, mas com uma ROI_2 menor;
c) ROI_2 parcialmente sobrepõe ROI_1 - Aqui, apenas parte da segunda região está dentro da primeira;
d) ROI_2 sobrepõe-se parcialmente a ROI_1, mas estende-se além da ROI_1.

Tem-se assim diferentes formas de delimitação de áreas para a deteção de veículos, onde a sobreposição entre as regiões pode influenciar a precisão da análise do algoritmo na tomada de decisão.

A partir da tabela 6 é possível entender como é que o sistema detetará apenas 2 saídas para cada estacionamento:

Tabela 6- condições para determinar a disponibilidade dos lugares [51].

Condição para a sobreposição da Região de interesse (ROI)	**Saída do sistema**
 Ambas as regiões estão sobrepostas.	A saída do programa considera “lugar ocupada”.
 As regiões não se sobrepõem.	A saída do programa considera “lugar livre”.

2.10 Algoritmo do raio (*Ray Casting Algorythm*)

O algoritmo do raio conta quantas vezes um raio horizontal a partir do ponto (*cx, cy*) cruza as arestas do polígono. Se o número de cruzamentos for ímpar, o ponto está dentro; se for par, está fora [52].

Sabendo que a região de interesse estática (ROI_2) não é um quadrado perfeito devido a posição da câmera, serão utilizadas técnicas de coordenadas baseadas no algoritmo do raio, que funciona da seguinte forma [52]:

Considerando um ponto *P= (cx,cy)* e um polígono com *vértices* $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.

Inicializa-se a contagem de interseções a partir de zero (0). Para cada aresta do polígono, é verificado se o raio horizontal a partir de *P* cruza essa aresta. Este teste, para uma aresta que conecta (x_i, y_i) *a* (x_{i+1}, y_{i+1}) começa por verificar se o ponto P está entre as coordenadas Y da aresta (ou seja, $y_i \leq cy < y_{i+1}$ ou $y_{i+1} \leq cy < y_i$).

Por fim calcula-se a coordenada *X* do ponto de Interseção **I** do raio horizontal a partir de *P* com a aresta:

$$\mathbf{I_x} = \mathbf{x_i} + \frac{(\mathbf{cy} - \mathbf{y_i}) \times (\mathbf{x_{i+1}} - \mathbf{x_i})}{\mathbf{y_{i+1}} - \mathbf{y_i}} \tag{26}$$

Se $I_x \geq c_x$, incrementa a contagem de interseções.

A implementação do algoritmo do raio em *Python* é apresentada no Anexo A.

3. Projeto do sistema inteligente de deteção de lugares de estacionamento

Este capítulo apresenta a metodologia adotada no desenvolvimento do sistema inteligente de deteção de lugares de estacionamento. A abordagem inicia-se com a definição do problema e dos requisitos do sistema, seguida da apresentação da arquitetura geral do projeto e da análise de diferentes abordagens técnicas. Com base nessa análise, justifica-se a escolha da solução implementada, incluindo a utilização do modelo YOLOv8 para deteção de veículos.

3.1 Definição do problema e requisitos do sistema

A gestão eficiente de estacionamento é um desafio em áreas urbanas devido à dificuldade na deteção e monitorização da ocupação dos lugares. A solução proposta visa utilizar visão computacional e *Machine Learning* para detetar a ocupação dos lugares em tempo.

Os principais requisitos do sistema são:

- Monitorização em tempo real da ocupação dos lugares;
- Utilização de visão computacional em substituição de sensores físicos;
- Acesso remoto às informações via uma interface *web*;
- Baixo custo e fácil implementação em diferentes locais;
- Operação em diferentes condições de iluminação, incluindo períodos noturnos.

3.2 Arquitetura do sistema

A figura 29 apresenta a arquitetura geral do sistema, destacando os principais componentes e as suas interligações:

- Câmeras (PiCâmera V2 e PiCâmera NoIR): Responsáveis pela aquisição de imagens em tempo real;
- *Raspberry Pi* 4 e *Raspberry Pi* 5: Executam o pré-processamento das imagens e enviam os resultados para a base de dados;
- Modelo de *Machine Learning* (YOLOv8): Realiza a deteção de veículos e classificação da ocupação dos lugares;
- Rede de Comunicação (*Wi-Fi*): Conecta os *Raspberry Pi* ao servidor de processamento;

- Servidor e Armazenamento: Processa os dados e disponibiliza os resultados através da interface *web*;
- Interface *web*: Exibe o estado dos lugares de estacionamento em tempo real.

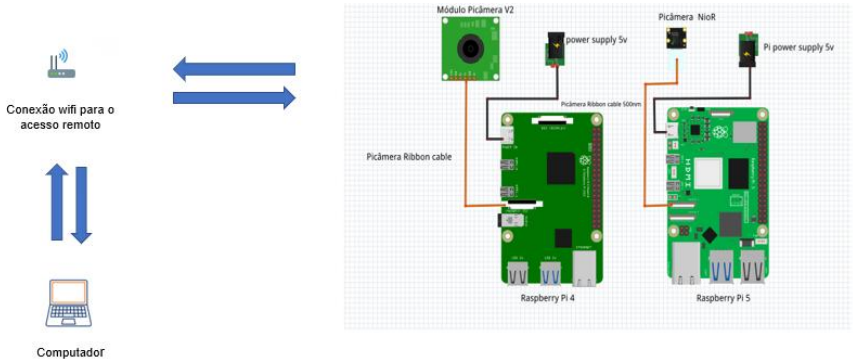


Figura 29- Visão geral do sistema.

O funcionamento do sistema segue as seguintes etapas:

- As câmeras capturam imagens da área de estacionamento;
- As imagens são processadas localmente nos *Raspberry Pi*, identificando veículos e determinando a ocupação dos lugares;
- Os dados processados são armazenados e apresentados numa interface *web* acessível remotamente.

3.3 Revisão de literatura e análise de alternativas

Para fundamentar a escolha desta solução técnica, foram analisadas diferentes abordagens utilizadas para a deteção de lugares de estacionamento. Começou-se pela análise dos métodos tradicionais, com sensores físicos (ultrassónicos, magnéticos, LIDAR), mas com custos elevados e desafios de manutenção. Passou-se então à visão computacional baseada em *Machine Learning*, que é uma alternativa flexível e escalável, dispensando sensores físicos.

Tendo-se optado por esta tecnologia mais avançada, foram analisados diferentes modelos de *Machine Learning* para a deteção de veículos, incluindo os algoritmos *RCNN* e *Faster RCNN*, que são modelos precisos, mas com elevado tempo de processamento. Depois estudou-se o *SSD (Single Shot MultiBox Detector)*, que oferece um melhor compromisso entre precisão e velocidade, mas no final optou-se pelo YOLO, que é um modelo otimizado para a deteção em tempo real, com elevada precisão e rapidez.

Refira-se que o YOLOv8 foi escolhido para o projeto devido às seguintes vantagens:

- Velocidade elevada, permitindo análise em tempo real;
- Alta precisão na deteção de veículos;
- Eficiência computacional, adequada para execução em dispositivos como o *Raspberry Pi*.

3.4 Implementação da solução técnica

A solução técnica adotada baseia-se em processamento local nos *Raspberry Pi* e utilização do modelo YOLOv8 para análise das imagens. Optou-se por utilizar duas câmeras diferentes, sendo que a câmera PiCâmera V2 capta imagens em condições normais de iluminação, e a câmera PiCâmera *NoIR* foi usada para melhorar a deteção em ambientes de baixa luminosidade.

O modelo YOLOv8 foi treinado com um *dataset* personalizado, contendo imagens de veículos em diferentes ângulos e condições de iluminação. O treino foi realizado num computador com *GPU*, utilizando a *framework Ultralytics* YOLOv8, e o modelo otimizado foi exportado para execução nos *Raspberry Pi*.

No processamento e exibição dos resultados as imagens captadas são analisadas pelo YOLOv8 para identificar os veículos. O estado dos lugares é atualizado e disponibilizado numa interface *web*.

3.5 Escolha do hardware e bibliotecas de configuração

Considerando a tendência da indústria, a utilização de câmeras para calcular os lugares de estacionamento foi a abordagem utilizada para o projeto.

Neste processo é necessário escolher a plataforma onde será desenvolvido o algoritmo principal para o reconhecimento dos veículos e mapeamento dos lugares. A escolha acabou por recair em dois *Raspberry* PI (4 e 5), apresentados nas figuras 30 e 31, cujas características estão indicados na tabela 7.

Figura 30- a) Raspberry Pi 5 com dissipador e ventoinha. b) Módulo da câmera NoIR.

Figura 31- a) Raspberry Pi 4. b) Módulo da câmera V2.

Tabela 7- Especificações de Hardware do Raspberry Pi 4 e 5 [53],[54].

Especificações	Raspberry Pi 4	Raspberry Pi 5
Processador	Broadcom BCM2711 Quad- core 64-bit @ 1.8 GHz	Broadcom BCM2712 Quad- core 64-bit @ 2.4 GHz
GPU	VideoCore VI @ 500 MHz OpenGL ES3.1, Vulkan 1.0	VideoCore VI @ 800 MHz OpenGL ES3.1, Vulkan 1.2
Display Output	2x Micro-HDMI 4K @ 60 Hz (Display único)	2x Micro-HDMI 4K @ 60 Hz (Display dúplo)
Memória	LPDDR4-3200 SDRAM, 4GB	LPDDR4X-4267 SDRAM 8GB
Conectividade	*Wi-Fi, Bluetooth, Gigabit Ethernert PoE enabled (with HAT)*	*Wi-Fi, Bluetooth, Gigabit Ethernert PoE enabled (with HAT)*
Portas USB	2x USB 2.0 2x USB 3.0	2x USB 2.0 2x USB 3.0 @ 5Gbps
Armazenamento	*Micro-SD card slot*	*Micro-SD card slot M.2 SSD suport with HAT*
Alimentação	5V/3A via USB-C (15W)	5V/3A via USB-C (27W)
Características Especiais	-	Botão liga/desliga, Relógio em tempo real (RTC) Porta de depuração UART
Preço	66 EUR	87 EUR

O sistema operativo escolhido, por recomendação dos criadores dos *Raspberry* Pi 4 e 5, foi o *Raspbian* versões *Bullseye* e *bookworm* [55]. Para a captação de imagem utilizaram-se dois tipos de câmera, a oficial do *Raspberry* Pi, presente na

figura 30 b) e a câmera de visão infravermelha *Raspberry* Pi *NoIR* presente na figura 31 b). Estas câmeras têm um sensor de 8 Megapixéis, mais do que o suficiente para a aplicação em causa, e conecta-se facilmente ao *Raspberry* Pi através de um conector próprio para cada modelo.

Em relação às bibliotecas, no módulo da câmera foi utilizada a biblioteca Picamera essencialmente para a captação e o armazenamento de imagens. Para o processamento de imagem propriamente dito foi utilizada a biblioteca *OpenCV* (*Open Source Computer Vision*) [56]. O *OpenCV* é uma biblioteca de código aberto utilizada para fins académicos e comerciais e possui atualmente mais de 2500 algoritmos relacionados com processamento de imagem e *Machine Learning* [56].

Para o projeto foi escolhida a linguagem de programação *Python* pela sua simplicidade e pelo seu vasto suporte e documentação no contexto do *Raspberry* Pi. A tabela 8 apresenta as principais funções do *OpenCV* utilizadas no projeto.

Tabela 8- Listas das principais funções do *OpenCV* utlizadas [56].

Nome	Descrição
cv2.VideoCapture	Esta função é usada para capturar vídeo da câmera ou de um arquivo de vídeo.
cv2.imshow	Função usada para exibir uma imagem ou um vídeo em uma janela.
cv2.flip	Função usada para inverter a posição de uma imagem. No código, é usada para inverter a imagem horizontalmente.
cv2.rectangle	Função usada para desenhar um retângulo em uma imagem. No código, é usada para delinear a caixa delimitadora dos objetos detetados.
cv2.circle	Função usada para desenhar um círculo numa imagem. No código, é usada para marcar o centro dos objetos detetados.
cv2.polylines	Função usada para desenhar polígonos (linhas conectadas) numa imagem. No código, é usada para marcar os lugares de estacionamento.
cv2.putText	Função usada para escrever texto numa imagem. No código, é usada para rotular as áreas e exibir contagens.
cv2.waitKey	Função usada para aguardar a entrada do teclado por um determinado período de tempo (ms).

A tabela 9 apresenta as principais funções do algoritmo *Ultralytics* utilizadas no projeto.

Tabela 9-Listas das principais funções do *Ultralytics* utlizadas no projeto [57].

Nome	Descrição
YOLO	Esta classe é usada para carregar e inicializar o modelo YOLO. No código é utilizado o model=YOLO ('best.pt')
predict	Método usado para fazer as previsões de deteção de objetos numa imagem.
Results[0].boxes.data	Permite aceder às caixas delimitadoras dos objetos detetados. No código as caixas delimitadoras são retomadas em forma de lista de coordenadas (x1,y1,x2,y2, confiança, classe).

3.6 Instalação das bibliotecas *OpenCV* e *Ultralytics*

Atualmente a instalação do *Opencv* e do *Ultralytics* no *Raspberry* Pi 4 e 5 não requer muitos procedimentos desde que ambos possuam as características descritas na tabela 6. Existem 3 requisitos básicos antes de preparar a instalação do ambiente *OpenCV*:

- Instalação do sistema operativo *Linux Raspbian* no hardware *Raspberry Pi*;
- Conexão do módulo da câmera *Raspberry Pi*;
- Atualização do software do sistema operativo para a versão mais recente.

A instalação do sistema operativo *Raspbian* é feito através do *download* do software *Raspberry Pi imager* no site oficial [58]. Para fazer a instalação do software no computador deve-se inserir o cartão de memória micro-SD no adaptador de *micro-SD* no computador. Feito isso, abre-se a aplicação *Pi imager* e escolhe-se o tipo de dispositivo (*Raspberry Pi* 4 ou 5), o tipo de sistema operativo e o cartão de armazenamento. O módulo da câmera é conectado fisicamente através da porta de câmera disponível na placa *Raspberry Pi*, como mostra a figura 32.

Figura 32- Ligação da câmera *Raspberry Pi* à placa principal.

Para este projeto foram usados como disco local do *Raspberry Pi* cartões microSD de 64 Gb, inserindos na porta SD da placa *Raspberry Pi*. Esses cartões SD contêm o sistema operativo *Linux* instalado no computador. Foi usado um monitor, teclado e rato para configurar a rede *wifi* do *Raspberry*, e depois passou-se a usar o ambiente de trabalho remotamente a partir do *RealVNC* [59].

A próxima etapa é fazer o *download* e instalação da versão mais recente do software *Raspbian OS*. Isso pode ser feito digitando o seguinte comando no terminal:

```
~$sudo apt-get update && sudo apt-get upgrade
```

Após a atualização do software, deve ser feita a instalação das bibliotecas OpenCV e *ultralytics* digitando os seguintes comandos no terminal:

```
~$sudo pip install opencv-python

~$sudo pip install ultralytics
```

Nota: Ao instalar o opencv-python no Raspberry Pi 5 na primeira tentativa dará um erro de instalação. Para solucionar este problema deve inserir os seguintes comandos:

```
~$sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED

~$sudo pip install  torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
```

Depois de executar o ambiente *OpenCV*, pode ser feita a preparação dos dados para este projeto. Existem dois pontos que precisam ser considerados e preparados antes de serem transferidos para o *Raspberry Pi* como protótipo de hardware:

1. Ficheiro do modelo YOLO: a saída dos dados da máquina treinada a ser usada para a deteção de veículos;
2. Ficheiro *Python*: o programa principal para rastrear o veículo e detetar a disponibilidade de lugares de estacionamento.

Esses requisitos serão explicados em detalhes nas secções seguintes. Trata-se de como treinar a deteção dos veículos a partir de um algoritmo de treino e como detetar a disponibilidade de estacionamento utilizando as teorias de região de interesse (ROI), via programação *Python*.

3.7 Preparação de dados para o programa de treino

Projetar um sistema para calcular disponibilidade de lugares de estacionamento através de técnicas de processamento de imagens é uma tarefa complexa. A preparação de dados para detetar veículos está dividida nas seguintes etapas [60]:

- **Recolha de imagens:** Para simular o programa, primeiro é feita a aprendizagem de máquina, para que a máquina aprenda a reconhecer vários tipos de veículos. Para este projeto cerca de 630 conjuntos de imagens tiradas pelo telemóvel (222 imagens) e pela câmera do Raspberry Pi 4 e 5 (408 imagens) foram usadas para treinar o modelo YOLO;
- **Definição das classes:** A partir das imagens recolhidas foi definida apenas uma classe: "***car***";
- **Anotação dos objetos:** Foram utilizadas ferramentas de anotação automática de cada imagem através da plataforma *Roboflow* para criar caixas delimitadoras (*bounding boxes*) ao redor de cada veículo presente nas imagens. A figura 33 apresenta um exemplo do processo de anotação de imagens realizado no âmbito deste projeto.

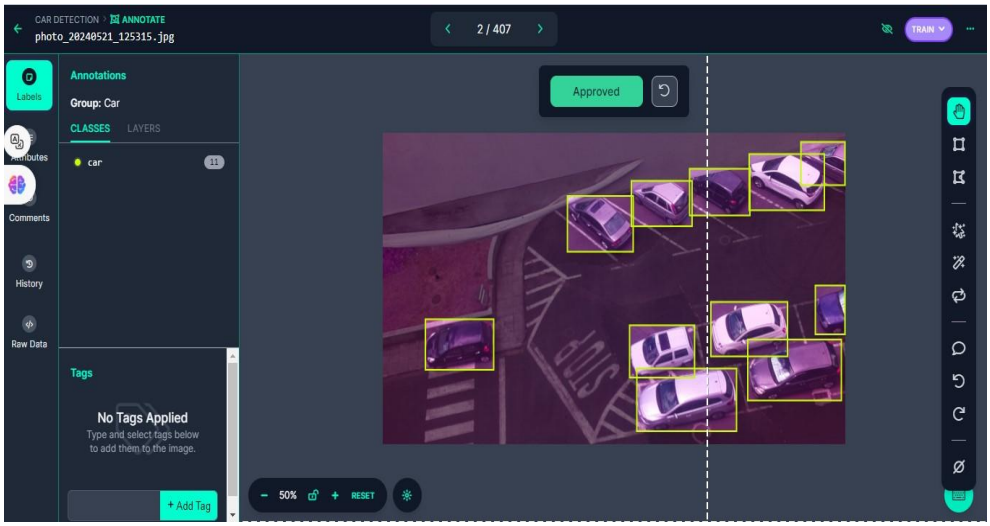


Figura 33- Anotação dos veículos na plataforma *Roboflow*.

Na imagem, observa-se a interface da plataforma *Roboflow* durante a fase de anotação. Os principais elementos destacados são:

- **Imagem original:** Captura aérea de um estacionamento, onde vários veículos estão identificados;
- **Caixas delimitadoras (*Bounding Boxes*):** Cada veículo na imagem foi marcado com uma caixa amarela, indicando a sua posição e dimensão;
- **Painel de anotações:** No lado esquerdo, observa-se que os objetos foram classificados na categoria "***Car***" (veículo), totalizando 11 veículos identificados;
- **Linha de separação:** A imagem está dividida ao meio, permitindo comparar a versão original com a versão anotada;
- Botão "***Approved***": Indica que a imagem foi revista e validada, ficando pronta para ser incluída no conjunto de treino do modelo;
- **Pré-processamento dos dados:** Para o pré-processamento dos dados foi realizada a orientação automática e redimensionamento das imagens. O redimensionamento muda o tamanho das imagens para o tamanho de entrada esperado pelo modelo YOLOv8. Para este projeto foi usada a opção de redimensionamento *Fit (black edges)* de 640x640, sendo aplicado a todas as imagens do *dataset*.
- **Aumento de dados (*Augmentation*):** Passo aplicado às imagens existentes no conjunto de dados, sendo que este processo ajuda a melhorar a capacidade do modelo treinado ao generalizar e assim atuar de forma mais eficaz em imagens não vistas. A plataforma *Roboflow* suporta as seguintes opções para o aumento artificial do *dataset*: inversão, rotação de 90 graus, rotação aleatória, recorte aleatório, desfocagem, ruído aleatório e exposição. No entanto recomenda-se a criação de um *dataset* sem a utilização do aumento de dados pois o mesmo só se aplica a *dataset* mais complexos, sendo que o seu uso pode prejudicar ou melhorar o desempenho de treino da máquina.
- **Divisão do conjunto de dados:** O conjunto de dados foi dividido em treino (441 imagens, cerca de 70%), validação (126 imagens, cerca de 20%) e teste (63 imagens, cerca de 10%) para o treino e avaliação do desempenho do modelo. A Figura 34 apresenta a divisão do conjunto de dados utilizado

para o treino do modelo de deteção de veículos, bem como as etapas de pré-processamento aplicadas.

