Universidade do Minho
Escola de Ciências

Luís Miguel Amorim Albuquerque

Melhoramento do Estacionamento Público

janeiro de 2021

Melhoramento do Estacionamento Público

Luís Albuquerque

UMinho | 2021

Universidade do Minho
Escola de Ciências

Luís Miguel Amorim Albuquerque

Melhoramento do Estacionamento Público

Dissertação de Mestrado
em Matemática e Computação

Trabalho efetuado sob a orientação da
Professora Doutora Ana Jacinta Pereira da Costa Soares
e do
Professor Doutor Luís Jorge Lima Ferrás

janeiro de 2021

Agradecimentos

O trabalho apresentado é o resultado de longos meses de planificação, de estudo, de trabalho e de dedicação. O mesmo é dividido por etapas, sendo a primeira a planificação e estruturação do trabalho. Durante esta primeira etapa, poucos acreditaram que fosse possível cumprir o plano que eu próprio tinha estipulado, num prazo tão apertado.

Por outro lado, muitas pessoas me apoiaram, e me deram muitos conselhos e força para conseguir cumprir o que prometera na planificação.

A essas pessoas dedico esta secção de agradecimentos.

Aos meus orientadores, Professora Ana Jacinta Soares, Professora Fernanda Costa e Professor Luís Ferrás, por estarem sempre disponíveis e demonstrarem constante interesse pelo projeto, tendo ajudado muitas vezes fora do horário laboral e acompanhado, inclusivamente, em reuniões com a Câmara Municipal de Braga.

Às minhas colegas e amigas, Cecília e Diana, que sempre me apoiaram e ajudaram, em todo o processo.

A todos os meus amigos e familiares, com especial apreço pela minha namorada Maria, e pelos meus pais e irmão, que acreditaram sempre em mim, mesmo quando eu próprio não acreditava. Sem dúvida, foram fundamentais nesta jornada.

Por fim, quero agradecer à *Accenture*, principalmente aos meus orientadores Nuno Oliveira e Carlos Jesus, que sempre me auxiliaram e ao Hugo Portela, que se mostrou sempre disponível.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Sistema de Estacionamento Inteligente

Na maior parte das cidades, encontrar um lugar de estacionamento disponível é uma tarefa complicada, o que leva ao aumento do trânsito, da poluição, do *stress* e do tempo perdido por parte dos condutores.

Nesta dissertação, é apresentada uma solução completa e genérica, para a gestão dos estacionamentos de veículos. Esta solução pode ser implementada em qualquer cidade, pois a mesma é capaz de se adaptar a diferentes necessidades e especificidades, através do uso de inteligência artificial ou através de leitura remota de dados.

A solução aqui apresentada consiste na elaboração de um sistema de detecção de lugares livres, um servidor, uma aplicação para telemóvel e um *back office*. O sistema de deteção é opcional, sendo somente instalado quando se pretende ter acesso ao número de lugares vagos em tempo real. O próprio sistema pode ser adaptado consoante as características do local, o orçamento disponível e a precisão desejada.

A aplicação para telemóvel permite que os utilizadores consigam ter acesso às informações dos estacionamentos disponíveis, indiquem as suas preferências e o seu destino e recebam depois sugestões de estacionamento por parte do servidor. Esta aplicação permite também que os utilizadores compartilhem informações sobre as lotações dos parques, as quais são recolhidas pelo servidor com a finalidade de treinar a rede neuronal e melhorar o seu desempenho. A rede neuronal fará as devidas previsões e, ao longo do tempo, irá aumentar a precisão do modelo de previsão. Um exemplo de funcionamento da aplicação pode ser visto em: `https://www.youtube.com/watch?v=G-MdzFcsBWw` (consultado a 7 de fevereiro de 2021).

O servidor recebe, armazena e distribui as informações dos estacionamentos, além de sugerir aos utilizadores um conjunto de estacionamentos que melhor se adaptam às suas preferências. A sugestão é apresentada com base nas características dos estacionamentos disponíveis, através de uma previsão de uma rede neuronal, treinada com todos os dados adquiridos do histórico de lotação do parque.

O *back office* permite que se gira o sistema de uma forma simples e agradável.

Pode então dizer-se que é possível implementar esta solução em todas as cidades e em todo o tipo de parques de estacionamento, auxiliando os utilizadores da aplicação, e permitindo que os mesmos

encontrem, de forma eficaz e eficiente, um lugar de estacionamento.

Palavras-chave: *Machine Learning*, *Deep Learning*, Inteligência Artificial, IoT, Redes Neuronais, CNN, YOLO, SSD, RetinaNet, Visão por computador, Estacionamentos Inteligentes, Aplicação móvel.

Abstract

Smart Public Parking

In most cities, finding a vacant parking space is a complicated task, which leads to increased traffic, pollution, stress, and a waste of drivers' time.

In this Master thesis, a complete and generic solution for the management of vehicle parking is presented. This can be implemented in any city, as it is able to adapt to different needs and specificities, through the use of artificial intelligence or through remote data reading.

The solution presented here consists of the elaboration of a system for the detection of free parking spaces, a server, a mobile application and a back office. The detection system is optional, being installed only when you want to have access to the number of vacant places in real time. The system itself can be adapted according to the characteristics of the park, the available budget, and the desired precision.

The mobile application allows users to access parking information, indicate their preferences and destination, and consequently receive parking suggestions from the server. It also allows users to share information about the park capacity, which is collected by the server for the purpose of training the neural network. The neural network will make appropriate predictions, and over time it will increase the accuracy of the forecasting model. An example of how the application works can be seen at:

`https://www.youtube.com/watch?v=G-MdzFcsBWw` (consulted on february 7^{th}, 2021).

The server receives, stores and distributes parking information, and, suggests to users a set of parking spaces that best fit their preferences. The suggestion is made based on the characteristics of the parking lots and through a forecast made by a neuronal network, trained with all the data acquired from the park's capacity history.

The back office allows one to manage the system in a simple and pleasant way.

It can then be said that it is possible to implement this solution in all cities and in all types of parking lots, assisting the users of the application, and allowing them to find, effectively and efficiently, a parking spot.

Keywords: Machine Learning, Deep Learning, Artificial Inteligence, IoT, Neural Networks, CNN, YOLO, SSD, RetinaNet, Computer Vision, Smart Public Parking, Mobile APP.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O simples ato de estacionar um carro pode revelar-se uma tarefa difícil. Esta acção pode dividir-se em três etapas: procurar um estacionamento ou um parque com um lugar disponível, estacionar o veículo nesse lugar, e pagar o estacionamento, se for esse o caso.

O processo de encontrar um lugar de estacionamento disponível é aquele que oferece mais dificuldades, pois não depende do condutor mas sim de fatores exógenos que o condutor não pode controlar.

Encontrar um estacionamento pode ser, na maior parte das vezes, uma tarefa difícil, principalmente, quando o automobilista não conhece a zona ou o local onde pretende estacionar. Neste caso, não sabe onde procurar um lugar para estacionar, e muito menos onde é mais provável encontrar um estacionamento com um lugar disponível.

Esta falta de conhecimento por parte dos automobilistas faz com que andem muitas vezes "às voltas", à procura de um estacionamento ou um parque com um lugar disponível, o que pode agravar o trânsito nessa zona e levar a um aumento da poluição (provocando problemas de saúde à população em geral).

Procurar estacionamento numa grande cidade pode ser ainda mais problemático. Existem cidades em que o número de estacionamentos disponíveis é bastante limitado, estando os seus estacionamentos constantemente lotados. Por outro lado, existem cidades que até têm bastante oferta de estacionamentos e de parques disponíveis, porém quando os automobilistas andam à procura de um parque não têm conhecimento prévio se esse parque tem lugares livres ou não. Como consequência, os automobilistas acabam por realizar uma procura ineficiente, tendo que percorrer várias ruas à procura de um lugar de estacionamento.

Em geral, quando os automobilistas estão habituados a estacionar em determinada zona ou local acabam por ter uma ideia da ocupação dos estacionamentos e dos parques dessa zona, com base nas circunstâncias do momento, como por exemplo, o dia da semana, a hora, o clima, e uma série de outros fatores. No entanto, quando os automobilistas não são conhecedores da zona/local, não têm qualquer noção sobre a existência de estacionamentos e muito menos do estado de ocupação desses mesmos estacionamentos ou parques.

1.1 Motivação e enquadramento

A procura por um lugar de estacionamento disponível contribui para alguns problemas da sociedade atual, nomeadamente: **poluição**, **trânsito**, **stress** e **tempo perdido**. Nos dias de hoje, o transporte pessoal é o meio de deslocação mais usado. Recorre-se a este meio, tanto para trabalho, como para lazer, resultando num aumento da procura por lugares de estacionamento, sendo este um problema mais crítico nas grandes cidades.

Na literatura é referido que cerca de 30% do trânsito urbano é causado pelos automobilistas que procuram estacionamento [2], e que esses automobilistas são responsáveis por 10% da poluição total causada pelos veículos [3], o que pode provocar problemas de saúde à população em geral, a médio e a longo prazo [4]. Mais, o tempo despendido na procura de estacionamentos livres, tem também como consequência perdas económicas, uma vez que cerca de 17 horas por ano são perdidas nessa procura, por condutor [5]. É de salientar que, em zonas com deficiência de estacionamentos, combinada com a falta de informação sobre possíveis lugares disponíveis nessa zona, leva a que os estabelecimentos comerciais ao seu redor possam perder possíveis negócios (o potencial cliente não consegue encontrar lugar e acaba por ir embora).

Por último, deve ser referido que este tema tem um impacto significativo ao nível individual. É sabido que 39% dos condutores referem que procurar um estacionamento é uma atividade frustrante e exaustiva, o que lhes pode causar problemas de saúde (stress e ansiedade).

Estas percentagens são bastante significativas, o que justifica a procura de uma solução eficiente para resolver este problema.

1.2 Objetivos

O objetivo desta dissertação passa por desenvolver uma aplicação móvel com o intuito de auxiliar os automobilistas a terem mais informação sobre os estacionamentos e parques com lugares livres em determinada zona ou local. Com esta aplicação, a procura por um estacionamento será mais simples, mais rápida e mais eficiente.

Para o desenvolvimento desta aplicação, o ideal seria instalar sistemas de controlo de ocupação nos diversos estacionamentos, os quais permitem fornecer a ocupação dos mesmos, em tempo real, à aplicação. Para os estacionamentos em que não seja possível instalar estes sistemas de controlo de ocupação,

executa-se uma previsão da ocupação dos estacionamentos e parques, com base numa inteligência artificial que vai aprender a reconhecer padrões (em imagens dos parque obtidas por câmara) e a indicar o número de lugares livres dos parques (esta terá em consideração as características dos parques).

A aplicação, para além dos dados que se consegue obter em tempo real pelos sistemas de controlo de ocupação (sensores e câmaras-inteligência artificial), está preparada para também receber dados inseridos, pelos próprios utilizadores da aplicação móvel. Deste modo, o algoritmo de inteligência artificial implementado, irá também aprender as especificidades de cada estacionamento para conseguir alcançar uma maior precisão na sua previsão de estacionamentos/parques com lugares livres.

A aplicação móvel, mediante a informação sobre o local de destino do utilizador, irá apresentar uma lista de sugestões de estacionamentos com lugares disponíveis. A aplicação permite também ter em consideração algumas preferências e exigências previamente indicadas pelo utilizador.

1.3 Estado da arte

O problema de arranjar um lugar de estacionamento é global e, como tal, existem diversos grupos de investigação a trabalhar e a desenvolver soluções para este problema.

No que diz respeito à ação propriamente dita de *estacionar o carro*, grande parte das marcas de automóveis já providenciam mecanismos para apoio ao estacionamento do veículo. Em alguns casos, o veículo detecta o lugar de estacionamento e o condutor apenas tem que usar o acelerador e o travão para ajudar na manobra.

Relativamente ao pagamento dos estacionamentos, existem também várias soluções. Em particular, em Portugal, existe a "Via Verde"[6], que disponibiliza um serviço que permite ao condutor estacionar e, de um modo muito simples, efetuar o pagamento. Para isso, o veículo tem que estar registado neste sistema. Este registo associa a matrícula da viatura à conta bancária do condutor, sendo o pagamento feito por débito direto. Para tal é necessário instalar um pequeno dispositivo (um identificador) dentro da viatura. Assim, quando a viatura entra num estacionamento que tem o sistema em questão, o dispositivo da viatura comunica com o dispositivo local. A viatura é identificada e a sua hora de entrada é registada. No final, quando a viatura sai do estacionamento e passa pelo sensor de saída, ela é identificada e é calculado o tempo de estacionamento, e o respectivo valor a ser cobrado.

Este tipo de solução pode ser de difícil aplicação em determinados estacionamentos. Por exemplo, em estacionamentos ao ar livre. Este sistema exige uma barreira e um dispositivo tanto na viatura como no estacionamento ou parque. Neste tipo de estacionamento pode não existir propriamente uma entrada e transformá-los nesta configuração pode ser muito dispendioso, sendo nalguns casos mesmo impossível (estacionamentos individuais em ruas).

Existem, no entanto, outras soluções alternativas a este serviço da "Via Verde"que, apesar de não serem tão práticas, abrangem um maior número de parques, uma vez que não necessitam da implementação de nenhuma infraestrutura. Estas consistem em aplicações móveis, que permitem fazer o pagamento. Mais ainda, estas aplicações podem também ser usadas para aumentar o tempo de permanência no estacionamento, caso necessite (evitando a deslocação do condutor ao local). Um exemplo deste serviço é também providenciado pela "Via Verde", com sua aplicação, "Via Verde Estacionar". Outra solução mais clássica é o uso de parquímetro.

Como já foi referido, para a etapa de estacionar o veículo e para a etapa de pagar o estacionamento (se for o caso), já existem boas soluções disponíveis. Relativamente à primeira etapa (procura de um lugar de estacionamento livre) existem na literatura algumas soluções. As mais comuns são aplicações ou sites que permitem consultar onde existem estacionamentos com lugares disponíveis, a respetiva localização, o número de lugares livres, e ainda se os lugares são pagos e qual o correspondente valor por hora.

A título exemplificativo tem-se o *Google Maps* [7], e o *Parkopedia* [8]. No *Google Maps*, quando se pesquisa por estacionamentos, é apresentado no mapa a sua localização e outras informações. O *Parkopedia* contém diversas informações sobre estacionamentos, nomeadamente a sua localização, se são pagos, o respectivo valor por hora e, por vezes, informações acerca da sua ocupação.

As aplicações deste tipo, (*Google Maps* e o *Parkopedia*), têm a desvantagem de apenas se focarem em parques pagos e uma grande parte dos parques não constam destas aplicações. Este facto é agravado em países ou cidades que estão longe do foco de interesse destas aplicações.

Outras abordagens passam pela colocação de sensores no chão (nos lugares de estacionamento), permitindo uma contagem em tempo real dos lugares ocupados. Porém, esta abordagem é bastante dispendiosa. Mais, poderá não ser eficaz devido aos seguintes problemas: locais sem marcações de estacionamento, carros mal estacionados, carros pequenos que conseguem por vezes estacionar sem

serem captados pelo sensor.

Com o passar do tempo e com a evolução da tecnologia, começou a ser usada a visão por computador para tentar solucionar este problema (através do uso de câmaras). No início foram usados algoritmos muito simples que apenas conseguiam identificar a presença de um objeto dentro da marcação do lugar de estacionamento. Esta solução, embora rudimentar, é de qualidade bastante superior à da colocação de sensores debaixo do pavimento, tendo um custo bastante inferior. De realçar que com apenas uma câmara podem identificar-se vários veículos.

O crescimento exponencial das capacidades de processamento dos computadores permitiu um uso em massa de algoritmos de inteligência artificial. Desta forma, a inteligência artificial foi ficando cada vez mais desenvolvida e, por exemplo, algoritmos como os do tipo *Deep Learning* aplicados à visão por computador foram ficando cada vez mais eficientes. Com estes algoritmos é possível detectar veículos numa imagem, independentemente da distância do veículo à câmara e independentemente da existência ou não de marcações de estacionamento no chão.

Existem, atualmente, no mercado, empresas que têm como modelo de negócio fornecer a gestão dos estacionamentos usando apenas câmaras de vigilância, [9]. No entanto, grande parte destas soluções são locais, ou seja, a recolha de dados é feita apenas para um estacionamento específico, não existindo um local centralizado (por exemplo, um site) onde as pessoas possam consultar os lugares livres dos estacionamentos e escolher o melhor local para estacionar.

Mesmo que tal *site* ou aplicação existisse, apenas poderia conter informação sobre parques com este tipo de solução (uso de câmaras de vigilância). Porém, grande parte dos parques de estacionamento não poderiam constar da aplicação, pois, poucas cidades no mundo seriam capazes de investir em soluções como esta (câmaras de vigilância por toda a cidade). Para ultrapassar esta dificuldade foram propostas soluções para compartilhar informação, onde os condutores, ativamente, comentam onde existem lugares disponíveis para estacionamento, isto é, indicam quantos lugares existem num certo estacionamento, próximo do lugar onde estacionaram.

Um exemplo de aplicação que usa esta estratégia é a *Parquist* [10]. Neste caso, o condutor quando sair do estacionamento partilha essa informação com os utilizadores da aplicação. Apesar da *Parquist* estar no bom caminho, informando as pessoas de locais onde existe momentaneamente um lugar livre de

estacionamento, ela apresenta a seguinte desvantagem: o condutor tem que obrigatoriamente registar-se na aplicação e, com a agravante de poder ser contactado para aguardar no local, por forma a garantir o lugar de estacionamento a outro utilizador que o solicite. Em contrapartida, tem a vantagem de compensar os utilizadores que partilham informação com um pagamento em moeda virtual.

1.4 Estacionamentos Inteligentes - solução proposta

Na procura por um lugar de estacionamento, podem identificar-se três grupos de condutores:

- Grupo1: os que não conhecem a zona ou o local, nem têm informação da localização de lugares de estacionamento;
- Grupo2: os que conhecem a zona ou o local, sabem onde existem estacionamentos e parques, mas não estão cientes dos lugares disponíveis existentes nesses estacionamentos;
- Grupo3: os que conhecem muito bem a zona ou o local e têm noção da ocupação dos estacionamentos ao longo dos vários dias da semana.

Para transformar uma pessoa do Grupo1 numa do Grupo2 basta encontrar uma forma de lhe transmitir onde estão os estacionamentos.

Contudo, para transformar uma pessoa do Grupo1 ou do Grupo2 numa do Grupo3 requer, não só indicar onde estão localizados os estacionamentos ou os parques, bem como *transmitir* a noção da possível ocupação desses estacionamentos. É de salientar que um condutor do Grupo3 terá um melhor conhecimento sobre a ocupação dos lugares de estacionamento dessa zona quanto mais informação tiver sobre a zona. A título de exemplo, ser conhecedor da ocorrência de eventos em datas específicas nesse local (que podem condicionar os estacionamentos).

Tendo em conta que o cenário ideal é ter a informação sobre a ocupação dos diversos estacionamentos em tempo real (para dar resposta aos condutores dos três grupos), nesta dissertação é proposta uma solução geral, que se baseia no uso de uma aplicação móvel. Esta solução é desenvolvida de forma modular, facilitando a adição de novas funcionalidades ao longo do tempo, e é geral o suficiente, abrangendo todo o tipo de funcionalidades existentes para a deteção de lugares livres, em todo o tipo de parques.

Para implementar a solução proposta nesta dissertação, foram desenvolvidas e utilizadas as seguintes ferramentas:

Servidor

- o servidor centraliza e armazena todas as informações numa base de dados;
- o servidor comunica com o exterior usando o protocolo **HTTP** fornecendo uma **API rest**, [11], para que as restantes entidades do sistema consigam comunicar, pedindo e enviando informações.

Modelos de Inteligência Artificial

- Acoplada ao servidor, há uma parte inteligente que vai ser responsável pela análise e previsão dos lugares de estacionamentos disponíveis. A previsão dos lugares de estacionamentos disponíveis será transmitida ao utilizador via servidor.
- Pode ser usada para detetar lugares de estacionamentos livres, através de imagens de câmaras, ou para análise dos dados recolhidos dos demais utilizadores, permitindo fazer uma previsão da lotação dos parques de estacionamentos baseada apenas em informação/dados (e não imagens).

Aplicação Móvel

- Tendo em conta que o objetivo principal desta dissertação é divulgar aos utilizadores a informação sobre os lugares de estacionamento livres, apresentamos o desenvolvimento de uma aplicação para telemóvel de fácil utilização. A aplicação móvel é o ponto de partida para todos os utilizadores que vão interagir com o sistema de controlo de ocupação dos lugares de estacionamento desenvolvido neste projeto.
- A aplicação móvel apresenta um mapa onde se encontram os lugares disponíveis para estacionar, incluindo informações sobre o tipo de parque. Quando o utilizador insere um destino na aplicação, são-lhe fornecidas diversas opções de estacionamento para que o utilizador consiga escolher o lugar a estacionar com base nas suas preferências e no número de estacionamentos disponíveis na zona.

Um ponto crucial para o desenvolvimento deste sistema é a recolha da informação sobre os estacionamentos existentes e a sua ocupação. Assim, para recolher esta informação é possível usar um dos seguintes métodos.

- Método 1 - implementar um sistema de reconhecimento de lugares livres, que comunica ao servidor quantos lugares estão disponíveis nesse momento nos diversos parques e estacionamentos.

  Sempre que possível, este método deve ser priorizado, uma vez que garante um volume maior de dados, tal como uma maior fiabilidade dos mesmos.

- Método 2 - incentivar os utilizadores a partilharem o estado da ocupação dos estacionamentos, dando-lhe permissões para inclusive adicionarem novos estacionamentos ou parques, caso não estejam ainda registados no sistema. De notar que este processo não será automático. É necessário implementar um filtro que valide se o estacionamento adicionado pelo utilizador é efectivamente um estacionamento, e, se ainda não consta no sistema. Para isso, é necessário desenvolver um sistema de **back office** que facilita a validação de estacionamentos, bem como a sua gestão.

Ressalva-se o facto de que a solução proposta deve estar preparada para receber as informações dos estacionamentos, independentemente do método utilizado. Assim, cada estacionamento ou parque deve ser submetido a uma análise meticulosa, onde será avaliado se vale a pena, ou não, implementar o Método 1. Em caso afirmativo, deve ser escolhido o sistema e o reconhecimento de lugares livres que melhor se adequa ao local, visto que tanto o Método 1, como a solução proposta deve funcionar sem problemas, qualquer que seja o sistema de reconhecimento de lugares livres implementado.

Para concluir, deve dizer-se que o trabalho desenvolvido nesta dissertação, além de apresentar o desenvolvimento de raiz de um sistema inteligente de procura de lugares de estacionamento, serve ainda como um guia para desenvolver e instalar um sistema de controlo de estacionamentos e sua divulgação (permitindo ainda a adição de novas utilidades por parte de terceiros).

1.5 Organização da dissertação

Depois de neste capítulo introdutório ter sido apresentado o estado da arte sobre estacionamentos em parques, e de ter sido apresentada uma solução para resolver o difícil problema de encontrar um lugar

de estacionamento disponível, é agora apresentada a estrutura usada nesta dissertação para descrever e implementar a solução desenvolvida.

Dado que a solução a adotar passa pelo uso de inteligência artificial, no Capítulo 2 são apresentados os conceitos básicos necessários para facilitar a compreensão do trabalho desenvolvido na área de redes neuronais. O principal foco serão conceitos de *Machine Learning*.

No Capítulo 3 são apresentados os diferentes sistemas de reconhecimento de lugares de estacionamento livres, designadamente o uso de sensores, o uso de uma câmara inteligente e o uso de algoritmos que permitem detetar a presença ou não de veículos nos lugares de estacionamento.

Neste capítulo são ainda apresentados resultados preliminares sobre o uso de modelos de *Machine Learning* para detecção de veículos a partir de imagens (obtidas por uma câmara).

No Capítulo 4 é apresentada uma alternativa aos parques não possuam um sistema de sensores nem de câmaras inteligentes. Neste capítulo é desenvolvida uma metodologia inovadora que permite fazer uma previsão dos lugares de estacionamento disponíveis, através do uso de inteligência artificial e de redes neuronais treinadas com dados partilhados pelos utilizadores. Como a aplicação ainda não foi distribuída por plataformas de distribuição (como por exemplo, a *Play Store* e a *App Store*), esses dados de utilizadores ainda não existem, tendo sido desenvolvido um algoritmo (baseado em dados reais observados) para gerar os dados de treino e teste do modelo de *Machine Learning* (o modelo vai prever a existência de lugares disponíveis). Assim, no momento em que a aplicação for lançada no mercado, os utilizadores já poderão obter uma sugestão da aplicação na primeira vez que a usam.

No Capítulo 5 é apresentada a aplicação móvel desenvolvida que irá permitir aos utilizadores procurar, de forma simples e rápida, um lugar de estacionamento. São discutidas as diferentes abordagens existentes para criação de aplicações móveis. É apresentada a interação da aplicação com o servidor, com o utilizador, e, possíveis problemas e soluções associados a estas interações.

No Capítulo 6 é apresentado o servidor desenvolvido, usado para armazenar as informações dos estacionamentos e para processar toda a informação (comunicar com as diferentes partes do projeto, sistema de reconhecimento de lugares livres, Aplicação, *back office*, envio e receção de informações sobre os estacionamentos e utilizadores). Neste mesmo capítulo, é também apresentado o *back office*: uma plataforma onde é possível fazer toda a gestão dos parques. Permite consultar as informações dos estacionamentos,

tal como ter acesso às reclamações e aos avisos direccionados ao estacionamento. Além disso, é possível validar estacionamentos introduzidos por um utilizador, tal como remover anomalias que sejam detetadas.

Por fim, no Capítulo 7 são apresentadas as conclusões e o trabalho futuro.

2 Conceitos básicos

Neste capítulo, são apresentados alguns conceitos básicos necessários para facilitar a compreensão do trabalho desenvolvido. O principal foco é colocado nos conceitos de *Machine Learning*.

2.1 Machine Learning

Ao longo de vários anos, os programadores limitavam-se simplesmente a escrever programas baseados em procedimentos genéricos (definidos através de uma sequência de regras e condições) e que eram capazes de resolver os problemas pretendidos.

Existem, no entanto, problemas para os quais nenhum procedimento genérico é capaz de os resolver. Em geral, para esses problemas, conhecem-se alguns dos seus resultados, designados por conjuntos de dados. Por exemplo, para um problema de classificação binária, este conjunto de dados é tipicamente da forma $D = \{(x^n, y^n), \quad n = 1, \ldots, N\}$, sendo

- x^n o vector dos atributos (*features*);
- y^n o rótulo ou etiqueta (*label*), associado ao vector dos atributos x^n

onde $x^n \in \mathbb{R}^I$, $y^n \in \{-1, +1\}$, com $n = 1, \ldots, N$.

A análise de um conjunto de dados consiste em encontrar uma função de previsão $\Phi(w; x)$ que associe atributos aos rótulos, em algum sentido ótimo

$$\Phi(w^*; x^n) \approx y^n, \;\; n = 1, \ldots, N$$

onde $w^* \in \mathbb{R}^{I+1}$ é o vector ótimo dos parâmetros da função de previsão.

Com esta nova abordagem de resolver problemas surgiu a área da aprendizagem automática (do inglês *Machine Learning* (ML)), em particular a aprendizagem supervisionada, que consiste num conjunto de algoritmos que são capazes de aprender, de forma autónoma, os parâmetros ótimos de funções de previsão, de modo a transformar e associar os dados de *Input* (atributos) aos dados de *output* (rótulos).

Apresenta-se, de seguida, um exemplo para ilustrar a dificuldade que existiria para detetar um veículo numa imagem RGB, caso não fosse implementado um algoritmo da área ML.

Exemplo:

Uma imagem é representada no computador por uma matriz de inteiros. Nessa imagem, um carro pode aparecer em qualquer parte (inclusive, a imagem pode conter mais que um carro), e em diferentes perspetivas e tamanhos. Assim, escrever um algoritmo que seja capaz de receber uma imagem e detetar onde estão os veículos nessa imagem é uma tarefa complexa e pode não produzir os resultados esperados.

Ao longo do tempo foram surgindo várias técnicas na área da visão por computador. Estas técnicas aplicam filtros na imagem, para apenas serem consideradas as partes tidas como mais importantes da mesma – os traços dos vários objetos contidos na imagem. Após a aplicação destes filtros são detectados e descritos os objetos, isto é, as suas formas e características. Porém, em geral, estes algoritmos são pouco robustos.

Com o aparecimento de algoritmos de *Deep Learning*, os problemas deste tipo (deteção de objetos em imagens) passaram a ser resolvidos de uma maneira mais eficiente e mais eficaz.

Uma das metodologias mais importantes do ML são as redes neuronais artificiais (do inglês *Artificial Neural Networks* (ANNs)), as quais se inspiram na estrutura e aspectos funcionais das redes neuronais biológicas, e, as Máquinas de Vectores de Suporte (do inglês, *Support Vector Machines* (SVMs)). Apresenta-se na secção seguinte uma breve descrição das SVMs e ANNs.

2.2 Support Vector Machine

Apresenta-se nesta secção um dos algoritmos de *Machine Learning* mais conhecido e utilizado para resolver problemas de classificação, designado por Máquina de Vectores de Suporte (do inglês, *Support Vector Machine* (SVM)).

É conhecido da literatura que o SVM, embora não tão complexo como uma rede neuronal, consegue produzir na maioria das vezes melhores resultados que as ANNs, para determinados problemas. Na figura 1 é ilustrado o hiperplano ótimo produzido pelo algoritmo SVM obtido para resolução do problema de classificação binária.

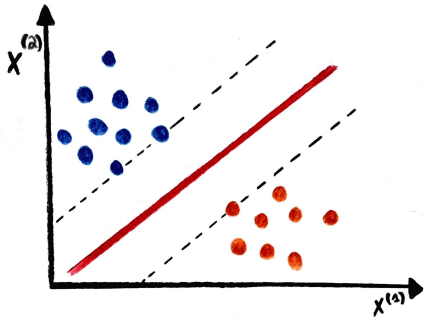

Figura 1: Problema de classificação binária - hiperplano ótimo obtido pelo algoritmo SVM.

Este algoritmo pode ser estendido a outros problemas além de classificação, nomeadamente, a problemas de regressão, de redução da dimensionalidade, da deteção de *outliers*, entre outros, [12].

No desenvolvimento do algoritmo SVM, designado por algoritmo SVM linear, assumiu-se que os conjuntos de dados para classificação são linearmente separáveis. O objectivo principal do algoritmo SVM é encontrar o hiperplano ótimo H que separa o conjunto de dados D, de tal forma que H tenha margem máxima aos vectores de suporte (os vectores x^n que estão mais próximos do hiperplano H, em cada classe).

O algoritmo SVM linear tem a desvantagem de ser muito sensível a *outliers* (valores muito discrepantes em relação à maioria dos valores da sua classe). Os *outliers* podem ter uma influência indevida na orientação e posição do hiperplano, levando a que este faça previsões erradas. Para ultrapassar esta dificuldade, surgiu na literatura o algoritmo SVM com margem flexível (do inglês *Soft margins SVM*), [13]. Este algoritmo é denotado por algoritmo C-SVM.

É de salientar que os algoritmos SVM podem também resolver problemas de classificação não linearmente separáveis. Assim, se D não é linearmente separável, é possível usar um método de kernel [14], para mapear todos os vetores x^n do espaço dos atributos original, χ, para um novo espaço de atributos de maior dimensão, $\mathcal{F}$, $\phi(x^n)$, com $n = 1, \ldots, N$, onde $\phi : \mathbb{R}^I \rightarrow \mathcal{F}$.

