UNIVERSIDADE TUIUTI DO PARANÁ

DANIEL DEDA
LUCAS FERREIRA MATEUS

SISTEMA AUTÔNOMO PARA IDENTIFICAÇÃO E GERENCIAMENTO DE VEÍCULOS EM ESTACIONAMENTOS

CURITIBA

2019

DANIEL DEDA
LUCAS FERREIRA MATEUS

SISTEMA AUTÔNOMO PARA IDENTIFICAÇÃO E GERENCIAMENTO DE VEÍCULOS EM ESTACIONAMENTOS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Faculdade de Ciências Exatas e de Tecnologia da Universidade Tuiuti do Paraná, como requisito à obtenção do grau de Bacharel.

Orientador: Prof. Diógenes Cogo Furlan

CURITIBA

2019

RESUMO

Visando facilitar o gerenciamento de vagas em estacionamentos, este projeto teve como objetivo o desenvolvimento de um protótipo para auxílio na identificação dos veículos e alocação das vagas. Foram utilizadas técnicas de Processamento Digital de Imagens, como *Grayscale* e Filtragem, e Redes Neurais Artificiais Convolucionais a fim de realizar a identificação do modelo dos veículos que forem utilizar o estabelecimento. Além de identificar o modelo do veículo, também foram desenvolvidos algoritmos utilizando Banco de Dados para que pudesse ser feito o registro da entrada do veículo, bem como a escolha de uma vaga adequada para o mesmo baseando-se no seu tipo.

Ao fim do desenvolvimento foram analisados os resultados obtidos com a utilização da Rede Neural Convolucional.

Palavras-chave: Visão Computacional, Inteligência Artificial, Estacionamento.

LISTA DE FIGURAS

FIGURA 1 – Áreas Relacionadas com Inteligência Artificial. 11
FIGURA 2 – Representação do processo de convolução com um filtro 3x3. . . 14
FIGURA 3 – Representação do processo de maximização com uma máscara 2x2. 15
FIGURA 4 – Comparação de imagens após a aplicação da técnica de Grayscale. 19
FIGURA 5 – Comparação de imagens após a aplicação da inversão de cores. 20
FIGURA 6 – Representação de uma máscara 3x3 aplicada de forma linear sobre a imagem. 21
FIGURA 7 – Comparação entre filtro de média e mediana. 23
FIGURA 8 – Máscaras utilizadas nos operadores de Roberts e Sobel. 24
FIGURA 9 – Comparação de imagens após a detecção de bordas de Sobel. . 24
FIGURA 10 – Comparação de imagens após a detecção de bordas de Roberts. 25
FIGURA 11 – Resultados atingidos pelos autores 28
FIGURA 12 – Imagem após a delimitação das regiões com caracteres. 29
FIGURA 13 – Resultado após a identificação da placa do veículo. 29
FIGURA 14 – Comparação de imagens após a binarização. 30
FIGURA 15 – Localização da placa 30
FIGURA 16 – Extração da placa. 31
FIGURA 17 – Local do caractere da placa 31
FIGURA 18 – Banco de Dados utilizado para comparação 32
FIGURA 19 – Comparação entre a imagem original e imagem após o afinamento. 33
FIGURA 20 – Resultado após a identificação da placa do veículo. 34
FIGURA 21 – Identificador de carros ocupando vagas. 34
FIGURA 22 – Exemplo de imagem de entrada e saída do sistema. 36
FIGURA 23 – Comunicação entre as APIs. 38
FIGURA 24 – Comparação entre a imagem original e imagem após o tratamento. 40
FIGURA 25 – Modelo da CNN utilizada 41
FIGURA 26 – Exemplo de requisições possíveis pelo *Swagger* 50
FIGURA 27 – Exemplo de requisição executada com o *Swagger* 51
FIGURA 28 – Página inicial do sistema 52
FIGURA 29 – Página de classificação e alocação de veículos 53
FIGURA 30 – Página de exibição das vagas do estacionamento 53
FIGURA 31 – Página de exibição do banco de dados 56
FIGURA 32 – Primeiro treinamento 58
FIGURA 33 – Segundo treinamento 58
FIGURA 34 – Terceiro treinamento 59
FIGURA 35 – Quarto treinamento 60
FIGURA 36 – Quinto treinamento 60

FIGURA 37 – Sexto treinamento 61

LISTA DE TABELAS

TABELA 1 – Resultado dos testes. 35

LISTA DE CÓDIGOS

CÓDIGO 1 – Método para pré-processamento da imagem. 41
CÓDIGO 2 – Método que executa a passagem adiante entre as camadas. . . . 42
CÓDIGO 3 – Método que executa a Retropropagação entre as camadas. . . . 42
CÓDIGO 4 – Método Forward da camada de convolução. 43
CÓDIGO 5 – Método Forward da camada de maximização. 43
CÓDIGO 6 – Método Forward da camada de ativação. 44
CÓDIGO 7 – Método Backpropagation da camada de ativação. 45
CÓDIGO 8 – Método Backpropagation da camada de maximização. 46
CÓDIGO 9 – Método Backpropagation da camada de convolução. 46
CÓDIGO 10 – Método de teste da CNN. 47
CÓDIGO 11 – Método de classificação de imagens. 48
CÓDIGO 12 – API da CNN. 48
CÓDIGO 13 – Método de classificação de veículos. 54
CÓDIGO 14 – Contexto do banco de dados. 55
CÓDIGO 15 – Exemplo de uma *migration* criada pelo sistema. 55
CÓDIGO 16 – Exemplo de consulta ao banco dentro do sistema. 56

SUMÁRIO

1 INTRODUÇÃO . . . 9
2 FUNDAMENTAÇÃO TEÓRICA . . . 11
2.1 INTELIGÊNCIA ARTIFICIAL . . . 11
2.1.1 Aprendizado de Máquina . . . 12
2.1.2 Redes Neurais Artificiais (RNA) . . . 13
2.1.3 Redes Neurais Convolucionais . . . 13
2.2 PROCESSAMENTO DIGITAL DE IMAGENS . . . 17
2.2.1 Transformações de cores . . . 18
2.2.2 Filtros . . . 20
2.2.3 Filtros Lineares . . . 20
2.2.4 Filtros Não Lineares . . . 23
2.3 VISÃO COMPUTACIONAL . . . 25
3 REVISÃO DA LITERATURA . . . 27
3.1 AN APPROACH TO LICENSE PLATE RECOGNITION . . . 27
3.2 AUTOMATIC VEHICLE IDENTIFICATION BY PLATE RECOGNITION . . . 29
3.3 RECONHECIMENTO DE CARACTERES ATRAVÉS DE REDES NEURAIS ARTIFICIAIS COM APLICAÇÃO A PLACAS DE AUTOMÓVEIS . . . 32
3.4 A VISION-BASED PARKING LOT MANAGEMENT SYSTEM . . . 34
3.5 DETECTION AND CLASSIFICATION OF OBJECT IN IMAGE FOR VEHICLE TRACKING . . . 35
3.6 COMPRATIVO DOS TRABALHOS . . . 36
4 METODOLOGIA . . . 37
4.1 FERRAMENTAS UTILIZADAS . . . 37
4.2 MÉTODOS . . . 38
4.2.1 Rede Neural Convolucional . . . 39
4.2.2 Classificação . . . 49
4.2.3 Controle do estacionamento . . . 49
4.2.4 Interface . . . 51
5 ANÁLISE DOS RESULTADOS . . . 57
6 CONCLUSÃO . . . 62
APÊNDICE A – Tabela de versões . . . 63
APÊNDICE B – Fluxograma de classificação e alocação de veículos. . . . 64
APÊNDICE C – Fluxo do treinamento da CNN. . . . 65
APÊNDICE D – Diagrama de Entidade e Relacionamento . . . 66
APÊNDICE E – Cronograma . . . 67
APÊNDICE F – Manual do usuário . . . 68
6.1 ALOCAÇÃO DE NOVO VEÍCULO . . . 68

6.2 CONSULTAR CARROS ALOCADOS E VAGAS DISPONÍVEIS 70
6.3 CONSULTAR CARROS E VAGAS CADASTRADOS NO BANCO 71
REFERÊNCIAS 73

1 INTRODUÇÃO

Não é de hoje que a quantidade de veículos automotores vem crescendo, principalmente nas cidades mais desenvolvidas. Uma pesquisa divulgada em maio de 2018 pela Companhia de Engenharia de Tráfego de São Paulo (CET) mostra que a cidade atingiu a marca de 8,6 milhões de veículos, incluindo carros, ônibus, motos e caminhões. Em 2015 o número era de 7 veículos para cada 10 habitantes, em 2016 este número subiu para 7,2, chegando a 7,4 no ano de 2017 (MENEZES, 2018). O crescimento foi tão grande que houve a necessidade de se implementar uma regra na cidade proibindo o paulistano de circular na região central um dia por semana, nos horários de pico.

Este crescimento não ocasiona apenas o aumento do trânsito, mas também agrava os problemas já existentes relacionados à poluição, aumento no valor dos fretes, gastos com combustível e à falta de um local adequado para estacionar.

E foi devido à falta de locais seguros para deixar os veículos que os primeiros estacionamentos surgiram. Logo nas décadas de 1920 e 1930, quando o número de motoristas começava a aumentar em escala global e os carros passaram a ocupar um tamanho precioso do espaço público (o espaço entre as casas e prédios, assim como espaço por onde as pessoas antes caminhavam e passavam o tempo livre (LAMAS, 2014)), criaram-se estacionamentos para todo o tipo de empreendimento e localização, tornando-se tão comuns que atualmente já estão enrustidos até mesmo nos costumes da sociedade.