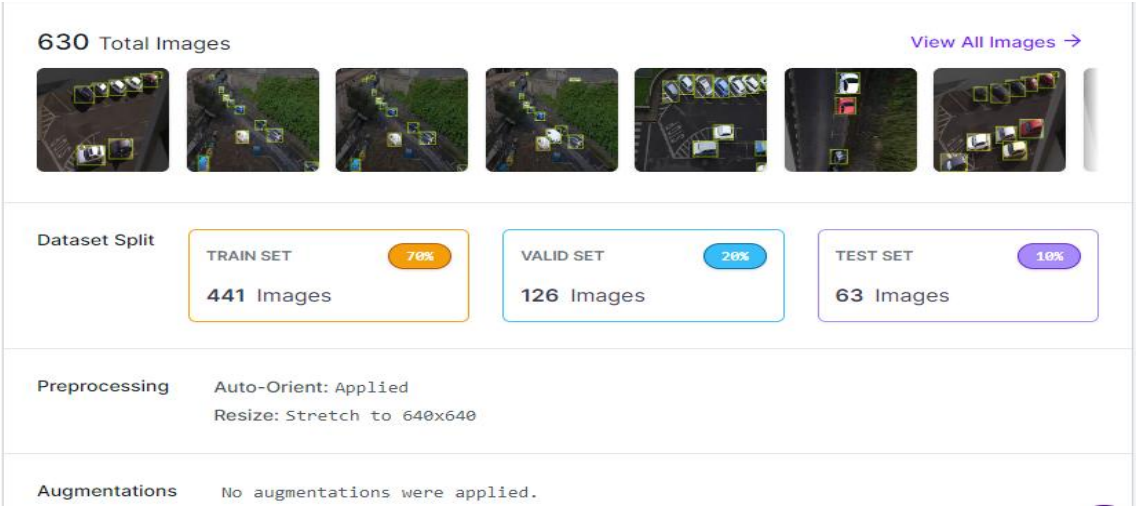


Figura 34- Divisão do *dataset*.

A figura 34 ilustra a interface da plataforma *Roboflow*, onde o *dataset* foi carregado, dividido e pré-processado.

Concluída a criação do *dataset* é feita a exportação em código Python para o treino. A figura 35 apresenta a interface da plataforma *Roboflow*, especificamente a secção onde é gerado o código para descarregar o conjunto de dados anotado. Este código permite a integração direta do *dataset* num ambiente de desenvolvimento Python, como *Jupyter Notebook* ou um terminal, facilitando a utilização do conjunto de dados para treino de modelos de *machine learning*.

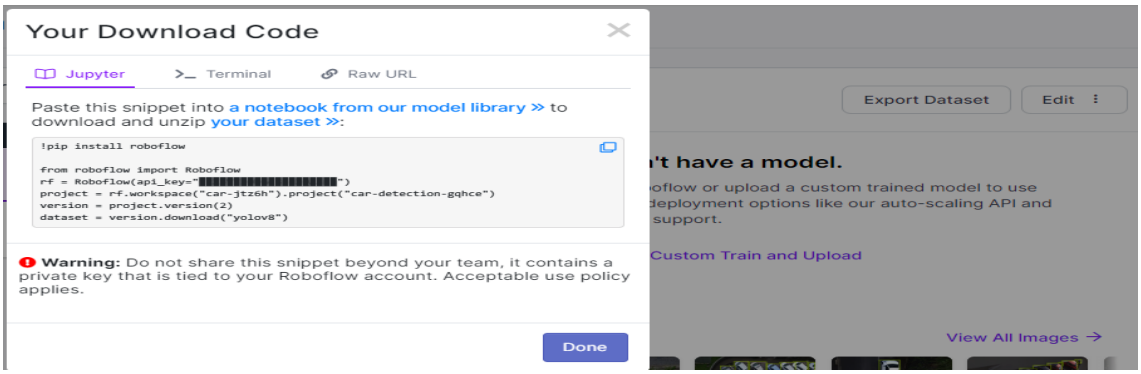


Figura 35- Exportação do *dataset*.

A imagem mostra um trecho do código Python fornecido pela *Roboflow* para *download* e extração do *dataset*. Os principais comandos são:

- **Instalação da biblioteca *roboflow***

```
!pip install roboflow
```

Este comando instala a biblioteca Roboflow, necessária para aceder aos conjuntos de dados armazenados na plataforma;

- **Autenticação com chave de API**

```
from roboflow import Roboflow

rf = Roboflow(api_key="**************")
```

A autenticação é feita através de uma API *key* privada (que não deve ser compartilhada). Esta chave permite aceder ao workspace do utilizador na *Roboflow*;

- **Especificação do conjunto de dados**

```
project = rf.workspace("car-5i2fh").project("car-detection-g9che")

version = project.version(2)

dataset = version.download("yolov8")
```

  - ***Workspace* e projeto**: Define o nome do *workspace* e do projeto onde o conjunto de dados foi criado;
  - **Versão do dataset:** Indica a versão do *dataset* que será descarregada (versão 2 no exemplo);
  - **Formato de *download*:** O *dataset* é descarregado no formato compatível com YOLOv8.

Refira-se que o *Roboflow* é uma plataforma *online* que permite que utilizadores criem bases de dados (*dataset*) personalizadas e treinem modelos de visão computacional. Neste projeto o *Roboflow* foi usado apenas para a criação da base de dados e o treino foi realizado através de um algoritmo de treino em Python [60].

3.8 Treino do modelo usando o algoritmo em python

Depois de concluído a fase de criação do *dataset* é realizado o treino da máquina. Sabe-se que esse processo pode consumir muito tempo de treino, dependendo da velocidade de processamento, memória RAM e placa gráfica do computador e do número de imagens a serem treinadas. Optou-se por isso em usar os processadores do google *Colaboratory* (plataforma da Google que permite executar códigos em Python e acesso a *GPU* sem custos financeiros) para o treino da máquina com as seguintes fases:

- **Instalação da biblioteca *Ultralytics***: A instalação da biblioteca é feita através do seguinte comando: **"pip install ultralytics**";
- **Importação do *dataset*:** O *dataset* criado é importado e extraído com o seguinte comando (figura 36):

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("car-jtz6h").project("car-detection-gqhce")
version = project.version(2)
dataset = version.download("yolov8")
```

Figura 36- Importação do *dataset*.

- **Treino da máquina**: Para o início do treino é usada a seguinte linha de código em Python:

```
from ultralytics import YOLO

!yolo task=detect mode=train model=YOLOv8n.pt data={dataset.location}/data.yaml epochs=50
imgsz=640 plots=True
```

Em que:

Task: Refere-se à tarefa de deteção de objetos que será realizada (*detect*);

Mode: Indica que o modelo será treinado (*train*);

Model: Especifica o modelo que será utilizado para a deteção de objetos, neste caso, o modelo a ser usado é o YOLOv8n.pt;

Data: Refere-se à localização dos dados que serão utilizados para o treino do modelo;

Epochs: Indica a quantidade de *epochs* (iterações completadas através do conjunto de dados de treino) que serão utilizadas durante o treino do modelo;

Imgsz: Especifica o tamanho das imagens que serão utilizadas como entrada para o modelo durante o treino;

Plots: Indica que serão gerados gráficos durante o treino para visualizar o progresso e os resultados.

Os demais parâmetros são autodefinidos com os parâmetros padrão para complementar o treino (conforme mostra a figura 37):

	from	n	params	module	arguments
0	-1	1	2320	ultralytics.nn.modules.conv.Conv	[3, 80, 3, 2]
1	-1	1	115520	ultralytics.nn.modules.conv.Conv	[80, 160, 3, 2]
2	-1	3	436800	ultralytics.nn.modules.block.C2f	[160, 160, 3, True]
3	-1	1	461440	ultralytics.nn.modules.conv.Conv	[160, 320, 3, 2]
4	-1	6	3281920	ultralytics.nn.modules.block.C2f	[320, 320, 6, True]
5	-1	1	1844480	ultralytics.nn.modules.conv.Conv	[320, 640, 3, 2]
6	-1	6	13117440	ultralytics.nn.modules.block.C2f	[640, 640, 6, True]
7	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
8	-1	3	6969600	ultralytics.nn.modules.block.C2f	[640, 640, 3, True]
9	-1	1	1025920	ultralytics.nn.modules.block.SPPF	[640, 640, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	3	1948800	ultralytics.nn.modules.block.C2f	[960, 320, 3]
16	-1	1	922240	ultralytics.nn.modules.conv.Conv	[320, 320, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	3	7174400	ultralytics.nn.modules.block.C2f	[960, 640, 3]
19	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
22	[15, 18, 21]	1	8718931	ultralytics.nn.modules.head.Detect	[1, [320, 640, 640]]

Model summary: 365 layers, 68153571 parameters, 68153555 gradients, 258.1 GFLOPs

Figura 37- Parâmetros do código para o processo de treino.

A figura 37 é um resumo do modelo YOLOv8 detalhando as camadas e os módulos que compõem a arquitetura do modelo da seguinte forma:

a) Estrutura da tabela:

- **From:** indica de onde a entrada para esta camada surge. O valor '-1' significa que a entrada vem diretamente da camada anterior e valores como [-1, 6] indicam que a camada recebe entradas de múltiplas camadas (a última camada e a camada de índice 6);
- **n:** indica o número de repetições de uma camada ou bloco;
- **params**: indica o número de parâmetros nesta camada;
- **module:** O tipo de módulo ou camada utilizada;
- **arguments:** argumentos ou parâmetros específicos para a camada.

b) Descrição das camadas:

- ***Conv layers*** (camadas de convolução): essas camadas aplicam operações de convolução para extrair características das imagens. As convoluções são definidas pelos argumentos [**in_channels, out_channels, kernel_size, stride**], como exemplo temos a seguinte camada:

0 - ultralytics.nn.modules.conv.conv [3, 80, 3, 2]

Esta camada tem 3 canais de entrada, 80 canais de saída, um tamanho de *kernel* de 3x3 e um *stride* de 2.

- ***C2f block*** (Bloco C2f): um bloco que inclui convoluções e operações adicionais, repetido várias vezes. Utiliza um esquema de convolução e é repetido 'n' vezes, onde 'n' é especificado nos argumentos. Por exemplo:

2 - ultralytics.nn.modules.block.c2f [160, 160, 3, True]

Este bloco recebe 160 canais de entrada e de saída, é repetido 3 vezes, e o último argumento geralmente é uma configuração booleana para a utilização de um tipo específico de convolução.

- **SPPF** (*Spatial Pyramid Pooling Fast*): técnica para aumentar o campo de visão da rede neuronal, essencialmente agregando informações de diferentes escalas. Por exemplo:

9 - ultralytics.nn.modules.block.SPPF [640, 640, 5]

Recebe 640 canais de entrada e saída, e utiliza um tamanho de *pooling* de 5.

- **Upsample** (Reamostragem): camada que aumenta a resolução das características extraídas, usando interpolação 'nearest'. Por exemplo:

10 - torch.nn.modules.upsampling.upsample [None, 2, 'nearset']

- **Concat** (Concatenação): camadas que concatenam a entrada de diferentes camadas. Por exemplo:

11 - ultralytics.nn.modules.conv.Concat [1]

- **Detect (deteção):** a camada final que realiza a deteção de objetos. Por exemplo:

22 - ultralytics.nn.modules.head.Detect [1, [320, 640, 640]]

Indica que o modelo está configurado para detetar uma classe (número de classe é 1) e os tamanhos de entrada são 320, 640 e 640.

O processo de treino da máquina pode levar algum tempo e geralmente depende de vários fatores, tais como:

- **A velocidade do processador:** quanto mais rápido for a velocidade de processamento, menor será o tempo necessário para o programa de treino YOLO ser concluído.
- **Memória do computador**: durante o processo de treino, a execução do programa exige uma grande quantidade de recursos, tanto de memória RAM como de armazenamento temporário, para processar os dados e otimizar os parâmetros do modelo de *machine learning*.
- **O tamanho das amostras:** Quando o tamanho da imagem for maior que o normal, a extração dos recursos será mais lenta, portanto, a conclusão do processo de treino levará mais tempo.

Após a preparação do conjunto de dados para treino e definição da arquitetura do modelo, o próximo passo fundamental consiste na configuração do optimizador, que desempenha um papel essencial na eficiência do processo de treino.

3.9 Configuração do optimizador

Em problemas de aprendizagem profunda (*Deep Learning*), o optimizador desempenha um papel fundamental na atualização dos pesos da rede neuronal para minimizar a função de perda e, consequentemente, melhorar a precisão do modelo. Ele ajusta os parâmetros do modelo com base nos gradientes calculados durante o processo de retro-propagação, garantindo que a rede aprenda de forma eficiente [61].

Para este projeto, foi escolhido o optimizador AdamW, configurado com uma taxa de aprendizagem de 0,002 e um momento de 0;9. Esses valores foram previamente definidos no código utilizado para o treino e não podiam ser alterados.

O *AdamW* é uma variante do optimizador *Adam* (*Adaptive Moment Estimation)* e é amplamente utilizado em *Deep Learning* devido à sua capacidade de acelerar a convergência durante o treino. A principal diferença entre Adam e AdamW reside

na regularização L2, que no *AdamW* é aplicada diretamente aos pesos, resultando em melhor generalização do modelo [61].

A atualização dos momentos (Eq. 27 e Eq. 28) e dos parâmetros (Eq. 29) é definida pelas seguintes equações [61]:

- **Estimativa dos momentos de primeira ordem (média dos gradientes):**

$$\boldsymbol{m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t} \quad (27)$$

Nesta equação, $\boldsymbol{m_t}$ representa a média móvel exponencial dos gradientes, que suaviza as atualizações e reduz a variabilidade do treino. O parâmetro $\boldsymbol{\beta_1}$ controla o peso dos gradientes passados, e foi fixado em 0,9 no código usado para o treino.

- Segunda ordem (média dos quadrados dos gradientes):

$$\boldsymbol{v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2} \quad \textbf{(}28)$$

Esta equação acompanha a variabilidade dos gradientes, permitindo ajustar a taxa de aprendizagem dinamicamente para cada parâmetro. O valor de $\boldsymbol{\beta_2}$ foi fixado em 0,999, garantindo uma atualização estável.

- **Atualização dos Parâmetros**

A atualização dos parâmetros no processo de otimização ocorre de acordo com:

$$\boldsymbol{\theta_t = \theta_{t-1} - \eta \left(\frac{m_t}{\sqrt{v_t + \epsilon}} + \lambda \theta_{t-1} \right)} \quad (29)$$

onde:

η - É a taxa de aprendizagem (0,002), que define o tamanho do passo na direção do gradiente;

λ - É o coeficiente de regularização (*weight decay*), usado para evitar *overfitting*;

β_1 e β_2 - São os coeficientes de decaimento exponencial para os momentos de 1ª e 2ª ordem;

ϵ - É um termo de estabilização numérica, que evita a divisão por zero.

O termo *weight decay* na equação (29) adiciona uma penalização aos parâmetros, ajudando a reduzir o risco de sobreajuste do modelo aos dados de treino. Este fator é especialmente importante em aplicações de visão computacional, como a gestão

inteligente de estacionamento, onde a variabilidade nas imagens pode afetar a capacidade do modelo de generalizar para diferentes condições.

Com essa abordagem, o optimizador AdamW equilibra a estabilidade das atualizações dos pesos e a generalização do modelo, garantindo um treino eficiente para o sistema de deteção de lugares de estacionamento utilizando o YOLOv8.

Quando a linha de código Python de treino é executada com os parâmetros definidos, a máquina inicia o treino. A figura 38 mostra como a máquina treina usando a imagem recolhida por meio do modelo YOLOv8n.pt.

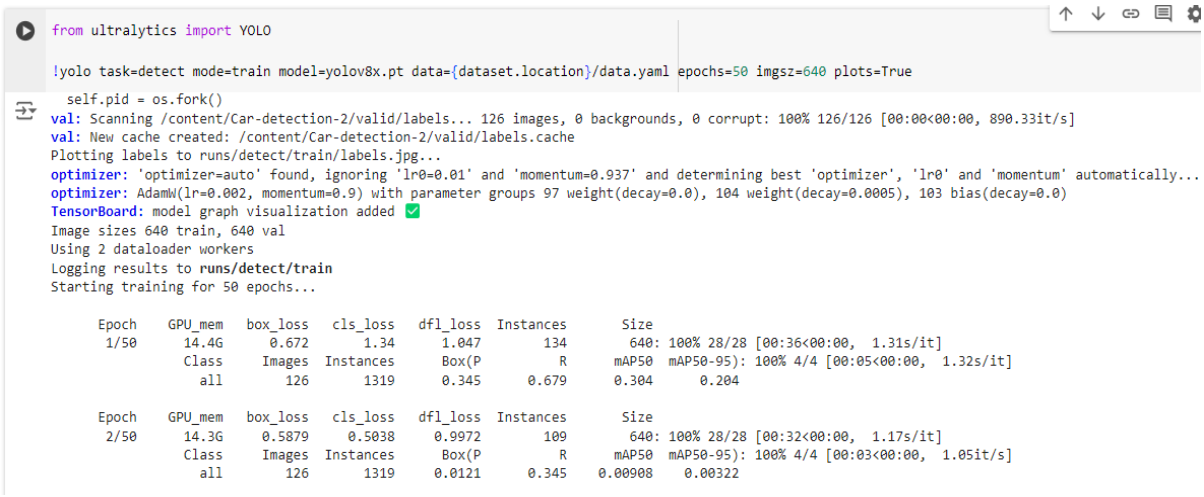


Figura 38- Processo de treino da máquina em 50 iterações.

Na Figura 38, observa-se o processo de treino do modelo YOLOv8 configurado para 50 épocas (iterações). Esse número foi escolhido para permitir que a rede neuronal aprenda os padrões relevantes das imagens, reduzindo os erros de deteção, como falsas classificações de objetos e falhas na identificação correta dos veículos.

Durante o treino, o modelo ajusta seus pesos a cada época, minimizando a função de perda (*box_loss, cls_loss e dfl_loss*) para melhorar a precisão da deteção. Além disso, métricas como mAP50 (média de precisão para um limiar de 50%) são monitorizadas para avaliar o desempenho progressivo do modelo. Após a conclusão do treino, a saída do arquivo gerado pode ser vista na figura 39.