Definição:

Um *Kernel* é uma função K, tal que para todo $x, z \in \chi$ retorna o produto interno das suas imagens no

espaço $\mathcal{F}$:

$$K(x, z) = \phi(x)^T \phi(z).$$

A figura 2 ilustra a aplicação do método de Kernel. Os dados $x^n \in \mathbb{R}^2$, representados no gráfico à esquerda, não são linearmente separáveis. Aplicando um kernel a estes dados, os dados são transformados em $\phi(x^n) \in \mathbb{R}^3$, gráfico à direita, os quais são linearmente separáveis $\mathbb{R}^3$.

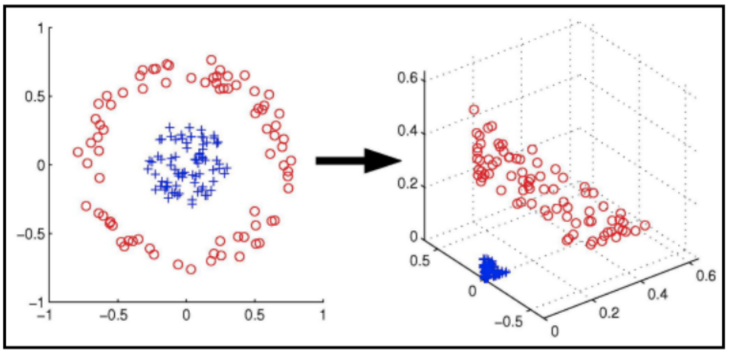

Figura 2: Ilustração da aplicação do método de kernel.

2.3 Redes neuronais artificiais

As ANNS surgiram para resolver problemas mais complexos, das relações entre dados de *Input* e de *Output*.

As redes Neuronais têm como elemento base o algoritmo do *Perceptron* (figura 3).

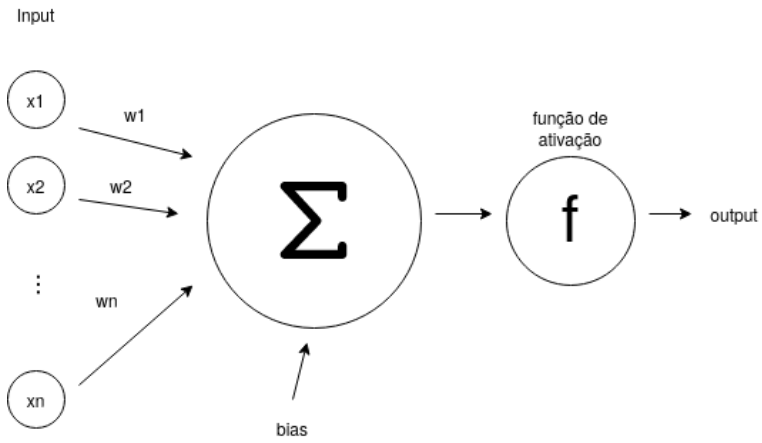

Figura 3: arquitetura do Perceptron

Um *Perceptron* é constituído por uma camada de *input* e um neurónio que recebe os dados de entrada. Os dados de entrada são pesados e combinados produzindo um sinal de saída. Existe um bias (viés) que representa o nível de ativação. O bias permite que a função tenha mais "liberdade". Existe uma função de transferência (função de activação) sendo o sinal de saída -1 (inibição) ou $+1$ activação.

O objectivo do algoritmo *Perceptron* é encontrar os pesos ótimos, w_i, $i = 1, \ldots, n$ e o viés (b), de forma a transformar/associar os dados de *Input* $x^n \in \mathbb{R}^I$, nos dados de *Output* $y^n \in \{-1, +1\}$, com $n = 1, \ldots, N$.

Dado um conjunto de dados para análise D, a função de previsão inerente ao algoritmo *Perceptron* é otimizada, para obter-se previsões corretas para novos dados sem rótulo. Neste caso, o *output* produzido pelo algoritmo de *Perceptron* será denotado por $\hat{y}$ sendo a previsão dada por:

$$\hat{y} = f\left(\sum_{i=0}^{I} w_i^* \tilde{x}_i\right) = f(w^{*T}\tilde{x}) \tag{1}$$

onde $w^* = (w_0^*, w_1^*, \ldots, w_I^*)$ com $w_0^* = b$, $\tilde{x} = (1, x) \in \mathbb{R}^{I+1}$ com $x \in \mathbb{R}^I$.

É de notar que o algoritmo *Perceptron* pode ser usado e otimizado com diferentes funções de ativação f. Nomeadamente, as que se apresentam na figura 4.

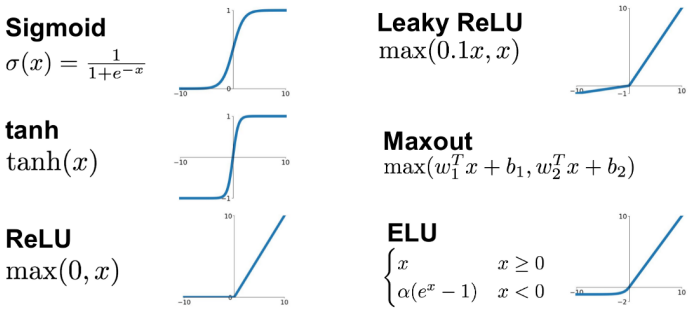

Figura 4: Funções de activação.

Como já referido, o objetivo é encontrar os parâmetros ótimos da função de previsão f, a qual é parametrizada pelo vector real $w \in \mathbb{R}^{I+1}$, sobre a qual será realizada a otimização. Uma previsão incorrecta será quantificada por uma função erro ou perda $Erro$. Isto é, dado um vector de atributos $\tilde{x}$, a diferença entre o valor previsto pela função de previsão $f(w^T\tilde{x}) = \hat{y}$ e o valor que é conhecido y, é medido pela função $Erro(f(w^T\tilde{x}), y)$.

Em geral, se a função $Erro$ é diferenciável, é possível usar o método do gradiente, [15], para calcular os parâmetros ótimos ($w^* \in \mathbb{R}^{I+1}$). É ainda de referir que, nos casos em que o conjunto de dados é muito grande, é habitual usar-se o método do gradiente estocástico [15]. Neste contexto, é ainda muito usual fixar-se um valor para o comprimento do passo η (conhecido por *learning rate*), usado na equação iterativa do cálculo dos parâmetros/pesos, do método de optimização.

Na prática, se η for grande, há o risco de nunca se encontrar um minimizante. De facto, se ele for muito pequeno, o algoritmo pode demorar muito tempo a encontrar o minimizante. Dado que o conjunto de dados pode ser muito grande, esta questão da escolha do valor para η é crucial.

A figura 5 ilustra o funcionamento do método do gradiente usado para calcular um minimizante de uma função.

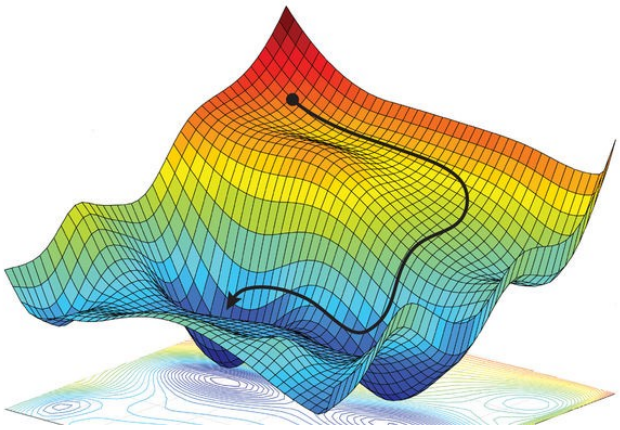

Figura 5: Método do gradiente

.

Um minimizante da função é obtido através da equação iterativa que usa a direcção de descida máxima:

$$w^{(k+1)} = w^{(k)} - \eta \nabla Erro(f(w^{(k)^T} \tilde{x}^n), y^n))$$

.

Uma rede neuronal é dada pela composição de *perceptrons*, onde o *output* de um perceptron passa a ser considerado o input para perceptrões seguintes. A figura 6 ilustra a arquitetura de uma rede neuronal artificial densa. Neste caso, a rede tem uma camada de input (com três dados de entrada), tem duas camadas escondidas (cada uma com 4 perceptrões) e uma camada de output (com dois dados de saída).

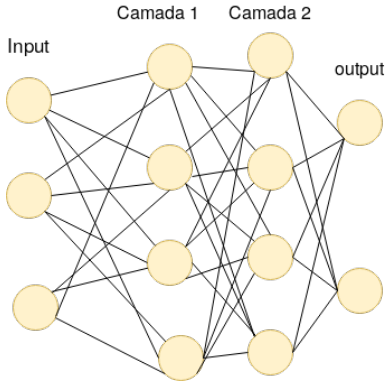

Figura 6: Arquitetura de uma rede neuronal

É de salientar que é possível definir várias arquiteturas de redes neuronais artificiais. A figura 7 ilustra alguns exemplos de arquiteturas de ANNs. A arquitetura de uma rede neuronal depende do tipo de problema

a resolver. Assim, para se resolver um problema específico, o primeiro passo é estabelecer/definir a arquitetura mais adequada.

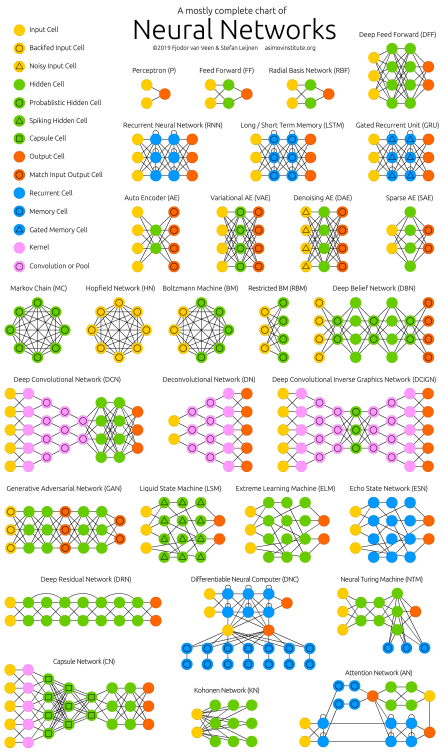

Figura 7: Diferentes arquiteturas de redes neuronais.

2.4 Visão por computador

A visão humana é extremamente complexa, foram precisos milhões de anos para organismos sofrerem mutações até se tornarem sensíveis à luz e assim permitir a visão humana como o desenvolvimento de mecanismos de visão similares.

A visão é formada por duas grandes entidades, o **olho** para capturar a luz (imagem), e o **córtex visual** para processar a imagem.

Desde o século XVIII, com a câmara obscura é possível capturar de forma artificial a luz e, deste modo, obter uma imagem semelhante à que é conseguida pelo olho humano. Recentemente, foi criada a primeira câmara digital, o que obrigou a uma definição computacional do que é uma imagem. Neste contexto, uma imagem é representada como um conjunto ou uma matriz de números. Mais recentemente, surgiu

a necessidade de se criar a visão por computador (interpretar imagens como um humano) e identificar objetos na imagem, sem auxílio humano. Apesar desta tarefa ser trivial para o cérebro humano, ela é, no entanto, uma tarefa complexa para o computador.

Nos últimos anos, devido às necessidades atuais (por exemplo, condução autónoma) e ao aumento significativo do poder computacional, foram desenvolvidos vários algoritmos de Aprendizagem Profunda (do inglês, *Deep Learning*), o que revolucionou por completo a forma como se processa e identifica objetos numa imagem. Assim, hoje em dia, é possível realizar certas tarefas, como identificar formas complexas e variáveis, como por exemplo um carro.

Os algoritmos de DL possibilitaram aos computadores aprender a identificar objetos, de forma similar a um humano.

2.4.0.1 A deteção de objetos é uma tarefa complexa, porquê? Ao longo desta dissertação, várias vezes será mencionado que a deteção de objetos é uma tarefa complexa. Nesta secção é explicado o porquê da complexidade da deteção de objetos numa imagem.

Uma das principais razões que contribui para esta complexidade é a variação de iluminação. É sabido que imagens de um mesmo objeto com diferentes iluminações fazem com que existam poucas semelhanças entre esse mesmo objeto nas várias imagens. Em casos mais drásticos, pode revelar-se mesmo muito complicado esta deteção pelos próprios humanos. Por exemplo, a dificuldade em identificar ou detectar pessoas numa passadeira com pouca iluminação.

Outro problema que afeta o reconhecimento de objetos é a variabilidade de posicionamento do objeto numa imagem. Por exemplo, um carro, apresenta características bastante diferentes, se for visto de cima, de lado, de frente ou de trás. Portanto, escrever um algoritmo que as descreva é bastante complicado.

Um outro problema é a existência de grande variabilidade intra-classe. Isto é, objetos diferentes da mesma classe têm características muito díspares entre si, podendo, por vezes, até confundir-se com objetos de outras classes. Um exemplo é a classe dos veículos, a qual contém motas, carros, carrinhas e camiões.

Para além dos problemas já mencionados, acresce ainda a dificuldade da imagem muitas vezes apenas conter uma parte do objeto ou conter um segundo objeto à frente do objeto de interesse, o que dificulta a identificação do objeto em estudo. Um exemplo é uma imagem de uma fila de carros vista de frente. Neste caso só é possível avistar por completo o veículo da frente e pequenas partes dos restantes veículos

dessa fila.

2.4.0.2 Como é que o cérebro humano identifica objetos? É de referir que, muitos dos algoritmos de DL foram baseados na forma como os humanos identificam um objeto. Para perceber melhor este mecanismo, é importante conhecer todo este processo inconsciente que ocorre no cérebro humano.

Quando um objeto é perceptível é porque parte da luz que é emitida pelo objeto é reflectida na direcção dos olhos. A seguir, o *Lateral geniculate nucleus* transmite essa informação desde o nervo óptico até ao lóbulo occipital, o que faz com que essa informação chegue ao córtex visual (parte do córtex cerebral responsável por processar informações visuais). O córtex visual é composto por seis camadas, constituídas por conjuntos de neurónios que são activados consoante a existência de determinadas características visuais. Conseguindo, assim, detectar várias informações da imagem, nomeadamente, linhas, ângulos e formas. Por último, essas informações são enviadas para o córtex infra-temporal, que junta todas as características permitindo fazer a identificação do objeto em questão.

A primeira camada do córtex visual é denominada de *córtex visual primário* e é denotada por V1. Esta é responsável por processar as informações visuais tanto de objetos estáticos como em movimento, e é especializada no reconhecimento de padrões. Esta transforma os estímulos visuais num *saliency map*, que é uma simplificação de toda a informação contida na imagem (linhas principais).

O *córtex visual secundário*, V2, é a primeira camada com área de associação visual, e, através das linhas identificadas por V1, esta detecta combinações de linhas (contornos).

A camada V3 é especializada na identificação de cor, o que muitas vezes é irrelevante para a deteção de objetos.

A camada V4, semelhante à V2, analisa os dados enviados pelas camadas anteriores e detecta combinações, neste caso os ângulos entre linhas.

As camadas V5 e V6 são responsáveis por efectuar um processamento de imagem mais avançado, o que permite identificar características mais especificas do objeto.

Por último, todas estas informações são enviadas para o *córtex visual infratemporal*, V6, a última camada no processo de análise de imagens pelo cérebro humano. Esta recebe informações de alto nível, permitindo identificar o objeto.

2.4.0.3 O que é uma imagem?

Uma imagem é constituída por duas partes, a parte visual e os seus metadados.

Os metadados são um conjunto de informações guardadas num ficheiro, como por exemplo, a extensão na qual foi armazenada (exemplo *png*, *jpeg*, etc) e a sua dimensão.

A parte visual da imagem é constituída por um conjunto de pixeis, que estão dispostos numa matriz, onde a ordem da matriz corresponde à resolução da imagem. Cada pixel, habitualmente, segue o padrão *RGB* (sigla inglesa para **Red, Green, Blue**), sendo um triplo com a informação da intensidade das três cores, nomeadamente (Red, Green, Blue), e a intensidade de cada cor varia de 0 a 255.

Assim, uma imagem será uma matriz de triplos. No entanto, ao longo desta dissertação, a imagem vai ser dada por 3 matrizes, onde cada matriz corresponde a um canal de cor *RGB*.

2.4.0.4 O que é feito para reduzir o ruído de uma imagem?

Num problema de classificação, grande parte da informação de uma imagem é considerada ruído. Isto deve-se ao facto de ser possível identificar o objeto considerando apenas os seus contornos.

Caso toda a informação da imagem fosse considerada (sem remover a informação extra/ruído), levaria a um aumento da complexidade do sistema de classificação, uma vez que todos os pixeis seriam considerados e processados pelo sistema de classificação em causa.

Para se obter sistemas de classificação mais eficientes, é usual a aplicação de filtros (matrizes de pequenas dimensões) para extrair as características mais importantes da imagem. Com essas características é construída uma nova imagem (de menor dimensão), a qual será mais facilmente processada pelo sistema de classificação.

2.4.0.5 Redes Neuronais Artificiais Convolucionais

Uma das classes de redes neuronais mais conhecidas são as redes neuronais artificiais convolucionais (do inglês *Convolutional Neural Networks* (CNN)), figura 8. As CNNs revolucionaram a área da visão por computador. São aplicadas no processamento de imagem devido à sua capacidade de extrair características importantes da imagem, o que facilita o processo de classificação.

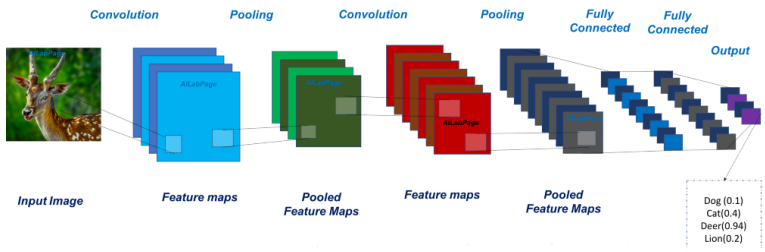

Figura 8: Etapas efectuadas por uma CNN para a deteção de objetos numa imagem.

É de referir que é possível aplicar uma ANN densa para classificar uma imagem. Porém, se a imagem for RGB, tal significa que esta contém muita informação, tornando extremamente difícil o seu processamento pela ANN. Notar que, neste caso, a imagem contém muita informação que é irrelevante para a sua classificação.

A abordagem de uma CNN passa por reduzir a dimensão da imagem através do uso de filtros. A redução é efectuada de forma a que não sejam perdidas as características principais da imagem. Neste contexto, o processo de classificação é simplificado em termos computacionais e conduz a um aumento da precisão devido à diminuição de ruído.

É de salientar que a parte visual de uma imagem é dada por uma única matriz se for uma imagem a preto e branco, e por três matrizes, se for a cores, seguindo o padrão RGB.

A um filtro também pode-se dar o nome de *Kernel*. Em certos casos, dependendo da dimensão do filtro, para que o mesmo possa ser aplicado, é adicionada à matriz dos pixeis (imagem) uma borda de pixeis com valor 0, para que todos os pixeis possam ser submetidos ao filtro. Esta estratégia apenas serve para que a ordem da operação com o filtro seja válida.

No caso de uma imagem RGB com resolução n por m, representada por 3 matrizes de dimensão $n \times m$, e de um filtro de ordem t, tem-se que $t < \min(n, m)$.

O filtro é aplicado até percorrer toda a imagem, começando no canto superior esquerdo, avançando, para a direita, um conjunto de pixeis definido à *priori*, sendo que quando chega à borda direita, a operação seguinte é feita com o filtro posicionado na primeira coluna e na linha seguinte. Esta operação é dada pela fórmula (2), onde F é a matriz do filtro, l_i é a linha que corresponde ao início da aplicação do filtro e t corresponde à ordem da matriz do filtro. A notação é dada por F[linhas][colunas] e [inicio:fim]

$$F[l_i : t + l_i][c_i : t + c_i], \forall l_i \; c_i, 0 \leq l_i \leq n, 0 \leq c_i \leq m. \tag{2}$$

Aplicando filtros, a dimensão da matriz (imagem) é reduzida. É ainda de salientar o uso de operações de *Pooling*, que tem por objectivo a redução da dimensão da imagem sem perder as suas características principais. Uma das operações mais conhecidas denomina-se por *Max Pooling*, que reduz um conjunto de pixeis a um pixel, o qual corresponde ao pixel de maior valor.

Assim, após aplicar este dois processos a matriz da imagem apresenta uma dimensão mais reduzida. Estes dois processos podem ser aplicados mais que uma vez (ver figura 8).

É de salientar que os filtros são como um peso (w_i) numa ANN. Dada uma matriz inicial, o filtro é inicializado. Ao longo do processo (classificação), os vários filtros vão sendo aprendidos e construídos de forma a extrair as características que melhor definem cada classe em estudo, o que permitirá obter resultados mais precisos, tal como referido em [13].

Assim, no fim de todo este processo, a imagem fica sintetizada e a qual será muito mais fácil de avaliar.

Apesar das CNNs serem muitas vezes usadas no processamento de imagens, podem também ser aplicadas a outros tipos de dados, nomeadamente, texto, som, entre outros.

Na área de visão por computador, principalmente quando são aplicadas técnicas de DL, 3 grandes abordagens são consideradas, e deverão ser escolhidas consoante a aplicação que se pretenda:

- Classificação – esta abordagem analisa a imagem como um todo, e dá como resposta a classe do objeto em estudo. Caso a CNN esteja preparada para classificar carros, se a imagem contiver dois carros, a CNN dá como resposta que a imagem tem um carro.

- deteção – esta abordagem é um pouco mais complexa, pois em vez de classificar a imagem como um todo, analisa várias partes, identificando os lugares com maior probabilidade de conterem os objetos em estudo. Caso a CNN esteja preparada para detectar carros, se a imagem contiver carros, a CNN dá como resposta a localização desses carros na imagem, bem como a sua classe. Esta abordagem pode ser usada para deteção de vários objetos diferentes numa mesma imagem, nomeadamente casas, carros, árvores, etc.

  Esta abordagem é nitidamente mais complexa do que a anterior, uma vez que pode realizar várias classificações. Mais ainda, o tamanho da sua resposta é variável e depende da imagem que é analisada. A CNN dá como resposta a localização de uma caixa que envolve o objetos detectado (4 pontos) e a sua classe, para cada objetos detectado, em estudo. Por exemplo, uma imagem

que contenha 4 carros dá como resposta quatro caixas (uma por cada carro detectado) e as suas classes. Caso uma imagem só contenha 1 carro, dá como resposta apenas uma caixa e a respectiva classe.

- Segmentação – a segmentação é usada quando se quer obter mais precisão, sendo feita a classificação pixel a pixel, indicando a que classe pertence cada um dos pixeis. É de notar que, na classificação, a resposta é sobre a imagem como um todo, e a deteção indica caixas de localização onde os objetos se encontram.

2.4.1 Principais algoritmos de deteção de objetos

Existem vários algoritmos para a deteção de objetos em imagens, que utilizam várias abordagens. Em geral, estes algoritmos podem ser divididos em dois grupos, consoante o número de etapas envolvidas (uma ou duas). Os algoritmos de duas etapas começam por fazer uma procura das regiões de interesse na imagem e posteriormente fazem a classificação. Exemplos de algoritmos de duas etapas são R-CNN, Fast R-CNN, Faster R-CNN, onde R-CNN vem do inglês *Region-based Convolution Neuronal Network*.

Os algoritmos de uma etapa classificam a totalidade da imagem. Exemplos de algoritmos de uma etapa são YOLO (do inglês *You Only Look Once*), SSD (do inglês *Single Shot Detector*), RetinaNet.

Estes algoritmos serão brevemente descritos no texto que se segue.

2.4.1.1 Region-based Convolution Neuronal Network (R-CNN) – O algoritmo R-CNN cria caixas em torno das regiões de interesse. Estas regiões são obtidas com auxílio do algoritmo *selective search*. Posteriormente, apenas estas regiões da imagem serão analisadas de forma a classificar os objetos que se encontram nessas regiões [16].

Um objeto pode ser detectado devido a variações numa imagem, como por exemplo: escalas, cores, textura, orientação de linhas, entre outras [17].

Quando o algoritmo R-CNN recebe uma imagem de input, são geradas as regiões de interesse através do algoritmo *selective search*, e, a cada região, é aplicada uma rede neuronal convolucional. A rede neuronal convolucional extrai as características mais importantes da imagem, alimentando uma *Support Vector Machine* (SVM) que faz a classificação dos objetos. Por último, para cada objeto identificado, é

usado um modelo de regressão linear que faz o ajuste das caixas aos objetos [18] (ver figura 9).

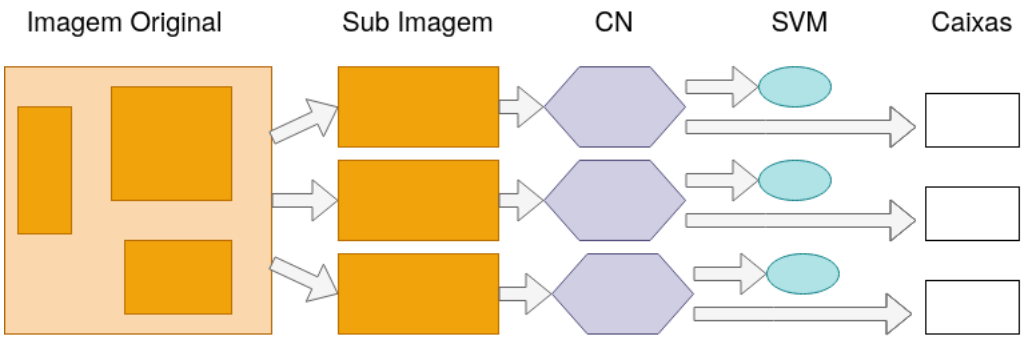

Figura 9: Etapas realizadas pelo algoritmo R-CNN para a deteção de objetos.

O R-CNN é computacionalmente dispendioso, pois usa o algoritmo *selective search* e uma CNN para extracção das características mais importantes em cada região. Além disso, o algoritmo de *selective search* usa sempre a mesma estratégia para identificar as regiões de interesse, não tendo a capacidade de aprender qual a melhor forma de detectar os objetos pretendidos.

2.4.1.2 Fast Region-based Convolutional Neural Network (Fast-RCNN) - Com o intuito de solucionar os problemas de performance do R-CNN, foi desenvolvido o Fast-RCNN, que agrega as três diferentes componentes do R-CNN (CNN, SVM e regressão) numa única arquitetura.

O Fast-RCNN começa por aplicar uma CNN à imagem como um todo, obtendo assim um *feature map* que identifica as regiões de interesse. De seguida, todas as regiões de interesse detectadas são convertidas para um único tamanho, sendo posteriormente alimentadas a uma rede neuronal artificial densa. Esta rede possui duas camadas que, em paralelo, fazem a classificação dos objetos com uma *softmax* e ajustam a caixa que envolve o objeto, ver figura 10.

Resumindo, o Fast-RCNN usa um único modelo (CNN) para extracção das regiões de interesse, classificação e ajuste das caixas. Contudo, tal como o R-CNN, usa o algoritmo *selective search*, o que faz com que o Fast-RCNN seja lento no processamento de *datasets* de tamanho elevado ou na deteção e classificação em tempo real [19].

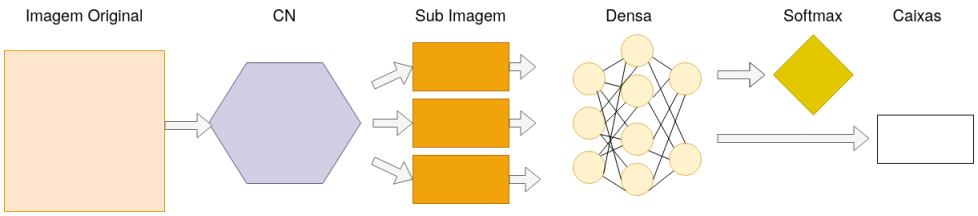

Figura 10: Etapas realizadas pelo algoritmo Fast R-CNN na deteção de objetos.

2.4.1.3 Faster Region-based Convolutional Neural Network (Faster R-CNN) - O algoritmo Faster R-CNN é uma versão melhorada do Fast R-CNN. A diferença entre os dois é a utilização de uma *Region Proposal Network* (RPN) em detrimento do *selective search*.

Através de uma CNN é obtido o *feature map* da imagem, ao qual é aplicada a RPN. Esta identifica as regiões que contêm objetos e passa-as a uma rede neuronal com duas camadas. Uma *softmax* que faz a classificação e uma camada de regressão para output das caixas ajustadas aos objetos [20], ver figura 11.

Tal como todos os algoritmos de deteção de objetos já abordados, o Faster R-CNN utiliza regiões para a identificação de objetos. Isto resulta na necessidade de fazer várias passagens pela mesma imagem para extrair todos os objetos. Outra desvantagem deste algoritmo é a utilização de várias componentes de forma sequencial, o que provoca propagação de erros.

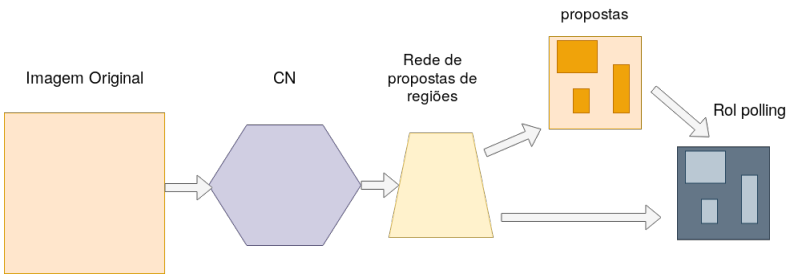

Figura 11: Etapas realizadas pelo algoritmo Faster R-CNN na deteção de objetos de uma imagem.

2.4.1.4 You Only Look Once (YOLO) – O algoritmo YOLO, ao contrário dos algoritmos anteriores, faz a deteção de objetos em apenas uma etapa [1], fazendo a classificação de toda a imagem sem haver uma primeira procura de regiões de interesse. O YOLO faz a previsão da localização das caixas e calcula

[1] Note-se que todos os algoritmos abordados anteriormente fazem deteção em duas etapas: a primeira onde a imagem é processada de forma a identificar as regiões de interesse (objetos); a segunda onde estas regiões são classificadas, levando à identificação dos objetos.

a probabilidade das classes, para toda a imagem, com o auxílio de uma rede neuronal semelhante a uma CNN. Este algoritmo é muito usado para deteção de objetos em tempo real (por exemplo, na condução autónoma de veículos) devido à sua elevada eficiência e rapidez, figura 13.

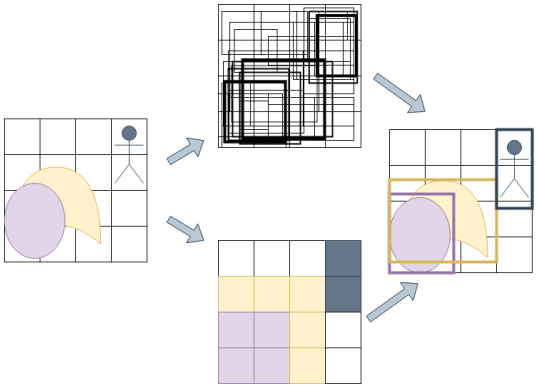

Figura 12: Estratégia utilizada pelo algoritmo YOLO na deteção de objetos numa imagem.