Mas se por um lado a quantidade de estacionamentos disponíveis para a população torna a vida mais fácil, por outro a concorrência acaba trazendo mais riscos e despesas aos proprietários destes locais. Em uma análise de despesas de operação de uma empresa de grande porte, foi constatado que os gastos com estacionamentos chegavam a 24% dos gastos totais, ultrapassados apenas pelos gastos com contratos e serviços (GONÇALVES, 2019). Grande parte deste custo se deve a um possível mau gerenciamento das vagas, além de despesas com funcionários e manutenção dos equipamentos, como cancelas, máquinas que operam os cartões de entrada e saída do estabelecimento e sistemas de segurança.

Visando facilitar o gerenciamento das vagas do estacionamento e melhorar a experiência das pessoas que o utilizam, este projeto tem como foco o desenvolvimento de um protótipo para fazer a identificação do modelo dos veículos e a escolha de uma vaga adequada. Para isso foi utilizada uma Rede Neural Convolucional para fazer a identificação do modelo do veículo a partir de uma imagem do mesmo, além de técnicas como *Grayscale* e Filtragem para realizar o pré processamento das imagens que foram

utilizadas.

Dessa forma, o sistema irá contribuir para redução do tempo gasto em busca de vagas adequadas para os veículos, melhorando a experiência dos motoristas e facilitando o gerenciamento das vagas do estabelecimento.

Tendo isso em mente, o Capítulo 2 apresenta a fundamentação teórica para o entendimento do trabalho, sendo que a Seção 2.1 aborda alguns temas relacionados a Inteligência Artificial, bem como alguns exemplos da sua utilização, e na Seção 2.2 são detalhadas as técnicas de Processamento de Digital de Imagens necessárias para o desenvolvimento do projeto. No Capítulo 3 são descritos algumas das obras literárias relacionadas com o projeto, bem como uma análise dos resultados obtidos pelos autores. Já no Capítulo 4, é definido a abordagem inicial para o desenvolvimento do projeto. Por fim é apresentada uma análise dos resultados obtidos no Capítulo 5 e no Capítulo 6 a conclusão do desenvolvimento do projeto.

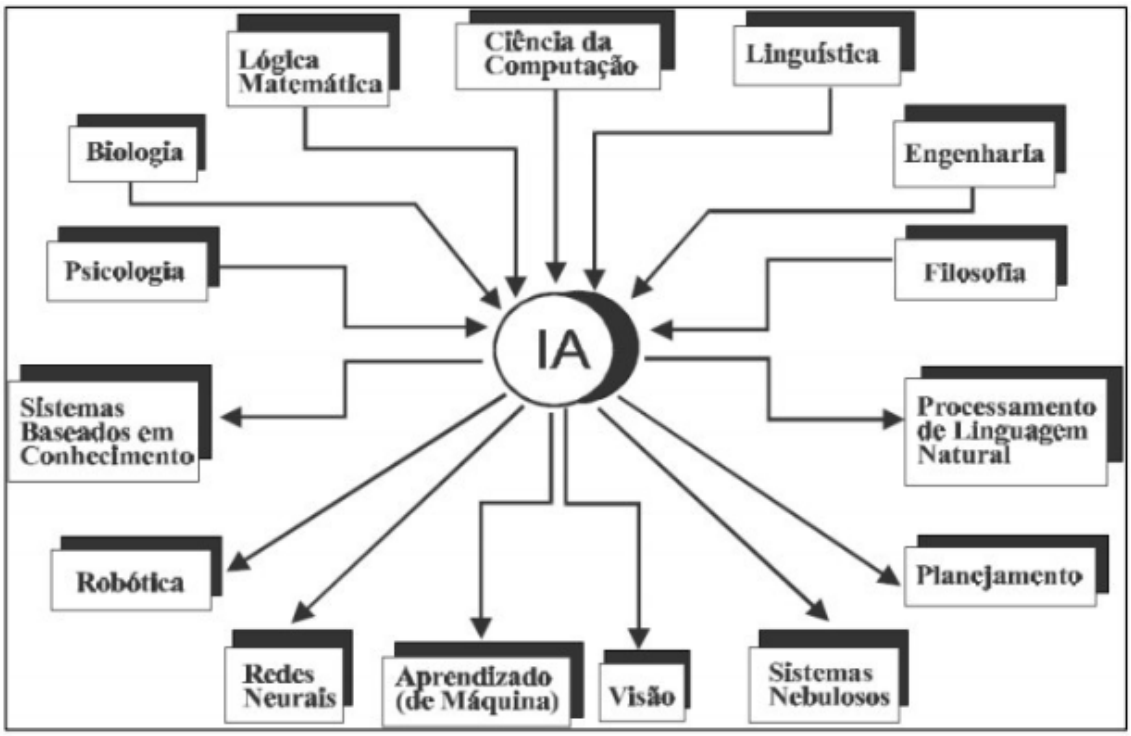
2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTELIGÊNCIA ARTIFICIAL

A mesma Inteligência Artificial (IA) que em um passado não tão distante era algo tratado como complexo e pouco acessível, que tinha seu uso restrito para resoluções de desafios e ainda não era concebida como uma aplicação clara no mundo real, se tornou essencial para o desenvolvimento e descobertas de novas tecnologias tanto na área de hardware como de software (MEDEIROS, 2018).

Pode-se definir a Inteligência Artificial como um ramo da Ciência da Computação que se ocupa da automação do comportamento inteligente (LUGER, 2013) e que tem o intuito de criar programas que sejam independentes dos humanos e ao mesmo tempo realcem sua capacidade de resolver problemas de modo que aparentem estar pensando (TEIXEIRA, 2009). Por ser um tópico muito amplo ela acaba englobando também as áreas de lógica matemática, linguística, engenharia, filosofia, psicologia, biologia, entre outras, como mostra a Figura 1.

FIGURA 1 – Áreas Relacionadas com Inteligência Artificial.



Fonte: MONARD e BARANAUSKAS (2000).

Nesta seção será abordada a área de Aprendizado de Máquinas (Seção 2.1.1) bem como o funcionamento de Redes Neurais Artificiais (Seção 2.1.2) e Redes neurais Convolucionais (Seção 2.1.3).

2.1.1 Aprendizado de Máquina

O Aprendizado de Máquina, também conhecido como *Machine Learning*, é um campo da área de Inteligência Artificial onde é feita uma tentativa de fazer com que computadores tenham capacidade de aprender e realizar uma determinada ação, através de dados coletados a partir de experiências exercidas por ela no ambiente residente (MITTCHEL, 1997).

No ano de 2014, a empresa *Deepmind* começou o desenvolvimento de uma nova inteligência artificial a qual, posteriormente, acabaria sendo adquirida pela Google. A IA nomeada de *AlphaGo Master* é uma versão mais recente e potente da *AlphaGo*, que se utiliza do sistema de aprendizado por reforço e obteve sua grande conquista em 2017, quando venceu o jogador campeão mundial Ke Jie do jogo Go. Comparado ao Xadrez, o jogo Go possui um número de possibilidades extremamente maior, enquanto o Xadrez possui 10^{50} o jogo chinês pode chegar a quantidade exorbitante de 10^{117} possibilidades. Em seu aprendizado a *AlphaGo Master* recebeu como parâmetros de entrada apenas as regras base do jogo e foi obtendo seu conhecimento e táticas enquanto simulava jogos contra si própria, analisando suas ações para que pudesse maximizar suas recompensas e torná-las uma vitória, que foi exatamente o ocorrido e em pouco tempo de aprendizado acabou superando o considerado melhor jogador do mundo. Após a *AlphaGo Master* foi desenvolvida a *AlphaGo* Zero, que em apenas 3 dias de aprendizado conseguiu superar suas versões passadas.

Existem três tipos de aprendizado que podem ser utilizados, sendo o primeiro deles é o Aprendizado Supervisionado. Durante o Aprendizado Supervisionado são apresentados ao algoritmo os dados a serem analisados e os dados esperados como resposta, dessa forma o algoritmo pode ajustar seus parâmetros comparando as respostas obtidas com as respostas esperadas. Esta é uma metodologia de aprendizado muito utilizada para algoritmos de classificação onde as classes possíveis são conhecidas ((HONDA; FACURE; YAOHAO, 2017)).

Outra metodologia de aprendizado é o Aprendizado Não Supervisionado, onde o algoritmo não conhece os grupos e classes dos dados. Neste tipo de aprendizado o algoritmo irá comparar os dados informados a ele e irá então formar grupos baseados na semelhança entre estes dados.

Por último, existe o Aprendizado por Reforço, onde cada uma das ações que o algoritmo pode realizar estão associadas a uma recompensa, de forma que o algoritmo irá avaliar todas as ações possíveis e escolher aquela que maximize suas recompensas baseando-se nos resultados anteriores.

2.1.2 Redes Neurais Artificiais (RNA)

O cérebro humano é composto de diversos neurônios, um tipo de célula que sozinha apenas recebe e transmite impulsos para as células vizinhas, mas que em conjunto pode formar todos os nossos pensamentos. Já uma Rede Neural artificial nada mais é do que uma técnica criada com o objetivo de imitar o funcionamento dos neurônios dos seres humanos. Atualmente redes neurais artificiais são utilizadas no processamento de linguagem natural, reconhecimento de voz, classificação de imagens, detecção de anomalias, dentre outros campos que necessitam de uma capacidade aprendizado por parte do algoritmo (CARVALHO, 2009).

Seu princípio remonta ao ano de 1943, com dois pesquisadores da Universidade de Chicago: Warren McCullough e Waler Pitts. Em seu artigo "*A Logical Calculus of the Ideas Immanent in Nervous Activity*", foi apresentada a ideia do neurônio como unidade básica de atividade do cérebro. Com o conceito popularizado, iniciaram-se pesquisas na área e, em 1959, surgiu na Universidade de Stanford a primeira Rede Neural para a resolução de um problema real, a MADALINE (*Multiple Adaptive Linear Elements*), desenvolvida para reduzir os ruídos em uma linha telefônica (DEMURO, 2018).