```
50 epochs completed in 0.791 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 136.7MB
Optimizer stripped from runs/detect/train/weights/best.pt, 136.7MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.2.31 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 68124531 parameters, 0 gradients, 257.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:09<00:00,  2.38s/it]
                   all        126       1319      0.988      0.976      0.994      0.976
Speed: 0.3ms preprocess, 28.8ms inference, 0.0ms loss, 4.0ms postprocess per image
Results saved to runs/detect/train
```

Figura 39- Arquivo gerado do modelo YOLOv8.

Após o treino do modelo por 50 épocas, é essencial avaliar seu desempenho para verificar se ele atingiu um nível satisfatório de precisão na deteção dos veículos. Para isso, são utilizadas métricas de avaliação que quantificam a qualidade das predições feitas pelo modelo.

A próxima seção apresenta as principais métricas utilizadas neste projeto, como precisão, *recall* e mAP, que permitem medir a capacidade do modelo de identificar corretamente os objetos de interesse e analisar os impactos de erros de deteção, como falsos positivos e falsos negativos.

3.10 Métricas de avaliação de desempenho

Para avaliar a eficácia das deteções, foram utilizadas métricas tradicionais como precisão, *recall* e mAP (*mean Average Precision*). Essas métricas permitem medir a capacidade dos modelos de identificar objetos de interesse e avaliar os custos associados a falsos positivos e negativos [62].

3.10.1 Matriz de correlação

Num modelo de deteção de objetos, a matriz de correlação é usada para avaliar a capacidade do modelo em detetar corretamente os objetos de interesse (neste caso veículos) e pode ser definida da seguinte maneira [62]:

- *True positive* (TP): indica que o modelo detetou corretamente a presença de um veículo onde realmente havia um veículo;
- *False positive* (FP): indica que o modelo detetou um veículo onde na verdade não havia um veículo (falso alarme);
- *False negative* (FN): indica que o modelo não detetou um veículo que estava presente (falha na deteção);
- *True negative* (TN): indica que o modelo identificou corretamente que não havia um veículo onde realmente não havia um veículo.

A matriz está dividida em quatro quadrantes, cada um representando um possível resultado da classificação realizada pelo modelo (figura 40).

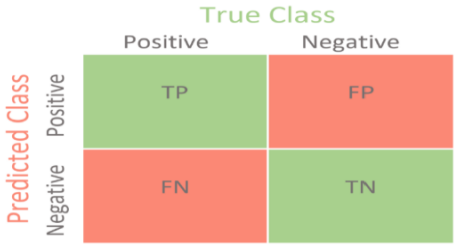


Figura 40- Matriz de correlação [62].

Esta matriz permite calcular métricas de desempenho do modelo, tais como [62]:

- **Precisão (*Precision*):** Mede quantas das previsões positivas estavam corretas;
- **Revocação (*Recall* ou Sensibilidade):** Mede quantas das instâncias positivas foram corretamente identificadas;
- **Exatidão (*Accuracy*):** Mede a proporção total de previsões corretas.

A matriz de correlação adaptada para o modelo de deteção de veículos pode ser representada na tabela 10.

Tabela 10- Matriz de correlação adaptada.

	Previsto: veículo	Previsto: Não veículo
Real: veículo	TP	FN
Real: Não veículo	FP	TN

Esta tabela representa uma matriz de correlação adaptada ao problema de deteção de veículos no contexto do projeto. A matriz permite avaliar o desempenho do modelo de classificação (com o YOLOv8) ao prever se uma determinada área contém um veículo ou não.

3.10.2. Taxa de classificação ou exatidão

A taxa de classificação ou exatidão é uma métrica que indica a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões, sendo calculada pela seguinte relação [62]:

$$\textbf{Exatidão} = \frac{\textbf{TP+TN}}{\textbf{TP+FP+FN+TN}} \quad (30)$$

A taxa de classificação apresenta algumas limitações. Ela assume custos iguais para ambos os tipos de erros (falsos positivos e falsos negativos), o que nem sempre reflete a realidade. Por exemplo, uma eficácia de 99% pode parecer excelente, mas a sua interpretação depende do contexto do problema. Se a classificação for aplicada a um sistema de deteção de veículos num parque de estacionamento, uma taxa de erro de 1% pode ser irrelevante. No entanto, num sistema de segurança que identifica veículos suspeitos, essa mesma taxa de erro pode ser inaceitável.

Além disso, a taxa de classificação pode ser enganosa quando os dados são desbalanceados, ou seja, quando há uma diferença significativa na quantidade de exemplos entre as classes. Por exemplo, num conjunto de dados onde 95% das imagens correspondem a "espaços vazios" e apenas 5% a "veículos estacionados", um modelo que prevê sempre "espaço vazio" terá uma precisão de 95%, mas será completamente inútil para identificar veículos.

Para superar essas limitações, utilizam-se outras métricas como a precisão (*precision),* a revocação (*recall*), que oferecem uma avaliação mais robusta do desempenho do modelo, especialmente em cenários com classes desbalanceadas.

3.10.3 Cálculo da precisão

A precisão é definida como a proporção de objetos de uma classe específica que foram corretamente classificados (TP) em relação ao total de objetos atribuídos a essa classe (TP+FP). A fórmula é a seguinte [62]:

$$\mathbf{Precisão} = \frac{\mathbf{TP}}{\mathbf{TP+FP}} \tag{31}$$

Essa métrica é calculada individualmente para cada classe presente no conjunto de dados.

3.10.4 Cálculo do *recall*

O *recall*, ou sensibilidade, mede a proporção de objetos da classe positiva que foram corretamente classificados. Este parâmetro avalia quantos objetos realmente pertencentes à classe positiva foram capturados pelo classificador em relação ao total de objetos dessa classe. A fórmula é [62]:

$$\boldsymbol{Recall} = \frac{\mathbf{TP}}{\mathbf{TP+FN}} \tag{32}$$

Assim como a precisão, o *recall* é calculado para cada classe do conjunto de dados.

A *mAP* (*mean Average Precision*) é uma métrica utilizada para avaliar o desempenho de modelos de deteção de objetos [62]. É uma das principais métricas de avaliação, amplamente adotada em aplicações de visão computacional, que oferece uma visão abrangente sobre a eficácia do modelo em detetar vários objetos.

De maneira geral, a *mAP* é a média da área sob a curva precisão-*recall* para todas as classes de objetos. Essa métrica fornece uma avaliação global do desempenho do modelo na tarefa de deteção de objetos, sendo que valores mais altos de mAP indicam melhor desempenho.

3.11 Simulação do programa Python

Para criar um arquivo *Python* para detetar a disponibilidade de estacionamento, foi desenhado um fluxograma para representar o algoritmo. Este fluxograma representa o mesmo procedimento do programa *Python* na simulação e no protótipo de hardware. A figura 41 mostra o fluxograma do programa *Python*, para detetar a disponibilidade de lugares de estacionamento.

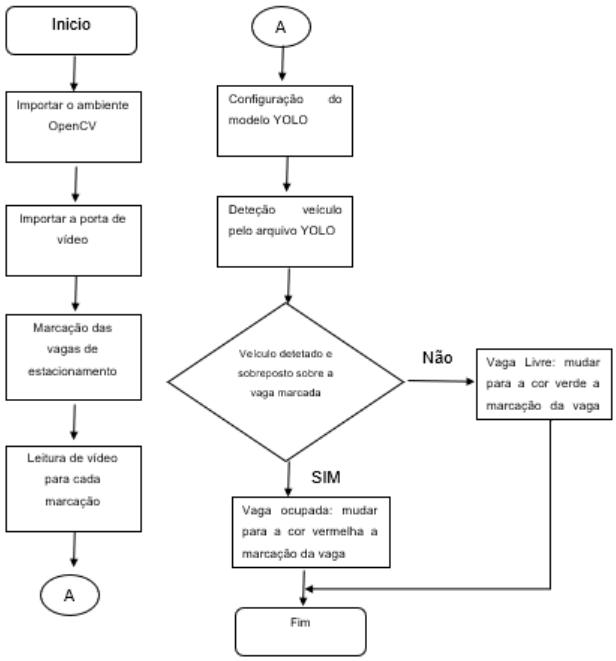


Figura 41- Fluxograma do arquivo *Python* para deteção de estacionamento.

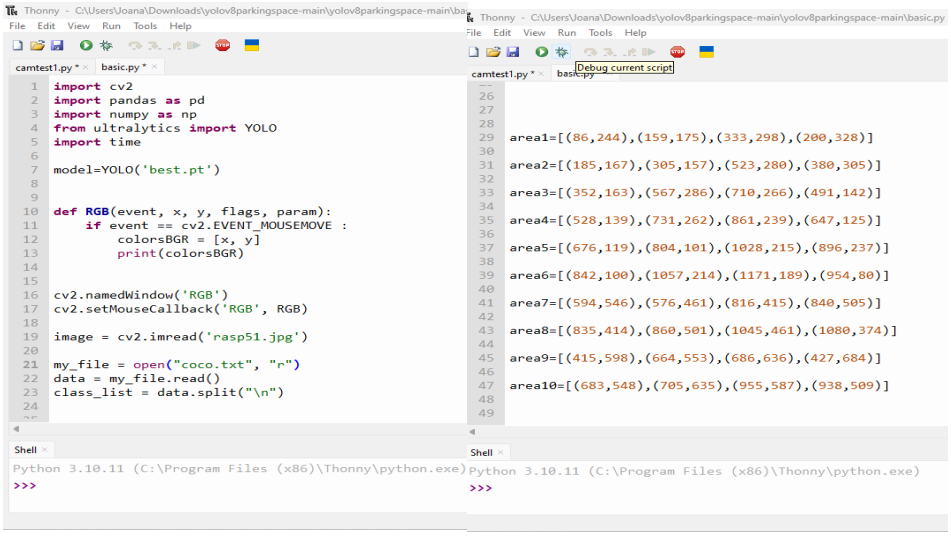
As etapas podem ser descritas da seguinte forma:

- **Início:** O sistema é inicializado e inicia a configuração do ambiente necessário para a execução do modelo;
- **Importação do ambiente e da porta de vídeo:** Representa o carregamento das bibliotecas necessárias para a execução do código (*OpenCV*). Define a fonte de vídeo que será utilizada para a análise (câmera ou vídeo gravado);
- **Marcação dos lugares de estacionamento:** Indica que o sistema define as áreas correspondentes aos lugares dentro da imagem/vídeo, onde será feita a verificação da ocupação;
- **Leitura do vídeo para cada lugar:** Indica que o sistema processa o vídeo quadro a quadro, analisando cada lugar previamente marcado;
- **Configuração do modelo YOLO:** Indica que o modelo YOLO é carregado e configurado para detetar veículos dentro das imagens capturadas;
- **Deteção do veículo:** Indica que o YOLOv8 processa o quadro de vídeo e verifica a presença de veículos sobre os lugares marcados;
- **Verificação da ocupação do lugar:** Se um veículo for detetado e sobreposto ao lugar marcado, o lugar é considerado ocupado e a sua marcação muda para vermelho. Se nenhum veículo for detetado, o lugar é considerado livre e a marcação muda para verde;
- **Fim do Processo:** Indica que o sistema continua a processar novos quadros de vídeo, repetindo o ciclo para cada nova atualização.

Refira-se ainda que a ação principal do programa para detetar a disponibilidade de lugares de estacionamento é a utilização da teoria da região de interesse (ROI). Para que o algoritmo detete a disponibilidade de lugares de estacionamento, o programa precisa identificar se a área de interesse selecionada, neste caso a área de estacionamento, irá sobrepor a deteção do veículo. Se houver sobreposição ocorrerá a mudança da cor da marcação da respetiva lugar para vermelho, indicando que está ocupada.

Caso não haja sobreposição (fora da região de interesse) o programa considerará a lugar como livre, mantendo a marcação do respetivo lugar na cor verde. Para desenhar a região de interesse é identificado um conjunto de coordenadas para mostrar a área do lugar de estacionamento. O projeto possui 10 áreas de

estacionamento, portanto existem dez (10) regiões de interesse a serem marcadas. A figura 42 mostra o código *Python* que representa o fluxograma da figura 41.



```python
import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time

model=YOLO('best.pt')


def RGB(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE :
        colorsBGR = [x, y]
        print(colorsBGR)


cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)

image = cv2.imread('rasp51.jpg')

my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")
```

```
Python 3.10.11 (C:\Program Files (x86)\Thonny\python.exe)
>>>
```

```python
area1=[(86,244),(159,175),(333,298),(200,328)]
area2=[(185,167),(305,157),(523,280),(380,305)]
area3=[(352,163),(567,286),(710,266),(491,142)]
area4=[(528,139),(731,262),(861,239),(647,125)]
area5=[(676,119),(804,101),(1028,215),(896,237)]
area6=[(842,100),(1057,214),(1171,189),(954,80)]
area7=[(594,546),(576,461),(816,415),(840,505)]
area8=[(835,414),(860,501),(1045,461),(1080,374)]
area9=[(415,598),(664,553),(686,636),(427,684)]
area10=[(683,548),(705,635),(955,587),(938,509)]
```

```
Python 3.10.11 (C:\Program Files (x86)\Thonny\python.exe)
>>>
```

Figura 42- Arquivo principal *Python* para deteção de lugares de estacionamento.

Após gerar o arquivo principal *Python* inicia-se a simulação onde será testada a deteção de veículos. Para executar o arquivo *Python* dentro do *sofware Thonny* [63], o arquivo de treino deve estar na mesma pasta que o arquivo *Python* (figura 43).

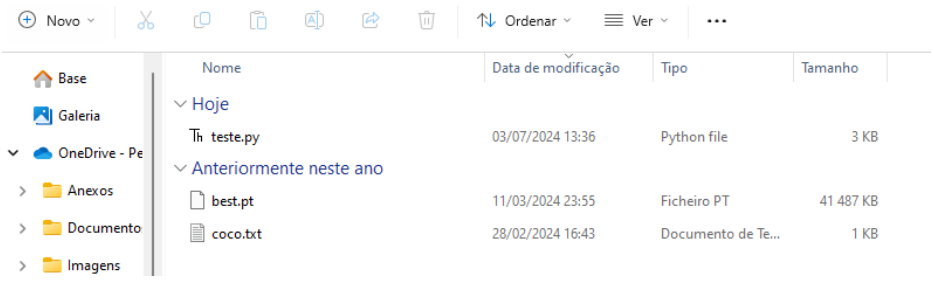


Figura 43- Localização dos aquivos.

3.12 Implementação de hardware

Após simular o programa foram aplicados os mesmos procedimentos para o protótipo de hardware. As figuras 44 e 45 mostram a configuração e a posição do protótipo, sendo que o mesmo foi instalado no terraço do edifício da Universidade da Madeira, na Penteada, para melhor visualização do parque de estacionamento, cobrindo assim diversos lugares de estacionamento.

Figura 44- Configuração do projeto em hardware real e a vista do módulo de câmera.

A figura 45 apresenta o sistema de captura de imagens baseado em *Raspberry Pi*, instalado dentro de uma caixa de proteção. A montagem inclui uma câmera *Raspberry Pi*, conectada via cabo *flat* à placa *Raspberry Pi*, que é responsável pelo processamento das imagens para o sistema de gestão inteligente de estacionamento.

Figura 45- *Raspberry Pi* instalado dentro do protótipo.

Na imagem da esquerda, a tampa da caixa está aberta, permitindo visualizar a placa *Raspberry Pi* e a sua câmera. A imagem da direita mostra a caixa fechada, com um recorte na tampa para que a lente da câmera possa capturar imagens do ambiente externo.

3.13 Marcação das lugares de estacionamento

Para a marcação dos 10 lugares do parque de estacionamento foi utilizada a seleção manual de Regiões de Interesse poligonais (figura 46).

Essa abordagem consiste na definição manual dos limites de cada lugar através de coordenadas específicas dentro da imagem. A resolução da imagem utilizada foi de 640x480 pixels, e cada lugar foi representada por um conjunto de pontos que delimitam a sua área na cena capturada.

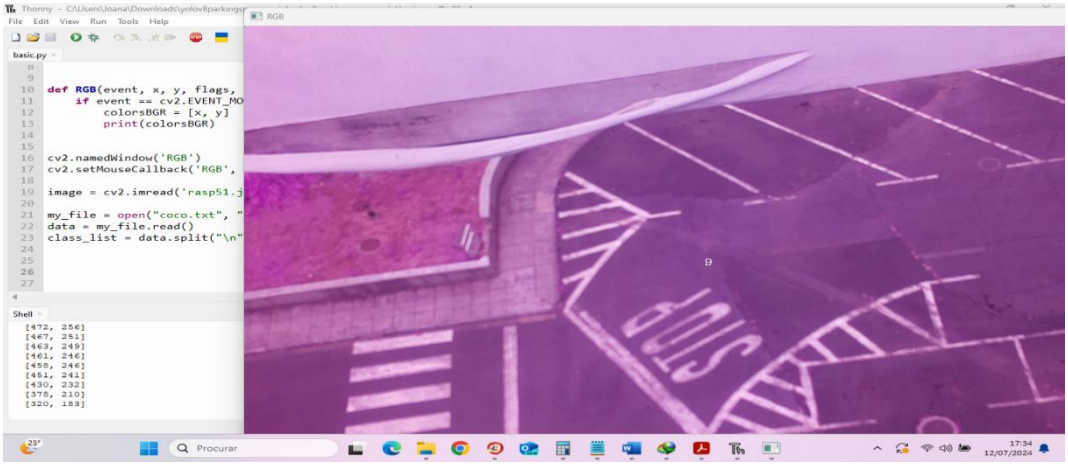


Figura 46- Coordenadas dos lugares de estacionamento.

A escolha da seleção manual de ROI poligonais deve-se ao fato de que os lugares nem sempre possuirem formatos perfeitamente retangulares, sendo necessário um método flexível que se adapte à geometria real do estacionamento. Assim, cada lugar foi definido por um conjunto de vértices (x, y), armazenados numa estrutura de dados para posterior análise.

Após a marcação dos lugares, o sistema utiliza o modelo YOLOv8 para detetar a presença de veículos sobre as ROI previamente definidas. Caso um veículo seja identificado dentro de uma região, a lugar é classificada como ocupada; caso contrário, é considerada livre. Essa abordagem permite um controle preciso das áreas de estacionamento, garantindo que o sistema possa monitorar com exatidão a ocupação dos lugares.

A figura 47 ilustra o resultado final da delimitação dos lugares de estacionamento após a definição das ROI. Os lugares foram contornados com linhas verdes, e cada uma recebeu um identificador numérico para facilitar o processamento e monitorização do estacionamento. Essa estrutura permite que, posteriormente, o

modelo YOLOv8 possa analisar cada lugar individualmente e determinar a sua ocupação.

Figura 47- Marcação dos 10 lugares de estacionamento.

Depois de projetar as ROI pode-se prosseguir com a implementação do hardware para obtenção dos resultados. Para fazer isso o código *Pyhton* foi transferido para o *Raspberry Pi*.

3.14 Criação da página *web*

Para informar os utilizadores sobre a disponibilidade dos lugares em tempo real foi usada uma página *web* em código PHP (*Hypertext Preprocessor)* que recebe os estados dos lugares através de uma base de dados (criada em MySQL) [64] e atualiza em tempo real a disponibilidade dos lugares de estacionamento.

3.14.1 Criação da base de dados

Para criar a base de dados MySQL, que recebe as atualizações do estado de cada lugar de estacionamento, é necessário criar a estrutura da base de dados e depois configurar o código *Python* para que ele comunique com a base de dados e insira os dados dos lugares livres e ocupados. A sua estrutura é composta por uma única tabela denominada *parking_spaces*, que contém os seguintes campos:

- *Id:* chave primária auto-incrementada que serve como identificador único para cada registro na tabela;
- space_id: campo responsável por identificar individualmente cada lugar de estacionamento;
- *Occupied*: estado do lugar, ‘1’ para ocupada, ‘0’ para livre;
- *Timestamp:* regista a data e hora da última atualização do estado do lugar.

Por fim é executado o seguinte código SQL na base de dados para a criação da tabela:

```
CREATE TABLE parking_spaces ( id INT AUTO_INCREMENT PRIMARY KEY, space_number INT NOT NULL, occupied TINYINT NOT NULL, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );
```

A figura 48 mostra a interface do *phpMyAdmin*, uma ferramenta de administração para bases de dados MySQL, acessada localmente via *localhost* (127.0.0.1).

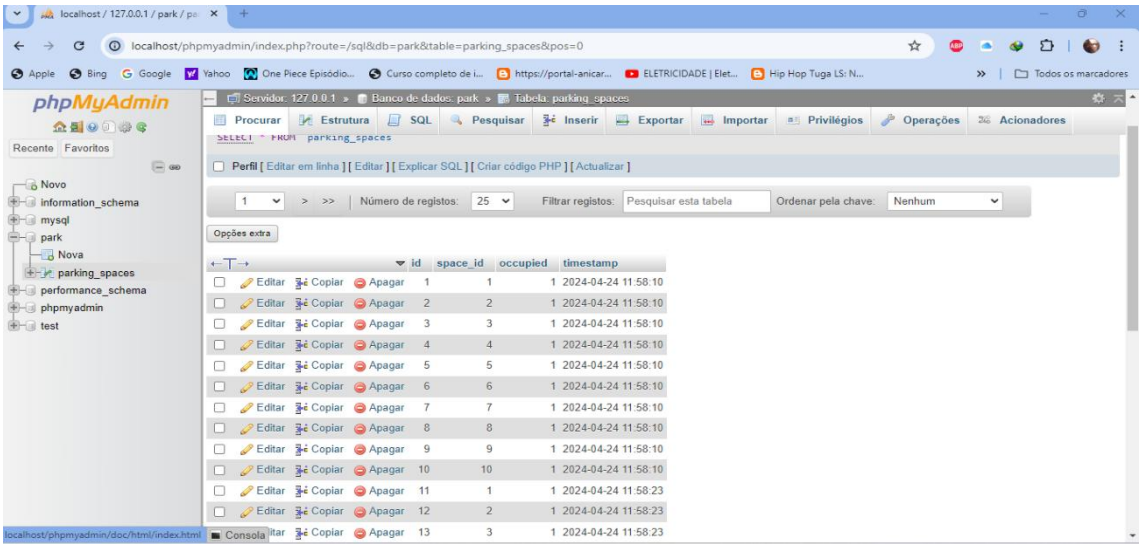


Figura 48- Estrutura da tabela *Parking_spaces.*

Na tela, a base de dados "*park*" é selecionada, e dentro dela, a tabela "*parking_spaces*" é visualizada. Cada linha da tabela representa um lugar de estacionamento e o seu respetivo estado. A interface permite operações como editar, copiar e apagar registos, além de consultas SQL e importação/exportação de dados.

A seguir é preciso configurar o código *Python* para haver comunicação com a base de dados de forma a inserir os dados dos lugares, tal como é descrito no anexo B. Com essas alterações o código *Python* inserirá automaticamente as atualizações do estado de cada lugar de estacionamento na tabela '*parking_spaces'* da base de dados MySQL.

3.14.2 Página *web*

O código PHP (anexo C), gera uma página com elementos dinâmicos (*slots* de estacionamento, fila e imagens) que são automaticamente chamados através do AJAX (mecanismo de atualização em tempo real) [65], que envia solicitações

assíncronas para um *script* PHP para verificar o estado atual dos lugares de estacionamento na base de dados e retorna apenas as mudanças (figura 49).

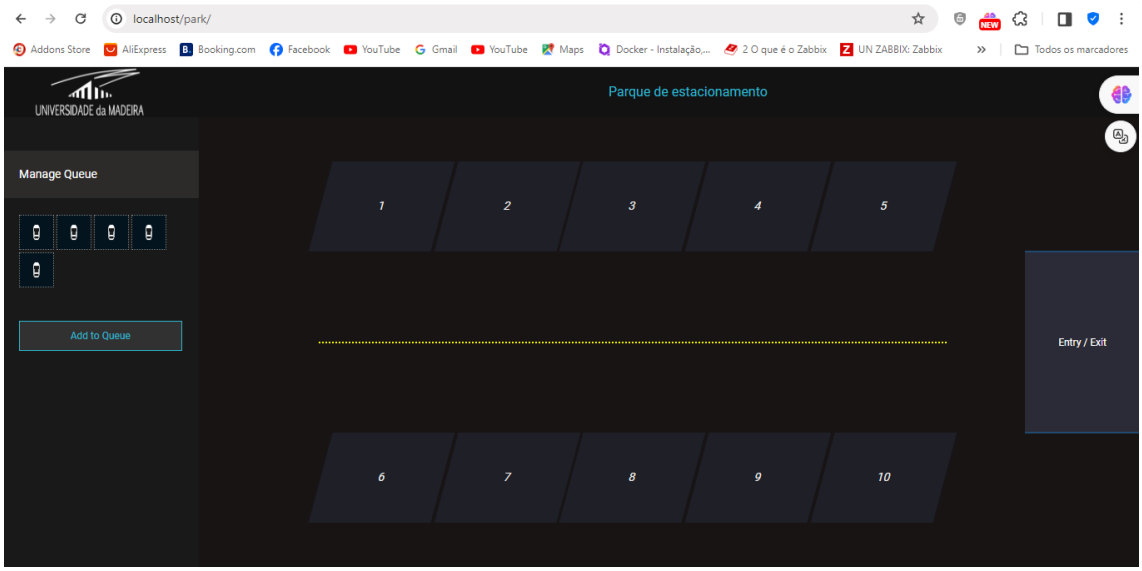


Figura 49- Página *web*.

A figura 49 mostra a interface *web* do sistema de gestão de estacionamento, acessado localmente através do navegador. A interface apresenta um *layout* escuro e minimalista com uma representação gráfica dos lugares de estacionamento. O parque de estacionamento está organizado em duas fileiras, contendo 10 lugares numerados de 1 a 10.

No processo de comunicação entre o *Raspberry Pi* 4 ou 5 com a base de dados deve-se ter em conta os testes de conexão realizados através do terminal de comando, no qual o *Raspberry Pi* fornece um IP específico para inserir dentro do servidor, e o IP fornecido deve ter permissão de acesso à base de dados, sendo um passo importante para permitir o funcionamento correto da página *web*.

Refira-se que tanto a base de dados como página *web* foram alocados dentro de um servidor local através do *software xaamp* [66]. Nos anexos D e E encontram-se os códigos necessários para o funcionamento do AJAX e do Javascript.

4 Análises e resultados

Neste capítulo são apresentados os resultados do projeto do sistema inteligente de deteção de disponibilidade de lugares de estacionamento.

4.1 Evolução do treino da máquina

Nesta secção, analisa-se a evolução das métricas de desempenho do modelo YOLO em função do número de imagens utilizadas no processo de treino. O objetivo é determinar o valor mínimo de imagens necessário para alcançar um desempenho aceitável na deteção de veículos, contribuindo assim para a otimização do sistema proposto.

Com base na documentação oficial da *Ultralytics*, recomenda-se a utilização de, pelo menos, 128 imagens para a criação de um *dataset* de treino [67]. Foram realizados três treinos distintos com conjuntos de dados compostos por 51, 100 e 129 imagens, respetivamente. As Figuras 50, 51 e 52 apresentam, por ordem, os resultados obtidos para cada um destes casos. Em cada figura são exibidas as seguintes métricas:

- Perdas de treino: *train/box_loss, train/cls_loss e train/dfl_loss*;
- Perdas de validação: *val/box_loss, val/cls_loss e val/dfl_loss*;
- Métricas de desempenho: *metrics/precision(B), metrics/recall(B), metrics/mAP50(B) e metrics/mAP50-95(B*).

Estas métricas permitem avaliar a capacidade do modelo de generalizar a partir dos dados de treino, assim como a sua precisão na deteção dos objetos.

A figura 50 apresenta os resultados obtidos com 51 imagens. Observa-se que as curvas de perda apresentam oscilações significativas, o que pode indicar instabilidade no treino ou sobre-ajuste. As métricas de precisão e *recall* situam-se entre 70% e 90%, revelando um desempenho moderado, mas ainda insuficiente para aplicações práticas robustas.

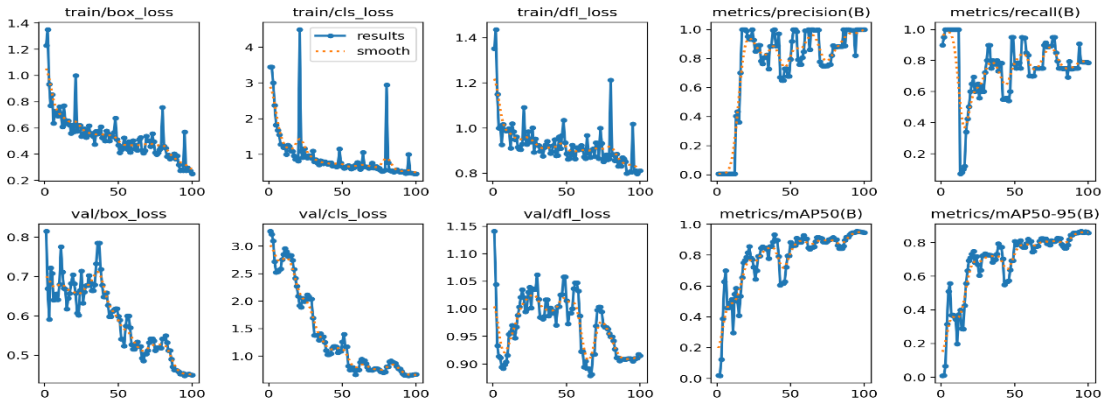


Figura 50- Resultados de treinos com 51 imagens.

A Figura 51 exibe os resultados com 100 imagens, onde se verifica uma melhoria na suavidade das curvas de perda, tanto no treino como na validação. As perdas diminuem de forma mais consistente ao longo das épocas. As métricas de precisão e *recall* aproximam-se de 90 %, enquanto os valores de mAP também revelam uma evolução positiva.

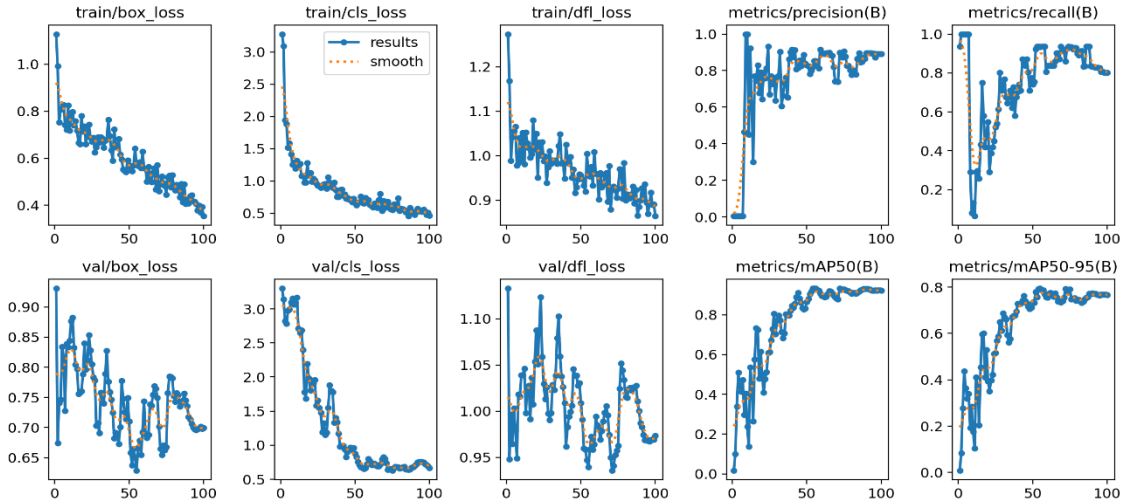


Figura 51- Resultados com treino de 100 imagens.

Na Figura 52, apresentam-se os resultados obtidos com 129 imagens, número superior ao mínimo recomendado. As curvas de perda tornam-se bastante estáveis e com tendência decrescente ao longo do treino, o que indica uma boa convergência do modelo. As métricas de desempenho ultrapassam consistentemente os 90 %, destacando-se os valores de mAP50 e mAP50-95, que revelam uma elevada fiabilidade na deteção.

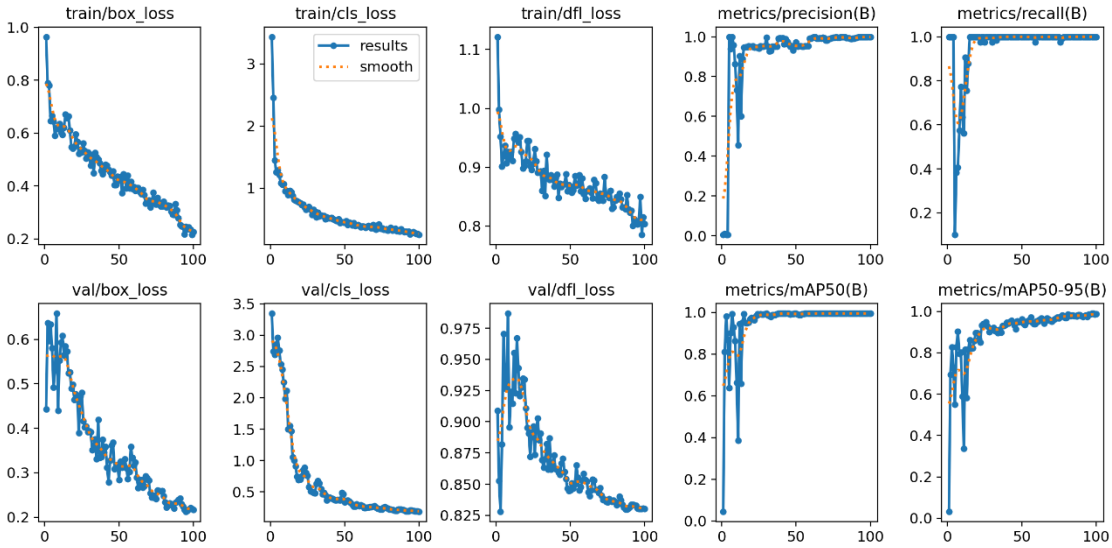


Figura 52- Resultados de treino com 129 imagens.

Com base na comparação dos três casos apresentados, conclui-se que o aumento do número de imagens utilizadas no treino tem um impacto direto na melhoria do desempenho do modelo. As perdas tornam-se progressivamente menores e mais estáveis, enquanto as métricas de precisão e mAP aumentam de forma consistente. Embora os valores de precisão sejam superiores a 90 % nos três treinos, a análise das perdas permite confirmar que um maior volume de dados contribui significativamente para a robustez do modelo.

4.2 Resultados de treino da máquina

Na fase de treino do modelo utilizou-se a máquina virtual do *Google Colaboratory* com as seguintes configurações: *RAM* de 12,7 GB, *GPU* (unidade de processamento gráfico) de 15 GB, armazenamento de 78,2 GB e CPU do tipo Tesla T4. Os resultados obtidos com este modelo são apresentados nas figuras 53, 54, 55, 56.

Ao analisar o tempo de treino, observou-se que o YOLOv8n leva aproximadamente 25 minutos para completar 50 *epochs* (iterações). Durante os testes, o modelo demonstrou uma maior precisão na deteção e classificação de veículos.

Conforme mostrado na figura 53, as imagens do conjunto de teste exibem os objetos detetados com as suas caixas delimitadoras e os respetivos níveis de confiança. Esses resultados indicam que o modelo treinado é eficaz na identificação

e classificação dos veículos, garantindo uma boa precisão na segmentação dos lugares de estacionamento e na distinção entre áreas ocupadas e livres.

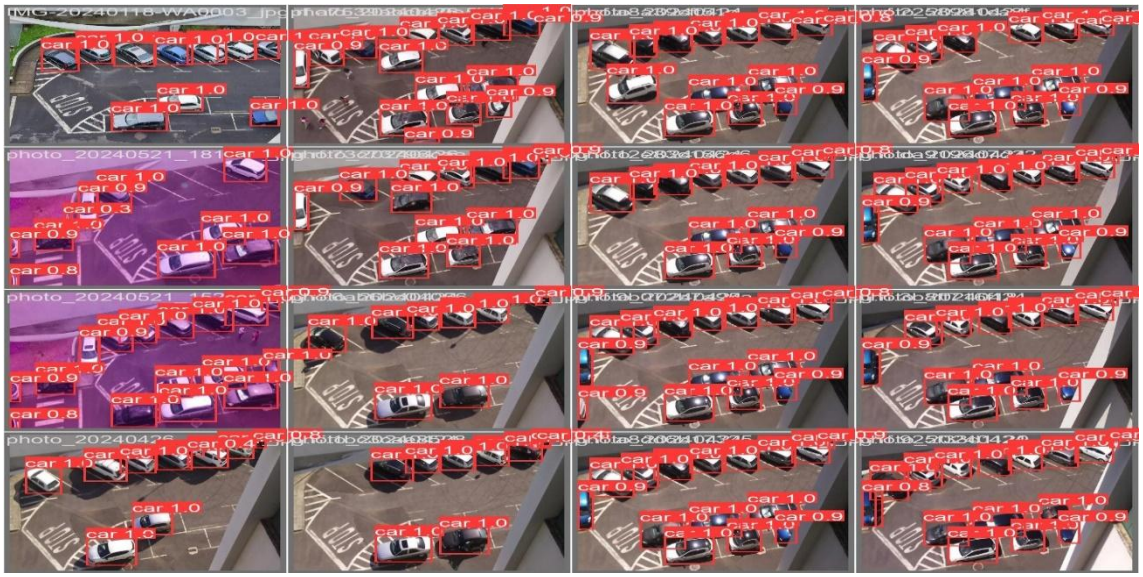


Figura 53- Deteções feitas pelo modelo treinado 1.

A figura 54 apresenta a matriz de correlação do modelo treinado 1, mostrando que a classe "***car***" não teve dificuldades em detetar os veículos. Isso deve-se à simplicidade dessa classe tanto nos detalhes de curvas, pontas e quantidade de ângulos nas suas delimitações. O modelo classificou corretamente 99% dos carros (valor 0,99 na diagonal principal).

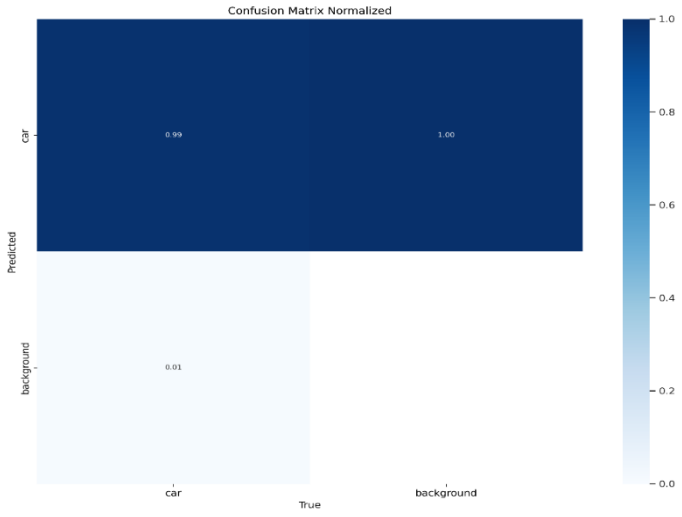


Figura 54- Matriz de correlação normalizada do modelo treinado 1.

Não houve falsos positivos para a classe "*background*" (o modelo não confundiu o fundo com veículos). Apenas 1% dos veículos foram erroneamente classificados como *background*, indicando um baixo número de falsos negativos.

O modelo apresenta um excelente desempenho na deteção de veículos, com alta precisão e baixa taxa de erro. A matriz confirma que o YOLOv8 foi treinado de maneira eficaz para distinguir entre carros e fundo, o que é essencial para a aplicação em gestão inteligente de estacionamento.

A figura 55 apresenta a curva de Precisão-*Recall,* que avalia o desempenho do modelo YOLOv8 na deteção de veículos. Esta curva foi gerada a partir dos resultados do conjunto de teste, utilizando as previsões do modelo comparadas com as anotações reais das imagens. O cálculo foi realizado a partir da relação entre precisão (*precision*) e revocação (*recall*) para diferentes limiares de confiança.

Observa-se que o modelo obteve um mAP@0.5 (*mean Average Precision* com IoU ≥ 0,5) de 0,994, o que indica uma elevada precisão na deteção de veículos. Esse desempenho superior é evidenciado pela forma da curva, que permanece próxima ao topo do gráfico, indicando que o modelo mantém uma alta precisão mesmo quando a revocação aumenta.

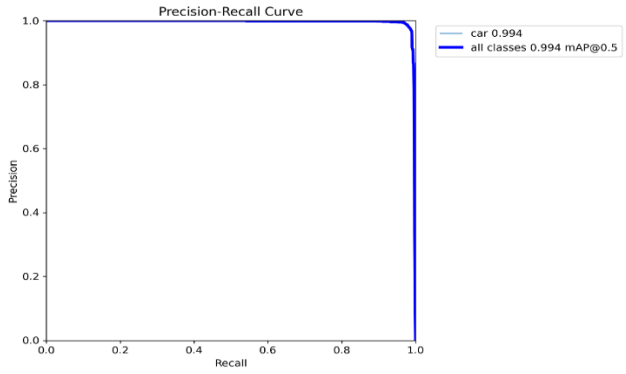


Figura 55- Curva de precisão-*Recall* do modelo treinado 1.

Esse resultado reflete a capacidade do YOLOv8 de minimizar tanto falsos positivos (classificações erradas de veículos) quanto falsos negativos (falha em detetar veículos reais), garantindo assim uma deteção robusta e confiável para a aplicação proposta.

A Figura 56 exibe os resultados de perda e as métricas de desempenho do modelo 1 ao longo do treino. No eixo x, tem-se o número de iterações (*epochs*), e no eixo y, os valores das métricas e das perdas. Os gráficos estão divididos em duas linhas:

- A primeira linha refere-se ao conjunto de treino;
- A segunda linha refere-se ao conjunto de validação.

As primeiras três colunas da figura representam as curvas de perdas ao longo do treino:

- ***Box Loss* (Perda das caixas delimitadoras)**: Mede o erro na predição das coordenadas das caixas delimitadoras dos objetos. Nota-se que essa perda diminui consistentemente, indicando que o modelo melhora na localização dos veículos;
- ***Classification Loss* (Perda de classificação):** Avalia a precisão da classificação dos objetos dentro das caixas detetadas. A curva mostra uma redução gradual, sugerindo que o modelo se torna mais confiável na identificação das classes;
- ***DFL Loss (Dual focal loss):*** Utilizada para otimizar a distribuição dos pesos nos *bounding boxes*, reduzindo erros na borda dos objetos detetados, essa perda também diminui ao longo do treino.

No conjunto de validação, observa-se um comportamento semelhante, indicando que o modelo não sofreu *overfitting* significativo. As três últimas colunas da figura representam as métricas de desempenho:

- **Precisão (*Precision*):** Mede a proporção de verdadeiros positivos entre todas as predições positivas. Atinge valores próximos a 1,0 rapidamente, demonstrando que o modelo tem poucos falsos positivos;
- ***Recall:*** Mede a proporção de verdadeiros positivos entre todas as instâncias reais da classe. Também melhora com as iterações, atingindo valores elevados, o que indica que o modelo deteta a maioria dos veículos corretamente;
- **mAP0.5 e mAP0.5-0.95 (*Mean average precision*):** São métricas que avaliam a precisão média do modelo em diferentes limiares de interseção sobre união (IoU). O mAP0.5 atinge valores próximos de 1,0, confirmando a eficácia do modelo 1. O mAP0.5-0.95, que é mais exigente, também apresenta um desempenho elevado.

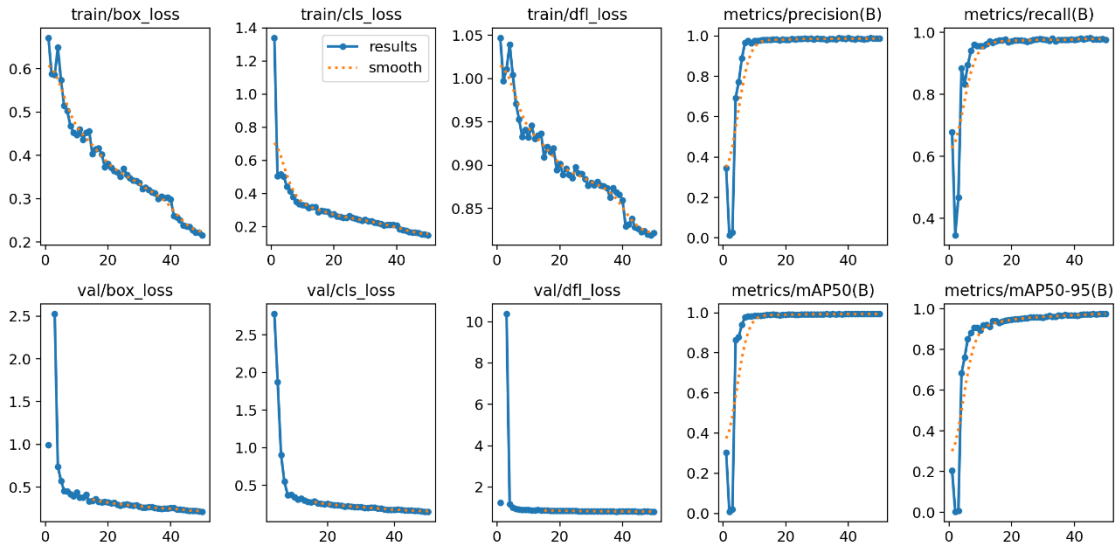


Figura 56- Gráficos de perdas e métricas do modelo treinado 1.

Além das métricas de desempenho, foram realizados testes para medir os tempos médios de pré-processamento e inferência. O modelo treinado apresentou um tempo de inferência médio de 17 ms e 0,6 ms de pré-processamento.

4.3 Comparação de dois cenários de *dataset*

Como discutido na secção 3.7, a opção de Aumento de Dados (*Data Augmentation*) não foi utilizada inicialmente na criação de *datasets* simples, pois pode impactar negativamente o desempenho do treino. No entanto, para avaliar os efeitos dessa técnica, foi criado um novo *dataset*, com as mesmas imagens, mas com modificações aplicadas por meio do aumento de dados.

As transformações aplicadas foram as seguintes:

- Inversão (*Flip*) Horizontal: as imagens foram espelhadas ao longo do eixo vertical, gerando versões invertidas lateralmente;
- Inversão Vertical: as imagens foram espelhadas ao longo do eixo horizontal, resultando em versões de cabeça para baixo;
- Escala (*rotation/scaling*): foi aplicada uma rotação aleatória entre -15º e +15º, modificando ligeiramente o ângulo das imagens para simular variações na captura;
- Saturação das cores (*Color saturation*): a saturação das cores foi ajustada dentro da faixa de -25% a +25%, alterando a intensidade das cores para simular variações de iluminação e contraste;

- Desfocagem (*Blur*): aplicou-se um filtro de desfoque com intensidade de até 2,5 pixels no raio do efeito, suavizando os detalhes da imagem para testar a robustez do modelo diante de imagens levemente desfocadas;
- Ruído (*Noise*): Adicionou-se ruído aleatório até 0,1% dos pixels da imagem. Esse ruído pode ser de diferentes tipos, como gaussiano, simulando interferências naturais nas capturas.

Com as mudanças feitas no novo *dataset* e realizado o treino da máquina, obtiveram-se os resultados mostrados nas figuras 57, 58 e 59.

Figura 57- Deteções feitas pelo modelo treinado 2.

A Figura 57 apresenta os resultados das previsões realizadas pelo modelo treinado 2 sobre o conjunto de dados de teste. Observa-se que o desempenho do modelo foi inferior ao esperado, com baixos níveis de confiança nas deteções. Em diversas imagens, os veículos não foram corretamente identificados ou receberam valores de confiança abaixo de 0,6, o que compromete a confiabilidade do sistema.

Através dos resultados de precisão-*Recall*, apresentados na figura 58, observa-se que o modelo treinado teve um desempenho inferior em termos de precisão e *recall*, atingindo um *mAP* de 0,495 confirmando assim uma péssima precisão na deteção de veículos em comparação com os valores de *mAP* de 0,994 nos resultados anteriores.

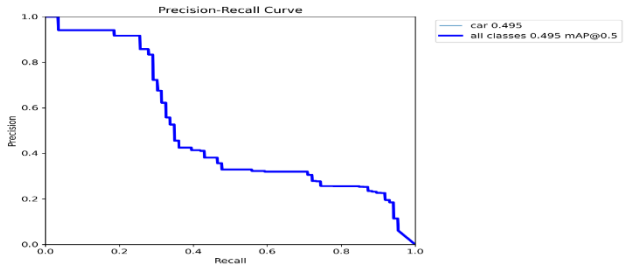


Figura 58- Curva de precisão-*Recall* do modelo treinado 2.

Os resultados das perdas e das métricas de desempenho do modelo (Figura 59) indicam que as métricas de qualidade ficaram abaixo do esperado (acima de 0,5), considerando que valores mais altos de mAP50 e mAP50-95 são desejáveis para um bom desempenho. Embora tenham mostrado uma melhora ao longo das iterações, as métricas de precisão e *recall* apresentaram uma grande variação entre as iterações, indicando um comportamento instável. Quanto às perdas, observou-se um pico elevado nas primeiras iterações do treino (nas primeiras 10 a 20 iterações), seguido de uma redução progressiva, demonstrando que o modelo ajustou seus parâmetros para minimizar os erros.

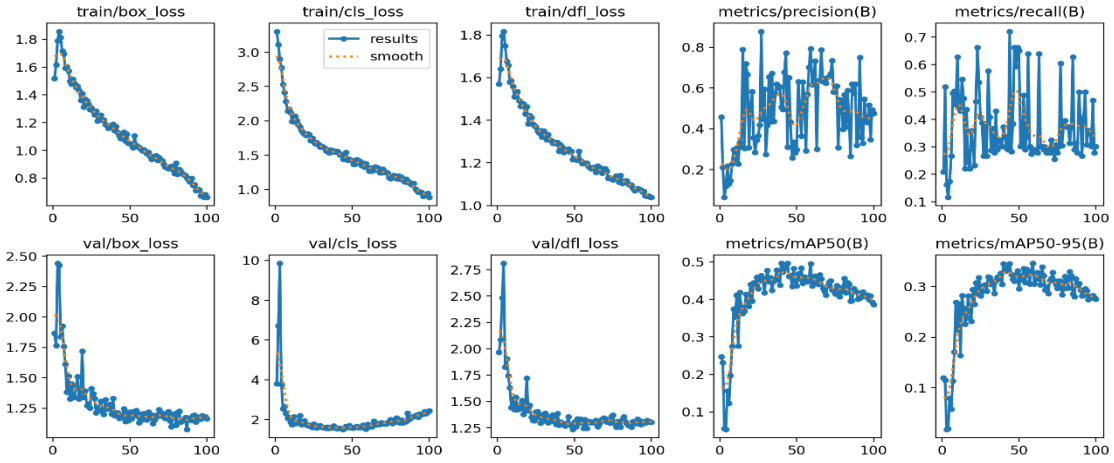


Figura 59- Gráficos de perdas e métricas do modelo treinado 2.

Com os resultados obtidos comprovou-se que o uso da opção *Augmentation* prejudica consideravelmente o treino do modelo 2 no qual atingiu no final do treino uma precisão média de 0,49 ou seja uma redução de 0,45 de precisão comparando com o modelo treinado 1.

Resumindo esta secção através da figura 60, realizaram-se dois cenários, o primeiro com o *dataset* original e o segundo com o data *Augmentation* sendo realizadas as etapas (capítulo 3.3) de criação dos *datasets*, pré-processamento de imagem, escolha da versão do modelo YOLO para o treino, treino dos conjuntos de dados e por fim a deteção e classificação dos mesmos.

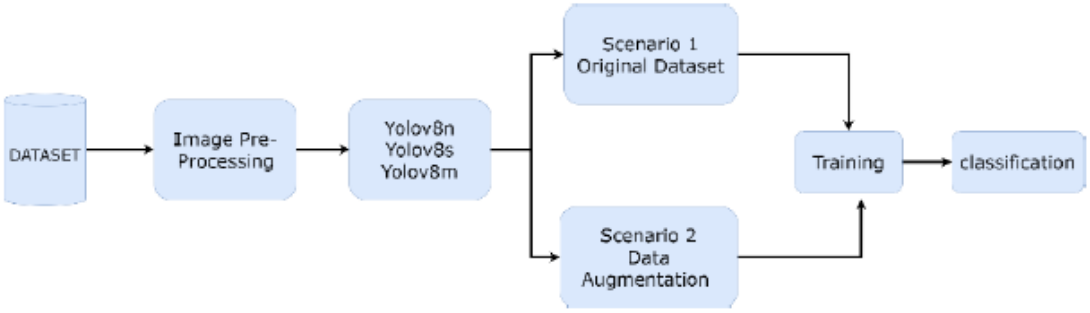


Figura 60- Processo de treino dos dois cenários [68].

As comparações dos resultados dos dois cenários podem ser vistas na tabela 11:

Tabela 11- Distribuição dos *datasets* por cenário.

cenários	treino	validação	teste	total
Cenário 1	441 (70%)	126 (20%)	63 (10%)	630
Cenário 2	1464 (88%)	132 (8%)	68 (4%)	1664

No cenário 2, a distribuição do conjunto de dados foi de 88% para treino (1464 imagens), 8% para validação (132 imagens) e 4% para teste (68 imagens), totalizando 1664 imagens.

As figuras 61 e 62 ilustram a diferença nas imagens de cada cenário. O treino no cenário 1 foi concluído em aproximadamente 40 minutos, enquanto no cenário 2 o tempo aumentou significativamente para cerca de 5 horas.

A pior performance do cenário 2, mesmo com um maior número de imagens, que relacionada a fatores como a maior complexidade do *dataset*, o uso de *augmentations,* não favoreceu a convergência do modelo.

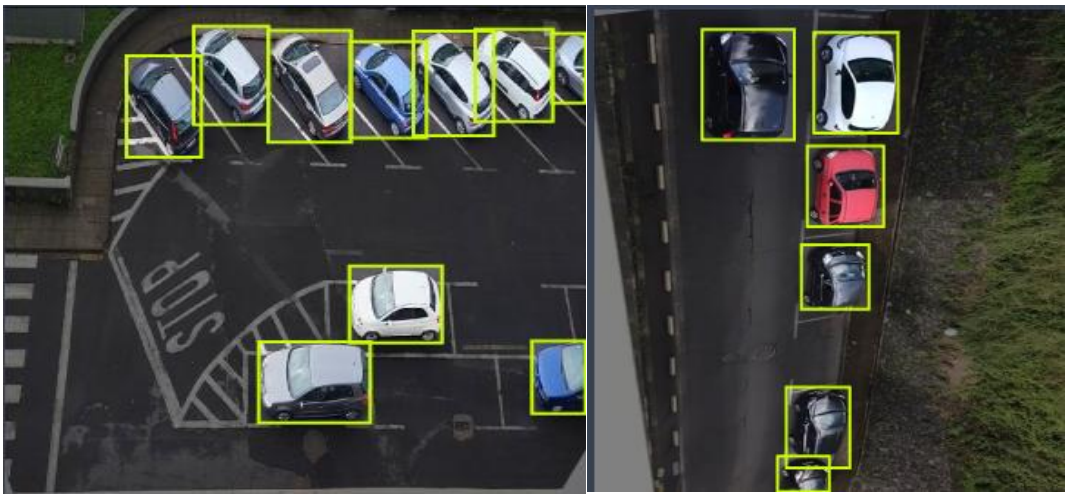


Figura 61- Cenário 1.

A figura 61 mostra um conjunto de estacionamentos com veículos bem definidos e visíveis. A segmentação dos carros foi feita com caixas delimitadoras amarelas, indicando que o modelo estava treinado para detetar os veículos em uma imagem nítida e com boa iluminação. Esse cenário utiliza um conjunto de dados mais reduzido e sem processamento de imagem adicional.

A figura 62 apresenta um tom roxo devido ao uso de técnicas de pré-processamento, uso de aumento de dados (*augmentations)*, que visam tornar o modelo mais robusto a diferentes condições. No entanto, essas transformações podem ter introduzido desafios adicionais para a deteção dos veículos, o que pode justificar os resultados inferiores em relação ao cenário 1.

Figura 62- Cenário 2.

4.4 Resultados da simulação