A deteção de objetos usando o algoritmo YOLO segue os seguintes passos (figuras 12 e 13):

1. Divide a imagem de input em regiões mais pequenas, através da construção de uma grelha de células de tamanho $S \times S$;

2. Para cada célula, faz duas operações em paralelo: cria N caixas de tamanho variável e calcula o valor da probabilidade de estas conterem um objetos; uma rede neuronal vai calcular a probabilidade de cada classe dos objetos identificados [2];

3. Por fim, as caixas sobrepostas e com maior probabilidade de classe são filtradas de forma a obter-se uma caixa por objetos;

[2] Note-se que apenas será feito o cálculo das probabilidades das classes uma vez por célula, independentemente de N

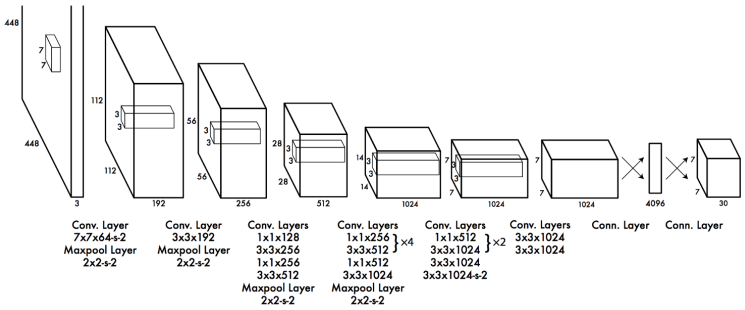

Figura 13: arquitetura do algoritmo YOLO.

Apesar da sua elevada velocidade de deteção e classificação de objetos, este algoritmo tem uma precisão mais baixa quando comparado com os algoritmos abordados anteriormente (caracterizados por duas etapas) [21].

2.4.1.5 Single Shot Detector (SSD) - Tal como o algoritmo YOLO, o SSD também faz a deteção de objetos numa única etapa. O algoritmo SSD usa a mesma estratégia que o YOLO para abordar o problema, mas consegue obter previsões mais precisas pelo facto de utilizar várias grelhas de células de tamanhos variáveis, em vez de uma única grelha de células de tamanhos iguais. Isto permite ao SSD melhorar a deteção de objetos de diferentes escalas, fazendo assim uma deteção mais fina (ver figura 14).

O algoritmo SSD é mais preciso que o YOLO, contudo o YOLO seria uma escolha mais acertada se a velocidade de deteção fosse mais importante que a precisão [22].

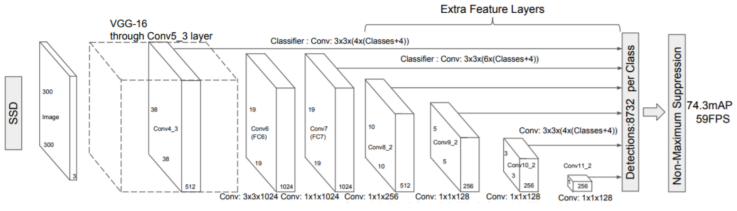

Figura 14: arquitetura do SSD

2.4.1.6 RetinaNet – Tal como já mencionado, os algoritmos de deteção de objetos com uma etapa são muito mais rápidos mas em contrapartida são menos precisos do que os de duas etapas (R-CNN, Fast R-CNN, Faster R-CNN). Para ultrapassar este obstáculo (de precisão mais baixa), surgiu o algoritmo

RetinaNet que possui duas estratégias de melhoria em relação ao YOLO e SSD: *Focal Loss* e *Feature Pyramid Network* (FPN).

1. **Focal Loss**

   Os algoritmos de deteção de objetos com uma etapa, como o YOLO e SSD, fazem a classificação de toda a imagem como um todo. Isto implica que a maioria das classificações correspondam ao fundo da imagem, que não contem qualquer objetos, levando a um grande desequilíbrio de classes no detector. Os algoritmos de deteção com duas etapas, fazem primeiro uma procura por regiões de interesse (objetos) e de seguida aplicam uma rede neuronal convolucional para fazer a classificação, evitando assim o fundo da imagem. Para corrigir este desequilíbrio de classes, de forma a manter a velocidade de deteção característica do YOLO e SSD, o algoritmo *RetinaNet* propõe a utilização de uma função custo modificada [23].

   Uma das funções custo mais utilizadas é a *Cross-Entropy*, a qual é dada por:

$$CE(p, y) = -\sum_{t} y_t . \ln p_t \tag{3}$$

   onde

símbolo	descrição
CE	Cross Entropy
p	vector de probabilidades
y	vector de classes
t	índice
y_t	denota $y[t]$
p_t	denota $p[t]$

   Não obstante, esta função de custo apresenta uma desvantagem: caso uma rede neuronal classifique 90% das caixas de uma imagem como sendo fundo, com elevada precisão (em inglês *accuracy*) e tenha dificuldade em classificar os restantes 10% (que se apresentam como objetos de interesse),

então esta última percentagem será praticamente insignificante em termos de contribuição para o valor da função custo. Como consequência, a rede não irá ter em atenção a minoria em que teve dificuldades de classificação devido ao número elevado de exemplos/dados que foram facilmente classificados (90%).

O *RetinaNet* usa uma função *Cross-Entropy* modificada, em que a contribuição dos exemplos e dos dados facilmente classificados como o fundo é reduzida. Desta forma é obtida uma aprendizagem focada nos dados de interesse. Esta nova função custo é designada por *Focal Loss* e é dada por [23]:

$$FL(p, y) = -\sum_{t} y_t.(1 - p_t)^{\gamma}. \ln p_t \tag{4}$$

onde

símbolo	descrição
FL	Focal loss
p	vector de probabilidades
y	vector de classes
t	índice
y_t	denota $y[t]$
p_t	denota $p[t]$

e $\gamma \in [0, 5]$ é um parâmetro que permite a redução da contribuição dos exemplos facilmente classificados. Na prática, é considerado um exemplo facilmente classificado aquele com uma probabilidade de previsão superior a 0.6.

Notar que, quando $\gamma = 0$ obtém-se a função *cross-entropy* ((3)). É de salientar que no caso de um exemplo ser classificado com uma probabilidade muito perto de 1, então o termo $(1 - p_t)$ terá um valor muito próximo de 0, levando a uma aprendizagem reduzida ou mesmo inexistente e focando a atenção nos casos de interesse [23].

2. **Feature Pyramid Network**

A deteção de objetos de diferentes escalas é uma tarefa complexa, especialmente para objetos de dimensões reduzidas. Este problema é comum a todos os algoritmos abordados anteriormente, sendo que, o SSD tenta resolver o problema através da utilização de grelhas com tamanhos variados. Porém, esta abordagem do SSD continua a apresentar grandes dificuldades na deteção de objetos de tamanho muito reduzido. Uma possível solução é alimentar a rede neuronal com várias cópias da mesma imagem, mas com dimensões progressivamente mais reduzidas (assemelhando-se a uma pirâmide). Contudo, esta estratégia é computacionalmente muito dispendiosa, tendo sido desenvolvida uma alternativa designada por *Feature Pyramid Network* (FPN).

A FPN faz a extração das características de uma imagem através da utilização de vários *feature maps* de diferentes escalas, com principal foco na performance. A FPN pode ser dividida em dois processos, o *bottom-up* e o *top-down* [1]:

(a) *Bottom-up*

Este processo é responsável pela recolha das características de uma imagem, no sentido ascendente da pirâmide, através da utilização de uma *ResNet*, tal como descrito em [24]. A cada camada, a resolução espacial é diminuída por um fator de $1/2$, permitindo que estruturas de alto nível [3] sejam detetadas, traduzindo-se num aumento do valor semântico à medida que se aproxima do topo da pirâmide. Cada camada devolve um *feature map*, que, por conexão lateral, será usado para enriquecer o processo *top-down*. Este processo encontra-se representado na figura 15 a).

(b) *Top-down*

Neste processo, a resolução espacial do *feature map* da última camada do processo *bottom-up* é aumentada segundo um fator de 2. Apesar da riqueza semântica, os *feature maps* das várias camadas apresentam localizações imprecisas dos objetos. Para corrigir isto, são realizadas conexões laterais entre camadas do mesmo nível das pirâmides dos dois processos. Este processo encontra-se representado na figura 15 b).

[3] As estruturas de baixo nível caracterizam-se por "primitivos"de imagens, por exemplo bordos, elementos de textura ou regiões. As estruturas de alto nível incluem objetos, cenas ou eventos.

O *RetinaNet* possui quatro componentes principais. As duas primeiras componentes são os processos *bottom-up* e *top-down*, que permitem a obtenção dos *feature maps*. As duas componentes finais são duas redes em paralelo, uma de classificação e uma de regressão. A rede de classificação é responsável por prever a probabilidade de um objeto pertencer a uma caixa e de fazer a classificação desses mesmos objetos (figura 15 c)). A rede de regressão é utilizada para obter a caixa que melhor se ajusta a cada objeto (figura 15 d)) [1].

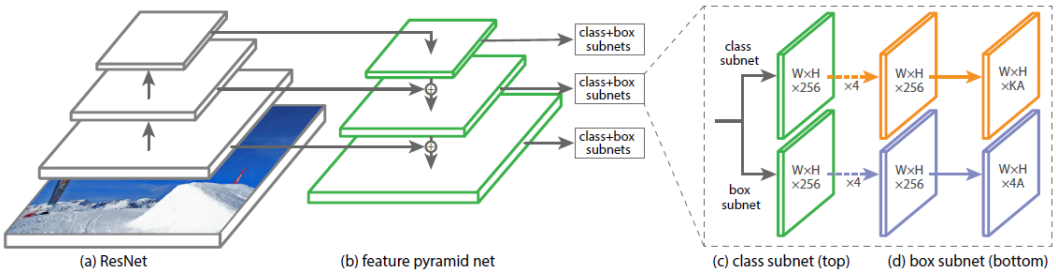

Figura 15: arquitetura de uma rede RetinaNet. [1].

Esta é composta por quatro componentes: a) processo *bottom-up*; b) processo *top-down*; c) rede de classificação; d) rede de regressão.

2.5 Métricas de desempenho

Um modelo construído por uma ANN (ou um SVM) requer um conjunto dados, designado na literatura por conjunto de dados de treino. Para avaliar o desempenho do modelo obtido na fase de treino, é necessário calcular uma matriz de confusão.

A matriz de confusão para classificação binária permite uma melhor compreensão do tipo de erros que estão a ser cometidos pelo modelo. Para a sua construção é necessário ter um conjunto de dados de teste, em que são conhecidos os respectivos valores de saída (labels).

É de referir que, em geral, o conjunto de dados é repartido em dois conjuntos, o conjunto de treino e o conjunto de teste.

Os valores de saída e as previsões dadas pelo modelo são comparadas, para obter os quatro valores da matriz de confusão: Verdadeiro positivo (do inglês *True Positive* (TP)), Falso positivo (do inglês *False*

Positive (FP)), Falso negativo (do inglês *False Negative* (FN)) e Verdadeiro negativo (do inglês *True Negative* (TN)).

Estes valores são obtidos contando o número de resultados em cada uma das seguintes categorias:

terminologia	definição
TP	o modelo prevê que um determinado dado pertence a uma classe e efectivamente pertence
FP	o modelo prevê que um determinado dado pertence a uma classe mas na verdade não pertence
FN	o modelo prevê que um determinado dado não pertence a uma classe mas efectivamente pertence
TN	o modelo prevê que um determinado dado não pertence a uma classe e efectivamente não pertence

Tabela 1: Classificar decisões

A tabela 2 ilustra a forma geral de uma matriz de confusão para classificação binária.

Tabela 2: Matriz de confusão binária.

		Classe Prevista	
		Positivo	Negativo
Classe Real	Positivo	**TP**	**FN**
	Negativo	**FP**	**TN**

Usando a informação da matriz de confusão, é possível calcular as seguintes métricas de desempenho, relativas ao modelo em estudo.

O **Recall** é a proporção de valores identificados correctamente como positivos (TP) relativamente ao número de exemplos relevantes (TP + FN).

$$Recall = \frac{TP}{TP + FN}. \tag{5}$$

A **Precision** é a proporção de valores identificados correctamente como positivos (TP) relativamente ao número de exemplos previstos como positivos (TP + FP).

$$Precision = \frac{TP}{TP + FP}. \tag{6}$$

A **Accuracy** é a proporção de valores correctamente classificados (TP + TN) relativamente ao total de previsões feitas pelo modelo (TP + TN + FP + FN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{7}$$

2.6 JSON

Ao longo deste documento serão frequentemente usados ficheiros do tipo JSON (*Java Script object notation*).

Os ficheiros do tipo JSON, como ilustrado na figura 16, têm uma formatação de dados muito conhecida, que se assemelha à forma de um objeto em *Java Script*. A sua estrutura é em forma de árvore, e a sua organização é bastante simples, contendo apenas três tipos de informação:

- *String* - é um conjunto de caracteres, que é delimitado por aspas.
- Lista – é um conjunto de valores de qualquer tipo, que é separada por vírgula, e delimitado por parêntesis rectos.
- Objetos – é um conjunto de correspondências. Associa valores a determinadas chaves, e são representados da forma $\{chave : valor\}$. A *chave* é uma String, e o *valor* pode ser de qualquer tipo (inclusive pode ser uma Lista).


```
[
  {
    "Identificador_do_estacionamento": "00001",
    "Rua do estacionamento": "Avenida da Liberdade",
    "lotação maxima": "15",
    "Dados_da_ocpação": [
      {
        "ano": "2020",
        "mês": "04",
        "dia": "06",
        "hora": "14",
        "minutos": "10",
        "segundos": "20",
        "lugares_livres": "5"
      },
      {
        "ano": "2020",
        "mês": "04",
        "dia": "06",
        "hora": "14",
        "minutos": "15",
        "segundos": "20",
        "lugares_livres": "6"
      }
    ]
  },
  {
    "Identificador_do_estacionamento": "00002",
    "Rua do estacionamento": "Avenida da Rocha",
    "lotação maxima": "5",
    "Dados_da_ocpação": [
      {
        "ano": "2020",
        "mês": "04",
        "dia": "06",
        "hora": "14",
        "minutos": "10",
        "segundos": "20",
        "lugares_livres": "0"
      },
      {
        "ano": "2020",
        "mês": "04",
        "dia": "06",
        "hora": "14",
        "minutos": "15",
        "segundos": "20",
        "lugares_livres": "1"
      }
    ]
  }
]
```


Figura 16: Exemplo do formato de um ficheiro JSON

3 Sistema de reconhecimento de lugares livres

Um dos métodos primordiais para obter dados de estacionamentos passa pela instalação de um sistema informático (sistema de reconhecimento de lugares disponíveis com recurso a um sensor por lugar – bastante usado em shoppings), que é capaz de comunicar, em tempo real, com o servidor, informando-o do número de lugares disponíveis existentes em cada instante de tempo.

Este tipo de sistema é capaz de indicar o número de lugares de estacionamento livres de um parque de forma precisa. Esta precisão deve-se a uma alta frequência de leitura realizada pelo sistema, o que permite detetar todas as alterações do parque (entrada ou saída de veículos no parque). Além disso, em geral, a incerteza de leitura do sistema é conhecida.

No entanto, um sistema deste tipo que seja capaz de satisfazer as necessidades de todos os parques, é problemático. Uma vez que existem diferentes tipos de parques, que por sua vez requerem diferentes abordagens. Exemplificando, este tipo de sistema não será o mais adequado para estacionamentos em ruas, ou para parques que requerem uma instalação otimizada.

Apresenta-se seguidamente uma breve descrição de cinco tipos de estacionamentos, para os quais este tipo de sistema não é o mais adequado.

3.0.0.1 Estacionamento sem marcações – figura 17 –

O estacionamento sem marcações é um estacionamento onde o formato não é conhecido e não tem qualquer marcação no chão a delimitar os lugares de estacionamento. Uma das características deste tipo de parques é que a lotação máxima do mesmo não é conhecida e é difícil de calcular.

Figura 17: Estacionamento sem marcações

3.0.0.2 Estacionamento em linha sem marcações – figura 18 – O estacionamento em linha sem marcações é um estacionamento onde o formato é conhecido e não tem qualquer marcação no chão a delimitar os lugares de estacionamento. A sua lotação máxima não é conhecida. Comparativamente ao estacionamento sem marcações, não é difícil de calcular, dado que a posição dos veículos é menos alterável, e a delimitação do local de estacionamento como um todo é melhor delineado.

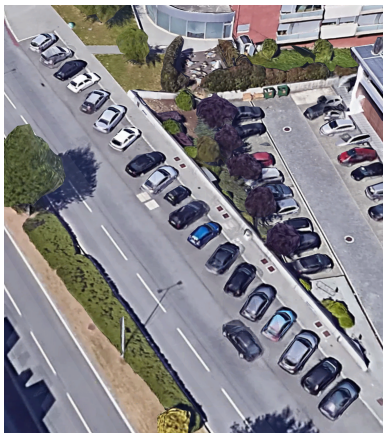

Figura 18: Estacionamento em linha sem marcações

3.0.0.3 Estacionamento em linha com marcações – figura 19 – O estacionamento em linha com marcações, ao contrário dos dois anteriores, a sua lotação máxima é conhecida. Em geral, este tipo de estacionamento tem uma grande extensão, pois vulgarmente ocorre ao longo de uma estrada.

Figura 19: Estacionamento em linha com marcações

3.0.0.4 Estacionamento com marcações – figura 20 – O estacionamento com marcações tem uma lotação máxima conhecida. Porém, poderá ter configurações diferentes (ao contrário do estacionamento em linha com marcações).

Figura 20: Estacionamento com marcações

3.0.0.5 Estacionamento com entrada e saída definida – figura 21 – Este tipo de estacionamento tem barreiras na entrada e na saída, permitindo assim fazer a contagem de veículos estacionados. Neste parque a lotação máxima é conhecida.

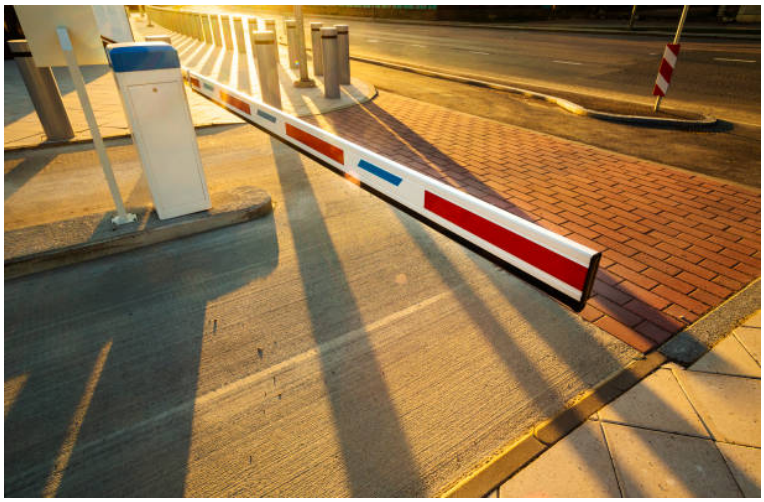

Figura 21: Estacionamento com entrada e saída definida

Para o sistema de reconhecimento de lugares livres, apresenta-se a seguir uma breve descrição de possíveis abordagens.

3.0.0.6 Sensores – O sistema de deteção de veículos mais consensual e consequentemente o mais utilizado para este fim, é o sistema de sensores. No entanto, apenas pode ser aplicado a estacionamentos com marcações.

3.0.0.7 Câmara inteligente: câmara combinada com um algoritmo de ML – O sistema que tende a ser mais promissor, com um custo de implementação reduzido, é aquele que usa uma câmara combinada com um algoritmo ML, que analisa as imagens da câmara e dá como resposta o número de lugares livres.

3.0.0.8 Comunicar ao servidor o número de lugares – Por último, o sistema mais simples, mas, no entanto, o mais restritivo, é o de apenas comunicar com o servidor. Este tipo de sistema só pode ser utilizado em estacionamentos que têm um sistema de deteção próprio, em que o sistema apenas comunica o número de lugares livres.

A solução ou a aplicação proposta nesta dissertação adapta-se a qualquer um dos sistemas de reconhecimento de lugares livres. É apenas necessário definir o modo de comunicação entre a solução e o tipo

de sistema de reconhecimento de lugares livres usado.

Apesar da solução estar preparada para funcionar com qualquer um dos três sistemas, este projeto focar-se-á substancialmente no sistema que consiste numa câmara e um algoritmo de ML. Uma vez que os restantes sistemas são de aplicação limitada e envolvem custos de implementação mais elevados.

3.1 Sensores

Um sensor é um dispositivo simples e capaz de detetar a que distância um objeto se encontra. O seu preço unitário é baixo e pode ser aplicado em diversos contextos. Uma vez que é de fácil instalação e funcionam relativamente bem, é um sistema de deteção de veículos bastante usado.

Apesar de não são ser muito comum encontrar este tipo de solução em parques de estacionamento ao ar livre, é habitual encontrar-se em parques de estacionamentos cobertos, como por exemplo, em centros comerciais.

Durante muitos anos os sensores foram a única solução para se saber a informação sobre o número de lugares livres e ocupados dos parques de estacionamento, na implementação de sistemas de controlo de estacionamentos.

Porém, o custo de instalação pode ser extremamente elevado em parques com um elevado número de lugares de estacionamento, pois é necessário aplicar um sensor por lugar. Os sensores têm também a desvantagem de apenas poderem ser aplicados em parques de estacionamento com marcações, e podem em certas situações fornecer um falso positivo (indicar que há lugar livre de estacionamento, quando efectivamente não há), como por exemplo, quando um carro está mal estacionado, ou, ser pequeno o suficiente e não ser detectado pelo sensor.

Actualmente, existem outras soluções alternativas à aplicação de sensores que funcionam incrivelmente bem e que têm um custo de instalação muito menor. A título de exemplo, citamos o uso de câmaras de vigilância.

No entanto, tudo depende da estrutura física do parque de estacionamento. Em certos parques, pode não ser possível instalar a câmara num lugar com grande cobertura. Por exemplo, no caso do parque de estacionamento ser coberto e ter um tecto baixo, será necessário instalar muitas câmaras para se conseguir mapear por completo todos os lugares de estacionamento.

Assim, se o objectivo for apenas contar os veículos, e não for interessante a opção de ter câmaras de vigilância por todo parque, os sensores será a opção mais adequada.

3.2 Câmara inteligente

Como referido, uma das principais desvantagens dos sensores é que não podem ser aplicados a qualquer tipo parque. E nos casos em que seja possível, se o parque em questão tiver um número elevado de lugares, o investimento pode tornar-se incomportável, o que impossibilita a sua utilização/instalação.

Em alternativa ao uso de sensores, uma opção mais económica passa pela utilização de uma câmara e um processador (para correr um algoritmo de ML). A câmara é colocada num local que maximize o número de veículos avistados, combinada com um processador (para correr um algoritmo de ML) que ficará encarregado de avaliar as imagens capturadas e assim averiguar o número de lugares de estacionamento livres e o número de lugares ocupados. Esta estratégia permite que, caso se consiga colocar a câmara num ponto ótimo (local que permite avistar todos os lugares de estacionamento), o investimento de implementação se cinja apenas a uma câmara e a um processador capaz de correr um algoritmo de ML. Os resultados dependem do algoritmo de ML usado para o processamento da imagem, podendo em muitos casos ser superiores aos alcançados pelos sensores, com a grande vantagem de poder ser melhorado, sem aumentar o custo de instalação. Portanto, com este sistema para reconhecimento de lugares livres – câmara inteligente – é possível realizar futuramente fiscalização autónoma (sem qualquer intervenção humana) ou intervenção assistida (optimizando o trabalho dos fiscais), o que faz com que este investimento seja rentabilizado.

3.3 Algoritmos para análise de imagens e contagem de lugares livres

Como mencionado, para implementar o sistema da câmara inteligente é necessário seleccionar e escolher um algoritmo de ML. O algoritmo de ML será responsável por analisar a imagem da câmara e dar como resposta o número de lugares de estacionamento disponíveis.

Na selecção do algoritmo de ML, deverá ter-se em consideração os requisitos da máquina e do processador que o vai executar, bem como a robustez e eficiência do algoritmo.

3.3.1 Especificidades do sistema da câmara inteligente

Dado que o sistema da câmara inteligente tem que trabalhar com as imagens capturados dos estacionamentos, é fundamental que o sistema de câmara inteligente não coloque em causa a privacidade das pessoas. Nesta dissertação é proposto um sistema de câmara inteligente com um funcionamento similar a um sensor. O sistema não guardará e nem enviará a imagem capturada, fará apenas um processamento/tratamento local.

Apresenta-se a seguir uma breve descrição de algumas das especificidades que devem ser tidas em consideração na implementação e instalação de um sistema de câmara inteligente, e que é proposto nesta dissertação.

3.3.1.1 Processamento local – Há alguns anos atrás, não havia legislação para a protecção de dados e as próprias pessoas não se importavam que as empresas reconhecessem os seus dados. No entanto, hoje em dia, grande parte das pessoas têm noção do poder dos dados e o que se pode extrair deles, estando bastante mais alertas em relação às questões da privacidade.

Assim, é de todo evitar projectar uma solução que dependa do envio de vídeos ou imagens, pois violaria a privacidade das pessoas (vias públicas e parques de estacionamento cobertos).

Portanto, para garantir a privacidade das pessoas todo o processamento da imagem deverá ser feito no local, para evitar a necessidade de se enviar ou mesmo guardar as imagens capturadas. Neste caso, o uso da câmara funcionará de modo similar a um sensor.

3.3.1.2 Algoritmo de ML eficiente – Dado que o processamento da imagem terá de ser feito no local, o processador ou o computador que irá executar o algoritmo de ML (algoritmo de deteção) pode ser bastante limitado computacionalmente.

Assim, na selecção do algoritmo de ML dever-se-á ter em consideração o processador ou o computador que é usado para o executar. Notar que, o tempo de execução de um algoritmo num computador com GPus de alto desempenho é muito diferente do tempo de execução desse mesmo algoritmo num mini computador como um *raspberry pi*. Portanto, o algoritmo de deteção deverá ser muito eficiente.

3.3.1.3 Processar as imagens frequentemente, mas sem necessidade de processar mais do que uma vez por segundo –

O parque de estacionamento deve ser monitorizado 24 horas por dia. Isto exige que se faça um conjunto de detecções consideráveis para que todas as entradas e saídas de veículos sejam contabilizadas e para que, em qualquer instante de tempo, se tenha acesso ao número de lugares livres ou ocupados do parque.

No entanto, pelo simples facto do sistema estar instalado, para se proceder à comunicação em intervalos de lugares, ao invés de se indicar o número absoluto dos mesmos, a margem de erro aumenta. Assim sendo, caso tenham havido detecções de 5 em 5 minutos, na maior parte dos parques, não haverá qualquer erro e o valor pode ser reduzido consoante a adesão ao parque. Não existe qualquer constrangimento na possibilidade de diferentes parques enviarem informações em tempos distintos, uma vez que o servidor estaria preparado para receber os dados a qualquer momento.

3.3.1.4 Veículos estacionados em zonas específicas do parque -

A maior parte dos parques de estacionamentos têm linhas que delimitam os lugares onde os veículos podem estacionar. Assim, neste tipo de parques, os veículos estacionados encontram-se nessas zonas delimitadas por tais linhas.

3.3.1.5 Só contabilizar os veículos estacionados

O objectivo deste módulo é conseguir detetar onde os veículos ocorrem na imagem, para depois poder contabilizar os lugares ocupados e comparar-los com a lotação máxima do parque em questão. No entanto, os veículos que não estão estacionados (por exemplo, os que estão em movimento) podem ser apanhados pelo ângulo de visão da câmara e, consequentemente, pode ser adicionado um erro na contagem de lugares ocupados desse parque. Portanto, este módulo deve ser capaz de contabilizar apenas os veículos estacionados no parque.

3.3.2 Sistema de comunicação com o servidor

Como já referido, independente do algoritmo de ML seleccionado, é necessário implementar localmente um sistema da câmara inteligente capaz de processar as imagens das câmaras e contabilizar o número de lugares livres do parque, e, assim poder comunicar essa informação a um servidor.

Em cada parque de estacionamento, as câmaras devem ser instaladas de forma a minimizar o número de câmaras necessárias para cobrir todos os lugares de estacionamento desse parque. Caso seja possível,

deve-se ligar todas as câmaras a um mesmo controlador. Caso não seja possível, deverá haver um controlador principal e os restantes devem ser considerados como secundários. Cada controlador, inclusive os secundários, poderá estar ligado a mais do que uma câmara.

Assim, os controladores secundários funcionaram como o controlador principal. A única diferença é que, como a área de deteção coberta pelos controladores secundários não é completa, os controladores secundários não podem comunicar directamente com o servidor. A informação recolhida pelos controladores secundários deve ser enviada directamente para o controlador principal do parque. O controlador principal recolhe e centraliza a informação de todos os controladores secundários, e envia ou comunica o número de lugares livres de estacionamento do parque ao servidor. Portanto, sendo a comunicação com o servidor apenas estabelecida pelo controlador principal, reduz-se o número de comunicações com o servidor.

Uma estratégia alternativa à anteriormente referida seria a dos controladores secundários apenas capturarem a imagem e enviarem a imagem para o controlador principal. O controlador principal procederia ao tratamento e processamento de todas as imagens do parque. Isto levaria a uma redução de custos, pois os controladores secundários poderiam ser muito mais rudimentares. Porém, esta estratégia tem a desvantagem de colocar em causa a questão da segurança e da privacidade, uma vez que a imagem teria que viajar pela rede. Como à partida os equipamentos usados serão de baixo poder computacional (pois o intuito seria reduzir o investimento), em muitos casos não seria possível encriptar a imagem para maior segurança. Outra desvantagem associada a esta estratégia seria o impacto na performance do todo o sistema. De facto, o controlador principal teria de processar todas as imagens, ao contrário da solução previamente apresentada, em que as operações serão realizadas em paralelo.

Em cada parque, após estar recolhida a informação sobre o número de lugares livres de estacionamento, essa informação deve ser enviada ao servidor. Para isso, o método usado foi o método **PUT**, com a rota **/identificador do parque** e as informações do estado do parque enviadas no corpo da requisição. As informações são **file**, que corresponde a uma imagem do parque (opcional) e, **state**, que corresponde ao número de lugares livres de estacionamento. No entanto, o número de lugares livres de estacionamento não deve ser enviado em valor absoluto, mas sim situado num intervalo. Todo este procedimento será explicado com mais detalhe na secção 6, em que falaremos do Servidor.

3.3.3 Algoritmos de ML a Adoptar

Como referido na secção 3.3.1 o objectivo é seleccionar um algoritmo de ML, nomeadamente um algoritmo de deteção que seja o mais eficiente possível. O algoritmo será responsável por analisar as imagens da câmara e por dar como resposta o número de lugares livres de estacionamento do parque.

Recordar que, na secção 2.4.1, apresentou-se uma breve descrição sobre seis algoritmos para a deteção de objetos em imagens, tendo sido apresentadas as vantagens e as desvantagens associadas a cada um desses algoritmos. Em função dos prós e dos contras descritos de cada um desses algoritmos, optou-se por treinar e testar os seguintes algoritmos de uma etapa:

- RetinaNet.
- YOLO.

Salienta-se o facto de, caso todos os parques de estacionamento tivessem marcações, seria possível usar uma abordagem totalmente diferente, e usar um algoritmo de classificação em vez de um algoritmo de deteção.

Esta abordagem consistiria em transformar o problema de deteção actual num problema de classificação.