Por se tratar de uma técnica que tenta imitar o funcionamento dos neurônios do cérebro humano, esse tipo de algoritmo é dividido em várias camadas compostas por neurônios que realizam operações individuais sobre os dados que são passados para ele, posteriormente esses dados são repassados para outros neurônios que irão continuar o processo. Estas redes neurais são divididas em dois grupos, as *Feedforward* e as *Feedback*. Nas redes *Feedforward* a informação segue apenas um sentido, sendo passada sempre para a próxima camada, esse tipo de Rede Neural normalmente é utilizada em programas para classificação de imagens e *data mining*. Já nas redes *Feedback* a informação pode seguir para qualquer direção, sendo esta comumente encontrada em algoritmos de otimização (BONNER, 2019).

2.1.3 Redes Neurais Convolucionais

As Redes Neurais Convolucionais, ou *Convolutional Neural Networks* (CNN), são um tipo de Rede Neural de aprendizado supervisionado que foi desenvolvido para se trabalhar com visão computacional, sendo utilizada principalmente em algoritmos de classificação e identificação de padrões em imagens. O que diferencia esta rede dos outros tipos de redes neurais é a presença de camadas convolucionais ocultas. Estas são responsáveis por aplicar o processo de convolução e maximização sobre os dados das camadas anteriores, realçando as características de acordo com os filtros utilizados por cada neurônio. O objetivo destas camadas é reduzir a quantidade de dados a serem processados de uma maneira que as as características essenciais

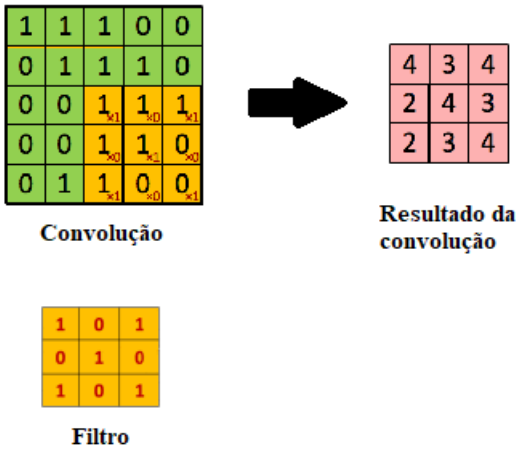
sejam mantidas para que se possa obter uma classificação precisa (SAHA, 2018).

Este tipo de Rede Neural tem grande vantagem sobre redes neurais tradicionais para classificação de imagens, já que numa Rede Neural tradicional cada pixel da imagem seria tradado como um sinal de entrada, ou seja, uma imagem colorida de 640x480 iria gerar mais de 300 mil sinais de entrada para cada um dos três espectros de cor, totalizando quase 921 mil sinais de entrada apenas na primeira camada, tornando este tipo de processamento inviável. Numa Rede Neural convolucional não são analisados os *pixels* individualmente, mas sim as características presentes em um conjunto de *pixels* através do processo de convolução.

Outra vantagem deste tipo de rede é o fato de que os objetos que se deseja identificar não precisam estar localizados sempre no mesmo local da imagem. Numa rede tradicional, se o objeto não estiver exatamente na mesma posição em qual foi treinado, os padrões de ativação dos neurônios seriam diferentes e afetariam o resultado da classificação.

Uma CNN é composta por combinações de quatro tipos de camadas de neurônios especializados. A primeira é a camada de entrada, onde a imagem será convertida para uma matriz que representa seus respectivos *pixels*. Já a segunda camada é a camada de Convolução, que dá nome ao algoritmo. Ela é composta de neurônios que aplicarão filtros nessa matriz de forma semelhante aos filtros detalhados na Seção 2.2.2. Como pode ser visto na Figura 2, o filtro é aplicado sobre a imagem original, de forma que o resultado da soma dos produtos da multiplicação de cada um dos *pixels* da imagem original pelos valores do filtro irá gerar uma matriz com os resultados da convolução.

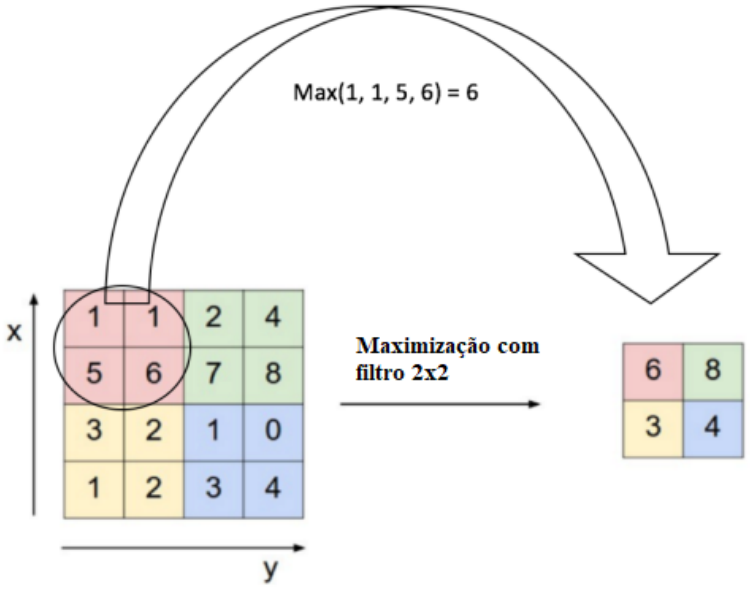
FIGURA 2 – Representação do processo de convolução com um filtro 3x3.



Fonte: Adaptado de Karn (2016).

A camada de maximização é a terceira camada e é responsável por realçar as características de uma imagem. Isso é feito iterando por regiões de área x, y nos *pixels* da imagem e escolhendo sempre o maior valor dentro desta região. Desta maneira tamanho da imagem original é reduzido, porém preservam-se as características mais relevantes obtidas na camada anterior (Figura 3).

FIGURA 3 – Representação do processo de maximização com uma máscara 2x2.



Fonte: Adaptado de Karn (2016).

Por último, a quarta camada da CNN é a camada de ativação, que é responsável por converter os dados obtidos em uma respectiva classe, normalmente convertendo a saída da camada anterior em um valor numérico que representa essa classe. Existem diversas maneiras de se realizar o processo de ativação, por exemplo a função ReLU (*Rectified Linear Unit*).

A ReLU é uma das funções de ativação mais utilizadas em redes neurais convolucionais devido a sua simplicidade e baixo custo computacional, sendo representada pela Equação 2.1 e seus resultados estão sempre entre 0 e infinito (SHARMA, 2017).

$$y = max(0, x) \tag{2.1}$$

A desvantagem da função ReLU é que todos os valores negativos obtidos na Rede Neural serão imediatamente convertidos para zero, tornando o mapeamento destes valores incorreto (SHARMA, 2017).

Uma alternativa para a função ReLU é a Softmax (Equação 2.2). Esta função de ativação converte os valores resultantes de uma Rede Neural em probabilidades daquele resultado pertencer a uma das classes conhecidas pelo algoritmo.

$$S_{(z_i)} = \frac{e^{zi}}{\sum e^{zj}} \tag{2.2}$$

Onde e é uma constante representando o valor obtido nas camadas anteriores, sendo elevado por zi, que representa cada elemento do vetor z de classes. Esta função converte os valores resultantes da Rede Neural em valores entre 0 e 1, representando a probabilidade do resultado pertencer a uma das classes conhecidas z.

Apesar de ser mais complexa, a função Softmax é amplamente utilizada em algoritmos de classificação por conseguir converter o resultado sem apresentar o mesmo problema da função ReLU, sendo muito mais eficiente (SHARMA, 2017).

Por fim, outro componente decisivo de uma Rede Neural Convolucional são os dados escolhidos para o treinamento, também chamado de *Dataset*. Ele deve conter imagens com exemplos do que se deseja classificar, bem como seus respectivos rótulos. A escolha de um bom *Dataset* durante a etapa de treinamento define o quão precisa será a classificação. Os exemplos de imagens devem ser diversos o suficiente para que o algoritmo aprenda a classificá-las em diversas situações (*Overfitting*), porém sem ser diversificado a ponto do algoritmo não conseguir identificar nenhuma semelhança (*Underfitting*) (BONNER, 2019).

O termo *Underfitting* é utilizado para se referir a falta da capacidade da Rede Neural generalizar o que foi aprendido durante o treinamento, sendo pela utilização de um *Dataset* muito disperso ou pela falta de treinamento. Já o termo *Overfitting* se refere ao excesso de especialização de uma Rede Neural, ele ocorre quando os dados do *Dataset* de treinamento são muito semelhantes entre si, ou até mesmo quando há o excesso de treinamento sobre os mesmo dados, de forma que o algoritmo se torna extremamente especializado em classificar apenas os dados utilizados no treinamento, perdendo a capacidade de generalizar o que foi aprendido para dados novos (BROWNLEE, 2018).

No entanto, apenas fazer com que a imagem percorra cada uma das camadas não irá gerar nenhum resultado consistente, para que a CNN possa aprender a identificar as características das imagens é necessário submetê-la a um processo de treinamento, que consiste em quatro etapas básicas:

1. Inicialização dos filtros.
2. Processamento dos dados.

3. Cálculo do erro.
4. Retro-propagação.

Na primeira etapa, todos os filtros das camadas convolucionais serão inicializados de forma aleatória. Na etapa de processamento dos dados, ou *Forward*, cada uma das imagens do *Dataset* passa por todas as camadas, sempre partindo da entrada em direção à última camada da rede. Ao final desta etapa irá se obter um possível resultado da classificação, este resultado então é subtraído do valor esperado na terceira etapa, isso é feito para se encontrar o erro do resultado obtido, ou seja, o quão longe do valor correto este resultado está.

Este erro então é utilizado pela etapa de retro-propagação, ou *Backpropagation*. Nesta etapa, o erro de cada classificação faz o caminho de volta ao início da rede, partindo sempre da última camada. Isso é feito para que todas as camadas possam ajustar seus parâmetros de acordo com o erro. A medida que as imagens do *Dataset* forem processadas, estes parâmetros irão ser atualizados de forma a reduzir o erro e aumentar a precisão da classificação (KARN, 2016).