Os resultados das simulações podem ser vistos nas figuras 63 e 64. As simulações foram realizadas utilizando uma imagem capturada por uma câmera *Raspberry Pi*, com o objetivo de testar a deteção de veículos por meio do programa *Python* no computador. Essa fase é essencial para garantir que o programa conseguirá identificar corretamente a disponibilidade dos lugares de estacionamento.

A figura 63 apresenta os resultados do modelo treinado com o cenário 2, sendo que se observa que a deteção dos veículos não foi completamente precisa. Alguns carros estacionados não foram corretamente identificados, o que pode comprometer a eficácia do sistema de monitorização de lugares. Esse resultado pode estar relacionado com o pré-processamento das imagens e com as técnicas de aumento de dados aplicadas nesse cenário.

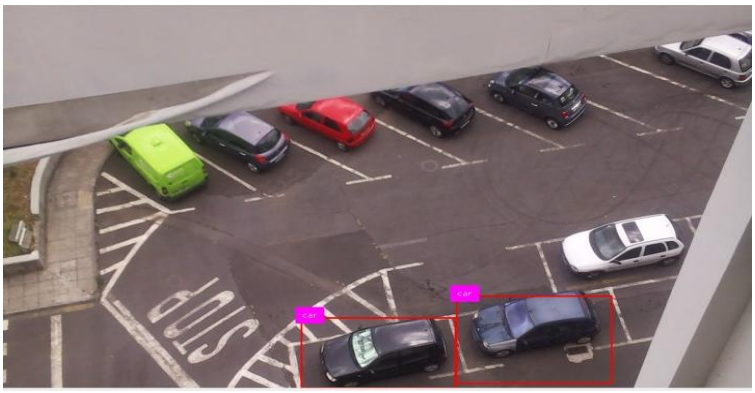


Figura 63- Resultado do modelo treinado 2.

Já a figura 64 exibe os resultados do modelo treinado com o cenário 1, sendo que a deteção dos veículos foi mais eficaz. A maior parte dos carros foi corretamente identificada, com caixas delimitadoras vermelhas, destacando cada veículo. Isso sugere que o modelo treinado com o conjunto de dados do cenário 1 teve um desempenho superior na identificação dos veículos em comparação com o modelo do cenário 2.

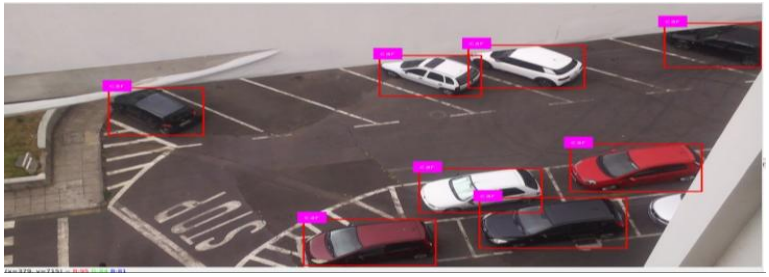


Figura 64- Resultado do modelo treinado 1.

4.5 Resultados obtidos com protótipo

Usando o arquivo *Python* (com o modelo treino do cenário 1) nos protótipos com o *Raspberry Pi* 4 (com o módulo da câmera normal) e o 5 (com o módulo de câmera infravermelho), o código foi executado em janela normal através do software *Thonny*. Refira-se que o arquivo *Python* e o arquivo do modelo treinado (1) foram colocados na mesma pasta para que a execução do programa seja feita sem erros.

Quando o programa é iniciado, as janelas reais do *Raspberry Pi* podem ser vistas na figura 65 onde o código *Python* é executado junto com o arquivo do modelo treinado que foi anexado a ele.

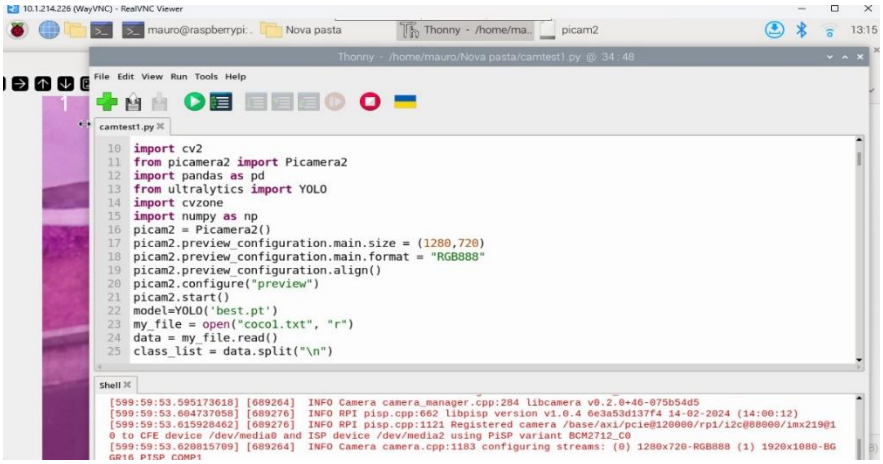


Figura 65- Execução do código no *sotfware Thonny*.

A figura 65 mostra a interface do *Thonny*, um ambiente de desenvolvimento integrado (IDE) utilizado para programar em *Python*, em execução num *Raspberry Pi* através de ligação remota via VNC (*Virtual Network Computing*). Neste contexto, o *Thonny* é utilizado para a execução do script responsável pela captura de vídeo através da câmera do *Raspberry Pi* e pela utilização do modelo YOLO para a deteção de veículos. O resultado da deteção da disponibilidade de estacionamento pode ser visto nas figuras 66 e 67.

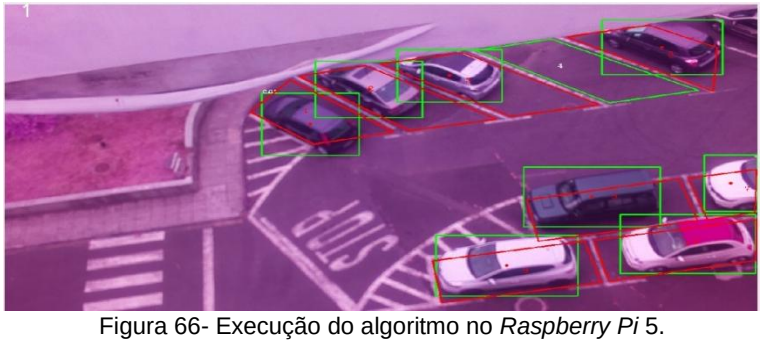


Figura 66- Execução do algoritmo no *Raspberry Pi* 5.

A figura 66 apresenta o resultado da execução do algoritmo de deteção de veículos num cenário real, utilizando o *Raspberry Pi* 5. É possível observar diversas viaturas identificadas com caixas delimitadoras verdes e vermelhas, que representam respetivamente lugares livres e ocupadas. A deteção foi realizada com base nas imagens captadas pela câmera, sendo processadas localmente pelo algoritmo implementado em *Python* com recurso ao modelo YOLOv8, evidenciando a capacidade do sistema em identificar corretamente os veículos estacionados.

A figura 67 ilustra a execução do algoritmo de deteção de veículos no *Raspberry Pi* 4, utilizando o modelo YOLOv8. Nesta fase do projeto, é possível observar que cada veículo foi identificado corretamente e delimitado por caixas coloridas. As caixas verdes indicam lugares de estacionamento livres, enquanto as caixas vermelhas indicam lugares ocupados. A figura demonstra a eficácia do sistema, mesmo com os recursos computacionais mais limitados do *Raspberry Pi* 4, evidenciando a boa performance do modelo treinado.

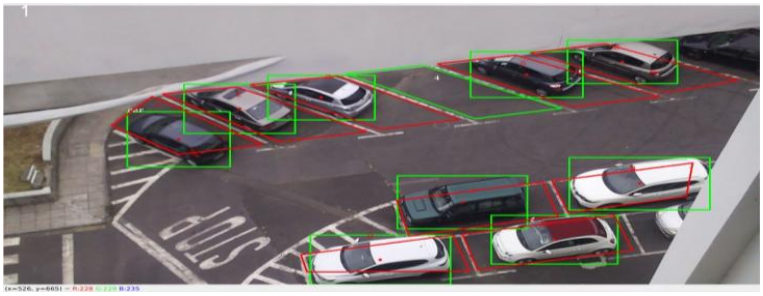


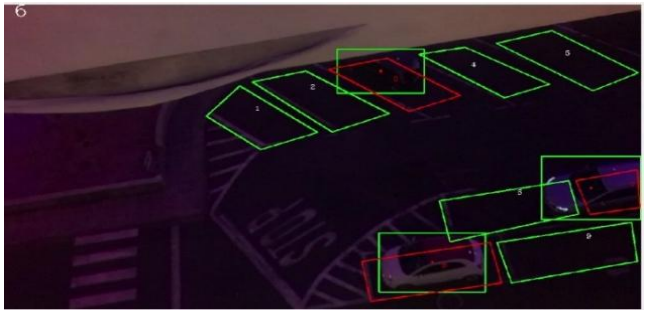
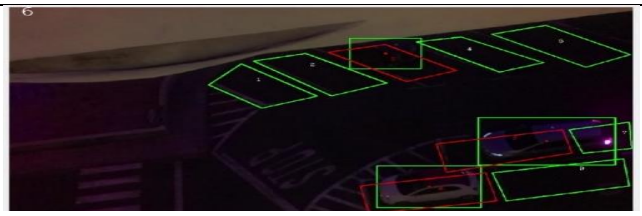
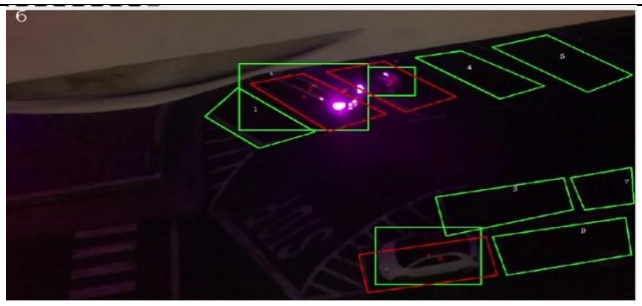
Figura 67- Execução do algoritmo no *Raspberry Pi* 4.

Para detetar a disponibilidade de lugares de estacionamento, a ROI deve estar sobreposta, conforme mencionado na secção 2.9, em que uma das áreas de estacionamento (ROI_2) deve-se sobrepor ao programa de deteção de veículos (ROI_1). Como ambas as ROI se sobrepõem, o algoritmo irá considerar como ocupado marcando a “vermelho” a ROI_2 onde o número 1 mostrado no canto superior esquerdo, representa os lugares disponíveis.

4.5.1 Desempenho de cada protótipo

Nesta secção é realizada uma avaliação do desempenho do algoritmo em cada protótipo e são anotados os comportamentos adquiridos num determinado período em função das condições climáticas e de luminosidade, como se pode ver nas tabelas 12 e 13 e 14.

Tabela 12- Imagens captadas no *Raspberry Pi* 5.

Data	Foto	Nº de veículos nos lugares	Precisão de deteção
13/06/2024 21:00h Período da noite com pouca luminosidade.		3	100%
13/06/ 2024 21:30h Período da noite com pouca luminosidade.		3	100%
13/06/2024 22:00h Período da noite com pouca luminosidade.		3	100%

A análise da tabela 12 sugere algumas conclusões importantes sobre o desempenho do modelo de deteção de veículos em condições de baixa luminosidade:

- Alta precisão de deteção: O modelo alcançou 100% de precisão em todas as capturas, o que indica que ele conseguiu identificar corretamente os veículos nos lugares, mesmo com pouca luz;
- Consistência dos resultados: No intervalo de uma hora, o número de veículos detetados permaneceu constante (3 veículos). Isso sugere que o modelo é estável e não sofre degradação significativa no desempenho ao longo do tempo;
- Impacto da baixa luminosidade: Apesar das condições adversas (noite com pouca luz), o modelo parece ter conseguido identificar os veículos sem grandes problemas. Entretanto, é importante verificar se houve algum falso

positivo ou falso negativo, o que poderia indicar dificuldades em certos casos;

- Possível influência da iluminação artificial: A última imagem (22h) mostra um reflexo mais intenso, possivelmente devido à iluminação artificial. Isso pode impactar a qualidade da deteção em situações futuras se houver variações maiores na iluminação.

O modelo mostrou bom desempenho em condições de baixa luminosidade, mantendo 100% de precisão e estabilidade nos resultados. Como o pré-processamento de imagem foi realizado usando a câmera Pi NoIR, notou-se ainda assim falhas na deteção dos veículos presentes nos lugares de estacionamento, como mostra a figura 68.

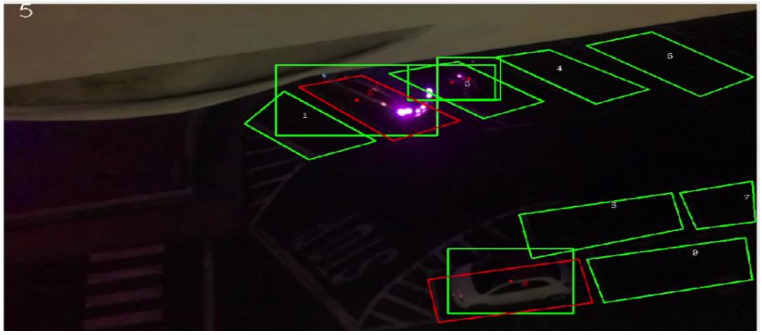


Figura 68- Falha na deteção dos veículos.

Na tabela 13, é apresentado um outro cenário com o *Raspberry Pi* 5 com as seguintes análises: No cenário de alta luminosidade (manhã), a deteção atingiu 100% de precisão, identificando corretamente os veículos nos lugares de estacionamento. Esse resultado demonstra que, em condições ideais de iluminação, o modelo tem um desempenho satisfatório.

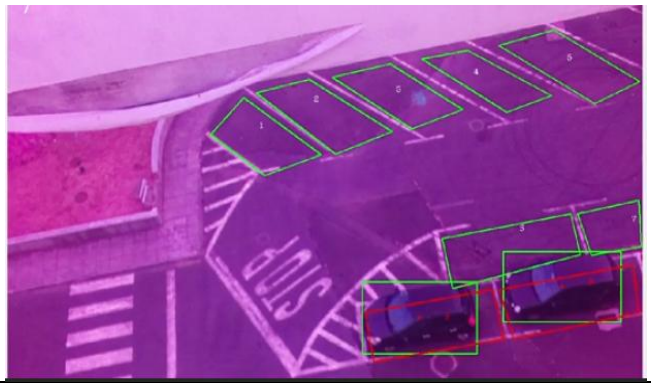
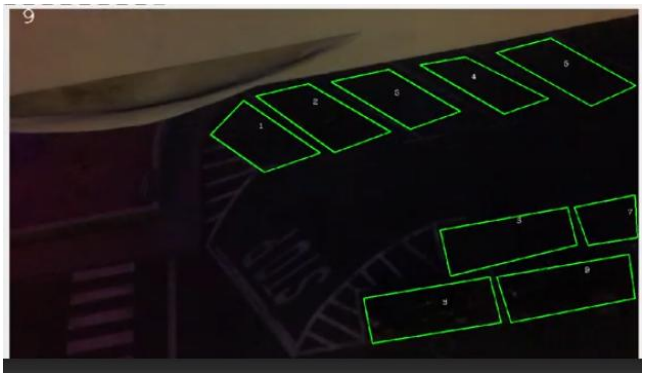
No cenário de baixa luminosidade (período noturno sem iluminação artificial), a precisão foi de 0%. Apesar de alguns veículos estarem presentes nos respetivos lugares, as caixas delimitadoras não foram corretamente posicionadas, resultando em falhas na contagem de veículos estacionados. Esse resultado evidencia que o modelo apresenta dificuldades para operar corretamente em ambientes com pouca luz.

No cenário de noite com iluminação artificial, a precisão voltou a ser de 100%, com todos os veículos corretamente identificados. Esse resultado sugere que a

utilização de fontes de luz adequadas pode mitigar os problemas observados no cenário anterior.

Com base nos testes realizados, verificou-se que a luminosidade do ambiente tem um impacto significativo na eficácia da deteção de veículos. O modelo apresentou alto desempenho em condições de boa iluminação, mas demonstrou limitações em cenários com pouca luz, o que pode comprometer sua aplicação em estacionamentos noturnos sem iluminação adequada.

Tabela 13- Imagens captadas no *Raspberry Pi* 5.

Data	**Foto**	**Nº de veículos de nos lugares**	**Precisão de deteção**
23/05/2024 08:17:00h Período da manhã com muita luminosidade.		2	100%
23/06/ 2024 21:40h Período da noite com pouca luminosidade.		2	0%
24/05/2024 9h:00h Período da manhã com muita luminosidade.		9	100%

A Tabela 14 apresenta os resultados obtidos com o sistema de deteção de veículos utilizando o *Raspberry Pi* 4 em conjunto com o módulo de câmera V2, em diferentes momentos do dia e sob diferentes condições de luminosidade. O objetivo desta fase foi verificar a fiabilidade do sistema em cenários reais, onde a iluminação natural varia significativamente ao longo do tempo.

No primeiro cenário, registado no dia 01/04/2024 às 12h53, durante o período da tarde, com clima chuvoso, foi possível detetar com sucesso 10 veículos, com uma precisão de 100%. Apesar da presença de nuvens e menor intensidade luminosa, a câmera demonstrou boa capacidade de deteção, revelando-se eficaz mesmo em dias nublados.

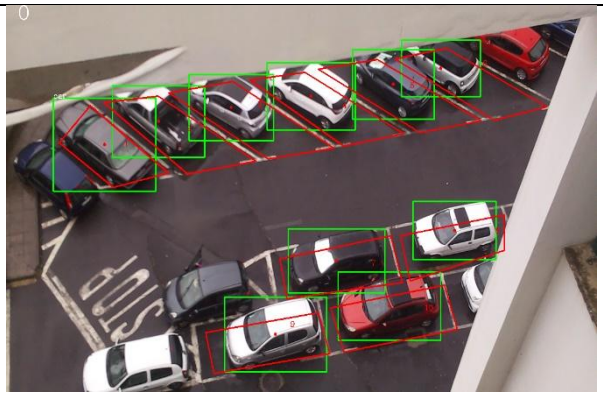
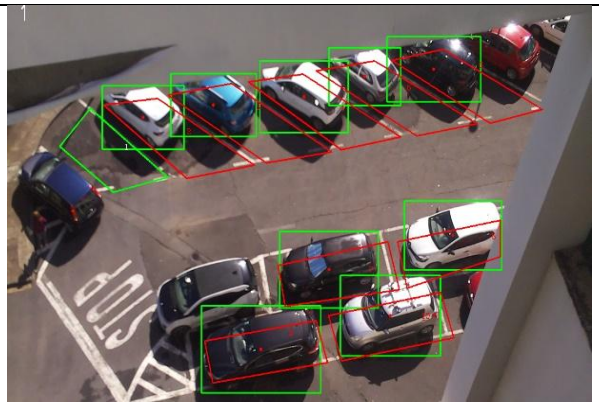
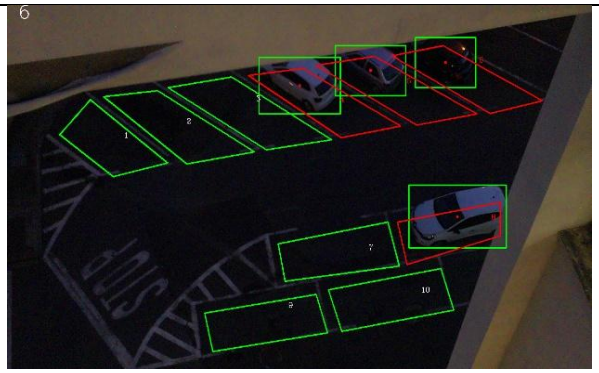
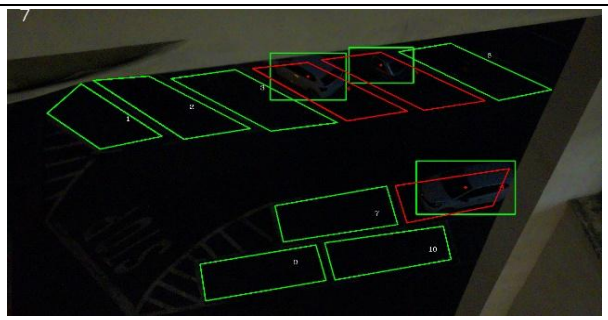
No segundo cenário, referente ao dia 02/04/2024 às 12h55, com muita luminosidade natural, foram identificados corretamente 9 veículos, também com uma precisão de 100%. Esta situação representa as condições ideais para o sistema, com elevado desempenho da deteção devido à abundância de luz solar.

À medida que o dia escureceu, observaram-se algumas limitações. No cenário das 20h30, com pouca luminosidade, foram detetados 4 veículos, mantendo-se a precisão de 100%. Este resultado mostra que o sistema ainda é funcional no início do período noturno, desde que exista alguma luz residual no ambiente.

Contudo, no cenário das 21h00, também com pouca luminosidade, apenas 3 veículos foram detetados e a precisão caiu para 75%. Este desempenho reduzido evidencia que, à medida que a noite avança e a iluminação natural desaparece, a performance da câmera V2 torna-se menos fiável, sendo afetada pela escassez de luz.

Em suma, conclui-se que o sistema baseado no *Raspberry Pi* 4 com câmera V2 apresenta excelente desempenho em períodos diurnos, inclusive sob condições atmosféricas adversas como a chuva. No entanto, à noite e em ambientes com fraca luminosidade, a sua eficácia é significativamente reduzida, o que destaca a necessidade de iluminação artificial adicional ou de uma câmera com melhor sensibilidade noturna para garantir resultados consistentes em todos os períodos.

Tabela 14- Imagens captadas no *Raspberry Pi 4.*

Data	Foto	Nº de veículos detetados nos lugares	Precisão de deteção
01/04/2024 12:53h Período da tarde com o clima chuvoso.		10	100%
02/04/ 2024 12:55h Período da tarde com muita luminosidade.		9	100%
02/04/2024 20:30h Período da noite com pouca luminosidade.		4	100%
02/04/2024 21:00h Período da noite com pouca luminosidade.		3	75%

4.5.2 Análise comparativa

A partir das tabelas apresentadas é possível comparar o desempenho do sistema de deteção de veículos em diferentes condições de iluminação utilizando duas configurações distintas de hardware:

a) Análise do *Raspberry Pi* 4 com câmera V2

Os resultados indicam que a deteção de veículos apresentou 100% de precisão durante o dia, tanto em condições de clima chuvoso quanto em dias ensolarados. No entanto, à medida que a iluminação diminui, o desempenho da deteção foi afetado de acordo as seguintes condições:

- Período noturno com pouca luminosidade (20:30h): A precisão ainda foi de 100%, mas com uma redução no número de veículos detetados;
- Período noturno com pouca luminosidade (21:00h): A precisão caiu para 75%, indicando que o sistema teve dificuldades em reconhecer todos os veículos corretamente.

b) Análise do *Raspberry Pi* 5 com câmera NoIR

No caso da câmera NoIR, que não possui filtro infravermelho, os testes mostraram um impacto significativo nas imagens capturadas durante a noite:

- Com iluminação suficiente: O sistema atingiu 100% de precisão, indicando que a câmera NoIR pode ser eficaz em ambientes bem iluminados;
- Com pouca iluminação: O desempenho caiu drasticamente para 0% de precisão em determinados testes, evidenciando que a ausência de luz afeta consideravelmente a capacidade de deteção.

O *Raspberry Pi* 4 com câmera V2 demonstrou um desempenho mais consistente em diferentes cenários, embora tenha perdido precisão à medida que a iluminação diminuiu. O *Raspberry Pi* 5 com câmera NoIR mostrou um bom desempenho quando há iluminação artificial, mas apresentou grande dificuldade em detetar veículos no escuro.

Portanto, para um sistema de monitoramento de estacionamento em ambientes de baixa luminosidade, o *Raspberry Pi* 4 com câmera V2 parece ser uma opção mais robusta. Caso a câmera NoIR seja utilizada, é essencial complementar o ambiente com fontes de luz artificial para garantir um desempenho satisfatório.

4.5.3 Falhas de deteção e de disponibilidade de lugares

Durante a fase de avaliação do desempenho do algoritmo em cada protótipo foram verificadas algumas falhas na deteção dos veículos, tal como é mostrado nas figuras 69 e 70.

Na figura 69, verifica-se que quando dois veículos são detetados e sobrepostos no mesmo lugar ocorre uma falha no algoritmo em decidir se o respetivo lugar está ocupado ou livre, sendo que no final acaba por decidir que o lugar está livre, o que não é verdade. A solução deste problema passa por reduzir a área de marcação dos lugares de estacionamento.

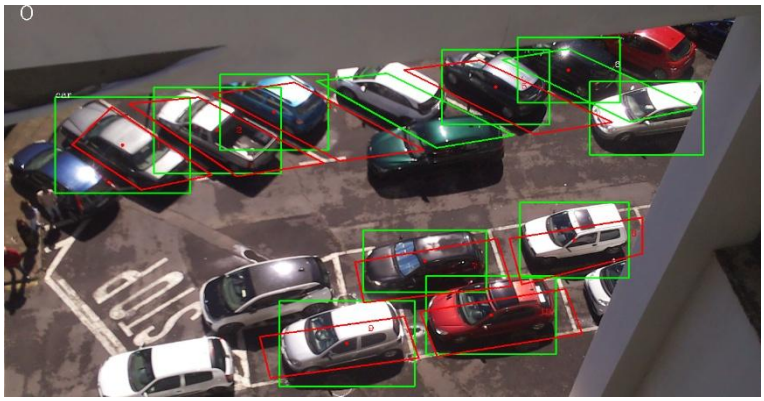


Figura 69- Resultados dois veículos sobrepostos no mesmo lugar.

Na figura 70, uma tampa de esgoto foi detetada com um veículo, verificando-se assim uma falha na parte de deteção, sendo que a solução deste problema passa por melhorar o modelo de treino, isto é, eliminar todas as marcações de objetos que não sejam veículos na fase de anotação.

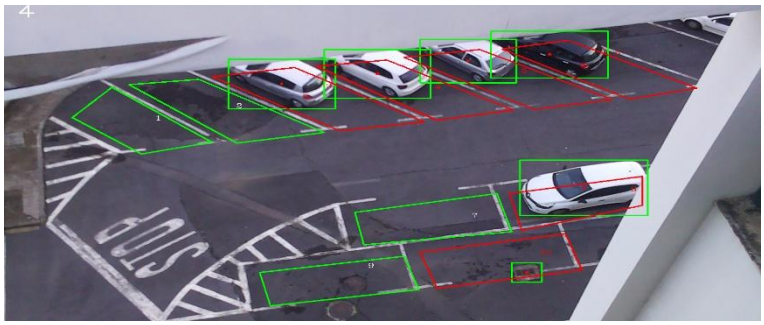


Figura 70- Uma tampa de esgoto detetada como um veículo.

Também se deu a situação em que a parte de deteção reconheceu um veículo a partir do reflexo de luz na porta de outro veículo (figura 68 da secção 4.5.1) fazendo com o algoritmo detetasse dois veículos sobre o mesmo lugar de estacionamento prejudicando a condição de disponibilidade da mesma.

4.6 Visualização dos lugares em tempo real

Nesta secção é mostrada a página *web* que apresentava em tempo real a disponibilidade dos lugares de estacionamento (figura 71). Os estados de cada lugar são guardados em variáveis no código *Python* e enviados para a base de dados num servidor local que atualiza continuamente a página *web*.

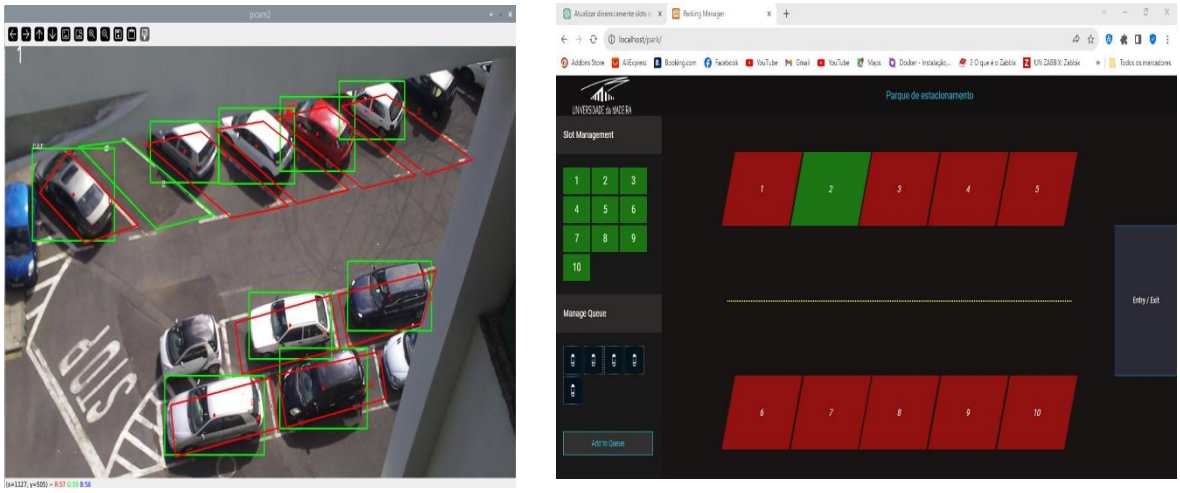


Figura 71- Visualização da disponibilidade dos lugares em tempo real.

A Figura 71 apresenta a visualização em tempo real da disponibilidade dos lugares de estacionamento, integrando o sistema de deteção baseado em visão computacional com a interface *web* desenvolvida para os utilizadores finais.

À esquerda, encontra-se a imagem captada pela câmera instalada no parque, processada pelo modelo treinado 1, sendo cada lugar delimitado por uma caixa colorida:

- Verde: representa um lugar livre;
- Vermelho: representa um lugar ocupado.

Este sistema realiza a deteção automática dos veículos e associa cada lugar ao respetivo estado de ocupação. Na parte inferior da imagem processada, surgem informações técnicas relacionadas com o ponto onde o cursor do rato está posicionado. Especificamente, são exibidas as coordenadas (x, y) da imagem, bem como os respetivos valores de intensidade RGB (vermelho, verde, azul). Estes dados são relevantes durante a calibração e validação do sistema, uma vez que permitem avaliar a qualidade da iluminação e a resposta da câmera em diferentes condições ambientais. No exemplo apresentado, o ponto (526, 665) apresenta os

valores R: 228, G: 229, B: 235, o que indica uma área com elevada luminosidade e tons claros, útil para aferir o contraste necessário para uma deteção precisa.

À direita, está representada a interface da aplicação *web*, acessível aos utilizadores, que apresenta o estado de cada lugar de estacionamento em tempo real. A disposição dos retângulos reflete a organização física dos lugares no local. Cada lugar é identificado por um número e uma caixa colorida conforme o seu estado atual.

Esta informação é continuamente atualizada a partir do sistema de processamento de imagem desenvolvido em *Python*, que monitora o estado dos lugares como variáveis internas. Estas variáveis são sincronizadas com uma base de dados alojada num servidor local, garantindo a comunicação em tempo real com a interface *web*. Assim, os utilizadores podem aceder à informação de disponibilidade de forma rápida e intuitiva, contribuindo para uma gestão mais eficiente do parque de estacionamento.

De forma geral, os resultados obtidos relativamente à deteção dos veículos nos lugares de estacionamento e à disponibilização, em tempo real, do estado de ocupação de cada lugar foram satisfatórios, tendo o sistema alcançado uma precisão média de deteção de aproximadamente 93,6%.

4.7 Desempenho do modelo YOLO