3.3.3.1 Nova abordagem – problema de classificação Os algoritmos descritos na secção 2.4.1 podem ser aplicados em diversas situações. Estes algoritmos foram desenvolvidos de modo a serem o mais genéricos possível, isto é, não dependem dos problemas de deteção a resolverem. Portanto, os algoritmos não têm na sua concepção especificidades particulares dos problemas, e em particular, do problema de deteção que se pretende resolver nesta dissertação.

Aliado a singularidade do problema em questão às dificuldades subjacentes aos algoritmos de deteção, é possível transformar este problema de deteção num problema de Classificação.

Uma dificuldade com que se deparam todos os algoritmos apresentados na secção 2.4.1 é a fase de identificar a localização dos veículos na imagem e com que tamanhos. E isto é crucial, pois cada algoritmo terá que ajustar, o melhor possível, as caixas à volta dos veículos detectados.

Dado que a maioria dos parques tem marcações a delimitar os lugares de estacionamento, isto obriga a que os veículos fiquem estacionados nesses lugares. Assim, neste tipo de parque é possível reduzir o custo de procura por onde se encontram os veículos ou os objetos estacionados. Tratando-se de um estacionamento com marcações e havendo a possibilidade de alguém os indicar na imagem fazendo caixas à volta dos veículos estacionados, resolver um problema de classificação é sem duvida a forma mais fácil de se obter o número de lugares livres e ocupados de um parque. Notar que treinar um algoritmo de classificação é um processo mais simplificado do que treinar um algoritmo de deteção.

Dado que as caixas à volta dos veículos já se encontram feitas, apenas é necessário classificar as imagens que correspondem a diferentes lugares de estacionamento, e perceber se contêm veículos ou não, nesses lugares de estacionamento.

Neste caso, basta simplesmente resolver um problema de classificação binária. As classes denominam-se por classe dos veículos, que significa que o lugar está ocupado, e a classe dos não veículos, que significa que o lugar está vazio.

Portanto, o problema que inicialmente é visto como um problema de deteção pode ser transformado num problema bem mais simples, nomeadamente, num problema de classificação.

Para resolver o problema de classificação é necessário treinar um algoritmo para o resolver. Este algoritmo, ao contrário dos anteriores, apenas precisa de imagens de veículos e de lugares vazios, para realizar o seu treino. Para facilitar este treino, as imagens não devem conter outros objetos.

Volta-se a salientar que esta abordagem apenas pode ser usada em parques de estacionamento com marcações. Para a usar é ainda necessário fornecer um ficheiro *JSON*, por lugar, com a seguinte estrutura:

```
{
    id: ObjectID,
    number_lots: Number,
    parking_lots: {
      minX: Number,
      minY: Number,
      maxX: Number,
```

```
        maxY: Number
        }
}
```

É de esperar que o algoritmo de classificação seja bastante mais leve em termos computacionais do que os algoritmos de deteção. Outra vantagem é que neste tipo de parques apenas se contabiliza os veículos que se encontram estacionados nos lugares marcados para estacionamento.

Uma vez que estes lugares estão pré-definidos, é apenas necessário avaliar se o lugar está ocupado ou não. Notar que o trabalho computacional é sempre o mesmo, sendo apenas necessário fazer tantas classificações como o número de lugares estacionamento existentes na imagem.

Como os lugares de estacionamento são independentes, é possível captar uma única imagem com vários lugares de estacionamento, e paralelizar a classificação dessa imagem usando vários *threads* do computador ou correr em diferentes computadores.

É, no entanto, de referir que, para que esta abordagem seja bem sucedida, é necessário que alguns aspectos técnicos estejam garantidos, nomeadamente um bom posicionamento da câmara de forma a que consiga captar na imagem todos os lugares de estacionamento, e pelo ângulo, nenhum veículo pode afectar a visão de nenhum lugar de estacionamento. Este último requisito é difícil de garantir, pois na maioria dos casos a câmara não está posicionada suficientemente alta para conseguir captar todos os lugares de estacionamento. Apesar disto ser um problema crítico para esta abordagem, não lhe é exclusivo, pois afecta também os algoritmos de deteção.

Portanto, se este problema crítico for ultrapassado, o algoritmo de classificação pode atingir uma precisão elevada, uma vez que tem os lugares de estacionamento bem definidos, podendo ser treinado devidamente.

Para finalizar, é de referir que, como esta abordagem só pode ser aplicada em estacionamentos com marcações, ela não foi usada no desenvolvimento do sistema de controlo de estacionamentos e sua divulgação, proposto nesta dissertação.

3.4 Conjuntos de treino e de teste

No âmbito desta dissertação, os conjuntos de treino são conjuntos de imagens. Estes conjuntos são usados para treinar os algoritmos de ML adoptados, nomeadamente os algoritmos RetinaNet e YOLO. É de salientar que, na fase de treino de um algoritmo, os conjuntos de treino usados devem estar o mais completos possível, isto é, devem conter imagens que sejam representativas e abrangentes do problema em estudo. Deste modo, após o treino, é expectável que se obtenha um modelo com um bom desempenho.

O conjunto de teste é um conjunto de imagens usado para avaliar o desempenho e a generalização do modelo obtido.

Cada conjunto de treino é acompanhado de um ficheiro de anotações (no formato *CSV*), no qual o primeiro campo do ficheiro indica o local onde a imagem está armazenada, seguindo-se de 4 números que correspondem a uma caixa em torno do veículo, sendo definida pelas coordenadas de 4 pontos. Por último vem a classe, sendo neste caso a classe **car**.

A estrutura do ficheiro de anotações é a seguinte:

$$ficheiro, \min(x1, x2), \min(y1, y2), \max(x1, x2), \max(y1, y2), classe$$

Nas subsecções seguintes apresentam-se os três conjuntos de treino e o conjunto de teste que foram usados para treinar e testar os modelos RetinaNET e YOLO.

3.4.1 Conjunto de treino1: Imagens de satélite

Este conjunto de treino é composto por 3748 imagens satélite (ver figura 22). Este conjunto está preparado para classificar mais objetos do que aquele que é requerido pelo problema em estudo, nomeadamente, veículos. Assim, removeu-se do conjunto de treino1 as imagens que não continham qualquer veículo, e, o ficheiro das anotações associado foi submetido a um pré-processamento, tendo sido removidas as anotações que correspondiam às imagens removidas.

No final, o conjunto de treino1 ficou com 3748 imagens de treino.

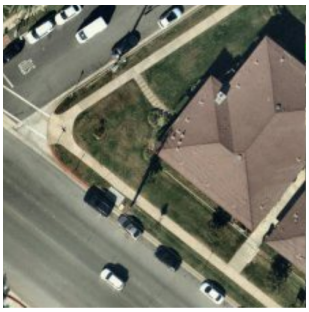

Figura 22: Exemplo de uma imagem de satélite.

3.4.2 Conjunto de treino2: Imagens obtidas por uma câmara no interior do veículo

O conjunto de treino2 é composto por 9423 imagens obtidas através de uma câmara colocada no interior de um veículo (ver figura 23). Este conjunto já se encontra devidamente anotado, não sendo necessário qualquer tipo de pré-processamento.

Figura 23: Exemplo de uma imagem obtida por uma câmara no interior de um veículo (*dash camera*).

3.4.3 Conjunto de treino3: Imagens obtidas em parques

O conjunto de treino3 é composto por 12417 imagens de carros estacionados (ver figura 24) em três parques de estacionamento distintos. As imagens que compõem este conjunto de treino foram obtidas através de câmaras instaladas num ângulo superior, considerando várias condições climatéricas. Este conjunto de treino requereu um pré-processamento das anotações. As anotações estavam em subdirectorias, uma por imagem, e em XML (num formato diferente do CSV a usar). Nas anotações apareciam as caixas referentes a cada lugar, independentemente do lugar de estacionamento estar livre ou ocupado. A caixa era definida por 4 pontos, correspondendo à sua localização, mais um valor que indicava a rotação da mesma.

Figura 24: Exemplo de uma imagem obtida num dos parques.

3.4.4 Conjunto de teste

O conjunto de teste foi construído manualmente, sendo constituído por 10 imagens obtidas em ângulos superiores. O seu ficheiro de anotações foi igualmente criado de raiz.

3.5 Treino e Teste dos algoritmos RetinaNet e YOLO

O algoritmo de DL a seleccionar será responsável por analisar as imagens da câmara e identificar na imagem os lugares ocupados de estacionamento do parque. No entanto, o algoritmo terá que fazer uma filtragem extra, uma vez que numa imagem nem todos os veículos devem ser analisados, pois determinados veículos poderão não se encontrar num lugar de estacionamento. Para solucionar este problema, é preciso indicar a zona de estacionamento, para que qualquer viatura que se encontre fora de um lugar de estacionamento seja simplesmente ignorada (ver figura 25).

Figura 25: Delimitação das zonas de estacionamento de um parque.

3.5.1 RetinaNet

O algoritmo RetinaNet dá pouca importância ao fundo das imagens e aos objetos que não são os objetos a detetar. Esta é uma das principais vantagens do algoritmo RetinaNet comparativamente aos outros algoritmos, sendo certamente uma boa opção para a resolução do problema em estudo nesta dissertação.

O conjunto de treino1 foi usado para treinar o algoritmo RetinaNet, e para a realização deste treino consideraram-se 2 épocas e 500 passos. Este treino demorou cerca de 3 horas. Após o treino, o modelo RetinaNet obtido foi testado com o conjunto de treino 1, mas os resultados obtidos com este modelo foram muito fracos. Foi notória a necessidade de se considerar um treino com mais épocas e passos, para se tentar obter um modelo com melhor desempenho.

Para isso, considerou-se para conjunto de treino a união dos conjuntos de treino1 e de treino2, e usou-se 4 épocas e 1000 passos. Após o treino, o modelo RetinaNet obtido foi testado com o conjunto de treino 2. Para imagens obtidas do ponto de visão do condutor, os resultados obtidos foram bastante razoáveis, pois o modelo identificava claramente a frente, a traseira e a lateral dos veículos. Porém, para imagens obtidas de um ângulo superior, os resultados obtidos foram fracos.

Para tentar obter-se um modelo RetinaNet com melhor desempenho, treinou-se o algoritmo RetinaNet com o conjunto de treino3. Na realização deste treino usaram-se 4 épocas e 5000 passos. Este treino demorou 4 dias num computador pessoal que apenas tem uma placa gráfica integrada. É de referir que, os conjuntos de treino usados não são os mais apropriados para o problema em estudo, pois o ângulo usado para tirar as fotos não é o ideal (ver figura 26).

Tendo em conta estas limitações, nomeadamente, o tempo de treino e a qualidade das imagens, pode dizer-se que os resultados obtidos com este modelo RetinaNet com o conjunto de teste foram bastante interessantes, mas continuam a ser inferiores aos resultados desejados. Como se pode ver na figura 26, alguns carros que estavam bem estacionados não foram detetados, outros carros que estavam estacionados fora do parque foram detetados como um bom estacionamento, e certos objetos que não são veículos, foram identificados como tal (ver por exemplo o *outdoor* representado na figura 27). Uma possível explicação para o modelo ter confundido um *outdoor* com um carro pode dever-se ao facto do algoritmo ter sido treinado com uma imagem com uma carrinha semelhante à da figura 28, que contém na sua lateral um painel semelhante a um *outdoor*. O uso de zonas de estacionamento solucionaria este problema.

Os resultados menos bons devem-se ainda ao facto de os veículos terem aparências muito dispares e complexas, o que dificulta a tarefa de identificação.

É de salientar que com um maior número de imagens, imagens essas mais diversas e mais representativas e captadas com uma melhor exposição solar, e ainda com um treino num computador mais potente, capaz de processar todas as imagens em tempo útil, é possível obter melhores resultados. Mesmo com as limitações mencionadas, é notável que o modelo RetinaNet tenha sido capaz de aprender a identificar veículos.

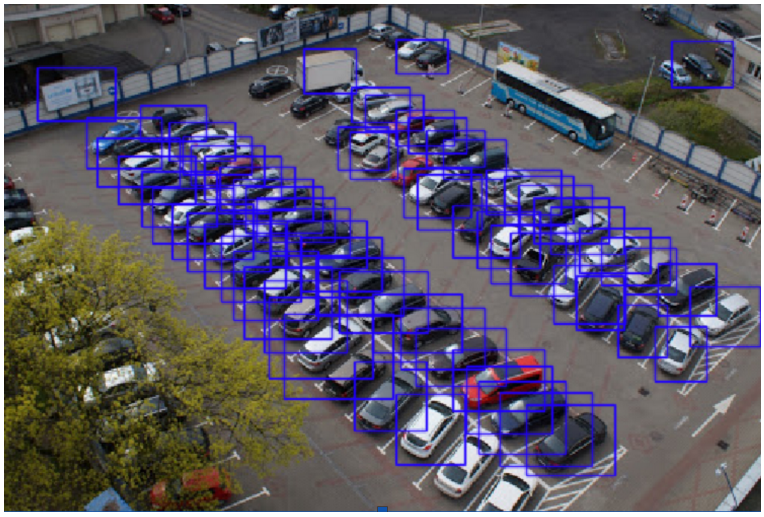

Figura 26: Resultados obtidos com um modelo Retina Net

Relativamente ao custo computacional do treino dos algoritmos, é de referir que, nos dias de hoje, já existem computadores com placas gráficas específicas para tarefas de DL ou computadores com placas gráficas suficientemente poderosas para que se consiga treinar este tipo de algoritmos. Existem ainda servidores como o *Google Colab*, que fornecem uma capacidade elevada de computação (o único senão é que o plano grátis apenas permite que se utilize o servidor durante algumas horas, por dia).

Figura 27: *Outdoor* identificado como um veículo

Figura 28: Veículo com bastantes semelhanças ao *outdoor*

Relativamente aos problemas de deteção e classificação de veículos, uma estratégia muito usada é a de criar uma nova classe com os falsos positivos encontrados.

Quando se considera apenas uma classe, qualquer objeto detetado tem uma certa probabilidade de pertencer a essa classe, contudo, só é classificado como tal, se esse valor for superior a um valor mínimo, definido *a priori*. Quando se recorre a mais que uma classe, permite-se que o modelo tenha a possibilidade de um objeto detectado pertencer a outra classe, aumentando assim a probabilidade do objeto ser identificado correctamente. Exemplificando, no caso do *outdoor* que foi classificado como um veículo, se fosse considerada outra classe com os falsos positivos, a percentagem de confiança de que o *outdoor* não seria um veículo seria maior. Isto acontece pois apesar de um *outdoor* poder ter semelhanças com uma carrinha, tem mais semelhanças com os outros *outdoor*.

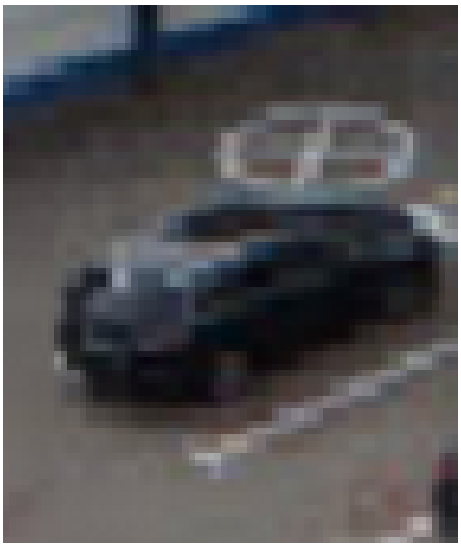

Figura 29: Veículo não identificado

É ainda de referir que também se podem obter falsos negativos. Ou seja, veículos que não são classificados como tal (ver figura 29). Para transformar os falsos negativos em verdadeiros positivos, é necessário perceber quais são as principais diferenças destes veículos para os restantes (que foram detectados) e perceber que a influência da luz, sombra, brilho ou chuva, podem ter dificultado o processo de classificar a imagem. Será necessário fazer um novo treino com mais imagens, e se possível, com imagens semelhantes às que forneceram falsos negativos.

Para finalizar, pode dizer-se que o algoritmo RetinaNet foi capaz de resolver o problema de deteção e classificação de veículos. No entanto, é importante enriquecer o conjunto de treino com novas imagens, criar novas classes, e utilizar uma ferramenta computacional adequada para a realização do seu treino, para a dimensão do problema em estudo.

3.5.2 YOLO (Darknet)

Como já referido, os conjuntos de treino usados não são muito adequados para o problema em estudo, uma vez que o ângulo usado para captar as imagens não é o ideal. Para ultrapassar esta dificuldade, na fase de instalação das câmaras nos parques, estas deverão ser colocadas em locais altos, de modo a captar o maior número de veículos possível. Deste modo, será possível obter um conjunto de fotos de vista

aérea dos lugares de estacionamento, o que permitirá ao algoritmo DL (RetinaNET ou YOLO) identificar veículos de qualquer ângulo. Para isso, é necessário treinar o algoritmo com um conjunto de imagens de vista aérea. Porém, este tipo de conjunto com anotações sobre os veículos não existe na literatura e fazer um de raiz é uma tarefa praticamente impossível.

Assim, perante a inexistência de um conjunto de treino adequado para o algoritmo YOLO, optou-se por usar um modelo disponível pré-treinado para detetar veículos.

A Darknet [25] é um programa ou uma estrutura de redes neuronais de código aberto, escrito em C e CUDA. É rápido, fácil de instalar, e suporta cálculo em CPU e GPU. Nomeadamente, possui as redes neuronais YOLO, ResNET e ResNEXT.

A Darknet possibilita treinar o modelo de uma forma fácil, apenas necessitando das imagens e das respetivas anotações (coordenadas das caixas que envolvem os objetos e o nome da sua classe). Mas como já referido atrás, o problema não é criar o modelo em si, mas sim treiná-lo. Para usar o modelo pré treinado é necessário fazer *download* do respetivo ficheiro de pesos ótimos, que se encontra disponível.

Estão disponíveis duas redes neuronais pré-treinadas YOLO. Uma com foco maior na precisão, sendo o seu modelo mais robusto, e uma versão mais leve, para ser usada em máquinas menos capazes, sendo este modelo menos preciso.

Além do ficheiro dos pesos ótimos da rede, é possível definir alguns requisitos de modo a potenciar o desempenho do modelo no computador que o vai correr. Exemplificando, é possível definir se se pretende que o computador use a placa gráfica dedicada, caso esta seja da marca Nvidea. É possível definir o valor para o parâmetro *threshold*, isto é, a precisão necessária para que o modelo considere a deteção de um certo objeto como pertencendo a uma determinada classe.

É de referir que o YOLO, além de detetar objetos numa imagem, pode fazer a deteção num conjunto de imagens, vídeos, e, inclusive, pode fazer deteção em tempo real a partir do vídeo captado pela câmara do computador.

É de salientar que o modelo foi testado num computador sem *GPU*, sendo que, muito provavelmente, será o caso do dispositivo que irá ser instalado no parque de estacionamento, para processar as imagens. Com este computador, o modelo demorou cerca de 20 segundos para fazer uma deteção.

Dada a inexistência de um conjunto de teste adequado (que contenha imagens de ângulos superiores), o modelo foi avaliado e testado usando um pequeno conjunto de imagens representativas do problema em

estudo.

A figura 30 ilustra a deteção de veículos num parque de estacionamento, realizada pelo YOLO.

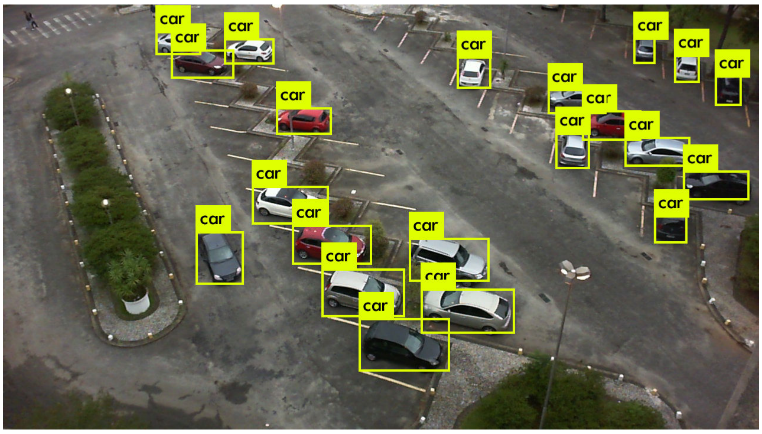

Figura 30: Resultado do YOLO

Como se pode verificar da figura 30, para esta imagem o modelo conseguiu identificar todos os veículos.

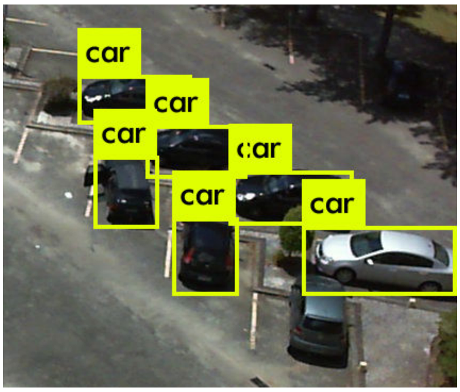

Figura 31: deteção de veículos com boa iluminação

A figura 31 apresenta uma imagem de um parque de estacionamento com boa luminosidade. Neste caso, apesar do parque estar bem iluminado, o YOLO não conseguiu detetar um dos veículos.

Figura 32: Problema na deteção de veículos devido à iluminação

Considerando agora o mesmo parque, mas com uma iluminação mais reduzida (ver figura 32), verifica-se que o YOLO teve dificuldade em detetar os veículos. Apenas conseguiu detetar um.

Tendo em conta as figuras 31 e 32, pode concluir-se que uma boa iluminação é essencial para um bom desempenho do modelo.

A figura 33 é representativa de um estacionamento onde existe uma péssima iluminação (sombra de árvores). Como se pode ver, as linhas do veículo não são visíveis, tornando-se, por isso, muito difícil o modelo conseguir fazer a sua deteção.

Figura 33: Veículo não identificado devido à falta de luz

Na figura 34 é apresentado um caso ainda mais extremo, onde até o próprio olho humano revela dificuldades na deteção do veículo. Uma vez mais, o YOLO não foi capaz de detetar este veículo.

Figura 34: Veículo não identificado devido à falta de luz

Para finalizar é de referir que os problemas da deteção de veículos, nas imagens com pouca iluminação, podem ser ultrapassados usando uma das seguintes estratégias:

- treinar o modelo com imagens que possuam as mesmas características das imagens em que modelo falhou a deteção do veículo;
- não existindo este tipo de imagens para treino, a solução passa pela adição de ruído a imagens com boa luminosidade, de modo a transformá-las em imagens com pouca luminosidade;
- usar uma câmara que permita obter fotos com melhor qualidade;
- trabalhar a imagem de forma a aumentar a exposição e o contraste, levando a que as linhas fiquem mais definidas e a que as sombras fiquem mais esbatidas, ou mesmo eliminadas. Estas operações são aplicadas utilizando a equação 8.

$$g(x) = \alpha f(x) + \beta \tag{8}$$

onde $\alpha, \beta, g(x), f(x)$ são, respectivamente, o contraste, a luminosidade, a imagem resultante e a imagem original. A aplicação desta estratégia está exemplificada nas figuras 36, 37 e 38

Figura 35: Imagem antes de ter sido processada

Figura 36: Imagem depois de ter sido processada

Figura 37: Veículo cinzento depois do tratamento, figura 33

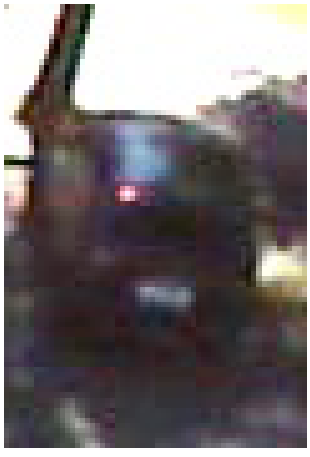

Figura 38: Veículo preto depois do tratamento, figura 34

3.6 Contagem dos lugares livres

A contagem dos lugares livres é feita através do seu complementar. Conhecendo *à priori* a lotação máxima de um parque e o número de veículo estacionados, o número total de lugares livres é dado pelo valor da lotação máxima, subtraindo o número de lugares de estacionamento ocupados.

Para que a contagem esteja correcta, a lotação máxima deve ser conhecida e a deteção dos lugares ocupados deve ser exacta.

Porém, nem todos os parques de estacionamento têm marcações e, além disso, nem sempre os veículos estão devidamente estacionados.

Para estas situações, o ideal seria aplicar um algoritmo mais inteligente. Tendo em conta os veículos já estacionados, o algoritmo deveria ser capaz de detetar um possível lugar de estacionamento e, consequentemente, determinar a sua lotação máxima.

4 Previsão da lotação de um parque e dos tempos de espera

Uma vez que nem todos os parques possuem um sistema de reconhecimento de lugares livres, a sua lotação tem que ser estimada, ou seja, tem que se prever o número de lugares livres. Se esta previsão tiver em conta os diferentes parâmetros que influenciam o estacionamento, e se a relação entre os diferentes parâmetros e o número de lugares de estacionamento for conhecida, o número de lugares pode ser previsto com exactidão.

De notar que ter este tipo de conhecimento é uma tarefa difícil, que necessita de muitos dados de históricos de estacionamento em cada parque. Contudo, como o mais importante é saber se existem lugares vagos de estacionamento e não o seu número exacto, existe alguma liberdade para ser feita uma boa previsão.

A previsão não será então a do número exacto de lugares livres, e serão estabelecidos intervalos. A previsão apenas incidirá sobre em que intervalo um dado estacionamento se enquadra, reduzindo a complexidade do problema. Esta abordagem tem como base o comportamento de muitos condutores, quando procuram um lugar de estacionamento em locais onde frequentemente estacionam. O que normalmente acontece é que, mesmo que de forma inconsciente, memorizámos vários padrões que fazem aumentar a probabilidade de um dado estacionamento ter lugares livres ou não. Podem ser situações tão simples como, ao procurar lugar, reparar que em certos estacionamentos é muito mais complicado de estacionar do que em outros. Com o passar do tempo, padrões mais complexos podem ser detectados, como correlacionar a hora, locais de interesse próximos, características do parque em si, ou até mesmo fatores externos como o clima. Essa previsão irá então deixar de estar baseada na experiência, e irá passar neste capítulo para uma inteligência artificial.

A previsão resultará num dos seguintes intervalos de número de lugares:

1. 0 lugares
2. 0-3 lugares
3. 3-10 lugares
4. >10 lugares

4.1 Relacionar os diferentes parques de estacionamento

Cada estacionamento possui um identificador, e para cada identificador são agrupados dados desse mesmo estacionamento. Esses dados sobre a ocupação dos parques são usados pelo modelo de *Machine Learning* para aprender padrões e inferir o comportamento futuro da lotação do parque.

Com o intuito de munir os novos parques de estacionamento (adicionados ao sistema e sobre os quais não temos informações sobre lotações passadas) com informações que permitam prever a lotação desse parque, foram consideradas as características semelhantes entre cada parque, de forma a que o histórico de um determinado parque possa ser usado para um novo parque que seja *semelhante* a um parque já existente, e assim ser desenvolvido um modelo de *Machine Learning* para o novo parque (o novo parque começa já com informação para os utilizadores).

Foram então consideradas as seguintes propriedades e características para estabelecer ligações entre parques:

1. Características do estacionamento

   Se é grátis, se é coberto, se tem uma estação de carregamento para os carros eléctricos, entre outros fatores. Todas estas propriedade podem ser usadas para traçar ligações entre um novo estacionamento, e um conjunto de estacionamentos, para os quais são conhecidos os padrões de estacionamento.

2. Coordenadas dos parques

   Uma das formas mais simples de relacionar estacionamentos, é usar as coordenadas dos mesmos. Estacionamentos próximos, minimamente semelhantes, tendem a ter afluências similares, sendo este um excelente parâmetro de comparação.

3. Agrupar estacionamentos por destinos

   Os condutores, quando estacionam o seu veiculo num parque, é com a finalidade de se dirigirem a um local de interesse nas proximidades. A afluência a esses locais, tem por vezes padrões delineados, o que poderá ser útil para relacionar diferentes parques de estacionamento, e perceber melhor assim os padrões dos parques que são afectados pela afluência a esses mesmos destinos.

Os destinos podem ser aglomerados em grupos:

- **escolas**
- **ginásios**
- **farmácias**
- **fábricas**
- **cafés**
- **restaurantes**
- **casas**
- **museus**
- entre outros.

A cada estacionamento será associado uma lista de destinos, e a cada combinação de destinos será atribuído um numero que caracterize essa combinação.

4.2 Formas de prever a lotação

1. Uso de padrões

   Uma vez que a lotação dos estacionamentos está relacionada com os pontos de interesse nas suas proximidades, uma abordagem possível é estudar a lotação dos diversos parques em diferentes horas, e encontrar um possível padrão que tente minimizar o erro (a uma certa hora, num determinado dia, sabendo que o estacionamento é afectado pela deslocação de pessoas para um determinado destino).

   Em primeiro lugar, serão registadas as lotações nos diversos parques, a diversos horários, e a cada parque serão associados os pontos de interesse, tal como mostrado na tabela seguinte.

ID do Estacionamento	Tipo	Hora	Minuto	Lugares livres
1	Escola	8	20	0
1	Escola	9	10	8
...	...	...	...	...
2	Ginásio	8	20	10
2	Ginásio	9	10	2
...	...	...	...	...
3	Escola	8	20	1
3	Escola	9	10	10
...	...	...	...	...

Tabela 3: Tabela completa para escrever os padrões de um parque.

Tendo sido os dados anotados, estes são organizados (Tabela 4), sendo ainda necessário aplicar uma função que permita unificar as várias instâncias de cada tipo de estacionamento. Essa função pode ser a média, a moda, a interpolação ou uma outra medida que consiga de melhor forma representar o grupo de dados como um único valor.

Tipo	Hora	Minuto	Lugares livres
Escola	8	20	1
Escola	9	10	9
...	...	...	...
Ginásio	8	20	10
Ginásio	9	10	2
...	...	...	...

Tabela 4: Tabela com os dados organizados por tipo e hora

De notar que os dados obtidos são discretos, e é preciso transformar esses valores discretos em

dados contínuos, de forma a ser possível realizar uma previsão a qualquer altura. A solução passa por interpolar os dados discretos.

2. Probabilidades

A ideia é semelhante à anterior. Em primeiro lugar é necessário estudar e anotar os dados de cada estacionamento, tal como exemplificado na tabela 5.

Estacionamento	Hora	Minuto	Lugares livres
1	8	0	5
1	8	0	4
1	8	0	3
1	8	0	5
...	...	...	...
1	8	10	2
...	...	...	...
2	8	8	2
...	...	...	...

Tabela 5: Tabela usada para calcular a probabilidade de haver estacionamento.

Depois de ter os dados anotados, é preciso uniformizar os dados e criar uma nova tabela. Desta vez, agrupando os dados por intervalos e calculando a probabilidade de cada um, como representado pela tabela 6.

Estacionamento	Hora	Minuto	0	<3	3-10	>10
1	8	0	0%	0%	100%	0%
...	...	...	...	...	...	...

Tabela 6: Tabela usada para calcular a probabilidade, por intervalos de lugares livres.

Depois da tabela estar construída será necessário uma interpolação dos dados discretos.

3. *Deep Learning*

   Tal como foi feito para previsão de lugares de estacionamento livres através de análise de imagens de câmaras, neste caso é usada uma rede neuronal capaz de identificar padrões mais complexos, podendo identificar relações entre atributos e diferentes propriedades dos parques, tendências, e rotinas dos parques e utilizadores. Estes dados seriam impossíveis de prever de forma correcta usando os dois últimos métodos (probabilidades e uso de padrões).