Na maior parte das vezes as imagens precisarão de um pré-processamento antes de serem enviadas para a CNN para reduzir a quantidade de processamentos necessários, já que uma imagem de resolução de 800 *pixels* por 600, por exemplo, irá gerar 480 mil valores para cada espectro de cor, por isso métodos como *grayscale* e o redimensionamento na escala da imagem são extremamente úteis para este tipo de rede para reduzir a quantidade de dados inciais (BONNER, 2019).

Alguns destes métodos de processamento de imagens serão detalhados na Seção 2.2.

2.2 PROCESSAMENTO DIGITAL DE IMAGENS

O Processamento Digital de Imagens pode ser definido como qualquer forma de processamento de dados em que a entrada e a saída são imagens e é muito mais complexo do que aparenta. Sua primeira aplicação foi para o envio de imagens a jornais através do sistema Bartline, em 1920. Este sistema consistia numa rede de cabos submarinos que ligavam Nova York a Londres com o objetivo de reduzir o tempo do transporte das imagens entre as cidades, antes transportadas fisicamente de um lugar para outro, de vários dias para 3 horas.

Dentre os problemas iniciais que o sistema apresentava, um deles era o nível do brilho das imagens, que eram armazenados em fitas perfuradas e era possível utilizar apenas 5 níveis de brilho. Em 1929 esse número foi expandido, passando para até 15 níveis de brilho. Este ajuste ajudou em muito na melhora da qualidade das

imagens armazenadas, mas hoje este sistema não se encaixa mais na realidade em que vivemos (GONZALES; WOODS, 2009).

O processamento de imagens como é conhecido hoje foi iniciado pela *Jet Propulsion Laboratory* (JPL) no final dos anos 1960, com o objetivo de que os sinais analógicos enviados pela sonda Ranger pudessem ser convertidos para imagens digitais (TOMAYKO, 1988). A área de processamento de imagens, antes pequena e restrita, cresceu rapidamente tornando-se presente no dia a dia de todas as pessoas. Procedimentos computacionais são utilizados para realçar o contraste das imagens radiográficas importantes para as indústrias e a medicina. Geógrafos utilizam técnicas para estudar padrões de poluição em imagens de satélite. Na arqueologia são utilizadas técnicas de processamento de imagens para restaurar e realçar detalhes de fotografias, mantendo o registro histórico de épocas onde só haviam fotografias analógicas. E mais recentemente, aliando técnicas de processamento de imagens a algoritmos de inteligência artificial, foi possível obter o primeiro registro fotográfico de um buraco negro, algo que até então era considerado impossível de se observar.

Tendo em mente o potencial do processamento de imagens, pode-se estudar algumas técnicas para o tratamento de imagens digitais: para um computador, uma imagem é uma matriz definida por X e Y, sendo que X é a largura da matriz e Y é sua altura. Cada coordenada XY da matriz representa um *pixel* da imagem, que é o termo utilizado para representar partes da imagem. As técnicas de processamento de imagens agem sobre estes *pixels*, alterando suas propriedades para se obter os resultados esperados (GONZALES; WOODS, 2009). Algumas destas técnicas atuam sobre os *pixels* individualmente, outras agem sobre um conjunto dos mesmos.

Neste capítulo serão abordadas algumas das técnicas utilizadas no processamento de imagens para realizar Transformações de Cores (Seção 2.2.1) e Filtragem (Seção 2.2.2), bem como o funcionamento da Visão Computacional (Seção 2.3).

2.2.1 Transformações de cores

A maioria das cores visíveis ao olho humano pode ser representada como a combinação das cores Vermelho, Verde e Azul, comumente referenciadas pelas letras R, G e B, respectivamente, e podendo assumir valores de 0 a 255 (um *byte*) (MARTINS, 2014).

As técnicas de transformação de cores trabalham com os valores R, G e B de cada *pixel* da imagem individualmente, alterando-os de forma a se obter resultados diferentes da imagem original.

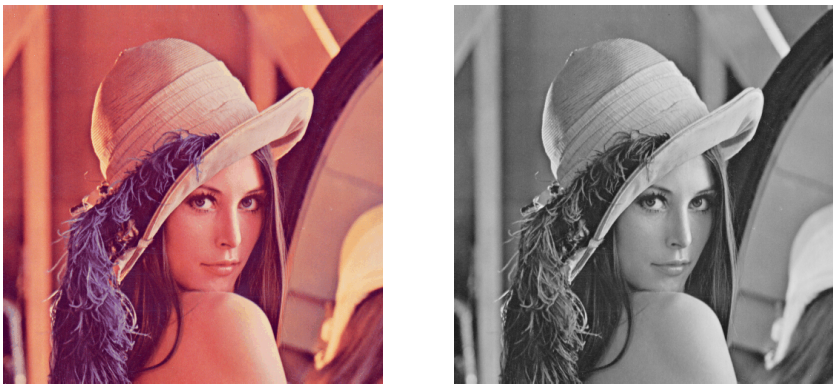
Imagens coloridas também podem apresentar uma quarta propriedade em cada *pixel*, sendo ela o nível de cinza, que representa a sua luminosidade. Esse valor pode

ser calculado com a seguinte fórmula:

$$C = 0.3 * (R/255) + 0.59 * (G/255) + 0.11 * (B/255) \tag{2.3}$$

Nesta Equação os valores R, G e B são o valor do *byte* que representa a cor vermelho, verde e azul, respectivamente, e C o *byte* resultante indicando o tom de cinza daquele *pixel*. Como pode ser visto na Figura 4, esta técnica remove todas as cores da imagem deixando apenas os tons de cinza.

FIGURA 4 – Comparação de imagens após a aplicação da técnica de Grayscale.



(a) Imagem original (b) Imagem em Grayscale

Fonte: Os próprios autores.

Outra transformação de cores comum no Processamento Digital de Imagens é o Negativo. O processo de gerar o Negativo de uma imagem consiste em inverter os valores dos *bytes* R, G e B na imagem através do cálculo:

$$X = 255 - C \tag{2.4}$$

Onde C é o valor do *byte* e X é o novo valor do *byte*. Fazendo isso com os três espectros de cores obtém-se a imagem negativa da original semelhante ao negativo de uma película de máquina fotográfica, como representado na Figura 5. Além de aplicações artísticas, o negativo de uma imagem pode ser utilizado para o realce dos detalhes brancos ou cinzas em regiões mais escuras da imagem (GONZALES; WOODS, 2009).

FIGURA 5 – Comparação de imagens após a aplicação da inversão de cores.

(a) Imagem original (b) Imagem negativa

Fonte: Os próprios autores.

2.2.2 Filtros

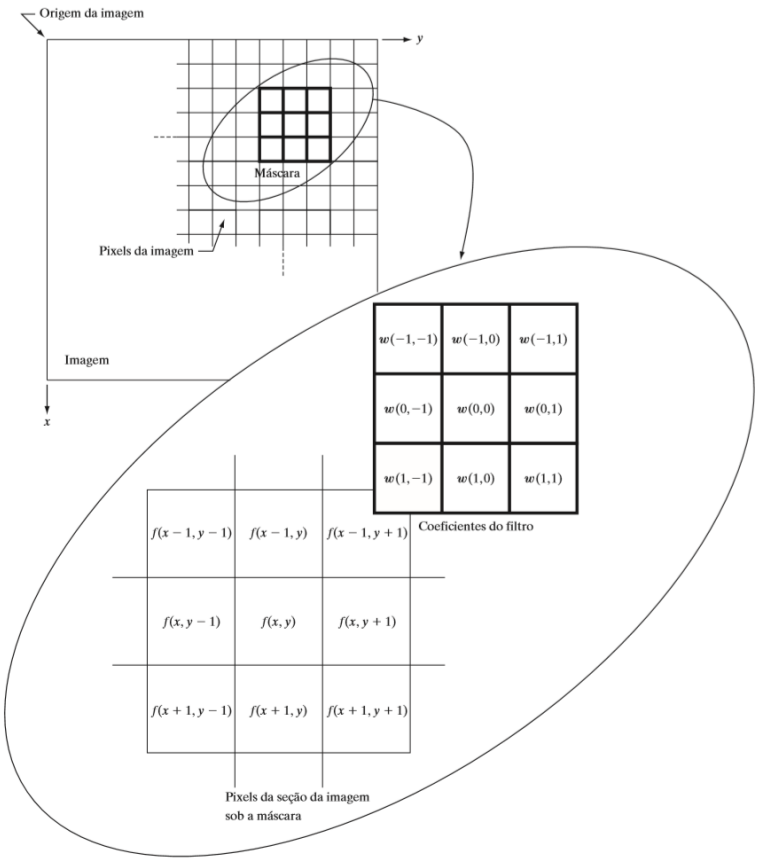
No processamento de imagens digitais, filtros podem ser utilizados para o tratamento de ruídos gerados pelo meio de captação da imagem, atenuação ou destaque de características da imagem, ou até mesmo para aplicações artísticas (PATEL, 2014).

A aplicação de filtros consiste em alterar os valores de um *pixel* utilizando operações matemáticas predefinidas, baseando-se nos *pixels* ao seu redor e aplicando uma máscara sobre a imagem. Essa máscara consiste em uma matriz quadrada, que será deslocada ao longo dos *pixels* da imagem, indicando quais deles serão considerados no cálculo (GONZALES; WOODS, 2009). O *pixel* localizado no centro da máscara é o que será alterado na operação, resultando em um novo *pixel* de mesmas coordenadas. Ao longo da aplicação da máscara sobre a imagem original, uma nova imagem será criada. Em aplicações onde a máscara se desloca de forma linear pela imagem este filtro é chamado de *Filtro Espacial Linear*, já para as aplicações onde a máscara percorre a imagem de forma não linear estes são chamados de *Filtros Espaciais Não Lineares* (GONZALES; WOODS, 2009).