Analisando a matriz de correlação apresentada na Figura 72, observa-se um elevado número de Verdadeiros Positivos (357), o que indica que o modelo YOLO conseguiu detetar corretamente a maioria dos veículos presentes nas imagens anotadas. Verifica-se também uma quantidade muito reduzida de Falsos Negativos (37) e Falsos Positivos (3) e Verdadeiros Negativos (0), o que evidencia que o modelo raramente deixou de detetar um veículo ou detetou erroneamente onde não existia nenhum. Estes resultados refletem uma elevada eficácia do sistema de deteção, com uma taxa de precisão, *recall* e *Accuracy* (Exatidão) próximas dos 100%.

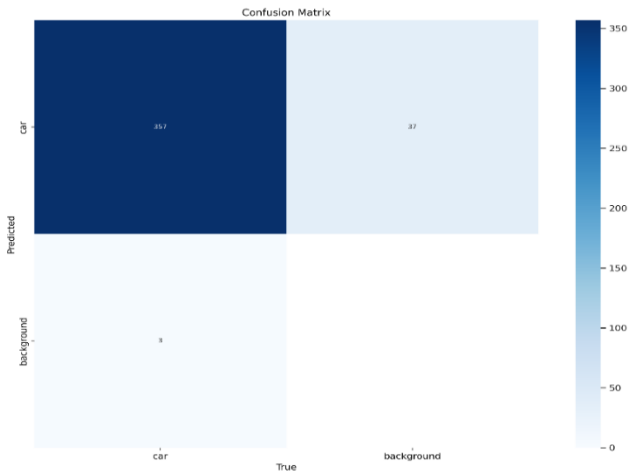


Figura 72- Matriz de correlação.

Através dos dados da matriz de correlação são calculadas as métricas da classe ***'car'*** com os seguintes resultados:

$$\text{Precisão} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{357}{357 + 3} = 0{,}99$$

$$\mathit{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{357}{357 + 37} = 0{,}91$$

$$\text{Exatidão} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} = \frac{357 + 0}{357 + 0 + 3 + 37} = 0{,}89$$

Com base nos resultados obtidos durante os testes de deteção de veículos, foi possível verificar que o sistema apresentou um desempenho global satisfatório. A análise dos valores de permitiu calcular uma exatidão aproximada de 89,92%.

Este valor demonstra que, na maioria das situações, o sistema é capaz de identificar corretamente os veículos presentes nos lugares de estacionamento. No entanto, a ausência de Verdadeiros Negativos (TN = 0) indica que o conjunto de testes considerou apenas situações com ocupação de lugares, o que limita a avaliação da capacidade do sistema em reconhecer corretamente lugares livres. Apesar disso, o elevado número de deteções corretas e a baixa taxa de falsos positivos revelam que o modelo apresenta uma elevada fiabilidade na tarefa proposta, sendo adequado para aplicações em ambientes reais.

5. Conclusões

Neste capítulo apresentam-se as conclusões gerais do trabalho desenvolvido, estabelecendo uma ligação direta com os objetivos propostos no início deste projeto de mestrado. Adicionalmente, são sugeridas propostas de trabalhos futuros que poderão servir como base para evoluções ou complementações deste sistema.

5.1 Conclusões gerais

O principal objetivo deste trabalho consistiu no desenvolvimento de um sistema inteligente de deteção de lugares de estacionamento, baseado em processamento de imagem, com disponibilização da informação em tempo real. Para tal, foram definidos e cumpridos vários objetivos específicos, distribuídos ao longo dos capítulos deste projeto.

Inicialmente, foi realizado um estudo aprofundado sobre técnicas de *machine learning* aplicadas à deteção de veículos, com ênfase na utilização do modelo YOLOv8. Foram treinados diferentes modelos de deteção de objetos, sendo analisados diversos cenários, com e sem aplicação de técnicas de aumento de dados, a fim de avaliar o impacto dessas abordagens na precisão da deteção.

Posteriormente, o modelo treinado foi integrado num algoritmo de tomada de decisão responsável por avaliar a disponibilidade de lugares de estacionamento. Esta integração permitiu não apenas identificar os veículos presentes, mas também inferir o estado de ocupação de cada lugar, de forma automática.

De seguida, foi desenvolvida uma página *web* para a disponibilização da informação em tempo real. Esta funcionalidade permite aos utilizadores acederem, de forma simples e intuitiva, à ocupação atual dos lugares de estacionamento.

Além disso, foi concebido e testado um protótipo físico com recurso ao *Raspberry Pi*, associado a diferentes módulos de câmera, de forma a validar a robustez do sistema em condições reais. Os testes demonstraram que o sistema atinge uma taxa de precisão média de 89,92% na deteção de veículos, sendo esta percentagem obtida a partir de um total de 397 previsões corretas (357 Verdadeiros Positivos e 37 Verdadeiros Negativos) em 460 tentativas, o que confirma a viabilidade prática da solução desenvolvida.

Por fim, foram feitas simulações e recolha de resultados experimentais em cenários distintos, permitindo concluir que a performance do sistema varia de acordo com as condições ambientais, a iluminação e o tipo de câmera utilizada.

5.2 Proposta de trabalhos futuros

Tendo por base o trabalho realizado, propõem-se algumas linhas de continuidade que poderão servir de ponto de partida para investigações futuras.

Uma possível evolução será a incorporação de algoritmos de compensação automática de deslocamento da câmera, com o objetivo de corrigir problemas verificados nos testes em campo, nomeadamente os desvios provocados por ventos fortes. Esta compensação poderá basear-se na análise automática de pontos de interesse e no ajuste dinâmico das regiões de interesse no código *Python*.

Outra proposta consiste na adaptação do sistema para diferentes ambientes urbanos, permitindo a deteção de lugares de estacionamento em ruas públicas, parques subterrâneos ou zonas comerciais, ajustando o modelo às diversas condições de iluminação e variação de ângulos de visão.

Adicionalmente, poderá ser interessante investigar o uso de técnicas de fusão sensorial, combinando o processamento de imagem com sensores ultrassónicos ou LIDAR, de forma a melhorar a precisão em condições adversas, como nevoeiro ou à noite.

Sugere-se também o desenvolvimento de uma aplicação móvel que complemente a página *web*, fornecendo notificações em tempo real aos utilizadores sobre a disponibilidade de lugares nas proximidades, com integração de rotas até ao local.

Por fim, uma proposta relevante será a realização de testes em larga escala com múltiplas câmeras e locais distintos, de modo a avaliar a escalabilidade do sistema, a sua fiabilidade a longo prazo e os requisitos computacionais associados à sua expansão.

6. Referências bibliográficas

[1] Oliveira, F. M. G. de. (2019). Gestão inteligente de estacionamento em ambiente urbano (Dissertação de mestrado). Universidade do Minho. Disponível em: https://repositorium.sdum.uminho.pt/bitstream/1822/64531/1/Filipe-Manuel-Gon%C3%A7alves-de-Oliveira-disserta%C3%A7%C3%A3o.pdf.

[2] I. Haritaoglu, D. Harwood, and L. S. Davis, “W4: Real-time surveillance of people and their activities,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 8, pp. 809–830, 2000.

[3] C. Stauffer and W. Grimson, “Learning Patterns of Activity Using Real-Time Tracking,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 8, pp. 747–757, 2000.

[4] T. E. Boult et al., “Into the woods: Visual surveillance of noncooperative and camouflaged targets in complex outdoor settings,” Proc. IEEE, vol. 89, no. 10, pp. 1382–1401, 2001.

[5] SHALEV-SHWARTZ, Shai; BEN-DAVID, Shai. *Understanding Machine Learning: From Theory to Algorithms*. 1. ed. Cambridge: Cambridge University Press, 2014.

[6] Wang, H. and He, W. (2011). A Reservation-based Smart Parking System. University of Nebraska-Lincoln.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neuronal Networks,” in Advances in Neuronal Information Processing Systems, vol. 25, 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[8] Z.-Q. Zhao, P. Zheng, S.-T. Xu, e X. Wu, “Object Detection with Deep Learning: A Review,” *IEEE Transactions on Neuronal Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

[9] DESHPANDE, Adit. "A Beginner's Guide to Understanding CNNs". Disponível em: https://adeshpande3.github.io. Consultado em: 25 de abril de 2025.

[10] ÜNAL, Mehmet. "Show Images Directly on Terminal: img2sh". A blog from Mehmet Ozan Ünal, 3 nov. 2019. Disponível em: https://mozanunal.com/2019/11/img2sh/. Consultado em: 25 de abril de 2025.

[11] Y. LeCun, L. Bottou, Y. Bengio, e P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] HEINRICH, Greg. "Image Segmentation Using DIGITS 5". NVIDIA Developer Blog, 2016. Disponível em: https://developer.nvidia.com/blog/image-segmentation-using-digits-5/. Consultado em: 25 de abril de 2025.

[13] ARNAB, Anurag *et al*. Conditional Random Fields Meet Deep Neuronal Networks for Semantic Segmentation: Combining Probabilistic Graphical Models with Deep Learning for Structured Prediction. IEEE Signal Processing Magazine. v. 35. n. 1 p. 37-52, jan. 2018.

[14] J. S. Denker e Y. LeCun, “Transforming Neuronal-Net Output Levels to Probability Distributions,” in *Advances in Neuronal Information Processing Systems*, vol. 3, 1990.

[15] J. S. Denker e Y. LeCun, “Transforming Neuronal-Net Output Levels to Probability Distributions,” in *Advances in Neuronal Information Processing Systems 3 (NIPS 1990)*, R. Lippmann, J. Moody e D. Touretzky, Eds. Denver, CO: Morgan Kaufmann, 1991, pp. 853–859.

[16] RIZWAN, Muhammad. "Convolutional Neuronal Networks – In a Nut Shell". engMRK, 17 set. 2018. Disponível em: https://engmrk.com/convolutional-neuronal-network-3/. Consultado em: 25 de abril de 2025.

[17] GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning. MIT Press. 2016.

[18] W. Luo, Y. Li, R. Urtasun, e R. Zemel, “Understanding the Effective Receptive Field in Deep Convolutional Neuronal Networks,” *arXiv preprint arXiv:1701.04128*, 2017.

[19] W. Luo, Y. Li, R. Urtasun, e R. Zemel, “Understanding the Effective Receptive Field in Deep Convolutional Neuronal Networks,” *arXiv preprint arXiv:1701.04128*, 2017.

[20] V. Dumoulin e F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.

[21] J. Brownlee, “A Gentle Introduction to Padding and Stride for Convolutional Neuronal Networks,” *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neuronal-networks/.

[22] V. Nair e G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[23] JORDAN, Jeremy. "Convolutional Neuronal Networks". Jeremy Jordan, 16 jul. 2017. Disponível em: https://www.jeremyjordan.me/convolutional-neuronal-networks/. Consultado em: 25 de abril de 2025.

[24] A. Ng, "Convolutional Neuronal Networks," *Deep Learning Specialization*, Coursera, 2017.

[25] Goodfellow, Y. Bengio e A. Courville, *Deep Learning*, Cambridge, MA: MIT Press, 2016.

[26] HINTON, Geoffrey; OSINDERO, Simon; TEH, Yee-Whye. A fast learning algorithm for deep belief nets. Neuronal Computation. v. 18. n. 7. p. 1527-1554. jun. 2006.

[27] GÖRNER, Martin. Modern Convolutional Neuronal Nets. 2018. Apresentação de slides. Disponível em: https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd. Consultado em: 25 de abril de 2025.

[28] ANIEMEKA, Ifu. "A Friendly Introduction to Convolutional Neuronal Networks". Hashrocket Blog, 22 ago. 2017. Disponível em: https://hashrocket.com/blog/posts/a-friendly-introduction-to-convolutional-neuronal-networks. Consultado em: 25 de abril de 2025.

[29] RANZATO, Marc’Aurelio *et al*. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. 2007 IEEE Conference on Computer Vision and Pattern Recognition. Mineápolis. p. 1-8. 2007.

[30] A. Karpathy, J. Johnson e F. Li, *CS231n: Convolutional Neuronal Networks for Visual Recognition*, Stanford University, 2016.

[31] S. S. Shanmugam, *Machine Learning for Computer Vision*. Cham, Suíça: Springer, 2021.

[32] S. Ren, K. He, R. Girshick e J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, n.º 6, pp. 1137–1149, 2017.

[33] R. Girshick, J. Donahue, T. Darrell e J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 580–587, 2014.

[34] J. Redmon, S. Divvala, R. Girshick e A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 779–788, 2016.

[35] C.-Y. Wang, A. Bochkovskiy e H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 7464–7475, 2023.

[36] U. Handalage and L. Kuganandamurthy, Real-Time Object Detection Using YOLO: A Review, preprint, May 2021. [Online]. Available: https://www.researchgate.net/publication/351411017_RealTime_Object_Detection_Using_YOLO_A_Review.

[37] Deep Learning Bible – YOLO V1. [Em linha]. Disponível em: https://wikidocs.net/177706. [Consultado em: abril 2025].

[38] A. Neubeck e L. Van Gool, "Efficient Non-Maximum Suppression," *Proceedings of the 18th International Conference on Pattern Recognition (ICPR)*, 2006, pp. 850–855.

[39] J. Redmon, S. Divvala, R. Girshick e A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Em linha]. Disponível em: https://doi.org/10.1109/CVPR.2016.91 [Consultado em: abril 2025].

[40] U. Handalage e L. Kuganandamurthy, *Real-Time Object Detection Using YOLO: A Review*, Preprint, 2021. Disponível em:

https://www.researchgate.net/publication/351411017_RealTime_Object_Detection_Using_YOLO_A_Review [Consultado em: abril 2025].

[41] Visão Computacional – Deep Learning. Disponível em: https://lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deteccao-de-objetos-em-imagens [Consultado em: abril 2025].

[42] Deep Learning Bible – YOLO V1. Disponível em: https:// wikidocs.net/167699. [Consultado em: abril 2025].

[43] Deep Learning Bible – YOLO V1. Disponível em: https:// wikidocs.net/167690. [Consultado em: abril 2025].

[44] J. R. Treven e D. M. Cordova-Esparaza, Uma revisão abrangente do YOLO: De YOLOv1 a YOLOv8 e além, 2023.[Online]. Disponível em: https://arxiv.org/pdf/2304.00501.pdf.

[45] SOLAWETZ, J.; F., [Nome completo de F.]. "What is YOLOv8? The Ultimate Guide". 30 abr. 2023. Disponível em: https://blog.roboflow.com/whats-new-in-yolov8/. Consultado em: 25 de abril de 2025.

[46] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye e D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," arXiv, 19 de novembro de 2019. Disponível em: https://arxiv.org/abs/1911.08287.

[47] J. Terven, D. M. Córdova-Esparza e J. A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," Machine Learning and Knowledge Extraction, vol. 5, pp. 1680–1716, 2023. [Online]. Disponível em: https://doi.org/10.3390/make5040083.

[48] A. J. Zimbico, "Análise comparativa de técnicas de compressão aplicadas a imagens médicas usando ultrassom," Dissertação de Mestrado em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2014. Disponível em: http://repositorio.utfpr.edu.br/jspui/handle/1/820.

[49] G. P. C. P. da Luz, G. M. Sato, L. F. G. Gonzalez e J. F. Borin, "Smart Parking with Pixel-Wise ROI Selection for Vehicle Detection Using YOLOv8, YOLOv9,

YOLOv10, and YOLOv11," Machine Learning and Knowledge Extraction, vol. 5, pp. 1680–1716, 2023. [Online]. Disponível em: https://doi.org/10.3390/make5040083.

[50] R. C. Gonzalez e R. E. Woods, Digital Image Processing, 4.ª ed., Pearson, 2018.

[51] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn e A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," International Journal of Computer Vision, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[52] J. D. Foley, A. van Dam, S. K. Feiner e J. F. Hughes, Computer Graphics: Principles and Practice, 2.ª ed., Addison-Wesley, 1995.

[53] Raspberry Pi Ltd., "Raspberry Pi 4 Model B Specifications," [Online]. Disponível em: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Consultado em: abril 2025].

[54] RASPBERRY PI LTD. "Raspberry Pi 4 Model B Specifications".Disponível em: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.
[Consultado em: abril 2025].

[55] Raspberry Pi Ltd., "Raspberry Pi OS (Raspbian) – Bullseye and Bookworm," [Online]. Disponível em: https://www.raspberrypi.com/software/. [Consultado em: abril 2025].

[56] OpenCV Team, "About OpenCV," [Online]. Disponível em: https://opencv.org/about/. [Consultado em: abril 2025].

[57] Ultralytics, "YOLO by Ultralytics Documentation," [Online]. Disponível em: https://docs.ultralytics.com/. [Consultado em: abril 2025].

[58] Raspberry Pi Ltd., "Raspberry Pi Imager," [Online]. Disponível em: https://www.raspberrypi.com/software/. [Consultado em: abril 2025].

[59] RealVNC Ltd., "RealVNC – Remote Access Software," [Online]. Disponível em: https://www.realvnc.com/. [Consultado em: abril 2025].

[60] Roboflow, "Roboflow: Give your software the sense of sight," [Online]. Disponível em: https://roboflow.com/. [Consultado em: abril 2025].

[61] D. P. Kingma e J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, 2014. Disponível em: https://arxiv.org/abs/1412.6980. [Consultado em: abril 2025].

[62] Z. Karimi, "Confusion Matrix," ResearchGate, 2021. Disponível em: https://www.researchgate.net/publication/355096788_Confusion_Matrix. [Consultado em: abril 2025].

[63] Thonny, "Python IDE for beginners," [Online]. Disponível em: https://thonny.org/. [Consultado em: abril 2025].

[64] MySQL, "The world's most popular open source database," [Online]. Disponível em: https://www.mysql.com/. [Consultado em: abril 2025].

[65] Mozilla Developer Network (MDN), "AJAX - Asynchronous JavaScript and XML," [Online]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX. [Consultado em: abril 2025].

[66] Apache Friends, "XAMPP: Apache + MariaDB + PHP + Perl," [Online]. Disponível em: https://www.apachefriends.org/. [Consultado em: abril 2025].

[67] Ultralytics, "YOLOv8 Documentation," [Online]. Disponível em: https://docs.ultralytics.com/. [Consultado em: abril 2025].

[68] J. Redmon, S. Divvala, R. Girshick e A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.

Anexos

Anexo A – Implementação do algoritmo do raio em *Python*.