   O modelo de *Machine Learning* vai então aprender tudo o que se passa em torno do parque, e essa informação deve ser fornecido ao modelo através da aplicação móvel a ser apresentada no próximo capítulo.

O modelo adoptado nesta dissertação, para fazer a previsão da ocupação de parques tendo em conta as informações fornecidas pelos utilizadores, é o de *Deep Learning*.

4.2.1 Arquitetura da Previsão

Para fazer a previsão da lotação de um determinado parque, é usado o seguinte processo:

1. São recolhidas as informações para a previsão;
2. As redes neuronais a usar na previsão são treinadas com o conjunto de informações recolhidas que possam influenciar na lotação do parque.
3. São escolhidos os modelos de *Machine Learning* para se realizar a previsão;
4. É feita a previsão através dos modelos propostos;
5. É feita a previsão final, tendo em conta a previsão dos diferentes modelos.

Os resultados obtidos através dos modelos serão tão bons quanto a qualidade e a quantidade de dados disponíveis. No caso de haver escassez de dados, são desenvolvidos diversos modelos, e assim a previsão poderá ser feita através de um modelo ou de uma combinação de modelos (ver figura 39).

- **Modelo principal**: é o modelo treinado com as informações de todos os parques, tendo assim uma visão geral de todos os padrões. É o modelo ideal quando existem ainda poucas informações sobre as lotações dos parques, pois é capaz de relacionar os diferentes estacionamentos. Sendo o modelo que conta com todos os registos, é o modelo mais provável de conseguir identificar padrões invulgares (uma vez que os restantes modelos, obtidos para cada parque, contam com poucos dados).

- **Modelos dos tipos de parque**: são modelos treinados com os dados dos parques que tenham os mesmos destinos por parte dos utilizadores, havendo tantos modelos como combinações de destinos. Estes modelos, devido à sua natureza, são especializados em aprender padrões mais gerais, associados ao destino. Conseguem detetar padrões mais específicos, uma vez que possuem apenas dados de parques semelhantes.

- **Modelos do parque**: cada modelo é treinado apenas com as informações do seu respetivo parque. Assim, este modelo é o que melhor consegue identificar padrões específicos do estacionamento de destino, sendo ideal quando existe um grande volume de dados do parque em questão.

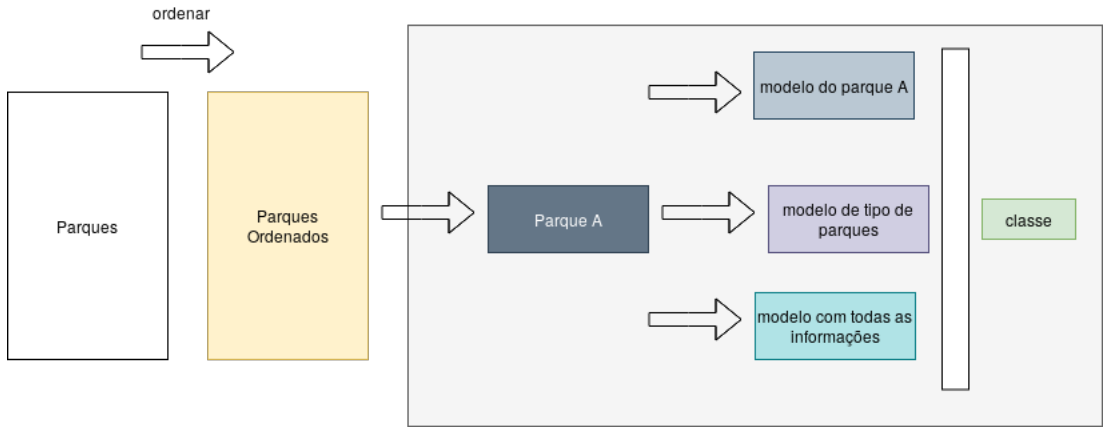

Figura 39: Esquema dos modelos usados para prever lotações em parques.

Os diversos modelos apresentados contam com a mesma arquitetura, tendo sido criados usando uma Rede Neuronal Densa, através do método de criação *Sequential* providenciado pela biblioteca do tensorflow. O modelo é constituído por 4 *camadas*.

A primeira camada, denominada por camada de *input*, é a camada responsável por receber os dados de input. Esta camada interpreta e processa os dados de forma a alimentar as próximas camadas da rede.

A camada de input corresponde aos seguintes atributos:

1. **identificador** do parque de estacionamento
2. **coordenada x** do estacionamento
3. **coordenada y** do estacionamento
4. **mês** em que foi efectuado o registo
5. **dia** em que foi efectuado o registo
6. **hora** em que foi efectuado o registo
7. **minuto** em que foi efectuado o registo
8. índice do grupo de **destinos** que afectam a lotação do parque
9. **meteorologia** no momento do registo
10. intensidade dos **eventos** próximos
11. se é **feriado**
12. se é **fim-de-semana**
13. intervalo de **lotação**

Em seguida surge a primeira camada escondida, esta possuí 128 neurónios e tem como função de activação a função ReLU.

A segunda camada escondida tem 100 neurónios e usa novamente a função de activação ReLU.

A terceira camada escondida tem 50 neurónios e volta a usar a mesma função de activação que as anteriores.

Por último, a quinta camada tem o numero de neurónios igual à dimensão do output, ou seja, 4, e tem como função de activação a Softmax.

Foram descritos 3 tipos de modelos capazes de realizar previsões (modelo principal, modelos dos tipos de parque, modelos do parque). No entanto, nem sempre devem ser usados os 3 modelos, pois dependendo da situação e das condições, estes podem não obter resultados significativos, e apenas introduzir ruído no calculo da previsão final.

A forma mais simples de combinar os modelos, para se gerar a previsão final, é a de escolher os modelos com base no volume de dados obtido sobre o histórico de lotação do parque (as redes neuronais precisam de um grande volume de dados para serem capazes de atingir bons resultados, ou seja, uma boa previsão da lotação de um parque).

Os modelos devem ser testados e avaliados de maneira a encontrar o número mínimo de dados necessários para um *bom* funcionamento do modelo. Assim, quando nenhum dos outros modelos, para além do modelo principal, tiverem alcançado o número mínimo de dados estipulado, a previsão deve ser feita utilizando apenas o **modelo principal**.

Quando somente o *modelo para cada parque* não alcançar o número mínimo de dados estipulado, será feita a previsão utilizando somente o **modelo dos tipo de parque**.

E, por fim, quando todos os modelos tiverem o número mínimo de dados estipulado, o modelo escolhido para realizar as previsões será o **modelo do parque**.

4.3 Linguagem

A linguagem escolhida para desenvolver o código para o modelo de **Machine Learning** foi *Python*. Esta é uma linguagem bastante popular, principalmente devido à sua crescente aplicação em áreas como **Data Science** e **Machine learning**, e pela facilidade de uso em manipulação numérica.

É sem duvida a linguagem que lidera o mercado quando se fala de **Machine Learning**. Isto acontece devido à existência de maior suporte e ferramentas.

Além disso, possui diversas bibliotecas, como o *tensorflow*, o *Pytorch*, *Scikit-learn*, *Scipy*, *Matplotlib*, *keras*, *Pandas*, *Numpy*.

Estas bibliotecas serão agora explicadas com mais detalhe.

4.3.1 Bibliotecas

1. Tensorflow

   Tensorflow [26] é das maiores bibliotecas no que diz respeito a *Machine Learning*. Foi criada pela equipa da *Google*, e neste momento conta com soluções para plataformas:

   - *browser*: Utilizando o **Tensorflow Js**.
   - telemóveis ou mini computadores/controladores: Utilizando o **Tensorflow Lite**.
   - Servidores: Utilizando o **Tensorflow**.

2. Matplotlib

   Matplotlib, [27], é uma biblioteca de visualização de dados. Permite construir qualquer tipo de gráfico ou esquema, de forma a interpretar os dados de uma maneira mais simples e clara. Esta biblioteca permite que se façam ajustes mais finos, de forma a que se analise e apresente os dados de acordo com as necessidades e interesses do projecto.

3. Keras

   Keras, [28], é uma biblioteca de *Machine Learning* de alto nível, que tem definidos os principais modelos de *Machine Learning* e várias arquiteturas de redes neuronais.

4. Pandas

   Pandas,[29], é das bibliotecas que permite, de uma forma simples, ler e estruturar os dados de um ficheiro.

5. Numpy

   O *Numpy*, [30], permite realizar operações matemáticas em *pyhton* de uma forma eficiente. Esta biblioteca é escrita em **C** para que seja o mais rápida e eficiente possível.

4.4 Dados

Havendo registos públicos da lotação do parque de estacionamento (seja ele coberto, lugares ao longo de uma avenida, etc), é possível fazer um estudo para se perceber que características (eventos, escolas nas

proximidades, etc) podem ter impacto no número de lugares livres disponíveis.

Porém, ao longo do desenvolvimento desta dissertação, não foi possível obter um historial de lotação de lugares de estacionamento, apesar de ter havido persistência para conseguir os dados sobre a cidade de Braga. Contactou-se a Câmara Municipal de Braga e os Transportes Urbanos de Braga, no entanto, os dados facultados passavam apenas pelo registo da lotação média estimada de cada parque, isto é, não era possível a correlação destas informações com outros parâmetros, o que tornou inviável identificar padrões com este tipo de dados. Consequentemente, não serviram de base para aplicação neste projeto.

A solução encontrada passou por gerar dados fictícios, baseados na tendência real de ocupação de lugares de estacionamento. Desta forma, os dados são semelhantes a dados reais, e, poderão ser usados para treinar e testar os modelos de *Machine Learning*.

Para gerar os dados, tanto de treino como de teste, foram construídas diversas soluções, que serão descritas de seguida.

1. Simulação em computador

   Abordagem que se baseia na criação de um conjunto de dados que descrevem padrões, com especial foco em padrões espaciais. Foi desenvolvida uma página *web* onde é possível configurar individualmente certas tendências de ocupação dos estacionamentos e indicar a sua localização num plano 2D.

   É possível indicar, numa escala de 1 a 10, um índice da tendência de ocupação dos lugares (0 indica que quase sempre há lugares livres e 10 que raramente existem lugares livres). Trata-se de uma forma simples de indicar o número de lugares livres que o estacionamento tem em média, sem que nenhum fator influencie a sua lotação.

   Além disso, é ainda possível indicar a confiança desses dados, assinalando se foram captados usando um sistema de controlo de estacionamentos ou compartilhados por um utilizador.

   Esta página *web* permite também indicar quais dos seguintes fatores e a que nível influenciam a lotação do parque de estacionamento.

   (a) Meteorologia;

   (b) Horas;

(c) Feriados;

(d) Eventos;

(e) Fins de semana.

Com estas informações, é gerado um *dataset* completo, que permite avaliar se o modelo é capaz de identificar que existem zonas que naturalmente são mais complicadas para se encontrar estacionamento, bem como avaliar se o modelo é capaz de aprender, como certas características podem influenciar a lotação do parque.

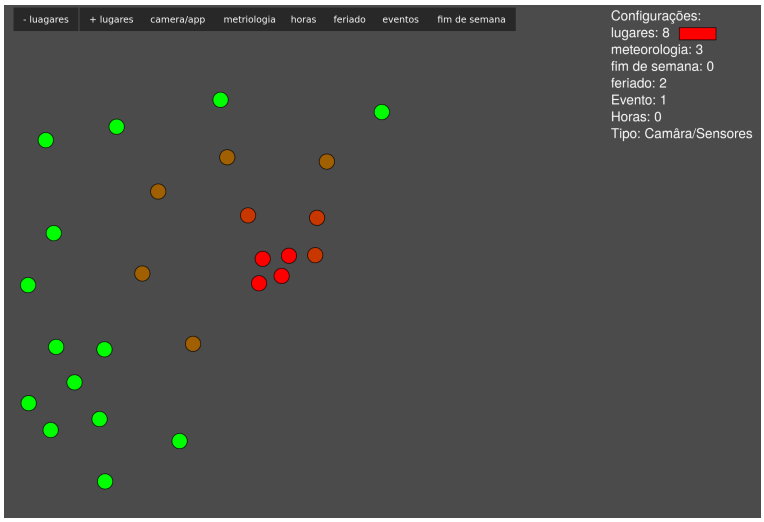

Figura 40: simulador

2. Script

Abordagem que se baseia na geração de dados, de forma a simular a afluência aos estacionamentos. O *script* criado gera um conjunto de estacionamentos fictícios, tal como um considerável volume de dados, sobre a sua suposta lotação, ao longo do tempo.

Para cada estacionamento foi considerado:

(a) **identificador do parque de estacionamento;**

(b) **Coordenadas da sua localização;**

(c) **Conjunto dos principais pontos de interesse nas proximidades;**

(d) **Indicativo da sensibilidade à alteração de certas características.**

Com estas informações, aliadas a configurações que permitem ter controlo sobre a entropia causada, isto é tornar a lotação menos determinística, foram criados todos os dados relevantes, de forma a que o modelo possa ser testado.

3. Recolha de dados manual

Recolheram-se dados reais da lotação dos estacionamentos. Foram anotadas as seguintes informações:

(a) **Identificador do parque de estacionamento;**

(b) **Coordenadas da sua localização;**

(c) **Número de lugares livres;**

(d) **Ano;**

(e) **Mês;**

(f) **Dia;**

(g) **Hora;**

(h) **Minuto.**

No entanto, devido à pandemia provocada pela *Covid-19*, apenas foi possível recolher 130 registos, pois, após ter sido imposto o confinamento obrigatório, muitos estacionamentos encontravam-se livres, o que não representava uma situação de normalidade.

Com isto, o número de dados recolhido foi muito reduzido não sendo o suficiente para treinar o modelo. Foi porém útil na construção do *script* que simula o caso real.

4.5 Análise dos dados

Depois de uma análise minuciosa dos dados obtidos por simulação em computador, através de um *script* e obtidos de forma manual, foi decidido usar apenas o *dataset* gerado pelo *script*. Isto deve-se à qualidade

do *dataset* e também ao facto de este ter sido o único usado para fazer o treino e o teste do modelo de *Machine Learning*.

Deve ser realçado que o *dataset* construído através de recolha manual, embora teoricamente mais promissor, uma vez que os mesmos não são manipulados, e contam com fatores aleatórios naturais, não foram contemplados na análise devido à escassez de dados. Deste modo seriam insuficientes para se retirar qualquer conclusão.

A análise do *dataset* é um passo fundamental para avaliar os resultados do modelo para perceber se os padrões foram detetados pelo modelo e se os resultados se enquadram na complexidade dos dados. Para esse fim, é preciso analisar várias propriedades dos dados, bem como tentar isolar algumas variáveis para que se possam perceber os padrões.

O primeiro ponto a ter em consideração é a distribuição dos dados pelos diferentes parques e grupos (estacionamentos com os mesmos destinos de interesse), para garantir que não há influências relativas à discrepância do volume de dados.

O *dataset* criado foi previamente baralhado e partido em dados para treino e teste, com um rácio de 80% (treino) para 20% (teste). Estes dados foram filtrados mediante o modelo a ser treinado/testado, pelo que o único modelo que foi treinado como os 80% completos foi o principal. Os modelos de um certo parque foram treinados e testados apenas com os dados referentes a esse parque, e analogamente para os grupos de parques. É de referir que os dados testados nunca foram usados para treino de nenhum dos modelos, ou seja, tal como os dados de treino têm como origem os mesmos 80%, apenas tendo sido filtrados para o modelo especifico, o mesmo se procede com os dados de teste.

Para um maior controlo do processo, foram avaliados todos os modelos intermediários, pois cada um é responsável por identificar padrões num conjunto de dados.

Através da tabela 7 verifica-se que os parques têm uma distribuição homogénea que ocorre tanto nos dados de treino como nos dados de teste, com uma distribuição de 80% (para os dados de treino) e 20% (para os dados de teste), em todos os parques.

Ao analisar os dados dos grupos (Tabela 7), facilmente se verifica que o parque 1 e o parque 2 pertencem ao grupo 1; já o parque 3 pertence ao grupo 2, sendo que o grupo contém todos os dados dos parques que o constitui.

Como esperado, o modelo relativo a um parque, tem todos os dados desse mesmo parque.

Dataset	Parque 1	Parque 2	Parque 3	Parque 4
modelo principal (Treino)	3621	3588	3659	3608
modelo principal (Teste)	903	936	865	916
Grupo 1 (Treino)	0	3588	3659	3608
Grupo 1 (Teste)	0	936	856	916
Grupo 2 (Treino)	3621	0	0	0
Grupo 2 (Teste)	903	0	0	0
Parque 1 (Treino)	3621	0	0	0
Parque 1 (Teste)	903	0	0	0
Parque 2 (Treino)	0	3588	0	0
Parque 2 (Teste)	0	936	0	0
Parque 3 (Treino)	0	0	3659	0
Parque 3 (Teste)	0	0	865	0
Parque 4 (Treino)	0	0	0	3608
Parque 4 (Teste)	0	0	0	916

Tabela 7: Distribuição dos dados pelos *datasets*.

Além de se analisar a distribuição relativa do volume de dados, é importante avaliar a distribuição relativa ao número de lugares livres, sendo que neste caso estuda-se o intervalo de lugares (Tabela 8).

Com estes dados é possível comprovar que os dados de treino e teste foram bem separados, tendo a mesma relação de lugares. Pode também ser verificado que, na maioria dos modelos, a maior percentagem diz respeito ao intervalo com mais de 10 lugares vagos. Isto acontece porque em grande parte das horas os parques encontram-se livres.

Contudo, o parque 2 é um parque extremamente solicitado, chegando a não ter qualquer lugar livre em 49 % do tempo. É interessante notar a distribuição de percentagens do parque 1, e consequentemente do grupo 1, que se encontram relativamente bem distribuídas pelos intervalos, o que poderá ser um indicativo

de maior dificuldade. Apesar do parque 2 não ter percentagens tão equilibradas, estas não seguem um padrão tão delineado como os restantes parques, uma vez que os parques 1, 3 e 4, na maior parte do tempo, têm mais de 10 lugares, sendo que o segundo intervalo com maior percentagem é o segundo com mais lugares livres, e assim sucessivamente. Já o parque 2 não segue o mesmo padrão, aumentando e diminuindo, e tendo o intervalo com maior densidade de valores mas com menos lugares (sendo o intervalo com menos valores nos restantes parques). É notável a melhor distribuição nos parques 3 e 4, contendo uma percentagem muito grande dos dados como possuindo lugares livres, tabela 8.

Dataset	0 lugares	0-3 lugares	3-10 lugares	>10 lugares
modelo principal (Treino)	17%	7%	17%	59%
modelo principal (Teste)	17%	7%	18%	58%
Grupo 1 (Treino)	18%	3%	15%	65%
Grupo 1 (Teste)	18%	3%	16%	64%
Grupo 2 (Treino)	15%	21%	23%	41%
Grupo 2 (Teste)	17%	20%	24%	40%
Parque 1 (Treino)	15%	21%	23%	41%
Parque 1 (Teste)	17%	20%	24%	40%
Parque 2 (Treino)	49%	3%	17%	30%
Parque 2 (Teste)	47%	3%	18%	31%
Parque 3 (Treino)	2%	2%	12%	83%
Parque 3 (Teste)	2%	2%	13%	82%
Parque 4 (Treino)	2%	2%	13%	82%
Parque 4 (Teste)	2%	3%	15%	81%

Tabela 8: Distribuição dos lugares pelos *datasets*.

Depois de se analisar a distribuição do volume de dados e a distribuição desses dados nos intervalos, será feita uma análise sobre que variáveis, e como essas variáveis influenciam os padrões obtidos.

Tendo em conta a importância do parâmetro **horas**, este foi escolhido para se fazer um estudo aprofun-

dado, uma vez que facilitará a compreensão de como os intervalos estão distribuídos. De realçar que outros parâmetros também influenciam os padrões obtidos, mas, tendo em conta que parâmetro **horas** também depende dos restantes parâmetros, pode ser assumido, com segurança, que este parâmetro representa bem a evolução da lotação do parque no tempo.

Uma vez que ao longo dos dados (muitos à mesma hora) constam diferentes intervalos, por forma a unificar e suavizar esses diferentes intervalos, foi feita uma média, catalogando os intervalos com os seguintes valores:

- 0 - quando não existirem lugares vagos;
- 1 - quando o número de lugares vagos estiver compreendido no intervalo de 0 e 3;
- 2 - quando o número de lugares vagos estiver compreendido no intervalo de 3 a 10;
- 3 - quando houverem mais de 10 lugares livres;

No modelo principal (figura 41) pode verificar-se facilmente que, das 6h às 7h, grande parte dos parques se encontram com mais de 10 lugares, começando a aumentar o fluxo e a serem ocupados a partir das 8 horas, sendo que as horas posteriores, digamos 9h, 10h e 11h, encontram-se entre o intervalo de 0-3 e 3-10. Voltando a subir perto do meio dia e baixando novamente por volta das 14h, continuando a baixar cada vez mais, até que chega à hora mais problemática, 18h, perto do intervalo de 0-3 lugares em média (figura 41).

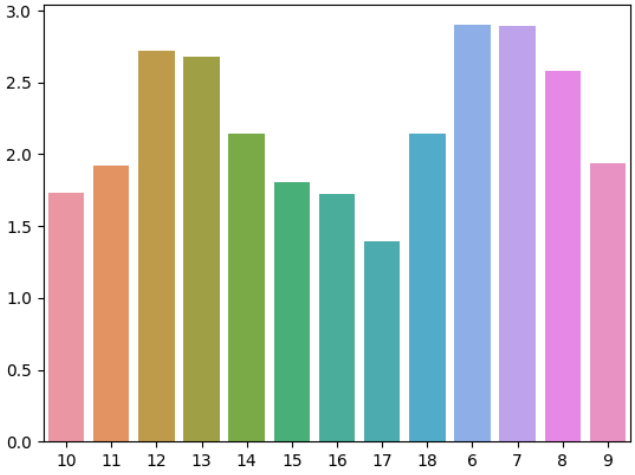

Figura 41: Modelo principal (treino). No eixo dos x temos as horas do dia, das 10 da manhã até às 18 da tarde, e das 6 às 9 da manhã. No eixo dos y temos a média dos valores representativos de cada intervalo.

Como mostrado na figura 42, os dados de teste representam perfeitamente os dados de treino, mantendo o mesmo padrão.

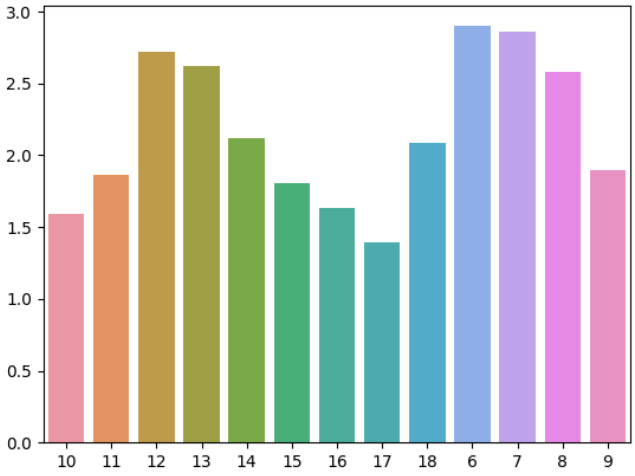

Figura 42: Modelo principal (teste). No eixo dos x temos as horas do dia, das 10 da manhã até às 18 da tarde, e das 6 às 9 da manhã. No eixo dos y temos a média dos valores representativos de cada intervalo.

Para o treino obtido com o Grupo 1 (figura 43) a diferença em relação ao modelo principal (treino) pode ser vista das 9h às 10h onde existem em média mais lugares próximos do intervalo de 3-10. Das 14h às 17h, apesar de continuar a ser a zona com menos lugares vagos, tem mais lugares em média que o modelo principal (treino), sendo que às 18h já se encontra perto do intervalo com mais de 10 lugares.

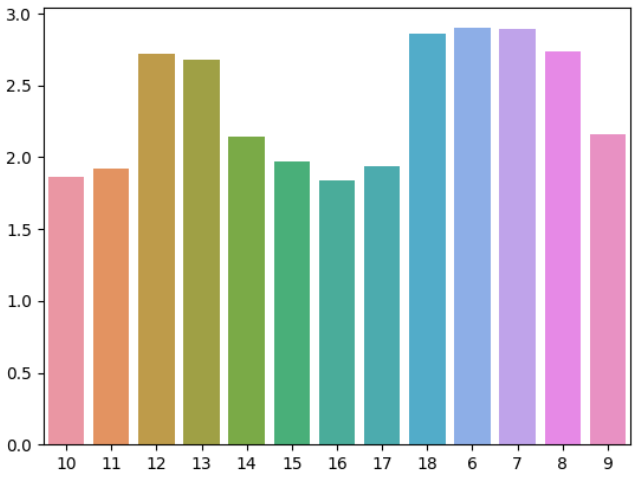

Figura 43: Grupo 1 (treino). No eixo dos x dispõem-se as horas do dia, das 10 da manhã até às 18 da tarde e das 6 às 9 da manhã. No eixo dos y temos a média dos valores representativos de cada intervalo.

Como seria de esperar, os dados de teste são semelhantes aos de treino (Figura 44), apenas com a ressalva que existem pequenas diferenças. Às 10h o valor de teste é inferior e o mesmo acontece às 16h e 18h.

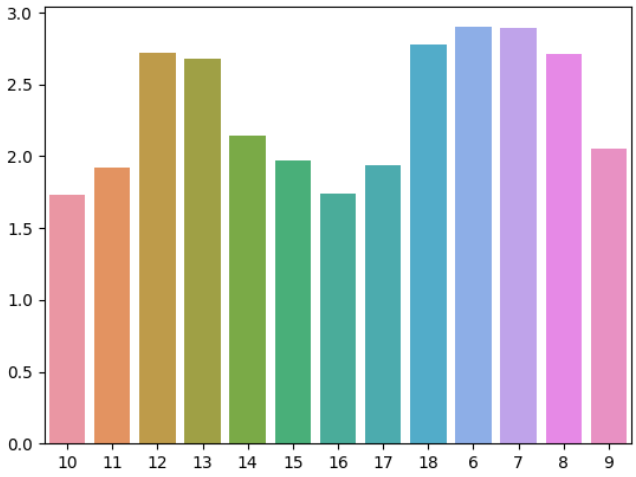

Figura 44: Grupo 1 (teste). No eixo dos x encontram-se as horas do dia, das 10 da manhã até às 18 da tarde, e das 6 às 9 da manhã. No eixo dos y, a média dos valores representativos de cada intervalo.

Já os dados de treino do grupo 2 descrevem zonas mais consistentes (figura 45), onde das 6h às 7h a média está perto do intervalo com mais de 10 lugares, baixando um pouco às 8h e voltando a baixar mais perto das 9h, ficando próximo do intervalo de 3-10. Baixa um pouco mais perto das 10h, voltando a subir às 11h para o intervalo de 3-10. Das 12h até 14h a média está no intervalo de mais de 10 lugares. Das 15 às 17 horas baixa para o intervalo de 3-10, e volta a subir a partir das 18h.

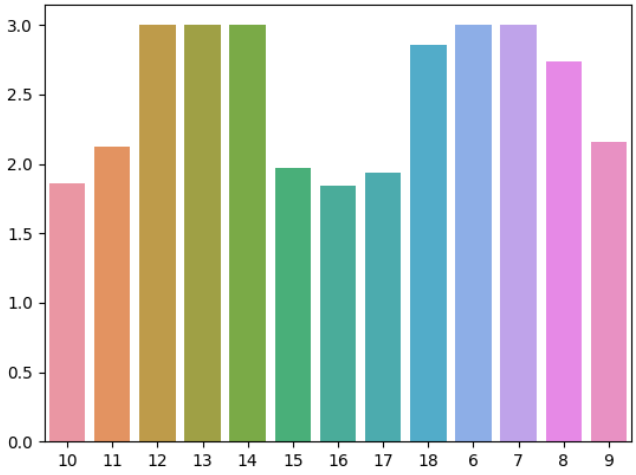

Figura 45: Grupo 2 (treino). No eixo dos x estão as horas do dia, das 10 da manhã até às 18 da tarde, e das 6 às 9 da manhã. No eixo dos y, a média dos valores representativos de cada intervalo.

Mais uma vez, os dados de teste representam correctamente os dados de treino, figura 46.

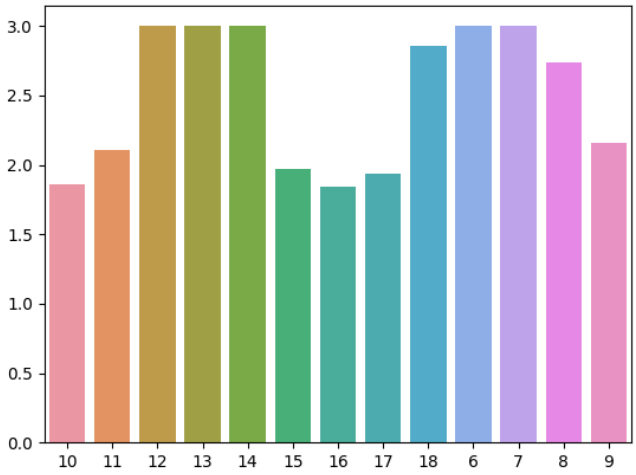

Figura 46: Grupo 2 (teste). No eixo dos x, encontram-se as horas do dia, das 10 da manhã até às 18 da tarde, e das 6 às 9 da manhã. No eixo dos y, a média dos valores representativos de cada intervalo.

Em relação aos parques, rapidamente se conclui que, em média, os padrões dos parques 1 e 2 são

bastante semelhantes aos do grupo 2, figuras 47, 48, 49, 50.

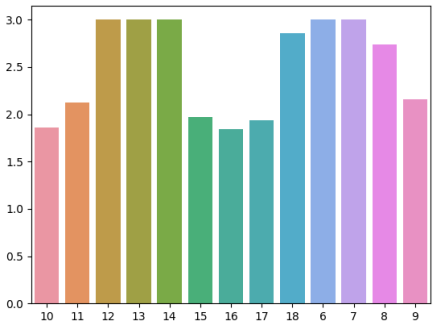

Figura 47: Parque 1 (Treino).

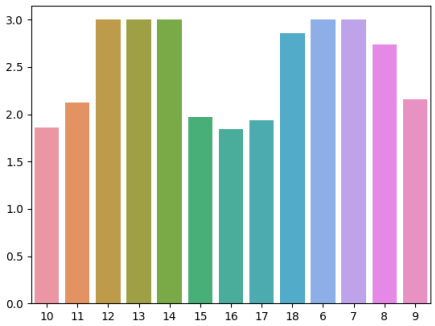

Figura 48: Parque 1 (Teste).

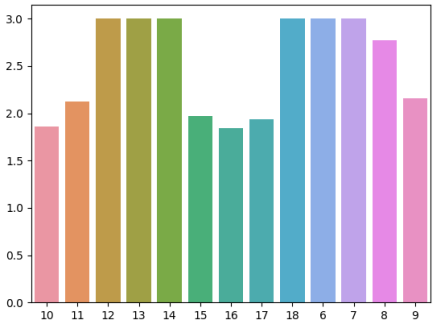

Figura 49: Parque 2 (Treino).

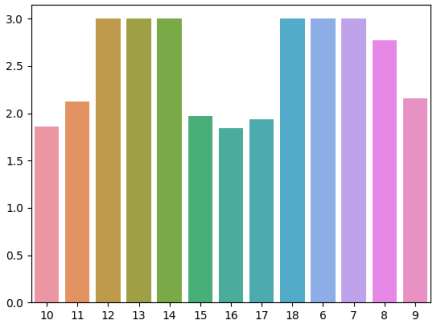

Figura 50: Parque 2 (Teste).