2.2.3 Filtros Lineares

Como dito anteriormente, a aplicação de um filtro espacial linear consiste em aplicar operações matemáticas sobre um *pixel* baseando-se nos valores dos *pixels* vizinhos. A Figura 6 representa uma máscara 3x3 aplicada sobre uma imagem onde em qualquer posição (x, y) na imagem, a resposta $g(x, y)$ é a soma dos produto dos

FIGURA 6 – Representação de uma máscara 3x3 aplicada de forma linear sobre a imagem.



Fonte: (GONZALES; WOODS, 2009).

coeficientes do filtro com os *pixels* da imagem que são englobados pelo Filtro 2.5

$$\begin{aligned} g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + ... \\ +w(0,0)f(x,y) + ... + w(1,1)f(x+1,y+1) \end{aligned} \tag{2.5}$$

Segundo (GONZALES; WOODS, 2009), a filtragem espacial linear de uma imagem de dimensões M x N com um filtro de dimensões m x n é definida pela Equação 2.6, onde x e y variam de forma que cada *pixel* em w percorre todos os *pixels* em f. É possível trabalhar com máscaras de tamanhos mistos, porém a utilização de matrizes quadradas de tamanho ímpar simplifica os cálculos, já que sempre apresentarão um *pixel* central.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)f(x+s,y+t) \tag{2.6}$$

Para gerar um filtro espacial linear m x n deve-se especificar os coeficientes da máscara mn, baseando-se no objetivo do filtro.

Uma das aplicações de filtros lineares é a suavização de imagens. Filtros de suavização são utilizados para dar um aspecto borrado à imagem, reduzindo assim seus ruídos. Eles são muito utilizados como pré-processamento no tratamento de imagens onde deseja-se fazer a extração de grandes objetos ou realizar conexões entre pequenas partes descontinuadas (GONZALES; WOODS, 2009).

Este resultado pode ser obtido através da utilização de filtros de média, também conhecidos como filtros *passa-baixa*. Esse tipo de filtro tem como objetivo suavizar a imagem substituindo o valor de cada *pixel* pela média dos níveis de intensidade dos *pixels* da vizinhança (RIBEIRO, 2015). Por exemplo, tendo a janela 3x3 abaixo:

$$\begin{bmatrix} 244 & 247 & 245 \\ 252 & 12 & 238 \\ 244 & 245 & 250 \end{bmatrix} \tag{2.7}$$

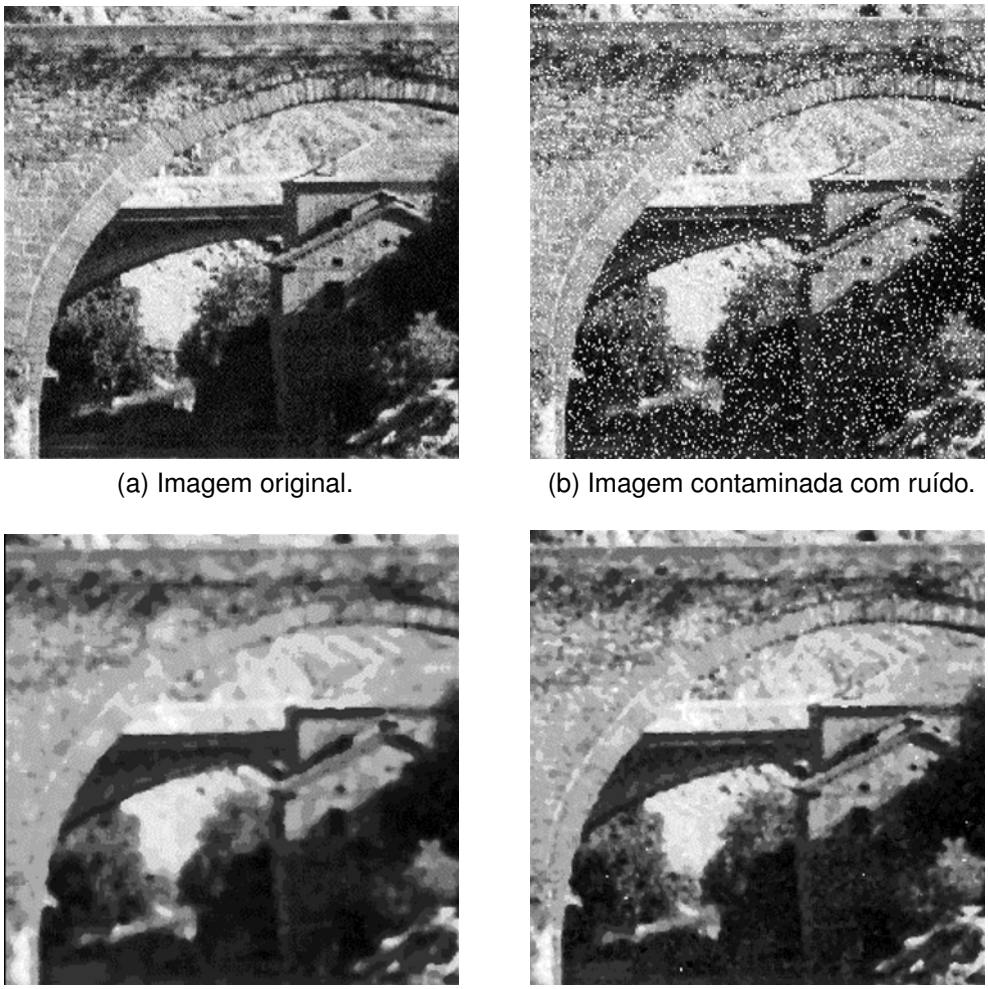
Calculando a média dos valores teríamos (12 + 238 + 244 + 245 + 245 + 247 +250 + 259) / 9 = 219, assim o *pixel* central que antes tinha o valor 12 agora receberá o valor 219.

Um dos pontos negativos na utilização desta técnica é a tendência a remover as bordas dos objetos na imagem, já que bordas são definidas por transições abruptas no padrão de cores ou luminosidade dos *pixels*, transições estas que serão suavizadas com a aplicação de um filtro *passa-baixa*.

Uma alternativa ao filtro de média é o filtro da mediana, que apresenta as mesmas características de suavizar ruídos da imagem, porém sem apresentar tanta perda de qualidade das bordas, como pode ser visto na Figura 7, tendendo a ser os melhores filtros para remoção de ruído (RIBEIRO, 2015).

Seu funcionamento é semelhante ao filtro de média, porém aplicando ao novo *pixel* o valor da mediana ao invés do valor da média. Por exemplo, ordenando os valores da mesma janela 3x3 do exemplo anterior se obtém a lista [12, 238, 244, 245, 245, 247, 250 e 252]. Sendo assim o valor da mediana que será aplicado ao novo pixel é 245. A desvantagem do filtro da mediana em relação ao filtro de média é o seu custo de processamento, já que é necessário ordenar toda a vizinhança de cada pixel da imagem individualmente.

FIGURA 7 – Comparação entre filtro de média e mediana.



(a) Imagem original. (b) Imagem contaminada com ruído.

(c) Imagem após filtro de média 3x3. (d) Imagem após filtro de mediana 3x3.

Fonte: (RIBEIRO, 2015).

2.2.4 Filtros Não Lineares

Os filtros não lineares, também conhecidos como filtros *passa-alta*, são utilizados para realce de bordas e detalhes finos da imagem, estando muito presentes em aplicações para detecção de bordas.

A detecção de bordas no processamento de imagens digitais se define pela detecção de regiões onde há uma variação abrupta nos níveis de cores entre os *pixels* da imagem (MARQUES; NETO, 1999).

Essa técnica normalmente se dá através do realce das linhas verticais e horizontais da imagem, utilizando-se dois filtros aplicados em sequência. Existem várias máscaras para a detecção de bordas, normalmente 3x3, como os operadores de Roberts e de Sobel (Figura 8).

FIGURA 8 – Máscaras utilizadas nos operadores de Roberts e Sobel.

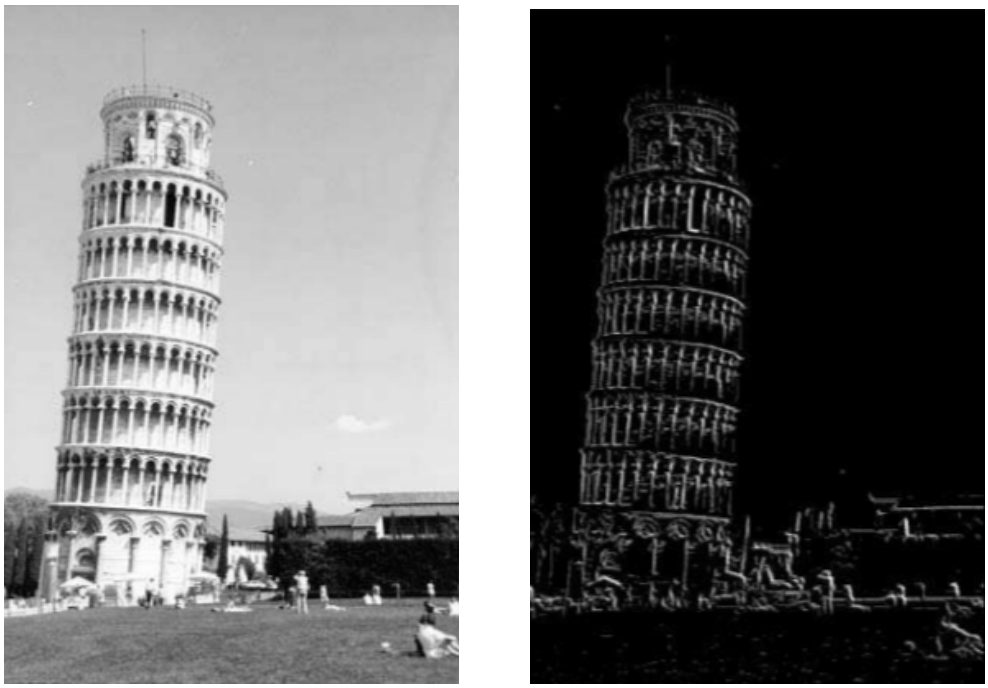
Operador	Vertical	Horizontal
Roberts	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Sobel	$\frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Fonte: (MARQUES; NETO, 1999).