```
def is_point_in_polygon(cx, cy, polygon):
    # Inicializar a contagem de interseções
    intersections = 0
    # Número de vértices no polígono
    n = len(polygon)
    for i in range(n):
        # Pegar o vértice atual e o próximo vértice (fechando o polígono)
        x1, y1 = polygon[i]
        x2, y2 = polygon[(i + 1) % n]
        # Verificar se o ponto está entre as coordenadas Y da aresta
        if (y1 <= cy < y2) or (y2 <= cy < y1):
            # Calcular a coordenada X do ponto de interseção
            intersect_x = x1 + (cy - y1) * (x2 - x1) / (y2 - y1)
            # Verificar se a interseção está à direita do ponto
            if intersect_x >= cx:
                intersections += 1
    # Se o número de interseções for ímpar, o ponto está dentro do polígono
    return intersections % 2 == 1



# Exemplo de uso
polygon = [(172, 275), (279, 350), (396, 320), (241, 217)]
point = (300, 300)
print(is_point_in_polygon(point[0], point[1], polygon))  # Retorna True ou False
```

Anexo B – Código Python para atualização da base de dados.

```
import sys

import datetime

import mysql.connector

# Conectar ao banco de dados

db_connection = mysql.connector.connect(

        host=”seu_host”

        user=”seu_user”

        password=”sua_senha”

        database=”seu_banco_de_dados”