Já em relação aos parques 3 e 4, embora existam diferenças em determinadas horas, a média está perto do intervalo de mais de 10, para qualquer hora, figuras 51, 52, 53, 54.

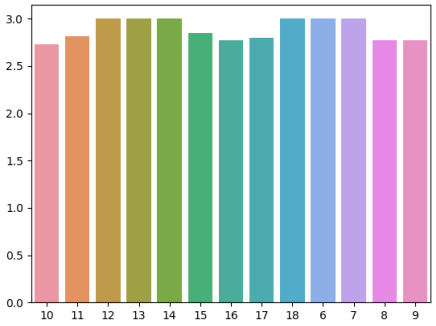

Figura 51: Parque 3 (Treino)

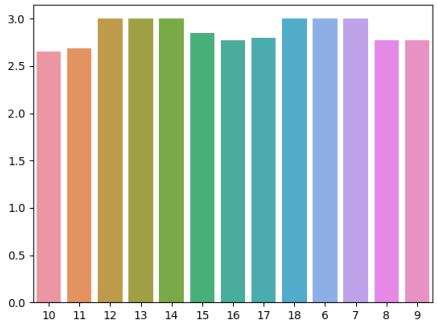

Figura 52: Parque 3 (Teste)

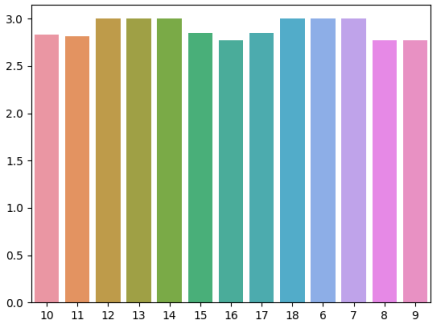

Figura 53: Parque 4 (Treino)

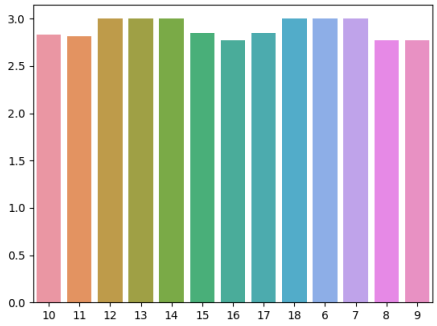

Figura 54: Parque 4 (Teste)

4.5.1 Resultados

1. Modelo de avaliação

   Como o objetivo da aplicação é recomendar um lugar de estacionamento, nem todos os erros têm a mesma importância, por isso, em vez de se avaliar de forma binária, certo ou errado, é mais interessante avaliar pela distância entre o real e a previsão, calculando o valor **previsto - valor real**. Assim podem ser obtidos os seguintes resultados:

   - *0*: quando o modelo acertou no intervalo
   - *1*: quando o modelo indicou que havia mais lugares do que o real, com um erro de um intervalo

- *-1*: quando o modelo indicou que havia menos lugares do que o real, com um erro de um intervalo
- *2*: quando o modelo indicou que havia mais lugares do que o real, com um erro de dois intervalos
- *-2*: quando o modelo indicou que havia menos lugares do que o real, com um erro de dois intervalos
- *3*: quando o modelo indicou que havia mais lugares do que o real, com um erro de três intervalos
- *-3*: quando o modelo indicou que havia menos lugares do que o real, com um erro de três intervalos

Dependendo do objetivo, pode ser considerado pior uma previsão com um número de lugares superior ao real, ou inferior ao real, uma vez que superior pode levar a que um utilizador vá para um estacionamento que não tem lugares, já o contrário dificulta a possibilidade de um estacionamento ser recomendado. Contudo, o importante é haver lugar, ou não, por isso foi feita uma avaliação complementar, sendo que apenas são consideradas duas classes, não haver lugar e haver lugar, descrevendo esses resultados numa matriz de confusão.

2. Modelo principal

O modelo principal foi treinado com todos os dados de treino, sendo capaz de prever qualquer parque, em qualquer circunstância (Figura 55). Avaliando a distância da previsão com o valor atribuído (0, 1, -1, 2, -2,3, -3), verifica-se que em 74% das vezes acertou, ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 26% de erro 56% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega aos 93%.

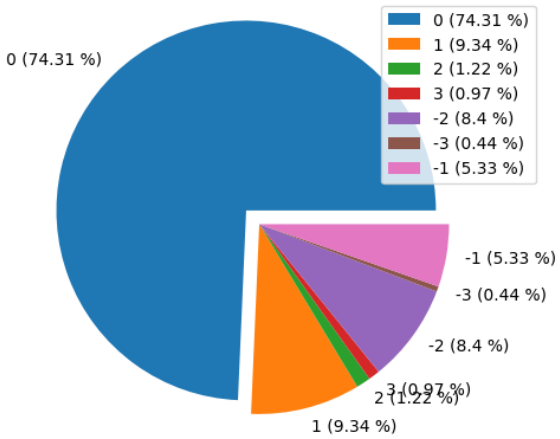

Figura 55: Modelo com todos os dados.

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 91% das vezes, indicando erradamente que há lugar em 8% e erradamente como não havendo lugar em 1%, tabela 9.

classificação	valor
TP	2694
TN	592
FP	299
FN	35

Tabela 9: Matriz de confusão do modelo principal.

3. Modelo do grupo 1

O modelo treinado apenas com os dados dos parques pertencentes ao seu grupo, em vez de ser um modelo criado para classificar todos os estacionamentos, é um modelo especializado em detetar os padrões de um certo conjunto de estacionamentos (Figura 56). Avaliando a distância da previsão

ao valor real, verifica-se que em 76% das vezes acertou, ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 24% de erro, 44% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega aos 77%.

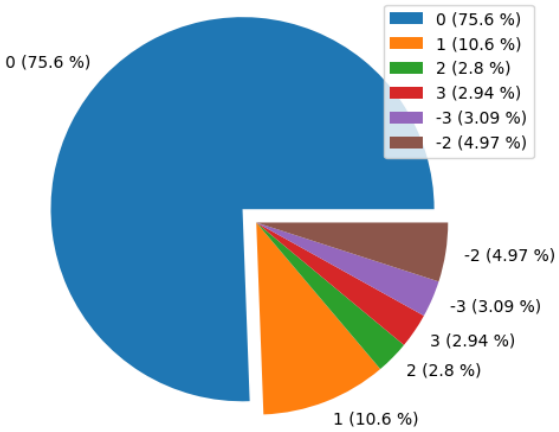

Figura 56: Modelo do grupo 1.

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 89%, indicando erradamente que há lugar em 8% e erradamente como não havendo lugar em 3%, Tabela 10.

classificação	valor
TP	2021
TN	397
FP	219
FN	80

Tabela 10: Matriz de confusão do grupo 1.

4. Modelo do grupo 2

Avaliando a distância da previsão ao valor real, verifica-se que em 62% das vezes acertou, ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 38% de erro, 49% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega aos 100%, Figura 57.

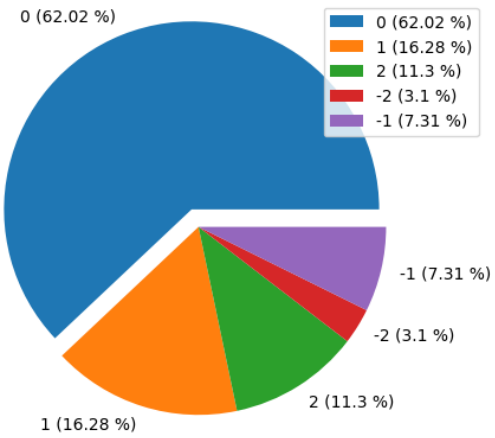

Figura 57: Modelo do grupo 2

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 91%, indicando erradamente que há lugar em 9% e erradamente como não havendo lugar em 0%, .

classificação	valor
TP	674
TN	150
FP	79
FN	0

Tabela 11: Matriz de confusão do grupo 2.

5. Modelo do parque 1

O modelo treinado apenas com os dados pertencentes ao seu parque, em vez de ser um modelo criado para classificar todos os estacionamentos, é um modelo especializado em detetar os padrões de apenas um parque de estacionamento.

Avaliando a distância da previsão ao valor real, verifica-se que em 54% das vezes acertou ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 46% de erro, 70% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega aos 100%, Figura 58.

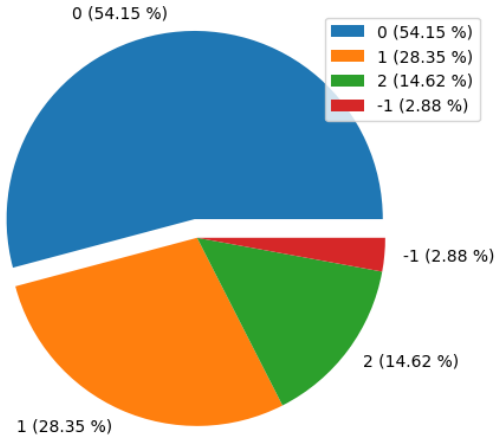

Figura 58: Modelo do parque 1.

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 92%, indicando erradamente que há lugar em 0% e erradamente como não havendo lugar em 8%, Tabela 12.

classificação	valor
TP	753
TN	81
FP	0
FN	69

Tabela 12: Matriz de confusão do parque 1

6. Modelo do parque 2

Avaliando a distância da previsão ao valor real, verifica-se que em 53% das vezes acertou ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 47% de erro, 7% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega apenas aos 46%, Figura 59.

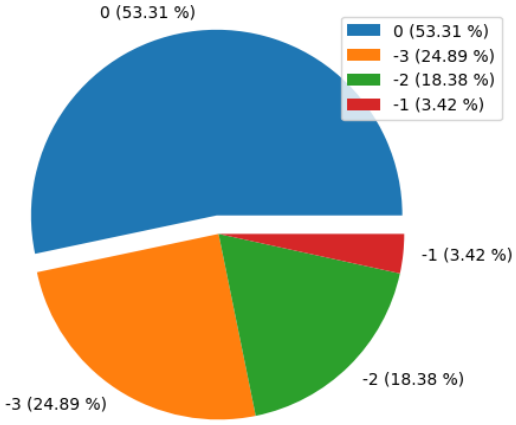

Figura 59: Modelo do parque 2

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 53%, indicando erradamente que há lugar em 47% e erradamente como não havendo lugar em 0%, Tabela 13.

classificação	valor
TP	57
TN	442
FP	437
FN	0

Tabela 13: Matriz de confusão do parque 2.

7. Modelo do parque 3

Avaliando a distância da previsão ao valor real, verifica-se que em 82% das vezes acertou ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 18% de erro 75% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega apenas aos 88%, figura 60.

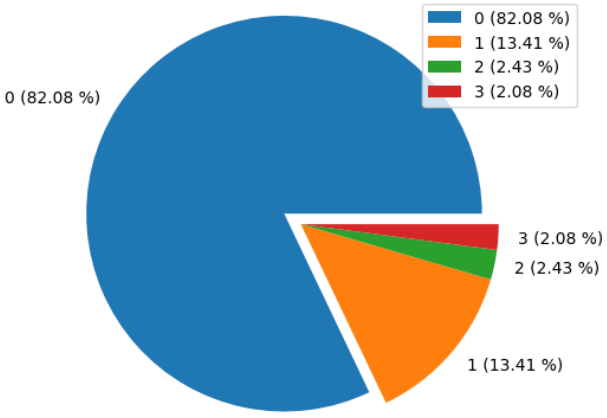

Figura 60: Modelo do parque 3.

classificação	valor
TP	847
TN	0
FP	0
FN	18

Tabela 14: Matriz de confusão do parque 3.

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 98%, indicando erradamente que há lugar em 0% e erradamente como não havendo lugar em 2%, tabela 14.

8. Modelo do parque 4

Avaliando a distância da previsão ao valor real, verifica-se que em 81% das vezes acertou ao enquadrar o estacionamento no seu devido intervalo. Sendo que dos 19% de erro 76% corresponde ao erro de apenas um intervalo, e considerando o erro de dois intervalos, este chega apenas aos 91%, figura 61.

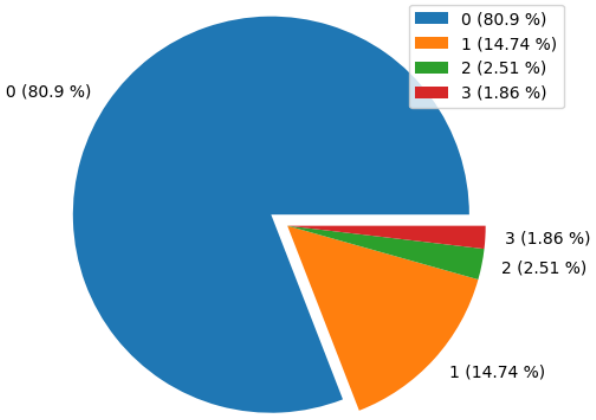

Figura 61: Modelo do parque 4

Analisando binariamente se o estacionamento tem ou não tem lugar, o modelo acerta em 98%, indicando erradamente que há lugar em 0% e erradamente como não havendo lugar em 2%, tabela 15.

classificação	valor
TP	899
FN	17
FP	0

Tabela 15: Matriz de confusão do parque 4.

Como conclusão desta análise de dados gerados pelo *script*, pode dizer-se que os resultados obtidos pelo modelo de *Machine Learning* são bastante fiáveis e poderão ser usados com segurança como dados dados iniciais de previsão quando um novo parque é adicionado à aplicação (a ser desenvolvida no próximo capítulo).

Deve ainda ser salientado que estes dados simulados foram ainda comparados com a realidade, ou seja, foram recolhidos dados manualmente, sobre parques de estacionamento com as características simuladas. Os resultados obtidos foram excelentes, tendo em conta que são usados intervalos de lugares.

4.6 Previsão do tempo de espera

Um dos fatores utilizado na escolha dos estacionamentos a sugerir é a estimativa de tempo de espera de um condutor até encontrar um lagar de estacionamento vago. Em certas situações, mesmo que o estacionamento se encontre lotado, dependendo do fluxo de veículos do parque (número de veículos que entram e saem do parque, num certo período de tempo) pode ainda ser um parque a considerar.

Este fator tem uma especial utilidade em diferenciar estacionamentos que estão constantemente lotados. Por um lado temos parques com veículos estacionados durante longos períodos de tempo, por outro lados temos parques com uma lotação próxima da sua lotação máxima, porém, a permanência de um veículo no parque é de tempo reduzido, o que permite um condutor conseguir estacionar em tempo útil.

Como tal, o que realmente importa, é classificar o parque de estacionamento, de forma a quantificar se é um parque onde frequentemente veículos entram e saem, e como tal deve ser valorizado, ou se a frequência é baixa, e como tal, deve ser desvalorizado, e não deve ser sugerido ao utilizador.

Uma forma de classificar os parques mediante o tempo de espera é através do cálculo da alternância de lugares.

Analisando os dados de um estacionamento é possível perceber se é um estacionamento onde o estado muda frequentemente ou não.

A estratégia adoptada é: construir um gráfico que contenha apenas lotação e tempo, onde a derivada do mesmo corresponde ao valor da discrepância da lotação (variação da lotação) numa certa altura do dia. Para isso basta considerar os dados referentes à lotação e ao tempo, isto é **ano**, **mês**, **dia**, **hora**, e **minuto**, sendo mais importantes o dia da semana, hora e minuto, desse mesmo dia. Para isso são criados para cada estacionamento, um gráfico por cada dia da semana, onde os eixos corresponderiam à lotação e ao tempo, considerando apenas horas e minutos.

É necessário tratar os dados para que essas análises possam ser feitas de forma correta. Um dos pontos essenciais é o tratamento de colisões, pois com tantos registos, na mesma hora e minuto, vão surgir registos com valores diferentes. Para resolver este problema existem diversas opções: uma simples seria calcular a média dos intervalos mas, caso se pretenda dar primazia à alternância de estado, pode-se analisar os momentos anteriores e posteriores e calcular o valor que maximiza a soma das distâncias ao valor anterior e ao valor posterior, isto é, para calcular o valor da lotação no instante t_i será encontrado de todos os valores t_i registados, aquele que maximiza, a alternância do momento calculado anterior t_{i-1}, e posterior t_{i+1}. Uma vez que o valor t_{i+1} poderá ter também ele colisões, será naturalmente considerado o valor com maior diferença a t_i.

Outro tratamento que é feito diz respeito à filtragem de determinados dados, nomeadamente, dados consecutivos com a mesma lotação, uma vez considerados, podem levar a interpretações erradas (tendo em conta que apenas se considera o ponto de lotação imediatamente anterior).

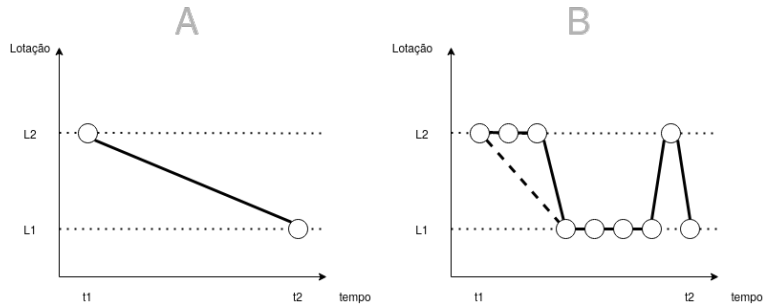

Figura 62: Exemplo que ilustra a razão pela qual deve ser feita uma filtragem dos dados.

Basta imaginar dois estacionamentos A (estacionamento com taxa de atualização baixa e com um tempo de espera elevado) e B (estacionamento com ma taxa de atualização alta e com um tempo de espera baixo). No entanto, se considerarmos apenas o ponto anterior (sem qualquer filtragem), no estacionamento A, a informação que obtemos é que há uma certa alternância de lugares (apesar de atenuada pelo passar do tempo). Por outro lado, no estacionamento B, podemos concluir, que este em certas ocasiões conta com uma alternância menor (o que é errado pois o estacionamento B conta com todos os pontos de A).

Para resolver esses problemas, basta filtrar todos os valores consecutivos com a mesma lotação.

A comparação de discrepâncias será executada através do somatório do percurso que resulta da união dos pontos de um gráfico filtrado. Assim, é possível construir um gráfico real, que condense a informação referente a um estacionamento, num determinado dia da semana. Isto permite calcular o valor que simboliza a variação da lotação, apenas através da derivada, no gráfico respectivo.

Estes gráficos apenas precisam de ser calculados uma vez, podendo ser calculados novamente, caso se pretenda refinar os dados.

5 Aplicação Móvel

Nesta dissertação, desenvolveu-se uma aplicação móvel, a qual tem como objetivo principal informar os seus utilizadores sobre lugares disponíveis nos diversos parques de estacionamento.

A aplicação móvel é o ponto de partida para todos os utilizadores que vão interagir com o sistema de controlo de ocupação dos lugares de estacionamento dos diversos parques. Esta será responsável por divulgar as informações sobre os lugares livres de estacionamentos aos utilizadores, funcionando como um intermediário entre todos os pontos de recolha e obtenção de informação (servidores, sensores, câmaras inteligentes e dos próprios utilizadores).

A aplicação móvel apresentará as informações mais pertinentes, para que o utilizador possa escolher o melhor possível o lugar onde irá estacionar a sua viatura. Para isso, a aplicação recebe e interpreta todos os pedidos e ações do utilizador, comunica com o servidor, e apresenta as informações enviadas de forma apelativa e simples, para que facilmente o utilizador consiga escolher o parque de estacionamento no qual irá tentar estacionar.

Para além de divulgar informações, a aplicação permite ao utilizador adicionar novos estacionamentos e informações sobre o número de lugares livres nos parques, permitindo melhorar a aplicação. Assim, quer a comunicação entre utilizadores, quer a criação de bases de dados mais ricas, irão permitir criar um padrão eficaz de estacionamento ao longo do tempo. Desta forma, serão criados mais dados sobre os estacionamentos que não possuem um sistema de comunicação com a aplicação móvel .

A aplicação poderá ser usada de duas formas distintas: ou de forma simples como um **GPS**, em que o condutor pode escolher o estacionamento e seguir o percurso indicado pela aplicação; ou como uma ferramenta para encontrar o estacionamento que melhor se adequa às necessidades do utilizador, caso em que a aplicação mostra um mapa, onde os estacionamentos são apresentados. Juntamente com uma descrição de cada estacionamento, são ainda fornecidas informações referentes à possibilidade de se conseguir arranjar lugar assim como ao tempo esperado para que um lugar fique livre.

É possível indicar o destino, através da rua e nome do local, ou através da localização no mapa, e indicar preferências, para que a sugestão seja personalizada mediante as necessidades do utilizador.

5.1 Tecnologias usadas

Esta aplicação pretende ser "amigável"e tem como foco a simplicidade e a praticabilidade, de forma a que qualquer pessoa a consiga utilizar e tirar benefícios na ajuda à procura de um lugar livre de estacionamento. Quanto mais pessoas usarem a aplicação, melhores serão os resultados. As sugestões dadas ao utilizador são alimentadas pelo *feedback* de outros utilizadores.

De modo a tornar a aplicação apelativa, os estacionamentos devem ser apresentados de forma clara e intuitiva, e a informação sobre os parques disponíveis deve estar acessível a partir de qualquer dispositivo. Para tal, foi escolhido o *framework Flutter*, conhecido pelo seu uso na construção de páginas *web* apelativas e possuindo apenas um código para gerar aplicações para as mais diversas plataformas, i.e. **Android**, **IOS**, **Windows**, **Mac Os**, **Linux**, e até **Web**).

5.1.1 Diferentes abordagens para a criação de aplicações móveis

Existem diferentes abordagens para a criação de aplicações móveis. Aqui destacam-se algumas, para as quais se apresenta uma descrição sucinta das suas principais características.

- Aplicações Nativas. Consistem em utilizar as linguagens oficiais de cada sistema operativo (por exemplo, Android e IOS). No caso do Android, as opções oficiais fornecidas pela empresa da *Google* são **Java** e **Kotlin**. No caso da *Apple*, as linguagens são **objet C** e **Swift**. Usando as linguagens nativas, é possível alcançar uma melhor performance. No entanto, exige-se um maior esforço para o desenvolvimento da aplicação, sendo ainda necessário desenvolver um código para cada sistema.

- Aplicações Híbridas. São aplicações que têm uma parte nativa e outra parte web. Toda a parte lógica e visual é escrita através da parte *web*. No entanto, quando a aplicação necessita aceder a uma parte nativa (câmara, localização, entre outros), esta comunica com a parte nativa da aplicação para que ela possa servir de intermediária no processo. Assim, é possível, com apenas um código, que sejam geradas aplicações para os dois sistemas operativos (Android e IOS). Um exemplo é o **IONIC**.

- Aplicações PWA. Têm uma abordagem semelhante às híbridas. No entanto, estas contam apenas com a parte *web*. São apenas páginas *web*, com o aspecto de aplicações mobile e, como tal,

podem ser usadas nas diferentes plataformas. No entanto, ao contrario das opções anteriores, não é possível aceder a partes nativas do equipamento, ficando restrita aos dados que o *browser* consegue transmitir. Estas aplicações, devido ao seu formato, não podem ser distribuídas através de lojas como *Play Store* e *App Store*.

- Aplicações Nativas compiladas. Permitem com um único código gerar aplicações para as duas plataformas, mas de uma forma mais eficiente. Estas abordagens, apesar de usarem a linguagem própria da solução escolhida, recorrem a um compilador e a um tradutor que as irá traduzir para uma linguagem interpretada pelos equipamentos a que se destinam. Estas opções, além das vantagens já descritas, estão repletas de opções que permitem criar facilmente uma aplicação eficiente e apelativa.

  Estas aplicações têm sido bastante usadas, devido às opções **React Native**, do Facebook, e, mais recentemente, devido ao **Flutter**, da Google.

Dadas as características da aplicação móvel que se pretende desenvolver nesta dissertação, iremos adoptar a abordagem das aplicações nativas compiladas para o desenvolvimento da aplicação. A principal razão desta escolha deve-se ao facto de as aplicações nativas compiladas permitirem o uso da linguagem nativa e ao mesmo tempo comunicarem com as diversas plataformas.

Para se desenvolver a aplicação poderá ser usado o **React Native** ou o **Flutter**. O **React Native** conta com uma comunidade maior, o que leva a que haja mais conteúdo e bibliotecas disponíveis, além de ser desenvolvido com a linguagem *java script*, que por si só já conta com um grande número de bibliotecas. Porém, o **Flutter** é uma solução mais sólida, a documentação existente é muito completa, e o projecto está melhor organizado, usando por base a linguagem *dart*, criada também pela Google, o que permite moldar a linguagem às necessidades do framework. Esta linguagem conta com um conjunto elevado de bibliotecas e módulos.

Esta framework permite ainda que se gerem aplicações para *Desktop* (Linux, Mac os e Windows) e *web* (gerar um site), funcionando num *browser*. Será então usada a *framwork flutter* para o desenvolvimento da aplicação.

5.1.2 Pacotes usados no flutter

O *Flutter* é um pacote de desenvolvimento de interfaces gráficas (UI), multi plataforma, sendo um [4]*framework* de **Dart**, uma linguagem de código aberto criada pela equipa de desenvolvimento da Google, em 2011. Esta linguagem não teve grande sucessor até ter sido implementada no *Flutter*. Desde então, tanto os programadores independentes, como a equipa de desenvolvimento do **Dart/Flutter**, têm tornado a linguagem muito mais evoluída, para a qual foram criadas muitas bibliotecas, tornando-a numa linguagem robusta. Todas essas bibliotecas podem ser encontradas em `https://pub.dev/`. As principais bibliotecas e módulos usados nesta dissertação são:

1. google_mapsflutter

   Módulo que permite a exibição no ecrã do mapa do Google Maps. Além de se instalar o módulo, é necessário ter uma conta Google e gerar uma chave para a sua API.

2. search_mapplace

   Permite que um local possa ser procurado, escrevendo o nome do local, das ruas e das cidades. Este módulo tem *auto-complete*, que sugere locais para que o utilizador não precise de escrever a morada por completo.

3. polyline

   Módulo usado para mostrar o percurso (*polyline*) desde o local onde o utilizador se encontra até ao parque de estacionamento escolhido.

4. Geolocator

   Módulo que fornece um conjunto de ferramentas para se trabalhar com coordenadas.

5. dio

   Uma vez que a aplicação comunica com um servidor, é necessário um módulo que torne possível a comunicação **HTTP**. O *dio* é um excelente módulo, permitindo que sejam enviadas várias requisições, de vários tipos, inclusive *multi-part-form*, essencial para enviar imagens para o servidor.

[4]Conjunto de classes e funções, implementadas numa linguagem de programação específica, que auxilia o desenvolvimento, ditando o fluxo e padrões da aplicação.

5.2 Comunicação com o servidor

5.2.1 Obter os estacionamentos

Quando um utilizador escolhe o seu destino, é feita uma requisição **GET** ao servidor com a rota principal, **/** , contendo os filtros da Tabela 16.

Filtro	definição
x	coordenada x
y	coordenada y
free	Apenas grátis
covered	Apenas parques cobertos
values	Numero de Parques
community	Se foi um utilizador que introduziu o parque
verificated	Se o parque foi validado
power_station	Se existe uma estação de carregamento, para carros elétricos

Tabela 16: Atributos enviados ao servidor para obter os estacionamentos disponíveis.

O servidor responde com uma lista de n parques, onde n é o atributo *values*, e onde cada parque é um objeto *JSON* com os atributos da tabela 17.

Filtro	definição
_id	coordenadas
lots	Número de lugares livres
community	Se foi um utilizador que introduziu o parque
free	Apenas grátis
images	Imagens de um parque
covered	Apenas parques cobertos
power_station	Se existe uma estação de carregamento, para carros elétricos
verificated	Se o parque foi validado

Tabela 17: Resposta ao pedido de listagem dos estacionamentos.

5.2.2 Adicionar um estacionamento

Quando um utilizador adiciona um parque na aplicação, é enviada uma requisição **POST** ao servidor com a rota /. O corpo da requisição contem os atributos enunciados na Tabela 18.

Atributo	definição
free	Apenas grátis
covered	Apenas parques cobertos
power_station	Se existe uma estação de carregamento, para carros elétricos
state	Número de parques
file	Imagem do parque

Tabela 18: Corpo da requisição da rota de adicionar um estacionamento.

Como se trata de adicionar um novo estacionamento ou um novo parque, deverá ser feita uma validação. Como tal, é obrigatório que o utilizador envie a imagem do parque, para que posteriormente a sua introdução possa ser validado. Caso a imagem não seja enviada, a requisição é ignorada.

5.2.3 Adicionar informações a um estacionamento já existente

Quando um utilizador adiciona uma informação nova sobre um parque, é enviada uma requisição **PUT** ao servidor com a rota **/:id**, onde o corpo da requisição tem os atributos da Tabela 19.

Atributo	definição
free	Apenas grátis
covered	Apenas parques cobertos
power_station	Se existe uma estação de carregamento, para carros elétricos
state	Número de parques
file	Imagem do parque

Tabela 19: Corpo da requisição da rota de adicionar uma informação a um estacionamento.

Neste caso, ao contrário do anterior, não é obrigatório enviar uma imagem do parque de estacionamento, sendo esse envio opcional.

5.3 Interação com o utilizador

Sendo a aplicação a ferramenta a usar por todos os utilizadores para usufruírem do serviço de procura de estacionamento, é essencial que a aplicação seja simples e intuitiva, de modo a que qualquer pessoa com o mínimo de experiência na utilização de aplicações possa navegar e aceder a todas as suas funcionalidades de forma natural.

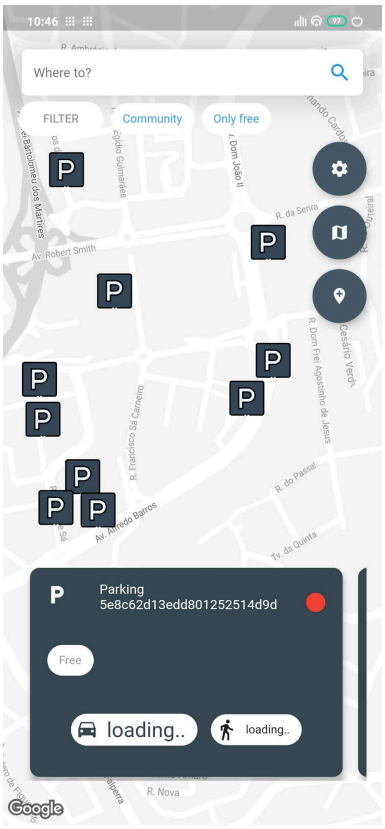

Figura 63: Ecrã principal da aplicação.

Quando o utilizador inicia a aplicação (Figura 63), é-lhe apresentada uma tela bastante simples, onde o destaque vai para um mapa que ocupa o ecrã inteiro. Na parte superior, consta uma barra de pesquisa e alguns botões de opções. Na parte direita surgem os botões de navegação e controlo da aplicação.

A barra de pesquisa é uma caixa onde o utilizador pode escrever o destino. À medida que vai escrevendo, são dadas sugestões através do *auto complete*. Quando o utilizador acaba de escrever ou clica em uma das sugestões, o destino é definido.