O operador de Sobel realça as linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados. Ele consiste na aplicação de duas máscaras em sequência, uma para realçar as linhas verticais da imagem e outra para realçar as linhas horizontais, compondo o resultado esperado (MARQUES; NETO, 1999).

Este operador tem a vantagem de ser menos suscetível a ruídos na imagem, porém sofre dificuldades para detectar objetos menores e mais granulados.

FIGURA 9 – Comparação de imagens após a detecção de bordas de Sobel.



(a) Imagem original. (b) Imagem após a detecção de bordas de Sobel.

Fonte: (MARQUES; NETO, 1999)

O operador de Roberts trabalha de maneira semelhante ao operador de Sobel, tendo a vantagem de conseguir detectar objetos menores e mais detalhados, porém acaba sendo mais suscetível aos ruídos presentes na imagem.

FIGURA 10 – Comparação de imagens após a detecção de bordas de Roberts.

(a) Imagem original

(b) Imagem após a detecção de bordas de Roberts.

Fonte: Os próprios autores.

2.3 VISÃO COMPUTACIONAL

A Visão Computacional pode ser descrita como o ato de simular o funcionamento da visão humana em um computador, sendo utilizada principalmente na detecção de objetos nas mais diversas áreas. Ela foi iniciada na década de 1950, com o uso de algumas das primeiras redes neurais para detectar os limites de um objeto e para classificar objetos simples em categorias como círculos e quadrados (THOMPSON, 2019).

Nos anos 1970 começou a ser utilizada comercialmente, através do processamento de textos digitados e manuscritos com o objetivo de interpretá-los para deficientes visuais. Já na década de 1990, com o grande avanço da *internet*, foi grande o aumento da utilização da visão computacional, devido à disponibilidade de imagens online (THOMPSON, 2019).

A Visão Computacional pode ser dividida em três etapas básicas:

1. Aquisição de Imagem.
2. Processamento de imagem.
3. Entendimento de imagem.

Na Etapa 1 é feita a captação das imagens de entrada, podendo ser imagens isoladas ou em vídeos. Na Etapa 2 é feito o processamento da imagem, utilizando algumas das técnicas mencionadas nas Seções 2.2.1 e 2.2.2, com o objetivo ressaltar as características que se busca na imagens para então, na Etapa 3, fazer a identificação de objetos ou da informação que se busca, normalmente utilizando algoritmos de IA que se baseiam em bancos de imagens conhecidas para reconhecer os padrões e encontrar o que se deseja (ACADEMY, 2018).

3 REVISÃO DA LITERATURA

Neste capítulo são descritas as obras literárias relacionadas ao projeto.

3.1 AN APPROACH TO LICENSE PLATE RECOGNITION

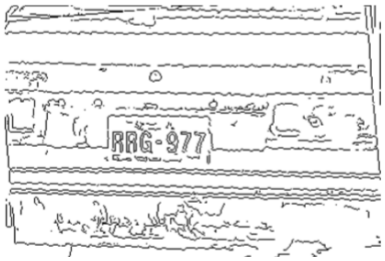
Em seu artigo, *An Approach To License Plate Recognition*, os autores, Parker e Federl definem uma sequência de passos em sua abordagem para detecção e leitura da placa de veículos, sendo estes:

- Entrada da imagem
- Processamento da imagem (conversões RGB e atenuação de ruídos)
- Detecção de bordas
- Localização dos caracteres
- Algoritmo genético para encontrar a placa
- Saída da imagem com a placa encontrada

No primeiro passo, o processamento da imagem foi feito utilizando a técnica *grayscale*, detalhada na Seção 2.2.1. Após a conversão para níveis de cinza, foi aplicado à imagem um filtro de mediana com uma máscara 5x5, atenuando assim os ruídos presentes na imagem. Com a imagem já processada, os autores utilizaram o algoritmo de detecção de bordas de *Shen-Castan* para realizar o segundo passo do fluxo. Um dos problemas encontrados na localização dos caracteres na imagem foi a separação das letras do resto da imagem, já que essa detecção é afetada por fatores como luminosidade, sujeira etc. A solução encontrada pelos autores foi a aplicação da técnica de binarização da imagem, resultando nas imagens mostradas na Figura 11.

FIGURA 11 – Resultados atingidos pelos autores

(a) Imagem original



(b) Imagem após detecção das bordas

(c) Imagem após binarização

Fonte: PARKER e FEDERL (1997)

Para fazer a identificação dos caracteres na imagem final, os autores utilizaram algoritmos para encontrar regiões que tivessem pelo menos 8 linhas conectadas, que definem o padrão das letras em placas de automóveis. Para eliminar possíveis falsos positivos, foram delimitadas condições aplicadas sobre essas regiões que deveriam ser atendidas, sendo elas:

- Largura superior a 4 *pixels*
- Altura superior a 8 *pixels*
- Área da região menor do que 100 *pixels*
- Proporção entre altura e largura menor do que 0,1
- Proporção entre altura e largura maior do que 1,3

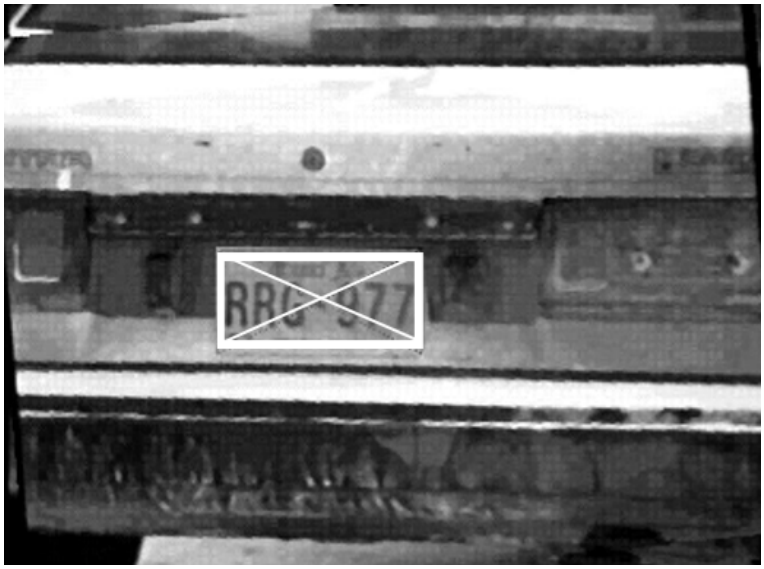
Quando as condições acima são atendidas, as regiões que possivelmente contém caracteres são delimitadas por retângulos, resultando na Figura 13.

FIGURA 12 – Imagem após a delimitação das regiões com caracteres.

Fonte: PARKER e FEDERL (1997).

Para encontrar a placa completamente, os autores utilizaram algoritmos de otimização e algoritmos genéticos, onde uma série de cálculos fazia a pontuação do resultado de cada iteração dos algoritmos, buscando sempre o resultado que mais se aproximava da placa ideal do veículo.

FIGURA 13 – Resultado após a identificação da placa do veículo.



Fonte: PARKER e FEDERL (1997).

3.2 AUTOMATIC VEHICLE IDENTIFICATION BY PLATE RECOGNITION

Os autores definem que o reconhecimento de placas ou *License plate recognition* (LPR), é cada vez mais importante pelo fato do papel de grande importância que os

carros possuem em nosso cotidiano. O algoritmo proposto pelos autores, com a imagem captada (Figura 14 (a)) por uma câmera alocada em uma distância de 4 a 5 metros do veículo, faz a separação da imagem em segmentos, sendo a primeira parte a extração da placa do veículo para ser tratada separadamente. A primeira parte do processo tem como ponto de início a binarização (Figura 14(b)) da imagem para a aplicação do algoritmo *Smearing*, no qual ele faz uma varredura vertical e horizontal comparando o número de *pixels* brancos e pretos e se suas quantidades estiverem localizadas em um intervalo determinado previamente os *pixels* de cor branca têm sua cor convertida para preto, fazendo com que a imagem obtenha um aspecto de borrada. Em seguida é aplicado uma operação morfológica, dilatação, a qual é responsável por localizar a região em que a placa se encontra. Esse processo pode ser observado na Figura 15.

FIGURA 14 – Comparação de imagens após a binarização.

(a) Imagem original (b) Imagem após binarização

Fonte: Ozbay e Ercelebi (2017).

FIGURA 15 – Localização da placa

Fonte: Ozbay e Ercelebi (2017)

Apenas depois que a placa fica visível na imagem, ela é extraída para que possa ser manipulada separadamente.

FIGURA 16 – Extração da placa.

Fonte: Ozbay e Ercelebi (2017).

Agora com a imagem contendo apenas a placa, é feita uma filtragem com um algoritmo de filtro de suavização para que ruídos possam ser retirados da imagem, assim sobrando apenas os caracteres da placa. Tendo essa imagem tratada, é aplicado novamente um processo de dilatação para que os caracteres pertencentes a placa sejam separados, se por motivos externos os mesmos estejam muito próximos um do outro. Depois, como o processo de dilatação o algoritmo de *Smearing* é aplicado novamente, mas agora com o propósito de achar a localização de cada um dos caracteres contidos na imagem da placa. O resultado obtido pode ser conferido na Figura 17.

FIGURA 17 – Local do caractere da placa

Fonte: Ozbay e Ercelebi (2017)

O próximo passo foi a extração de cada caractere separadamente, já que no passo anterior realizado pelos autores foi encontrada a localização de cada um deles, tendo cada um dos caracteres devidamente retirados eles passam por um processo de refinamento para que eles fiquem sólidos e não sobre nenhuma lacuna em branco entre cada pixel. Feito isso, todos foram convertidos para um tamanho de 36x18, para que possam ser comparados com o banco de dados que contém todos os caracteres do alfabeto conhecido.