)

db_cursor = db_connection.cursor()

# Atualize o estado das vagas no banco de dados

For i, space_count in enumerate([a1, a2, a3, a4, a5, a6, a7, a8, a9, a10], start=1):

        space_occupied = 1 if space_count == 1 else 0

        db_cursor.execute(“INSERT INTO parking_spaces (space_number, occupied) VALUES (%s, %s)”,(I, space_occupied))

        db_connection.commit()
```

Anexo C – Código PhP da página *web*.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Parking Manager</title>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@100;400&display=swap" rel="stylesheet" />
    <link rel="stylesheet" href="parking.css" />
    <script src="parking.js"></script>
</head>
<body onload="setupparkingmanager()">
    <div class="navbar">
        <div class="brand"> <img src="Logo_UMa.png" width="130" height="60"></div>
        <div class="texto"> Parque de estacionamento </div>
    </div>
    <div class="dashboard">
        <div class="parking-manager">

            <div class="new-car-entry" id="parkingSlots">
                <!-- Lugares de estacionamento serão adicionadas aqui -->
            </div>
            <div class="new-car-btn">Manage Queue</div>
            <div class="queue">
                <?php
                // PHP loop to generate queue images
                for ($i = 1; $i <= 5; $i++) {
                    echo "<img id=\"queue$i\" src=\"carfaded.png\"></img>";
                }
                ?>
            </div>
            <div class="addtoqueue" onclick="addtoqueue()">Add to Queue</div>
        </div>
        <div class="parking-container" id="parkingspace">
            <div class="parking-slots-holder">
                <?php
                // PHP loop to generate parking slots (first row)
```

```
            for ($i = 1; $i <= 5; $i++) {
                echo "<div class=\"parking-slot\" id=\"slot-$i\">$i</div>";
            }
            ?>
        </div>
        <div class="parking-way" id="entry-way"></div>
        <div class="parking-way"></div>
        <div class="parking-slots-holder">
            <?php
            // PHP loop to generate parking slots (second row)
            for ($i = 6; $i <= 10; $i++) {
                echo "<div class=\"parking-slot\" id=\"slot-$i\">$i</div>";
            }
            ?>
        </div>
    </div>
    <div class="outside">Entry / Exit</div>
  </div>
</body>
</html>
```

Anexo D – Código *AJAX* para atualização da página em tempo real.

```
<?php
// Conexão com o banco de dados
$conn = new mysqli("10.1.215.139", "root", "1234", "park");

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Consulta para obter o estado atual das lugares de estacionamento
$sql = "SELECT space_id, occupied FROM parking_spaces";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // Array para armazenar os dados das lugares de estacionamento
    $parklist = array();

    // Percorre os resultados da consulta e armazena os dados no array
    while ($row = $result->fetch_assoc()) {
        $parklist[$row["space_id"]] = $row["occupied"];
    }

    // Retorna os dados como JSON
    echo json_encode($parklist);
} else {
    echo "0 results";
}
$conn->close();
?>
```

Anexo E – Código *javascript.*

```
var w, h;
var parklock = false;
var parklist = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
var queueitems = 0; // Inicialmente é 0

function setupparkingmanager() {
    w = document.getElementById('parkingspace').offsetWidth;
    h = document.getElementById('parkingspace').offsetHeight;

    // Criando as animações - parte importante
    var anim = document.createElement('style');
    var rule1 = document.createTextNode('@-webkit-keyframes car-park {' +
        'from { transform: rotate(270deg) }' +
        '80% { transform: rotate(270deg) translate(0px,-' + w + 'px) }' +
        '90% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) }' +
        'to { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,-' + h * .25 + 'px)}' +
        '}');
    anim.appendChild(rule1);
    var rule2 = document.createTextNode('@-webkit-keyframes car-bottom {' +
        'from { transform: rotate(270deg) }' +
        '80% { transform: rotate(270deg) translate(0px,-' + w + 'px) }' +
        '90% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) }' +
        'to { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,' + h * .25 + 'px)}' +
        '}');
    anim.appendChild(rule2);
    var rule3 = document.createTextNode('@-webkit-keyframes car-exit-top {' +
        'from { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,-' + h * .25 + 'px) }'
+
        '80% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,-' + h * .25 + 'px)
translate(0px,' + h * .25 + 'px)}' +
        '90% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,-' + h * .25 + 'px)
translate(0px,' + h * .25 + 'px) rotate(90deg)}' +
        'to { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,-' + h * .25 + 'px)
translate(0px,' + h * .25 + 'px) rotate(90deg) translate(0px,-' + w + 'px)}' +
        '}');
    anim.appendChild(rule3);
    var rule4 = document.createTextNode('@-webkit-keyframes car-exit-bottom {' +
        'from { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,' + h * .25 + 'px) }'
+
```

```
        '80% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,' + h * .25 + 'px) translate(0px,-' + h * .25 + 'px)}' +
        '90% { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,' + h * .25 + 'px) translate(0px,-' + h * .25 + 'px) rotate(90deg)}' +
        'to { transform: rotate(270deg) translate(0px,-' + w + 'px) rotate(90deg) translate(0px,' + h * .25 + 'px) translate(0px,-' + h * .25 + 'px) rotate(90deg) translate(0px,-' + w + 'px)}' +
        '}');
    anim.appendChild(rule4);
    document.getElementById('parkingspace').appendChild(anim);

    // Iniciar a atualização contínua dos dados
    setInterval(updateParkingData, 5000); // Atualiza a cada 5 segundos (5000 milissegundos)
}

function updateParkingData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var parklist = JSON.parse(this.responseText);
            updateParkingSlots(parklist);
        }
    };
    xhttp.open("GET", "get_parking_data.php", true);
    xhttp.send();
}

function updateParkingSlots(parklist) {
    // Atualize as lugares de estacionamento com base nos dados recebidos do servidor
    for (var slot_id in parklist) {
        var occupied = parklist[slot_id];
        var slotElement = document.getElementById('slot-' + slot_id);
        if (occupied == 1) {
            slotElement.style.backgroundColor = 'rgb(146, 18, 18)'; // Slot ocupado
        } else {
            slotElement.style.backgroundColor = 'rgb(27, 118, 19)'; // Slot livre
        }
    }
}
```

```
function addtoqueue() {
    var freeslotflag = 0;
    for (var j = 0; j < 10; j++) {
        if (parklist[j] != 1) {
            freeslotflag = 1;
            alert("Free slots available");
            break;
        }
    }
    if (freeslotflag != 1) {
        queueitems = queueitems + 1;
        if (queueitems > 5)
            alert("Queue Limit Reached");
        else
            updatequeue();
    }
}

function updatequeue() {
    for (var i = 1; i <= 5; i++) {
        if (i <= queueitems) {
            document.getElementById('queue' + i.toString()).src = 'car.png';
        } else {
            document.getElementById('queue' + i.toString()).src = 'carfaded.png';
        }
    }
}

function queuecheck(slot) {
    if (queueitems > 0) {
        queueitems = queueitems - 1;
        updatequeue();
        carenter(slot);
    }
}

function carexit(slot) {
    if (!parklock) {
```

```
        parklist[slot] = 0;
        console.log(parklist);
        parklock = true;
        document.getElementById('slot' + (slot + 1).toString()).style.background = 'rgb(27,118,19)';
        if (slot <= 4)
            document.getElementById('car' + (slot).toString()).style.animation = 'car-exit-top 2s both';
        else
            document.getElementById('car' + (slot).toString()).style.animation = 'car-exit-bottom 2s both';
        setTimeout(function() {
            document.getElementById('car' + (slot).toString()).remove();
            parklock = false;
            queuecheck(slot);
        }, 2000);
    }
}

function generatenewcar(slot) {
    var space = document.getElementById('parkingspace');
    let img = document.createElement('img');
    img.src = 'car.png';
    img.className = 'new-car-origin';
    img.style.width = (w * .8) * .1 + 'px';
    img.id = 'car' + slot.toString();
    space.appendChild(img);
}



function carenter(slot) {
    if (!document.getElementById('car' + (slot).toString()) && !parklock) {
        parklist[slot] = 1;
        console.log(parklist);
        parklock = true;
        generatenewcar(slot);
        document.getElementById('slot' + (slot + 1).toString()).style.background = 'rgb(146,18,18)';
        if (slot != 4 && slot != 9)
            document.getElementById('car' + (slot).toString()).style.right = (-w + (w * .1) + (((5 - (slot + 1) % 5)) * ((w
* .8) * .2)) + ((w * .8) * .05)) + 'px';
        else
            document.getElementById('car' + (slot).toString()).style.right = (-w + (w * .1) + ((w * .8) * .05)) + 'px';
```

```
        if (slot <= 4)
            document.getElementById('car' + (slot).toString()).style.animation = 'car-park 2s both';
        else
            document.getElementById('car' + (slot).toString()).style.animation = 'car-bottom 2s both';
        setTimeout(function() {
            parklock = false;
        }, 2000);
    } else {
        carexit(slot);
    }
}
```

Anexo F - Código *Python* para mapeamentos dos lugares.

```
import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time
model=YOLO('yolov8n.pt')
def RGB(event, x, y, flags, param):
if event == cv2.EVENT_MOUSEMOVE :
colorsBGR = [x, y]
print(colorsBGR)

cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)
image = cv2.imread('rasp51.jpg')
my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")

area0=[(0,0),(0,0),(0,0),(0,0)]
while True:
frame  = image.copy()

frame=cv2.resize(frame,(1280,720))
results=model.predict(frame)
a=results[0].boxes.data
px=pd.DataFrame(a).astype("float")
list0=[]
for index,row in px.iterrows():
x1=int(row[0])
y1=int(row[1])
x2=int(row[2])
y2=int(row[3])
d=int(row[5])
c=class_list[d]
if 'car' in c:
cx=int(x1+x2)//2
```

```
cy=int(y1+y2)//2
results0=cv2.pointPolygonTest(np.array(area0,np.int32),((cx,cy)),False)
if results0>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list0.append(c)
a0=(len(list9)
if a0==1:
cv2.polylines(frame,[np.array(area0,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('0'),(591,398),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area0,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('9'),(591,398),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)

cv2.imshow("RGB", frame)

if cv2.waitKey(0)&0xFF==27:
break
cap.release()
cv2.destroyAllWindows()
#stream.stop()
```

Anexo G – Código para implementação real usando o *Raspberry Pi.*

```
import sys

import datetime

import cv2

from picamera2 import Picamera2

import pandas as pd

from ultralytics import YOLO

import mysql.connector

import cvzone

import numpy as np


# Estabelecer conexão com o banco de dados

db_connection = mysql.connector.connect(

host="10.1.215.139",

user="root",

password="1234",

database="park")

db_cursor = db_connection.cursor()

picam2 = Picamera2()

picam2.preview_configuration.main.size = (1280,720)

picam2.preview_configuration.main.format = "RGB888"

picam2.preview_configuration.align()

picam2.configure("preview")

picam2.start()

model=YOLO('best.pt')

my_file = open("coco.txt", "r")

data = my_file.read()

class_list = data.split("\n")
```

```
area1=[(86,244),(159,175),(333,298),(200,328)]
area2=[(185,167),(305,157),(523,280),(380,305)]
area3=[(352,163),(567,286),(710,266),(491,142)]
area4=[(528,139),(731,262),(861,239),(647,125)]
area5=[(676,119),(804,101),(1028,215),(896,237)]
area6=[(842,100),(1057,214),(1171,189),(954,80)]
area7=[(594,546),(576,461),(816,415),(840,505)]
area8=[(835,414),(860,501),(1045,461),(1080,374)]
area9=[(415,598),(664,553),(686,636),(427,684)]
area10=[(683,548),(705,635),(955,587),(938,509)]
count=0
while True:
frame= picam2.capture_array()
count += 1
if count % 3 != 0:
continue
frame=cv2.flip(frame,-1)
results=model.predict(frame)
a=results[0].boxes.data
px=pd.DataFrame(a).astype("float")
#    print(px)
list1=[]
list2=[]
list3=[]
list4=[]
list5=[]
list6=[]
list7=[]
list8=[]
list9=[]
```

```
list10=[]
for index,row in px.iterrows():
#       print(row)
x1=int(row[0])
y1=int(row[1])
x2=int(row[2])
y2=int(row[3])
d=int(row[5])
c=class_list[d]
if 'car' in c:
cx=int(x1+x2)//2
cy=int(y1+y2)//2
results1=cv2.pointPolygonTest(np.array(area1,np.int32),((cx,cy)),False)
if results1>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list1.append(c)
cv2.putText(frame,str(c),(x1,y1),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
results2=cv2.pointPolygonTest(np.array(area2,np.int32),((cx,cy)),False)
if results2>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list2.append(c)
results3=cv2.pointPolygonTest(np.array(area3,np.int32),((cx,cy)),False)
if results3>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list3.append(c)
results4=cv2.pointPolygonTest(np.array(area4,np.int32),((cx,cy)),False)
if results4>=0:
```

```
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list4.append(c)
results5=cv2.pointPolygonTest(np.array(area5,np.int32),((cx,cy)),False)
if results5>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list5.append(c)
results6=cv2.pointPolygonTest(np.array(area6,np.int32),((cx,cy)),False)
if results6>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list6.append(c)
results7=cv2.pointPolygonTest(np.array(area7,np.int32),((cx,cy)),False)
if results7>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list7.append(c)
results8=cv2.pointPolygonTest(np.array(area8,np.int32),((cx,cy)),False)
if results8>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list8.append(c)
results9=cv2.pointPolygonTest(np.array(area9,np.int32),((cx,cy)),False)
if results9>=0:
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)
cv2.circle(frame,(cx,cy),3,(0,0,255),-1)
list9.append(c)
results10=cv2.pointPolygonTest(np.array(area10,np.int32),((cx,cy)),False)
if results10>=0:
```

```
cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)

cv2.circle(frame,(cx,cy),3,(0,0,255),-1)

list10.append(c)

a1=(len(list1))

a2=(len(list2))

a3=(len(list3))

a4=(len(list4))

a5=(len(list5))

a6=(len(list6))

a7=(len(list7))

a8=(len(list8))

a9=(len(list9))

a10=(len(list10))

o=(a1+a2+a3+a4+a5+a6+a7+a8+a9+a10)

space=(10-o)

print(space)

if a1==1:

cv2.polylines(frame,[np.array(area1,np.int32)],True,(0,0,255),2)

cv2.putText(frame,str('1'),(255,262),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)

else:

cv2.polylines(frame,[np.array(area1,np.int32)],True,(0,255,0),2)

cv2.putText(frame,str('1'),(255,262),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)

if a2==1:

cv2.polylines(frame,[np.array(area2,np.int32)],True,(0,0,255),2)

cv2.putText(frame,str('2'),(392,234),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)

else:

cv2.polylines(frame,[np.array(area2,np.int32)],True,(0,255,0),2)

cv2.putText(frame,str('2'),(392,234),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)

if a3==1:

cv2.polylines(frame,[np.array(area3,np.int32)],True,(0,0,255),2)
```

```
cv2.putText(frame,str('3'),(544,189),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area3,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('3'),(544,189),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a4==1:
cv2.polylines(frame,[np.array(area4,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('4'),(728,191),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area4,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('4'),(728,191),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a5==1:
cv2.polylines(frame,[np.array(area5,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('5'),(874,154),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area5,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('5'),(874,154),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a6==1:
cv2.polylines(frame,[np.array(area6,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('6'),(1032,113),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area6,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('6'),(1032,113),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a7==1:
cv2.polylines(frame,[np.array(area7,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('7'),(790,484),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area7,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('7'),(790,484),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a8==1:
cv2.polylines(frame,[np.array(area8,np.int32)],True,(0,0,255),2)
```

```
cv2.putText(frame,str('8'),(1059,424),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area8,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('8'),(1059,424),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a9==1:
cv2.polylines(frame,[np.array(area9,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('9'),(615,598),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area9,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('9'),(615,598),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
if a10==1:
cv2.polylines(frame,[np.array(area10,np.int32)],True,(0,0,255),2)
cv2.putText(frame,str('10'),(905,568),cv2.FONT_HERSHEY_COMPLEX,0.5,(0,0,255),1)
else:
cv2.polylines(frame,[np.array(area10,np.int32)],True,(0,255,0),2)
cv2.putText(frame,str('10'),(905,568),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),1)
cv2.putText(frame,str(space),(23,30),cv2.FONT_HERSHEY_PLAIN,3,(255,255,255),2)
cv2.imshow("picam2", frame)
for i, space_count in enumerate([a1, a2, a3, a4, a5, a6, a7, a8, a9, a10], start=1):
space_occupied = 1 if space_count == 1 else 0
space_id = i
db_cursor.execute("INSERT INTO parking_spaces (space_id, occupied) VALUES (%s, %s)", (space_id, space_occupied))
db_connection.commit()
if cv2.waitKey(1)&0xFF==27:
break
db_cursor.close()
db_connection.close()
cv2.destroyAllWindows()
#stream.stop()
```

Anexo H – Código *Python* para o treino de deteção de veículos.

```
pip install ultralytics

!pip install roboflow

from roboflow import Roboflow

rf = Roboflow(api_key="TzX5VTlOUGqBpuXQAsnn")

project = rf.workspace("car-jtz6h").project("car-color-2-kmbqv")

version = project.version(1)

dataset = version.download("yolov8")

from ultralytics import YOLO

!yolo task=detect mode=train model=yolov8n.pt data={dataset.location}/data.yaml epochs=100 imgsz=640 plots=True

!yolo task=detect mode=val model=/content/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

#export your model's weights for future use

from google.colab import files

files.download('./runs/detect/train2//weights/best.pt')
```