Também é possível escolher o destino através do mapa. Para isso, o utilizador apenas precisa de pressionar no lugar para onde pretende ir, e para se mover no mapa basta arrastar o mapa com um dedo em qualquer uma das direcções. Se o utilizador pretender mudar o ângulo de visão ou alterar o *zoom*, apenas tem que fazer gestos de rotação ou de pinça com dois dedos, respectivamente.

Uma vez escolhido o destino, aparece na parte inferior um conjunto de 10 cartões contendo as informações mais relevantes de parques de estacionamento. Um cartão ocupa a largura do ecrã, para que

seja fácil ler as informações que nele constam, e para que se vejam outros cartões, basta deslizar o dedo horizontalmente na zona dos cartões.

Um cartão contém informações como a morada, se é grátis ou a pagar, se é coberto, se tem carregamento para veículos eléctricos, a distância a que se encontra o estacionamento viajando de carro, a distância do estacionamento até ao destino a pé, e um indicativo da possível lotação do parque. Essa indicação ocorre no formato de um pequeno círculo, onde vermelho significa que a probabilidade de haver lugar é baixa, laranja indica que poderá haver um lugar, amarelo indicando que o mais provável é haver lugar, e verde que o mais certo é haver lugar, podendo ter mais que um.

Quando o utilizador clica num destes cartões, é-lhe mostrado o percurso desde a sua localização até ao destino, Figura 64.

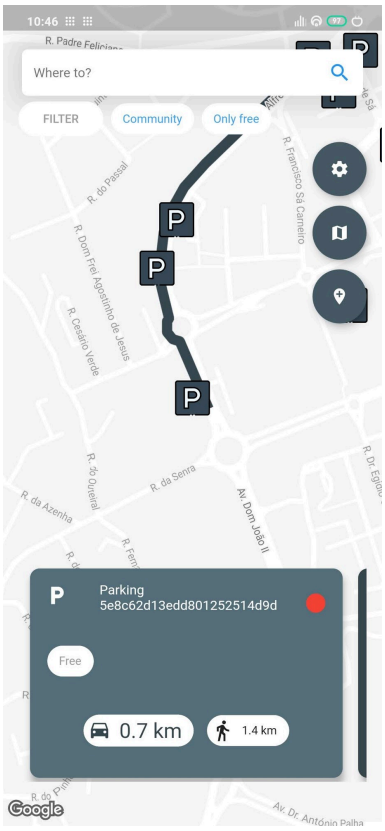

Figura 64: Parque escolhido e respetivo trajecto.

Se o utilizador pretender obter mais informações sobre o estacionamento, basta pressionar durante algum tempo o cartão.

Tal como referido anteriormente, abaixo da barra de pesquisa surgem 3 botões, um botão com os filtros que, quando clicado, mostra ao utilizador uma janela onde pode configurar todos os filtros que pretende na aplicação, e dois botões com filtros rápidos (mostrar parques grátis, e permitir ou não a listagem de parques adicionados pela comunidade).

No lado direito surgem 3 botões. Um que redireciona o utilizador para o ecrã de definições, outro que permite que se veja o mapa em modo satélite, e outro que permite que sejam adicionados novos parques, Figuras 65 e 66.

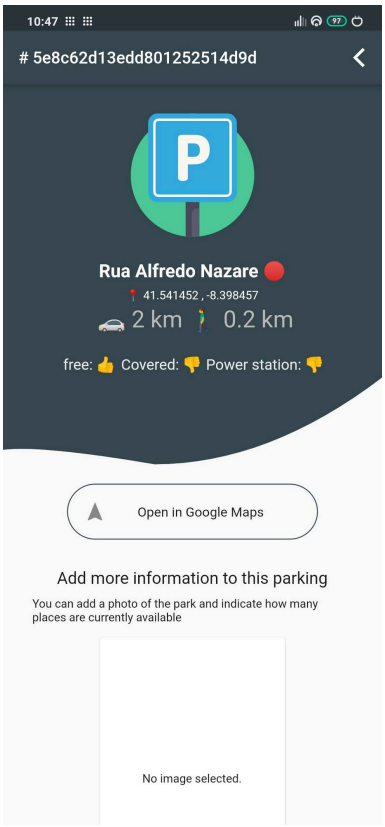

Figura 65: Página de um parque de estacionamento.

A Figura 65 mostra o menu onde o utilizador consegue consultar todas as informações de um dado parque. Esta página é na realidade constituída por três partes. Uma que contém as informações do parque de estacionamento e que possibilita que sejam adicionadas novas informações sobre o parque. Neste menu existem ainda 4 botões para que o utilizador possa indicar qual é o estado no parque, tendo um botão para indicar se não existem lugares vagos, se tem 0 a 3 lugares vagos, se tem 3 a 10 lugares e por

último, se tem mais que 10 lugares. Outra parte contém todas as fotografias do parque, e outra indica onde é possível escrever uma mensagem de *report* (o utilizador pode escrever uma mensagem para informar algum problema sobre algum dado do estacionamento, ou indicar um problema ou sugestão).

Figura 66: Página para adicionar um parque de estacionamento.

Tendo em conta a impossibilidade de todos os estacionamentos serem introduzidos pelos criadores da aplicação, a funcionalidade de *adicionar parque* (figura 66) tenta solucionar esse problema. O utilizador pode introduzir um novo parque de estacionamento. Para isso é necessário estar no local, tirar uma fotografia e mencionar o estado do parque, tal como mostrado na figura 66.

5.4 Problemas e soluções

5.4.1 Informações falsas sobre o parque de estacionamento

Uma vez que os utilizadores têm a possibilidade de adicionar parques de estacionamento, podem adicionar os parques em locais onde não existe qualquer parque de estacionamento.

- Solução

  Para que um utilizador consiga adicionar um estacionamento, ele deve enviar uma fotografia do local, para que mais tarde o parque possa ser verificado. Até ser verificado, o parque não aparece ao utilizadores; no entanto, é possível ter acesso a todos os estacionamentos neste estado, se a opção **apenas parques verificados** não estiver seleccionada (que por defeito está).

5.4.2 Informações falsas sobre o estado

Em vários parques, a única informação sobre a lotação dos parques é dada pelos utilizadores. No entanto, é possível que a informação recebida não seja verdadeira.

- Solução

  Para que nem todos os dados compartilhados tenham a mesma importância, é-lhes associado um fator de confiança. Esse fator aumenta com base na quantidade de vezes que o utilizador compartilhou informações verdadeiras, e diminui quando se suspeita de estarem erradas. Para avaliar se uma certa lotação é certa ou não, esta é comparada com o histórico anterior, sendo avaliado se há alguma discrepância. A confiança aumenta num maior grau, quando a informação compartilhada por um utilizador é verificada por outro. Isto ocorre quando, num curto espaço de tempo, outro utilizador, passa pelo mesmo estacionamento, e indica uma lotação que é plausível, tendo em conta o tempo que passou desde a última informação fornecida sobre esse parque.

5.4.3 Imagens que podem conter informação sensível

A aplicação permite aos utilizadores publicarem imagens do estacionamento, que podem conter dados sensíveis ou privados, e conteúdos de divulgação não apropriada.

- Solução

  As imagens, quando recebidas pelo servidor, podem passar por um filtro inteligente, que é capaz de identificar certos objetos sensíveis na imagem, adicionar um desfoque ou uma faixa preta em certas regiões da imagem, por exemplo sobre uma matrícula ou sobre rostos de pessoas, entre outras.

5.4.4 Estacionamentos esquecidos

Estacionamentos que, frequentemente, não têm lugares vagos podem levar o modelo a generalizar que nunca existem lugares vagos nesse local e, como tal, não recomendar mais o local aos condutores. Uma vez que o estacionamento não é mais sugerido, não se obtêm mais dados do local, e o mesmo continuará sem ser recomendado.

- Solução

  Um estacionamento pode ter frequentemente a sua lotação próxima da sua capacidade máxima, mas o importante é identificar quais são os fatores que levam a tal situação. Acrescentar informações sobre o parque, assim como indicar pontos de interesse próximos desse parque, será fundamental para que esse parque possa ser comparado com os restantes. Será fundamental identificar informações que possam evidenciar ocasiões especiais ou determinadas situações ou padrões ao longo do dia, de modo a que a probabilidade de haver lugar neste parque aumente.

  Para garantir que o parque não fica sem informações, será indicado um percurso a seguir para procurar lugares. Assim, mesmo que o estacionamento não surja com probabilidade alta de ter lugares livres, pode ser recomendado passar por ele, porque fica a caminho de outros parques, e assim recolher mais informações sobre o seu estado em termos de lugares disponíveis, e garantir que ele não é esquecido.

5.5 Sugestões da aplicação para o utilizador

Esta aplicação apresenta um conjunto de características que a distinguem das restantes aplicações desenvolvidas com o propósito de ajudar o condutor a encontrar um lugar de estacionamento.

Esta aplicação possui um filtro inteligente que tem em conta as preferências e as necessidades do utilizador, e somente oferece um conjunto de 10 estacionamentos que melhor corresponde ao pedido do utilizador. Além de serem consideradas as características dos estacionamentos, é feita uma filtragem com base no historial de lotações apresentado até ao momento da pesquisa, e a sua tendência (dados adquiridos através da informação cedida pelos utilizadores, através da inteligência artificial e através dos sistemas de reconhecimento de lugares livres).

Esta abordagem, além de melhorar a experiência de utilização, tornando-a mais simples e rápida, devido ao cruzamento e análise de um grande volume de dados, permite ao utilizador escolher o melhor estacionamento, usando informação privilegiada fornecida pela aplicação.

5.5.1 Sugestões a fornecer ao utilizador

Existem várias abordagens possíveis para realizar uma sugestão ao utilizador.

- Sugestão por características

  Está é a sugestão mais simples. Normalmente denominada por **Content-Based Filtering Recommender**, esta é uma sugestão feita com base nas características do produto a recomendar (neste caso, parques de estacionamento). Os estacionamentos contam com um conjunto de informações que os caracterizam (por exemplo, a sua localização, se é grátis, se é coberto). Estas características podem ser usadas para se realizar uma sugestão ao utilizador, como por exemplo, sugerir os estacionamentos mais próximos e que vão de encontro às preferências do utilizador.

- Sugestão colaborativa

  Outra abordagem, normalmente denominada por **Colaborative Filtering Recommender**, é uma recomendação em função das escolhas dos restantes utilizadores, e as suas semelhanças de perfis e gostos por determinados produtos (parques de estacionamento, neste caso). O seu funcionamento é simples. Cada utilizador tem um perfil, no qual são armazenadas informações sobre as suas preferências/escolhas, e a estas é associado um valor de satisfação. Assim, a sugestão é feita, encontrando perfis próximos dos do utilizador, e serão escolhidos os produtos (parques) que melhor satisfazem os demais utilizadores (com perfis semelhantes ao utilizador em questão), ou seja, os

parques onde os condutores conseguiram estacionar o veículo.

- Sugestão híbrida

  Como o nome sugere, esta abordagem corresponde a uma conciliação das abordagens anteriores. Dito de outro modo, a recomendação tem como base as características dos parques e o histórico das escolhas anteriores por utilizadores com perfil semelhante, bem como o facto de terem conseguido estacionar ou não. Assim, é possível apresentar uma solução mais robusta, sendo mais precisa que a sugestão por características, e sem a limitação da sugestão colaborativa, que exige um grande volume de dados para que se consiga atingir bons resultados (recomendações de acordo com os gostos do utilizador).

Dentro destes três grandes grupos gerais de sugestões, podem ainda ser dadas as seguintes sugestões particulares de estacionamentos:

1. Sugerir os estacionamentos mais próximos do destino

   Uma das sugestões mais simples, enquadrada nas sugestões por características, consiste em apresentar os estacionamentos mais próximos do destino. Esta abordagem funciona bem quando os estacionamentos em questão têm habitualmente lugares disponíveis. Esta abordagem pode ser aplicada mesmo quando não existem informações sobre o histórico de estacionamentos.

   O objetivo desta abordagem é o de encontrar os estacionamentos mais próximos do destino escolhido pelo utilizador. No entanto, a ordenação dos parques pode ser feita de diversas formas, priorizando a eficiência dos cálculos ou a sua precisão. No caso do foco ser a eficiência, o algoritmo usado, abdica de alguma precisão no seu cálculo, isto é, não calcula a real distância que seria percorrida pelo utilizador, porém, realiza o cálculo com uma menor complexidade, e assim responde a mais requisitos. Por outro lado, se o tempo de resposta não for um problema, o cálculo pode ser mais preciso, mesmo que para isso o custo computacional aumente.

   (a) Distância Euclidiana

   Esta é a abordagem mais simples, porém, a menos precisa e a menos eficiente. Calcula a distância através da norma Euclidiana. Sendo (x_1, y_1) as coordenadas do estacionamento e (x_2, y_2) as coordenadas do destino, a distância é dada por:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Para encontrar o estacionamento mais próximo, a complexidade do calculo é de $\mathcal{O}(N)$. Para seleccionar os 10 estacionamentos mais próximos, pode ser feita uma ordenação que tem o custo computacional de $\mathcal{O}(N \log N)$.

(b) Diagrama de Voronoi

Com o objetivo de reduzir o custo computacional e, assim, dar uma resposta mais rápida, o diagrama de Voronoi [31] é muitas vezes usado. Este diagrama permite encontrar o estacionamento mais próximo de um destino com um custo $\mathcal{O}(\log N)$. A redução da complexidade e custo é causada pela mais adaptada organização/estruturação da posição dos estacionamentos.

Na construção do diagrama de Voronoi, são consideradas apenas as coordenadas (pi) dos N estacionamentos.

$$\{P_i \mid i = 1, 2, \ldots, N\}$$

Para cada um destes N estacionamentos (pontos P_i), é construída uma célula c_i que delimita a região constituída pelos pontos cuja distância ao parque P_i é menor do que a distância a todos os outros parques $P_j \neq P_i$. Assim,

$$c_i = \{P \mid \text{ dist}(P, P_i) < \text{dist}(P, P_j), \quad \forall P, \quad \forall P_j \neq P_i, \quad i, j = 1, 2, ..., N\}$$

Uma das muitas formas de construir o diagrama de Voronoi, ou seja, o conjunto de células de Voronoi (ver a Figura 71), cuja união corresponde a todo o espaço considerado, é recorrendo ao algoritmo **Fortune**, que tem a complexidade/custo temporal $\mathcal{O}(N \log N)$, e $\mathcal{O}(N)$ espacial.

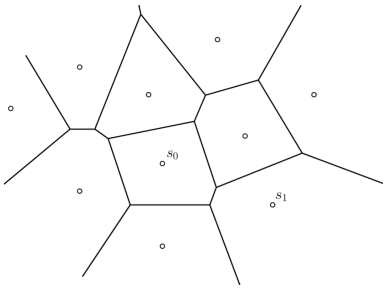

Figura 67: Diagrama de Voronoi.

(c) Distância geográfica

Uma outra abordagem, cujo foco é aumentar a precisão, é o cálculo da distância geográfica, através da fórmula de *Haversine*, como descrito em [32]. Esta distância tem em conta a curvatura da superfície terrestre do nosso planeta.

Os estacionamentos são definidos pelo par (latitude, longitude). Definindo o estacionamento e o destino por (latitude1, longitude1) e (latitude2, longitude2), respectivamente, a distância geográfica é dada por:

$$\text{distância} = Rc,$$

onde R corresponde ao raio da terra, que tem sensivelmente o valor de 6371 km, e c é dado por

$$c = 2\sin^{-1}(\sqrt{a}),$$
$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) \times \sin^2\left(\frac{\Delta\lambda}{2}\right) \times \cos(\varphi_1) \times \cos(\varphi_2)$$

com

$$\varphi_i = \frac{\text{latitude}i \times \pi}{180}$$
$$\Delta\varphi = \frac{(\text{latitude2} - \text{latitude2}) \times \pi}{180}$$
$$\Delta\lambda = \frac{(\text{longitude2} - \text{longitude2}) \times \pi}{180}$$

(d) Distância Real

Esta abordagem minimiza o erro quando comparada com as abordagens anteriores. Calcula a distância do estacionamento ao destino, usando o percurso com menor distância entre os dois locais. Isto é, considera as ruas, sentidos, e condicionantes, para que seja calculada a distância real de uma forma eficaz.

Para usar esta distância, é necessário armazenar a informação sobre as ruas e os locais dos estacionamentos, e organizar esta informação segundo um grafo pesado, no qual os vértices são os estacionamentos considerados, cada aresta representa a rua que liga os dois estacionamentos e o peso atribuído a cada aresta será a distância entre os estacionamentos representados pelos vértices que definem essa aresta. A distância entre dois vértices é dada pelo caminho com menor peso (menor valor da soma das distâncias das ruas que unem os dois lugares). Esta distância é encontrada através do algoritmo *Dijktra*, bem explicado em [33].

2. Sugerir estacionamentos tendo em conta a distância e a ocupação dos parques

   Além da distância do parque ao destino, pode ainda ser tida em conta a ocupação do parque.

   Em estacionamentos que contenham um sistema de reconhecimento de lugares livres, as leituras do mesmo podem ser consideradas para a eleição dos melhores estacionamentos. Nos parques que não possuem um sistema de reconhecimento de lugares de estacionamento, será usada a previsão obtida através da inteligência artificial, que irá indicar a probabilidade de existência de lugares vagos.

   Por uma questão de uniformização, os estacionamentos que contêm sistemas capazes de indicar, em tempo real, os lugares livres serão também submetidos a uma previsão.

   Depois de escolhidos os estacionamentos mais próximos, eles são submetidos a uma ordenação tendo em conta a importância da distância e da probabilidade de existência de lugares livres de estacionamento.

3. Sugerir os estacionamentos segundo a distância, possível ocupação, e menor tempo de espera

   Neste caso, além de se considerar a probabilidade de haver lugar de estacionamento, é também tido em conta o tempo de espera para obtenção de um lugar vazio, quando o parque está cheio.

Depois de escolhidos os estacionamentos mais próximos, eles são submetidos a uma ordenação tendo em conta a importância atribuída à distância, à probabilidade de haver lugar vago, e ao tempo de espera.

4. Sugerir os estacionamentos segundo as preferências demonstradas pelo utilizador

   Além de considerar a distância, a probabilidade de haver lugar e o tempo de espera, são também consideradas as preferências do utilizador.

   Estas preferências podem ser indicadas de forma explícita, através da aplicação, ou adquiridas de forma implícita, através da aprendizagem tendo em conta as escolhas anteriores.

   Os estacionamentos são ordenados, tendo em consideração essas preferências.

5.5.2 Arquitectura da Sugestão

A sugestão dada ao utilizador consiste num conjunto de 10 parques de estacionamentos ótimos, segundo critérios previamente definidos. A escolha dos parques de estacionamentos segue o seguinte processo, exemplificado na Figura 68.

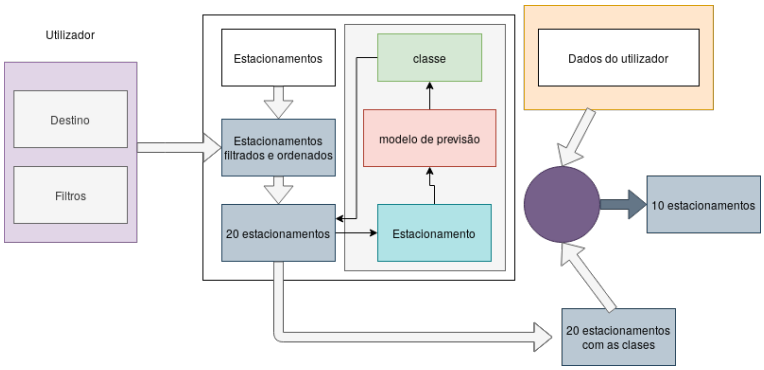

Figura 68: Arquitetura da sugestão.

Quando o servidor recebe um pedido de sugestão, são contempladas os seguintes parâmetros:

- Destino
- Filtros

  - Apenas estacionamentos com sistemas de detecção
  - Permitir estacionamentos adicionados por outros utilizadores
  - Apenas parques grátis
  - Apenas parques cobertos
  - Apenas parques com estações de carregamento
  - Apenas parques verificados

- Preferências explícitas
  - Priorizar a distância ao local
  - Priorizar a probabilidade de haver lugar vago

- Preferências implícitas

  Sugestões baseadas no histórico de escolhas do utilizador.

De acordo com estes parâmetros, são seleccionados unicamente 10 estacionamentos, através do seguinte processo.

1. Os estacionamentos são filtrados de acordo com os filtros definidos pelo utilizador.
2. Os estacionamentos que satisfazem os filtros escolhidos são ordenados em relação à distância ao destino.
3. São considerados apenas os 20 estacionamentos mais próximos do destino.
4. Às informações dos estacionamentos, é adicionada uma informação sobre o número de lugares disponíveis de momento. No caso de ser um estacionamento que não tenha um sistema de detecção, é feita uma previsão.
5. É adicionada às informação de cada estacionamento o tempo médio de espera (tempo que poderá demorar ate surgir um lugar vago), quando o parque se encontra lotado.

6. Ordenam-se novamente os estacionamentos, tendo em conta as novas informações adicionadas e de acordo com as preferências do utilizador.

   Esta ordenação tem como objetivo priorizar os diferentes parques de acordo com as escolhas do utilizador, porém, primeiramente, procede-se a uma associação de um peso/importância a cada característica do parque (sendo que cada parque é definido segundo as características descritas nas alíneas abaixo).

   (a) Probabilidade de ter lugar vago (p_P);

   (b) Tempo de espera (p_{Te});

   (c) Distância de carro até ao estacionamento (p_{Dce});

   (d) Distância a pé desde o estacionamento até ao destino (p_{Ded});

   (e) Se é grátis[5](p_G);

   (f) Se é coberto[5] (p_C);

   (g) Se contém uma estação de carregamento para carros eléctricos [5] (p_{Ec}).

   Após esta quantificação, segue-se com a ordenação dos parques, tendo em conta as preferências do utilizador, listadas abaixo.

   p: importância quantitativa de escolher um estacionamento cuja probabilidade de lotação máxima seja baixa;

   te: importância quantitativa de escolher um estacionamento cujo tempo de espera seja baixo;

   dce: importância quantitativa da valorização de escolha de um estacionamento próximo do local onde se encontra;

   ded: importância quantitativa da valorização de escolha de um estacionamento próximo do destino;

   g: importância quantitativa do parque ser gratuito;

   c: importância quantitativa do parque ser coberto;

[5]Tratando-se de uma característica booleana, esta toma o valor 0, caso seja falsa, ou 1, caso verdadeira.

ec: importância quantitativa do parque ter uma central de carregamento eléctrico;

É de notar que os parâmetros devem satisfazer a seguinte condição:

$$p_P + p_{Te} + p_{Dce} + p_{Ded} + p_G + p_C + p_{Ec} = 1$$

Essas características são agregadas através da fórmula

$$\text{pontuação} = (p_P \times p) + (p_{Te} \times te) + (p_{Dce} \times dce) + (p_{Ded} \times ded) + (p_G \times g) + (p_C \times c) + (p_{Ec} \times ec), \quad (9)$$

A equação (9) transforma os estacionamentos num único indicador, para que possa ser feita a ordenação.

Quando não existem informações dos utilizadores, são definidos por defeito os pesos iniciais:

$$\text{pontuação} = (p_P \times 0.4) + (p_{Te} \times 0.2) + (p_{Dce} \times 0) + (p_{Ded} \times 0.4) + (p_G \times 0) + (p_C \times 0) + (p_{Ec} \times 0), \quad (10)$$

Estes pesos podem ser modificados, de acordo com as preferências explícitas, e podem ser reajustados, através das preferências implícitas aprendidas.

As preferências explícitas aumentam determinados pesos e, como consequência, diminuem os restantes. Quando é dada preferência a uma característica, ela toma o peso de 80%, e o seu peso previamente estipulado será distribuído pelas restantes características.

As preferências implícitas são aprendidas através do histórico de escolhas, e devem ser reflectidas em todas as sugestões posteriores. A cada nova escolha, os pesos são ajustados, tendo em conta a possível razão pela qual o utilizador a escolheu. Esses ajustes seguem o seguinte procedimento:

(a) São criadas listas de estacionamentos para cada característica considerada;

(b) As listas são ordenadas;

(c) São identificadas as posições de cada estacionamento, perante cada uma das características;

(d) É anotada a posição do estacionamento escolhido em cada lista;

(e) São ajustados os pesos de forma a que na próxima sugestão dos mesmos estacionamentos, para o mesmo utilizador, o estacionamento escolhido apareça em primeiro lugar.

7. Por fim, são seleccionados os primeiros 10 estacionamentos listados.

6 Servidor e Back Office

6.1 Servidor

É no servidor que são armazenadas as informações dos estacionamentos e onde é feita grande parte do processamento da informação. O servidor tem como função comunicar com as diferentes partes do projecto (sistema de reconhecimento de lugares livres, Aplicação, *back office*), enviando e recebendo informações sobre os estacionamentos e os utilizadores.

À medida que novas informações são recebidas (envidadas por utilizadores ou por sistemas de deteção), estas são armazenadas numa base de dados. Posteriormente, a informação é usada para treinar um modelo de redes neuronais capaz de realizar previsões acerca da lotação dos parques.

O utilizador, através da aplicação, indica o seu destino (local para onde se pretende deslocar), juntamente com as suas preferências e imposições. O servidor processa essa informação (pedido do utilizador), e sugere um conjunto de 10 estacionamentos. Estes são ordenados por ordem do número de lugares livres.

Desde o pedido até à sugestão, é feita uma ordenação dos principais candidatos (parques que cumpram as restrições impostas pelo utilizador), tendo em conta a distância do utilizador ao destino, a probabilidade de haver lugar, o tempo que pode demorar até se conseguir lugar e as preferências que o utilizador tenha manifestado.

As informações podem ser relativas às características do parque, ditas estáticas, ou referentes ao estado actual do parque, ditas dinâmicas.

Os dados estáticos são introduzidos quando um parque é criado no sistema, pela equipa de desenvolvimento ou pelos utilizadores da aplicação. Já os dados referentes ao estado do parque, podem ser enviados pelos utilizadores da aplicação ou automaticamente enviados pelos estacionamentos que tenham algum detector de veículos. Além de armazenar, enviar e receber dados dos parques de estacionamento, o servidor, com base nesses dados, é capaz de fazer uma sugestão personalizada para cada utilizador. Para a formulação dessa sugestão é usado um algoritmo de inteligência artificial, nomeadamente uma rede neuronal artificial, Figura 69. Este algoritmo é usado para aprender padrões dos condutores e dessa forma prever a existência de lugares de estacionamento livres.

Assim, mesmo em lugares que não tenham nenhum sistema de deteção, a informação da lotação do mesmo consegue ser prevista com alguma precisão, através do uso de inteligência artificial.

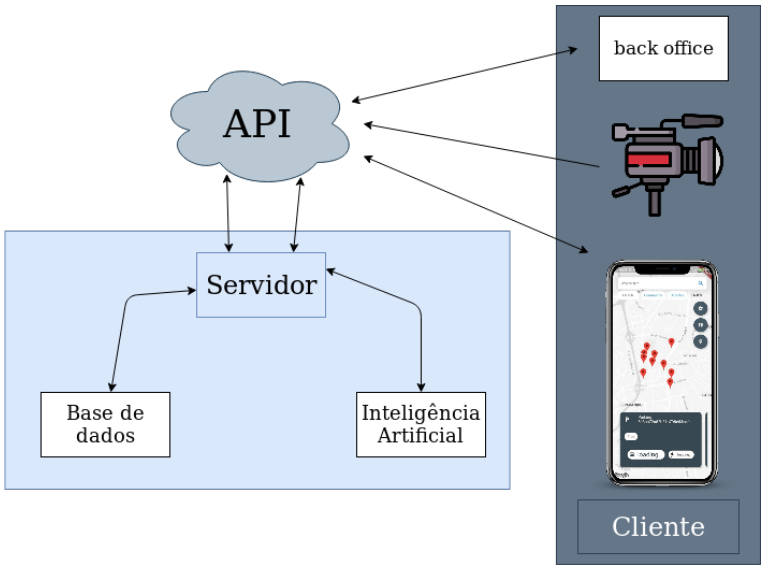

Figura 69: Diagrama da API.

6.1.1 Linguagem do servidor

O servidor foi construído usando o **Node Js**, um *software* escrito em C++ que permite que o código **java script** seja usado no **back end**. A interpretação do mesmo dá-se através do motor *V8 javaScript Engine*, o mesmo usado pelo *browser Chrome*.

Além das capacidades inerentes à linguagem *java script* e às suas bibliotecas, foram acrescentadas funcionalidades que permitem ter acesso a mais informações do dispositivo no qual é processado (como gestão de ficheiros e executar comandos do terminal).

Esta abordagem tem sido cada vez mais adoptada, devido ao crescimento da linguagem **java script** e ao facto de permitir uma maior especialização, uma vez que pode ser usada tanto no **front-end** como no **back-end**.

Um dos principais *frameworks* usado na construção deste servidor foi o **Express**. É um *framework* muito usado no auxílio da construção de um servidor **HTTP**, simplificando a forma como são geridas as rotas, e permitindo que sejam automaticamente providenciados os ficheiros estáticos. Este permite ainda

uma boa organização das pastas, tornando o servidor mais robusto (o que facilita a sua gestão aquando de uma reformulação).

6.1.2 Estrutura do servidor

O servidor está estruturado em módulos:

- API de dados
- Base de dados
- Modelos de *Machine Learning*
- Scripts usados para recolher informação actual

Estes módulos apenas podem ser utilizados por intermédio da API de dados. A API recebe os pedidos por parte dos clientes (quem utiliza o serviço fornecido), avalia se estes cumprem todos os requisitos de autenticação para realizar o pedido em questão, e procede à realização do pedido, coordenando e executando os restantes módulos (base de dados, modelos de *Machine Learning*, *scripts*).

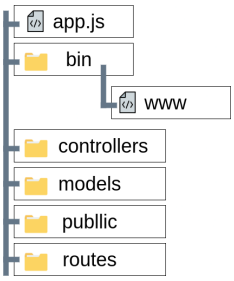

Figura 70: Arquitetura de pastas da API de dados.

Na Figura 70 é ilustrada a estrutura de pastas usada para desenvolver a API de dados. Esta estrutura segue o *desing pattern Model View Controller* (MVC) [34] (ver Figura 71), que representa uma arquitetura de *software* que visa a distribuição de responsabilidades entre diferentes partes do código, permitindo obter um código mais organizado.

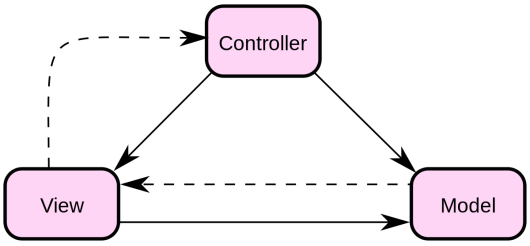

Figura 71: Design patter MVC

- **Model** – corresponde à definição da estrutura dos dados, em geral refere-se à estrutura da base de dados.
- **View** – constitui uma representação da informação que é enviada para o utilizador; parte gráfica usada pelo utilizador para interacção com a aplicação.
- **Controller** – onde se encontra toda a parte lógica da aplicação; parte do código onde pode ser consultado/alterado/eliminado um dado da base de dados; todas as operações mais significativas devem ser feitas nesta parte do código.

A configuração da API de dados é feita através do ficheiro *app.js*. Neste ficheiro, são adicionadas dependências, é feita a ligação com a base de dados e é feita a distribuição das responsabilidade das rotas pelos ficheiros. As configurações mais profundas ou de baixo nível (como definir a porta do servidor), encontram-se no ficheiro *www*, que está dentro da pasta *bin*.