Para a realização da comparação com o banco de dados, os autores utilizaram um método correlação desenvolvido por Horowitz. Esse método mede o coeficiente de correlação entre uma imagem conhecida com uma desconhecida de mesmo tamanho. Existem dois métodos de correlação: o *Auto-correlation function (ACF)* e o *Cross-correlation function (CCF)*, no problema proposto os autores utilizam do método CCF que ao contrário do ACF utiliza duas imagens para fornecer um resultado que aponta uma medida de semelhanças e propriedades compartilhadas pelas duas. A correlação

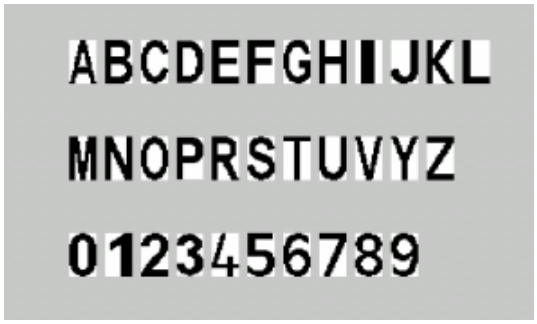
cruzada é obtida através de:

$$R(m,n) = \frac{\sum_j \sum_k F_1(j,k)F_2(j-m+(M+1))/2, k-n+(N+1)/2)}{[\sum_j \sum_k |F_1(j,k)|^2]^{1/2}[\sum_j \sum_k |F_2(j-m+(M+1))/2, k-n+(N+1)/2|^2]^{1/2}} \tag{3.1}$$

Sendo $F_1(j,k)$ e $F_2(j,k)$ para $1 \leq j \leq J$ e $1 \leq k \leq K$, representam o par de imagem sendo comparados, e m e n sendo valorados como $1,2,3...M$ e $1,2,3...N$ respectivamente , o qual M e N são números ímpares inteiros.

Para comparar o par de imagem, os autores utilizaram um banco de dados preenchido com os caracteres do alfabeto turco que englobam no total de 33 caracteres sendo eles 23 letras e 10 números.

FIGURA 18 – Banco de Dados utilizado para comparação



Fonte: Ozbay e Ercelebi (2017)

Os autores realizaram esse teste no MatLab 6.5 com um total de 340 imagens de tamanho padrão 1600x1200, onde cada foto foi tirada em uma condição de iluminação do ambiente diferente. O resultado obtido foi o seguinte:

- Localização da região da placa: 97.6%
- Separação dos caracteres: 96%
- Comparação e reconhecimento da placa: 98.8%

Assim totalizando 92.57% de precisão no desempenho final.

3.3 RECONHECIMENTO DE CARACTERES ATRAVÉS DE REDES NEURAIS ARTIFICIAIS COM APLICAÇÃO A PLACAS DE AUTOMÓVEIS

Para facilitar o processamento das imagens, a solução proposta pelos autores faz a captação das imagens com câmeras digitais de menor resolução que produzia fotografias já em tons de cinza, já que imagens com alta resolução apresentam

problemas no momento da detecção das bordas com luminosidade e maior grau de granulação, devido ao alto nível de detalhamento das imagens.

Na solução dos autores, primeiramente a imagem, que já era representada em tons de cinza, foi submetida a uma binarização para então aplicar um filtro Gaussiano, com a finalidade de atenuar os ruídos da imagem e facilitar a detecção dos caracteres. Após isso, foi aplicada à imagem uma técnica chamada Afinamento, para que regiões da imagem que apresentavam várias linhas paralelas fossem reduzidas para apenas uma única linha. Isto é feito avaliando-se *pixel* por *pixel* da imagem e verificando em sua vizinhança quais dos *pixels* poderiam ou não ser removidos da imagem.

FIGURA 19 – Comparação entre a imagem original e imagem após o afinamento.

(a) Imagem original. (b) Imagem após a aplicação da técnica de afinamento.

Fonte: FONTANA e MARIM (2010).

A imagem tratada, foi utilizada então uma Rede Neural do tipo *feedforward*, treinada 20 imagens dos dígitos de 0 a 9 e de cada letra individualmente, obtendo aproximadamente 70% de taxa de acerto dentre os dígitos, resultado estes que segundo os autores poderia ser ainda maior com o aumento do banco de imagens utilizadas no treinamento e uma melhoria na capacidade de processamento da máquina utilizada.

Após este treinamento o algoritmo foi capaz de reconhecer corretamente 55 de 76 placas analisadas, utilizando imagens de 1024x768 *pixels* de resolução. Após a redução para imagens de 360x70 *pixels*, este número aumentou para 220 acertos dentre 222 placas analisadas, já que o tempo de processamento e a quantidade de detalhes na imagem reduziu drasticamente.

FIGURA 20 – Resultado após a identificação da placa do veículo.

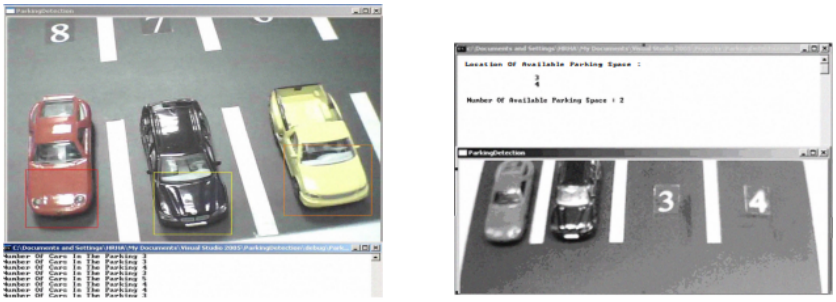
Fonte: FONTANA e MARIM (2010).

3.4 A VISION-BASED PARKING LOT MANAGEMENT SYSTEM

Os autores propõem a utilização de câmeras no auxílio do gerenciamento do estacionamento, bem como para a melhoria na segurança do estabelecimento, trazendo a resolução através de um sistema desenvolvido com a utilização *Haar-like features* e da biblioteca de processamento de imagem *Open Computer Vision Library* (OpenCV). Para treinamento da classificação dos objetos foi utilizado o *haartraining* que está disponível na biblioteca utilizada pelos autores o OpenCV,o algoritmo necessita para seu treinamento que sejam disponibilizados dois conjuntos de imagens para processamento, denominados de imagens positivas e negativas, no qual o grupo de imagens negativas não possui o objeto a ser detectado na imagem enquanto o grupo de imagens positivas possui. Além do objeto estar presente na imagem, são informadas também as dimensões e a posição do objeto na imagem.

O sistema desenvolvido pelos autores foi treinado para reconhecer as vagas e se estão ou não ocupadas, assim fazendo seu treinamento com imagens de carros em vários ângulos diferentes. Como o custo de sensores poderia impactar na implementação desse sistema nos estacionamentos, toda a parte de captura de imagens foi feita a partir de uma simples *WebCam*. A localização de cada vaga é especificada pela ferramenta *"Object Maker"* da biblioteca. Essa localização tem como objetivo determinar onde o algoritmo deve detectar a presença ou não de um veículo.

FIGURA 21 – Identificador de carros ocupando vagas.



Fonte: Lin, Chen e Liu (2006).

TABELA 1 – Resultado dos testes.

Total de Leituras	**1 Carro**		**2 Carros**		**3 Carros**	
	Acerto	Erro	Acerto	Erro	Acerto	Erro
30	30	0	28	2	27	3
	100%	0	93,3%	6,7%	90%	10%

Fonte: Lin, Chen e Liu (2006).

Ao realizar os testes os autores perceberam que a precisão da identificação diminuía de acordo com a quantidade de vagas a serem identificadas como ocupadas aumentavam, como indicado na Tabela 1. No total foram testados 300 casos e em 271 deles foi obtido o retorno desejado, o que convertido significa 90,33% de taxa de precisão.

Os autores afirmam que a implementação de um sistema como esse em um estacionamento é factível, porém melhorias podem ser realizadas no modo de implementação no ambiente, como câmeras de melhor qualidade, melhor posicionamento na captação das imagens, sistema de instalação da câmera que permita alterações no ângulo e inclinações, assim possibilitando que as câmeras alcancem um maior espaço, além de aumentar o número de imagens utilizadas no treinamento do algoritmo.

3.5 DETECTION AND CLASSIFICATION OF OBJECT IN IMAGE FOR VEHICLE TRACKING

Nesse artigo, o autor apresenta uma proposta para detecção de objetos e rastreamento dos mesmos utilizando imagens captadas a partir de um *drone* aéreo.

O algoritmo utilizou uma base de 287 imagens retiradas de vídeos de uma rodovia gravados pelo próprio *drone* em alturas variadas. Essa base de imagens foi então categorizada utilizando técnicas como a *Bag-of-features*, que é uma abordagem popular para a classificação visual de objetos por ser eficaz e simples. Essa técnica calcula as características de uma imagem e através disso, classificá-las em grupos. Estes grupos são então utilizados para gerar um dicionário visual, onde cada imagem representa um grupo de características encontradas pelo algoritmo.

A detecção dos objetos em cena se baseou em três tipos distintos de veículos, sendo eles: Carros, caminhões e motos, onde cada um dos tipos foi testado individualmente, chegando a uma taxa de acerto de 83%, resultado que foi afetado devido à variações na iluminação das imagens captadas em relação às imagens utilizadas no treinamento do algoritmo.

FIGURA 22 – Exemplo de imagem de entrada e saída do sistema.

(a) Imagem de entrada. (b) Veículos detectados na cena.

Fonte: MONTANARI (2015).

Os objetos detectados dessa forma foram armazenados em uma lista. A cada nova imagem captada, é feita uma nova busca por objetos que então são comparados aos objetos da lista considerando-se dois fatores: O histograma da imagem e a correspondência das características entre eles. Se a comparação dos objetos for positiva, então a posição do objeto é atualizada na lista e uma nova imagem é associada a ele, caso não seja encontrado um objeto correspondente na lista o novo objeto será adicionado a ela.