A pasta *routers* possui os ficheiros responsáveis por receber os pedidos dos clientes e por dar uma resposta. A resposta é dada através do formato *View*.

Os restantes módulos da arquitetura **MVC**, (*Model*, *Controller*), podem ser encontrados nas pastas *Models* e *Controllers*, respetivamente.

A pasta *public* é especial e possui apenas os ficheiros estáticos servidos automaticamente pelo próprio *express*.

6.1.3 Base de dados

As Base de dados mais utilizadas continuam a ser as relacionais (nomeadamente a SQL). Estas têm um esquema rígido e garantem a consistência dos dados, fazendo desta opção a mais robusta.

Por norma, as base de dados **SQL** são construídas através de um modelo conceptual, com uma estrutura definida, onde são considerados todos os atributos e todas as ligações. Só são admitidos dados nesse mesmo formato do *SQL*, respeitando as regras do esquema criado e as estipuladas pela própria natureza do **SQL**.

Por outro lado, as base de dados **NoSQL** têm a particularidade de não ter uma estrutura definida, o que pode ser positivo por ser extremamente mutável, e de permitir que o esquema da base de dados seja alterado, sem ser necessário alterar os dados armazenados.

A abordagem *NoSQL* foi desenvolvida para tentar solucionar alguns problemas de *performance* em empresas que possuem um grande volume e fluxo de dados, focando-se na rapidez da leitura dos mesmo. As leituras em **SQL** não conseguem ser tão rápidas, devido às verificações/validações que são feitas para garantir a consistência dos dados. Essas mesmas verificações/validações aumentam ainda a complexidade da distribuição base de dados por várias máquinas, sendo que esta distribuição é essencial em empresas que lidam com grandes volumes de dados (por exemplo, *Google* e *Facebook*).

Tendo em conta a versatilidade da *NoSql*, os dados usados na aplicação são armazenados e tratados através de uma **NoSql**, mais precisamente, a **MongoDB** [35]. Esta é uma base de dados orientada para documentos e coleções e usa a notação **JSON**.

6.1.4 MongoDB

A MongoDB [35] é uma base de dados **NoSQL**, sem esquema definido, orientada para documentos e coleções, e os seus dados são guardados no formato **JSON**.

Esta base de dados é normalmente usada quando o volume de dados é muito grande, e há uma maior incidência nas leituras do que na escrita. Os dados devem ser gravados tendo isso em mente, e cada documento deve conter a informação necessária para responder a qualquer pedido, mesmo que para isso os mesmos dados tenham que estar presentes em mais do que um documento, podendo resultar num aumento do tempo de escrita.

No caso de ser necessário alguma consistência nos dados, terá que haver um programa *third party* que garanta essa consistência. Esta base não tem qualquer validação nesse sentido.

A **Mongo DB** foi escolhida para este projeto também devido ao facto de o servidor ter sido feito em

Node Js, levando a que a interacção com a base de dados seja feita de uma forma natural. Outro ponto a seu favor é o facto de permitir que os dados da base de dados tenham diferentes formatos (este atributo pode ser necessário caso mais tarde surjam outros tipos de estacionamentos com diferentes características)

A **MongoDb** não tem um esquema definido, no entanto, quando se usam linguagens de programação para gerir a base de dados, a existência de um esquema definido facilita o processo. Para ajudar na gestão da base de dados é então usada uma biblioteca (**Mongoose**) que permite a criação de um esquema, indicando desde a estrutura, o tipo de dados de cada atributo e até se alguns atributos são obrigatórios ou não. Esta biblioteca permite ainda fazer as *querys* de forma mais simples, permitindo fazer operações que não seriam possíveis utilizando apenas os comandos nativos no *Mongo*.

O esquema da base de dados usada é o seguinte:

```
var LocationSchema = new mongoose.Schema({
  x: Number,
  y: Number
 })

var stateSchema = new mongoose.Schema({
  year: Number,
  month: Number,
  day: Number,
  hour: Number,
  min: Number,

  intensity_event: Number,
  weekend: Boolean,
  holiday: Boolean,

  cars: String
```

```
});

var ReportSchema = new mongoose.Schema({
  content: String,
  viewed: Boolean,
  parking: String
});

var APISchema = new mongoose.Schema({
    id: String,
    cords : LocationSchema,
    images : [String],

    lots: [String],
    community : Boolean,
    free : Boolean,
    covered : Boolean,
    power_station: Boolean,
    verificated: Boolean,

    before_training: [stateSchema],
    after_training: [stateSchema],

    reports: [ReportSchema],
  });
```

6.1.5 Informações estáticas e informações sobre o estado do parque

Os parques de estacionamento têm determinadas características/propriedades que são consideradas estáticas. São propriedades que, além de identificarem o parque, permitem ainda estabelecer comparações com os restantes parques. Essas propriedades são apresentadas na seguinte tabela:

Propriedade	definição
id	identificador do parque de estacionamento
coordenadas	posição geográfica do estacionamento
imagens	imagens do local
comunidade	Se foi um utilizador que introduziu
grátis	Se o estacionamento é gratuito
coberto	Se o estacionamento é coberto
carregamento	Se tem um posto de carregamento para carros elétricos

Tabela 20: Informações estáticas.

São também armazenadas informações sobre o histórico de ocupação do parque (representadas na Tabela 21). Estes dados são fundamentais para que se possa treinar o modelo de *Machine Learning*, prever a probabilidade de existência de lugares livres e perceber quanto tempo será necessário para se conseguir um lugar livre.

Propriedade
Ano
Mês
Dia
Hora
Minuto
Metereologia
Eventos
Numero de lugares

Tabela 21: Informações sobre o estado de ocupação do parque.

6.1.6 Rotas HTTP

O servidor segue a arquitectura *REST* e, como tal, diferentes operações semânticas são feitas através de diferentes métodos *HTTP*. Desta forma, através do método usado é possível identificar o tipo de operação no servidor.

6.1.6.1 GET

As rotas que têm como método *GET* (ver Tabela 22) correspondem a rotas de leitura. Estas rotas são usadas quando o cliente pretende aceder a alguma informação do servidor.

Rota	descrição
/	Listar os parques de estacionamento
/coords	Listar a localização de todos os parques
/coordsId	Listar a localização e Id de todos os parques
/reports	Listar todos os *reports*
/:id	Toda a informação de um determinado parque

Tabela 22: Rotas **GET**.

1. Rota **/**

   A rota principal, /, lista os parques de estacionamento. Esta pode ser configurada através dos filtros. Se os filtros contiverem a localização do destino em vez dos parques, apenas são enviados para o utilizador os parques com base na proximidade do local de destino, e a probabilidade de haver lugares nesses parques.

Filtro	definição
x	coordenada x
y	coordenada y
free	Apenas grátis
covered	Apenas parques cobertos
values	Numero de parques
community	Se foi um utilizador que introduziu o parque
verificated	Se o parque foi validado
power_station	Se existe uma estação de carregamento, para carros elétricos

Tabela 23: Filtros da rota **/**

Os filtros disponíveis (ver Tabela 23) são configurados através da *query string* da *request*. A *query string* é uma lista de atributos que começa com um ponto de interrogação, **?**, e cada elemento é

constituído por um par chave, valor,representado por **chave** = **valor**. Para separar os diferentes elementos, usa-se o símbolo **&**.

2. Rota /coords ou /coordsId

   A rota que lista todas as coordenadas, /coords ou /coordsId, não recebe qualquer tipo de *query string* ou *req body*. Apenas responde com uma lista de coordenadas (um objeto contendo a coordenada **x** e a coordenada **y**). No caso da rota /coordsId, além das coordenadas, o objeto contém o identificador, **id**, do parque.

3. Rota /reports

   Tal como as rotas mencionadas anteriormente, esta também não recebe qualquer tipo de informação, respondendo com a lista das mensagens de erro, onde nelas consta uma pequena descrição do problema.

4. Rota /:id

   Na maior parte dos casos, a informação da rota **/** é suficiente, no entanto, se for acrescentado o identificador do parque (/:id), passa a ser possível consultar todos os dados de um determinado parque.

6.1.6.2 Post

As rotas que têm como método *POST* (ver Tabela 24) são usadas quando o intuito é criar algo no servidor, como por exemplo acrescentar uma nova informação.

Rota	descrição
/	Adicionar um parque
/training	Treinar um modelo de machine learning

Tabela 24: Rotas **POST**.

1. Rota **/**

   Esta rota é usada para adicionar um novo parque de estacionamento. As informações do parque são introduzidas através do *body* da requisição, Tabela 25.

Atributo	descrição	Tipo
file	imagem do parque de estacionamento	jpg
state	número de lugares livres	0 , <3, 3-10, >10
free	se o estacionamento é grátis	True, False
covered	se o estacionamento é coberto	True, False
power_station	se o parque de estacionamento tem carregamento para carros elétricos	True, False
cords	coordenadas do estacionamento	{x: Float, y: Float}

Tabela 25: Atributos da rota **/**

   O parque é adicionado e etiquetado como um estacionamento inserido pela comunidade. Consequentemente, este carece de validação por parte de um membro confiável, que irá analisar os dados através do *back office* e assim admitir ou rejeitar o parque.

2. Rota **/training**

   Esta rota não será usada pelos utilizadores. A rota funciona como um mecanismo para remotamente treinar o modelo de inteligência artificial, com a existência de novos dados. Quando é enviado para o servidor uma actualização do estado actual do parque, esses dados são armazenados numa lista, que corresponde à acumulação de dados desde a última vez que o modelo foi treinado. Com esses novos dados, o modelo de *Machine Learning* é treinado novamente. Todos os elementos dessa lista são apagados e movidos para uma outra lista que contem todos os estados já treinados. Este procedimento é feito para todos os parques.

6.1.6.3 Put

As rotas que têm como método *PUT* são usadas quando se pretende alterar alguma informação do servidor. Para tal, foram implementadas as rotas apresentadas na Tabela 26.

Rota	descrição
validate:id	Valida um parque adicionado por um utilizador
report:id	Reporta um problema num parque
/:id	Adiciona informação a um parque

Tabela 26: Rotas **PUT**

1. Rota **validate:id**

   Como mencionado anteriormente, quando um estacionamento é adicionado, este não é automaticamente validado, tendo que ser validado por um membro da organização. Esta rota é responsável por validar um estacionamento.

2. Rota **report:id**

   Os parques podem ter sido adicionados com algumas informações erradas, por isso, é essencial haver uma forma de indicar qualquer problema que possa existir. Assim, o utilizador pode enviar uma mensagem pelo corpo da requisição, cuja chave é **report**.

3. Rota **/:id**

   Os novos dados são fundamentais para que se consiga treinar o modelo de *Machine Learning*. Esta rota é responsável por acrescentar informações a um parque de estacionamento, como, por exemplo, o número de lugares livres. As informações são enviadas através do corpo da requisição, e os atributos são:

Atributo	descrição	Tipo
file	imagem do parque de estacionamento	jpg
state	numero de lugares livres	0 , <3, 3-10, >10

Tabela 27: Atributos da rota **/:id**.

6.1.6.4 Delete

As rotas cujo método é o *DELETE* (ver Tabela 28) são usadas para eliminar algum registo no servidor.

Rota	descrição
/:id	Eliminar um parque

Tabela 28: Rotas **DELETE**

1. Rota **/:id**

   Alguns parques de estacionamento adicionados pelo utilizador podem não ser válidos. Esta rota é usada para eliminar um parque de estacionamento considerado inválido.

6.2 Back office

O **Back office** é uma plataforma onde é possível fazer toda a gestão dos parques, tornando o processo mais agradável e mais simples, e aumentando a segurança do sistema, não expondo a base de dados nem expondo outro serviço que consiga manipular directamente os dados.

Esta é uma ferramenta essencial para que se possa consultar as informações dos estacionamentos, tal como ter acesso às reclamações e aos avisos direccionados a cada estacionamento. Além disso, é possível validar estacionamentos introduzidos por um utilizador, tal como remover anomalias que sejam detectadas.

O *back office* foi construído como uma aplicação *web*, sendo um site que interage com o servidor principal. Para isso, o servidor do *back office* foi desenvolvido tal como o principal, usando *node* como linguagem de programação e *express* como *framework* para facilitar a comunicação *HTTP*.

Porém, este servidor, ao contrário do principal, não responde em *JSON* mas sim em *HTML*, *JS* e *CSS*, esperando que quem consuma os seus dados sejam os *browsers* e que possam mostrar as informações de uma forma visual.

Para esse processo, o *express* foi uma grande ajuda, uma vez que as páginas *JS* e *CSS* são ficheiros estáticos, e que são automaticamente geridas pelo *framework*, possibilitando que o programador não precise de se preocupar com a entrega desses pacotes, apenas enviando uma página com links para esses ficheiros que serão automaticamente entregues, sem que essas entregas tenham que ser desenvolvidas pelo criador da aplicação.

Para facilitar o processo de construção de páginas *HTML*, foi usado um gerador de *HTML*, chamado **PUG** que, com uma linguagem própria, facilita o processo de criar páginas *HTML* inteiras. Assim é possível construir *layouts*, que serão herdados por outras páginas, e usar sintaxe mais avançada para gerar páginas como usar iteradores de objetos e criar variáveis, entre outras facilidades, que não são possíveis usando *HTML* puro, e que facilitam a escrita e leitura do código.

Quando a página é servida ao cliente, a comunicação do cliente com o servidor principal é feita unicamente através do servidor do *back office*, encapsulando a comunicação com o servidor principal.

A comunicação entre o cliente e o servidor de *back office* é feita através de uma biblioteca *JS* que é o **axios**, ela permite e facilita o envio de mensagens *HTTP*, tornando esse processo bem simples.

A comunicação entre o servidor de *back office* e o servidor principal, ocorre de forma similar, com a única ressalva, neste caso a biblioteca não é importada através do *HTML*, mas instalada através de um gestor de pacotes do *node*, como o **npm** ou o **yarn**.

Essa comunicação é estabelecida por *HTTP* em formato *JSON*, e assim abstraída, permitindo que os sistemas comuniquem entre si sem conhecerem aqueles com os quais comunicam.

Para isso, apenas necessitam de seguir o protocolo, previamente definido. O servidor principal define um conjunto de rotas e os parâmetros que está à espera de receber. Assim, quaisquer sistemas que pretendam comunicar com ele, terão apenas de seguir essas mesmas instruções.

6.2.1 Ecrã principal

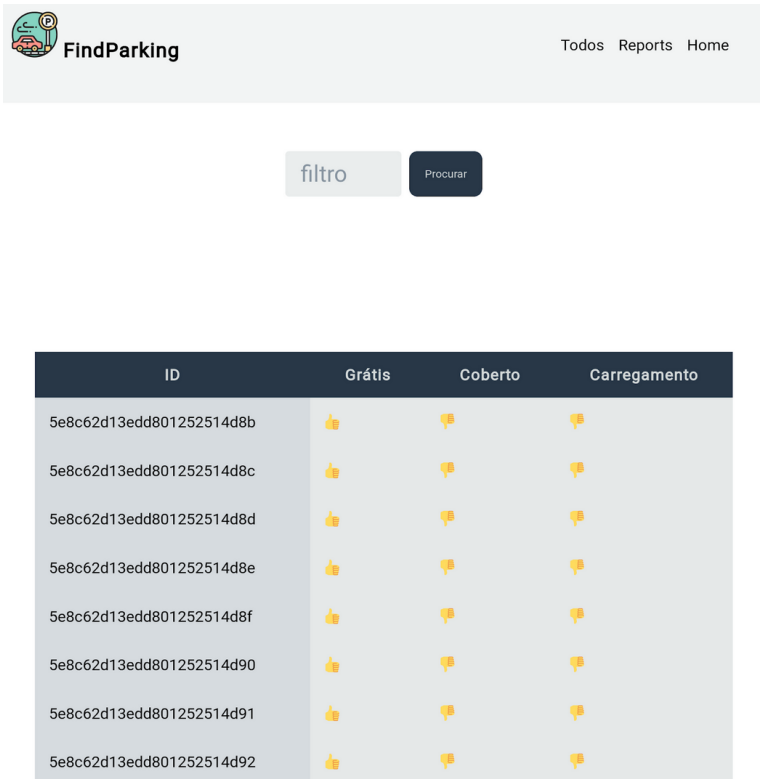

Figura 72: Ecrã principal

O primeiro ecrã contém um menu na parte superior, onde é possível navegar entre as abas *Todos*, *Reports* e *Home*, Figura 72. Quando a aplicação inicia, a mesma abre por defeito na aba *Home*, por uma questão de prioridade. Esta aba é onde se encontram todos os parques que ainda não foram validados, ou seja, quando um utilizador adiciona um novo estacionamento através da aplicação, o mesmo é adicionado na base de dados, mas com uma etiqueta indicando que ainda não foi validado.

Através desta página, são listados todos os parques nessas condições, bem como as informações mais relevantes desses parques.

Outra operação muito importante é a de listar todos os *reports*, pois ter acesso a essas informações pode ser fulcral para identificar anomalias na aplicação ou no próprio parque de estacionamento. Tendo isso em conta, esta aba lista todas as mensagens de *report* e identificadores do respectivo parque, para que, de forma fácil, seja possível consultar todos os dados do mesmo.

Embora as últimas duas abas sejam as mais importantes, ter um método que possibilite a listagem e consequente consulta das informações detalhadas de todos os estacionamentos revela-se muito útil.

6.2.2 Ecrã de um parque de estacionamento

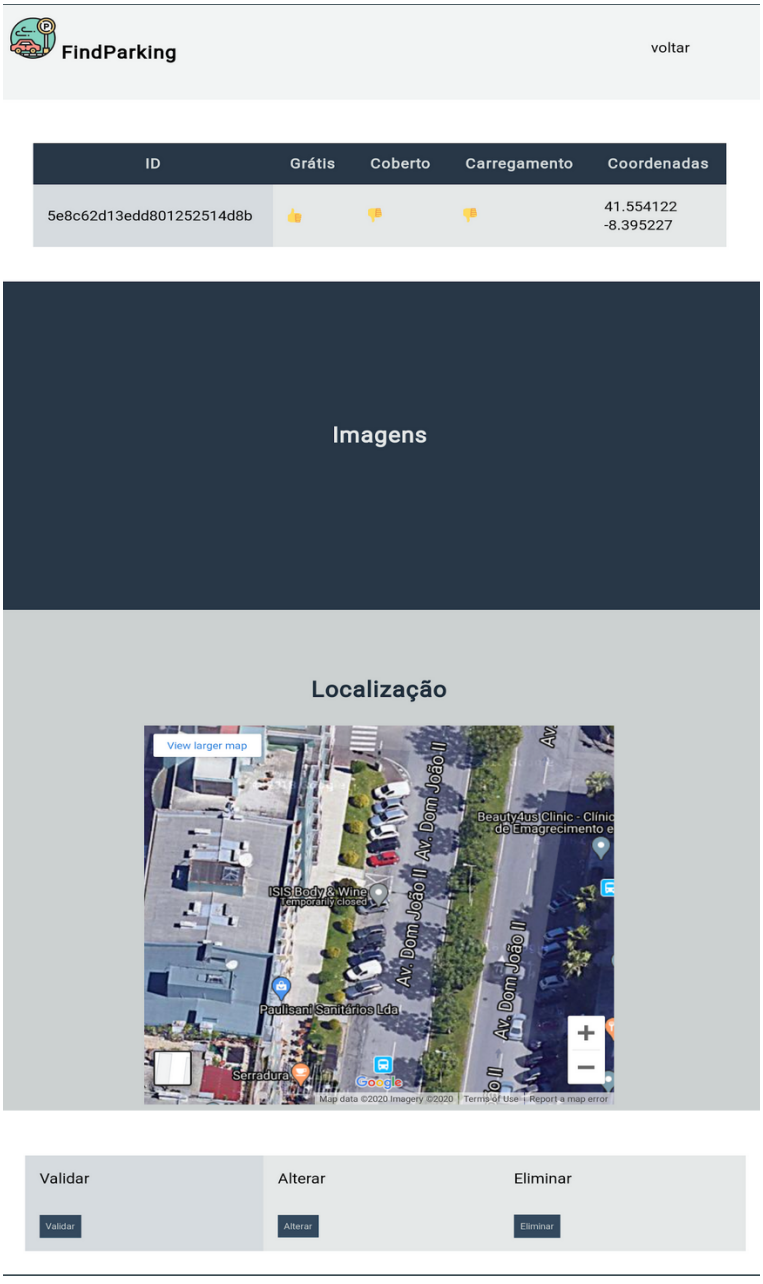

Figura 73: Ecrã de um parque de estacionamento

Tal como mencionado acima, quando os parques são adicionados têm que ser verificados, uma vez que qualquer utilizador pode adicionar um parque e o mesmo pode não existir ou já ter sido adicionado, ou mesmo conter alguma informação que não reflicta a realidade. Esse processo é feito através desta aplicação (Figura 73), onde o gestor tem acesso aos dados introduzidos pelo utilizador, coordenadas, imagens tiradas pelo utilizador na hora em que adicionou o parque, e uma visão real do local, através do **Google Maps**.

Assim, é possível um administrador avaliar se o parque deve ser adicionado, alterado ou eliminado, conseguindo filtrar todos os estacionamentos, e é possível também não deixar o utilizador adicionar informações sem filtro. Este é o ecrã que é disponibilizado em qualquer das abas, que contêm alguma referência a um parque, Figura 73.

7 Conclusão e trabalho futuro

7.1 Conclusão

Esta dissertação apresenta a arquitetura de uma solução completa para procura de parques de estacionamento, que pode ser implementada e adaptada a qualquer cenário. O projeto cobre todo o processo, desde a recolha de informações, até à divulgação de dados do estacionamento, passando por apresentar sugestões usando inteligência artificial que, por sua vez, detecta os padrões, identifica qual é a probabilidade de num dado estacionamento haver lugares livres, e, com base nas preferências do utilizador, sugere um conjunto de estacionamentos.

Ao longo da dissertação foram abordados os temas e discutidos as problemas principais, tendo sido sugeridas possíveis soluções.

Apresentaram-se os conceitos básicos necessários para facilitar a compreensão do trabalho desenvolvido na área de redes neuronais e *Machine Learning*.

Demonstraram-se os diferentes sistemas de reconhecimento de lugares de estacionamento livres: uso de sensores, uso de uma câmara inteligente e uso de algoritmos que permitem detectar, a presença ou não, de veículos nos lugares de estacionamento. Foram treinados modelos de redes neuronais usando o YOLO e RetinaNet. Estes modelos permitiram detectar a presença de veículos através de imagens recolhidas por câmaras.

Desenvolveu-se uma alternativa para o caso de o parque não possuir um sistema de sensores, nem câmaras inteligentes, que passou por uma metodologia inovadora, que permite a previsão dos lugares de estacionamento livres, através do uso de inteligência artificial e de redes neuronais treinadas com dados partilhados pelos utilizadores.

Produziu-se e apresentou-se uma aplicação móvel, que irá permitir aos utilizadores procurar, de forma simples e rápida, um lugar de estacionamento. Discutiram-se as diferentes abordagens existentes para criação de aplicações móveis; apresentou-se a interação da aplicação com o servidor e com o utilizador, e foram apresentados possíveis problemas e as respetivas soluções.

Criou-se um servidor para armazenar as informações dos estacionamentos, e para processar toda a informação (comunicar com as diferentes partes do projecto, sistema de reconhecimento de lugares livres,

aplicação, b*ack office*, enviar e receber informações sobre os estacionamentos e utilizadores). Foi também desenvolvido o *back office*: uma plataforma onde é possível fazer toda a gestão dos parques. Permite consultar as informações dos estacionamentos, tal como ter acesso às reclamações e avisos direccionados ao mesmo.

Os resultados apresentados mostram que a ferramenta desenvolvida é extremamente eficiente e que permite, de facto, o manuseamento inteligente dos lugares de estacionamento.

O resultado final é deveras satisfatório, permitindo o *download* da aplicação por parte dos utilizadores e o seu uso imediato como forma de procura de lugares de estacionamento livres.

Em suma, pode então concluir-se que os objetivos propostos foram totalmente alcançados.

7.2 Trabalho Futuro

Em jeito de síntese, conclui-se que apesar da presente dissertação propor uma solução completa para o problema proposto, existem várias melhorias que poderão ser implementadas e muitas outras contempladas.

No entanto, devido ao mesmo ter sido desenvolvido de forma genérica, pode ser adaptado a vários requisitos, bem como evoluir por diversos caminhos.

7.2.1 Aplicação

1. Pagamento através da aplicação

   A aplicação reúne inúmeras informações sobre os parques e, como tal, poderia ser estendida, de forma a permitir que se realize o pagamento do parque de estacionamento.

   Tal, irá aliciar mais utilizadores a usarem a aplicação e, indiretamente, fará com que mais pessoas compartilhem informações acerca das lotações dos diversos parques de estacionamento, enriquecendo a própria aplicação e melhorando o seu desempenho. A grande vantagem deste processo é também fornecer informações sobre o tempo estimado de espera até que surja um lugar e, assim, refinar a sugestão, nomeadamente no cálculo do tempo de espera.

2. Tempo de viagem

   Neste momento, apenas é calculada a distância desde a localização do utilizador ao parque escolhido. Constituiria uma mais valia, para o utilizador, mostrar o tempo de viagem correspondente a esse trecho.

   Essa informação adicional pode ser usada para ajustar melhor o tempo de previsão, aumentando assim a precisão do modelo.

3. Percurso mais eficiente

   Ao invés de se sugerir um conjunto de estacionamentos para que o utilizador escolha um, é-lhe apresentado o trajeto mais eficaz, isto é, aquele que minimiza o tempo de procura de um lugar de estacionamento.

Esta funcionalidade permite que se obtenham dados de parques que, devido ao seu historial registado, surjam como não tendo lugares disponíveis, e assim reduzir a probabilidade de deixarem de ser recomendados. Uma vez que os trajetos sugeridos podem contemplar pequenos desvios, os mesmos podem ser essenciais para recolher mais dados desses parques, percebendo quando os mesmos se encontram com lugares livres.

4. Autenticação

   Implementar autenticação na aplicação, de forma a que cada utilizador possa ser identificado, restringindo o seu uso apenas a determinadas permissões.

7.2.2 Detecção de Veículos

1. Implementar uma nova abordagem (problema de classificação)

   Ao longo da dissertação foram estudados diversos algoritmos de Machine Learning, tendo sido usados os algoritmos:

   (a) RetinaNet

   (b) YOLO

   No futuro, deveria ser implementada uma *nova abordagem (problema de classificação)*, que, apesar de ter a condicionante de apenas funcionar em estacionamentos que tenham marcações, é uma solução promissora, sendo muito provavelmente a solução mais precisa perante os parques de estacionamentos em que se especializa. Para implementar esta solução, apenas é preciso implementar um modelo capaz de classificar imagens e, em seguida, treinar o modelo com imagens de lugares vazios e de lugares ocupados.

2. Identificar estacionamentos irregulares

   Implementar algoritmos capazes de identificar veículos mal estacionados, infrações ou anomalias.

   Esta funcionalidade poderá ser mais um fator que justifique o investimento em parques e, como tal, venha a incentivar a implementação de um sistema de controlo de veículos em mais estacionamentos.

3. Ler matrícula do veículo

   Implementar um algoritmo capaz de identificar matrículas e, consequentemente, capaz de identificar as viaturas estacionadas num certo parque de estacionamento.

   Esta funcionalidade permitirá fazer o pagamento automático, tal como identificar os veículos que se encontrem mal estacionados ou em falta de pagamento.

   A funcionalidade irá ainda permitir saber a rotina de um determinado veículo, sem colocar em risco a confidencialidade de informação.

7.2.3 Servidor

1. Autenticação

   Implementar autenticação no servidor, de forma a proteger,contra terceiros, os dados e determinadas funcionalidades.

   Uma das formas de autenticação mais comuns é o uso de um *web token*, que é uma informação que é enviada ao utilizador depois de fazer *login*. Este *token* permite ao servidor identificar o utilizador e verificar se o mesmo tem permissões para realizar a operação pedida. Um dos *tokens* mais utilizados de momento é o *JWT -JavaScript Web Token*.

2. Estudar novos fatores que podem influenciar a procura de um lugar de estacionamento

   A rede neuronal construída para prever a lotação de um estacionamento recebe, como parâmetros, um conjunto de valores; no entanto podem ser retirados alguns parâmetros ou acrescentados outros, de modo a aumentar o eficácia do modelo.

   Para averiguar quais parâmetros que devem ser considerados, devem ser recolhidos dados reais da lotação de diversos parques. O próximo passo deve passar por um estudo cuidadoso, de forma a perceber a relação entre os demais parâmetros e o número de lugares do parque. No final, devem ser escolhidos os parâmetros que se consideraram mais relevantes.

3. Refinar parâmetros

   Um dos parâmetros que pode ser refinado é a **Intensidade de um evento**.

Por uma questão de generalidade, o parâmetro intensidade de um evento corresponde apenas a um número natural, o que apenas permite indicar a intensidade (quantitativa) de eventos no redor, de modo a que a rede neuronal seja capaz de identificar relações entre certas intensidades e o número de lugares disponíveis.

No entanto, o cálculo da intensidade depende dos eventos na área circundante do estacionamento, com um certo raio. E, dependendo do evento e da sua distância ao parque, calcula-se a intensidade do evento.

Deste modo é impossível relacionar eventos específicos com a lotação de alguns parques.

Uma solução plausível passaria por considerar um conjunto vasto de possíveis eventos e, para cada evento, considerar a distância aos diferentes parques e a outros eventos:

(a) *jogo de futebol*;

(b) *festa popular*;

(c) *corrida*;

(d) *concerto*;

(e) *festival*;

(f) outros.

Referências

[1] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[2] Chase Dowling, Tanner Fiez, Lillian Ratliff, and Baosen Zhang. How much urban traffic is searching for parking. *arXiv preprint arXiv:1702.06156*, 2017.

[3] Paul G Höglund. Parking, energy consumption and air pollution. *Science of the Total Environment*, 334:39–45, 2004.

[4] Car park anxiety. https://www.huffingtonpost.co.uk/. Accessed: 2020-02-07.

[5] Drivers spend an average of 17 hours a year searching for parking spots. https://eu.usatoday.com/. Accessed: 2020-02-07.

[6] Via verde. `https://www.viaverde.pt/particulares`. Accessed: 2020-02-07.

[7] Google maps. `https://play.google.com/store/apps/details?id=com.google.android.apps.maps`. Accessed: 2020-02-07.

[8] Parkopedia. `https://www.parkopedia.pt/`. Accessed: 2020-02-07.

[9] Pixevia. `https://www.pixevia.com/smart-parking`. Accessed: 2020-02-07.

[10] Parquist. https://play.google.com/. Accessed: 2020-02-07.

[11] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", 2011.

[12] Jinbo Bi, Kristin Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3(Mar):1229–1243, 2003.

[13] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[14] J. A. K. Suykens. Nonlinear modelling and support vector machines. In *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, volume 1, pages 287–294 vol.1, 2001.

[15] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[16] Karel Lenc and Andrea Vedaldi. R-cnn minus r. *arXiv preprint arXiv:1506.06981*, 2015.

[17] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[19] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[24] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. pages 936–944, 2017.

[25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[26] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[27] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[28] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[29] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[30] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[31] Joseph o'Rourke et al. *Computational geometry in C*. Cambridge university press, 1998.

[32] Nitin R Chopde and Mangesh Nichat. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013.

[33] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[34] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings fifth ieee international enterprise distributed object computing conference*, pages 118–127. IEEE, 2001.

[35] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.