Uma das dificuldades encontrada pelo autor foi a taxa de captação das imagens, já que a câmera captava 60 quadros por segundo, aumentando a quantidade de processamento necessária e dificultando a detecção de múltiplos veículos na imagem, isso foi resolvido reduzindo a taxa de captação para um único quadro por segundo.

3.6 COMPRATIVO DOS TRABALHOS

Como pode se observar, o tratamento da imagem possui uma grande importância na eficiência das soluções apresentadas. Transformações de cores foram utilizadas em conjunto com Filtros sobre as imagens a fim de se realçar e extrair as informações desejadas. Esse tratamento pode ser observado principalmente no artigo escrito por FONTANA e MARIM (2010), onde foram utilizados tanto a filtragem quanto a transformação de cores para preparar a imagem para ser enviada a uma Rede Neural. Um processo semelhante foi utilizado no trabalho desenvolvido por PARKER e FEDERL (1997), onde técnicas de filtragem foram utilizada sobre as imagens a fim de re realçar as bordas presentes, permitindo que o algoritmo reconhecesse com mais facilidade os caractéres nas placas dos veículos.

4 METODOLOGIA

Para que o objetivo do projeto pudesse ser alcançado, foi desenvolvida um Rede Neural Convolucional para identificar o modelo e o tipo dos veículos. Também foram desenvolvidos algoritmos para fazer o cadastro e alocação destes em suas respectivas vagas em um banco de dados.

Neste capítulo são apresentadas as ferramentas utilizadas para o desenvolvimento do projeto, bem como o funcionamento da Rede Neural e do algoritmo de alocação de vagas.

4.1 FERRAMENTAS UTILIZADAS

Na primeira etapa, a escolha das linguagens utilizadas bem como os ambientes de programação de cada uma tiveram um papel fundamental. Foram escolhidas C# .NET Core para as APIs de controle e de classificação, e Python foi aplicado á Rede Neural.

Como citado, para o desenvolvimento das APIs de controle de estacionamento e de classificação, trabalhou-se com C# .NET Core, já que um *framework* multiplataforma elimina possíveis problemas de compatibilidade entre sistemas operacionais. Além disso, é uma linguagem ágil, simplificando o desenvolvimento do projeto. O C# .NET Core possui ainda bibliotecas que ajudam na criação de interfaces, como o Blazor, empregue nas telas de utilização do sistema, e o *Entity Framework*, uma das principais ferramentas da linguagem para se trabalhar com banco de dados. O ambiente de desenvolvimento escolhido foi o Visual Studio 2019 Comunity, da Microsoft, que possui ferramentas voltadas especialmente à programação utilizando C#, bem como outras ferramentas que auxiliam na depuração do código e controle de versão do mesmo.

Já para o desenvolvimento da Rede Neural, utilizou-se Python 3.7, que, assim como o C# .NET é uma linguagem de desenvolvimento ágil e com bibliotecas que simplificam o trabalho voltado à Inteligência Artificial, além de ter uma vasta comunidade nessa área. O ambiente de programação escolhido foi o PyCharm, da JetBrains, específico para se trabalhar com Python, contendo ferramentas que auxiliam na depuração do código e versionamento.

O *Dataset* utilizado para o treinamento e teste da CNN foi desenvolvido pela Universidade de *Stanford*, contendo 16185 imagens de veículos distribuídos em 196 modelos diferentes. Este *Dataset* foi escolhido por conter imagens em vários ângulos diferentes, além do grande número de veículos variados armazenados no mesmo.

Como banco de dados optou-se pelo SQL Server, uma ferramenta robusta e

confiável que atende bem aos requisitos do sistema. Outro motivo para sua escolha foi a familiaridade dos desenvolvedores com a ferramenta, já que a mesma foi utilizada no decorrer do curso e em ambiente profissional dos mesmos e possui uma ótima integração com o Entity Framework, fazendo com que o gerenciamento das tabelas seja realizado de forma simples e eficiente.

Por fim, também foi utilizada a ferramenta *Swagger*, que auxilia no gerenciamento e desenvolvimento de sistemas web e APIs, e ainda ajuda a testar e visualizar os resultados de cada parte do sistema durante a fase de desenvolvimento. Através dele podem ser feitas requisições para o sistema, possibilitando a manutenção do banco de dados.

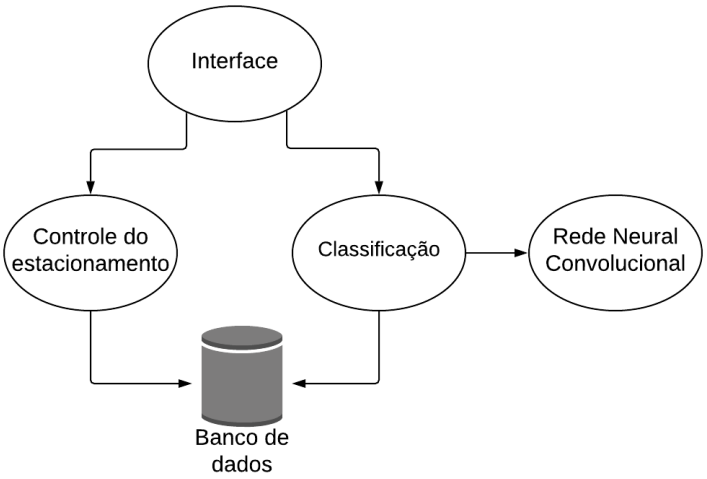
As versões destes e outros recursos utilizados podem ser vistos no anexo A.

4.2 MÉTODOS

O sistema foi dividido em 4 APIs, sendo três desenvolvidas utilizando .NET Core e uma outra desenvolvida em Python, cada uma responsável por parte do fluxo de classificação do veículo. A comunicação entre as APIs é feita através de requisições HTTP, conforme a Figura 23.

- Rede Neural Convolucional (Python)
- Interface (.NET Core)
- Classificação (.NET Core)
- Controle de estacionamento (.NET Core)

FIGURA 23 – Comunicação entre as APIs.



Fonte: Os próprios autores.

O processo de classificação e alocação do veículo pode ser visto no Anexo 6, se iniciando com a escolha da imagem do veículo. Essa imagem é enviada para API de Classificação, que por sua vez faz a comunicação com a API da CNN. Esta API então irá fazer um pré-processamento da imagem e então classifica-la utilizando a CNN e retornando o *Label* que obteve a maior probabilidade de acerto para API de Classificação. Este *Label* será utilizado para verificar se o veículo esta cadastrado no banco, caso esteja os dados dele serão exibidos na tela, caso contrário uma mensagem de erro será exibida. Com a classificação bem sucedida é habilitada a opção de alocação do veículo em uma vaga do estacionamento. Isso é feito na API de Controle do Estacionamento, onde é buscada por uma vaga compatível com o veículo em questão e então exibindo a vaga selecionada na tela.

4.2.1 Rede Neural Convolucional

No início do projeto, o desenvolvimento da CNN foi feito utilizando o *framework* de desenvolvimento de Inteligência Artificial da Microsoft, o ML.NET, que havia se mostrado promissor e apresentou diversas ferramentas e a possibilidade de usar modelos prontos para realizar as classificações. No entanto, o mesmo tinha um alto nível de encapsulamento, que impossibilitava saber exatamente a maneira com que a Rede Neural funcionava, dificultando alteração de parâmetros dos modelos desenvolvidos. Com isso, a Rede Neural passou a ser desenvolvida em Python, tendo como base o algoritmo apresentado por Victor Zhou, que fez a classificação de imagens de números escritos a mão do *Dataset* MNIST, um dos *Datasets* mais utilizados para demonstrar o funcionamento de uma CNN.

O fluxo do funcionamento da Rede Neural segue as etapas:

1. Carrega os *Labels* do arquivo local
2. Relaciona os arquivos do *Dataset* com as respectivas *labels*
3. Para cada imagem:
    a) Pré-processamento da imagem.
    b) Camada de convolução.
    c) Camada de maximização.
    d) Camada de ativação.
    e) Cálculo da precisão e perda de entropia.
    f) Retropropagação na camada de ativação.
    g) Retropropagação na camada de maximização.

h) Retropropagação na camada de convolução.

O processo inicia-se carregando as *labels* de um arquivo local e organizando-as em um dicionário, sendo a chave a *label* em si e um valor numérico que representará aquela *label*. Em seguida todos os diretórios das imagens do *Dataset* são armazenadas em uma lista acompanhadas da sua respectiva *label*. Com labels e dataset em mãos, o algoritmo então será aplicado em cada uma das imagens executando as etapas do treinamento.

A imagem passa por um pré-processamento com o intuito de reduzir a quantidade de dados que serão enviados para as próximas etapas. Passando por um redimensionamento para uma resolução de 200x200 pixels, tem seu tamanho reduzido, de forma que todas as entradas da Rede Neural sejam padronizadas. Então os espectros RGB são convertidos para *Grayscale*, detalhado na Seção 2.2.1, utilizando as ferramentas da biblioteca do *OpenCV* (Código 1). Por fim, a imagem passa por um processo de Filtragem, utilizando um Filtro Gaussiano, que é um tipo de Filtro Linear, detalhado na Seção 2.2.2. A Filtragem é utilizada para reduzir a quantidade de ruídos na imagem que poderiam interferir na classificação (Figura 24). O Filtro Gaussiano foi escolhido por ser um filtro de suavização que consegue atenuar os ruídos, porém mantendo as bordas da imagem relativamente intactas.

FIGURA 24 – Comparação entre a imagem original e imagem após o tratamento.

(a) Imagem original.

(b) Imagem em Grayscale.

(c) Imagem filtrada.

Fonte: Os Próprios Autores.

Com a imagem processada, os dados são enviados para cada uma das camadas da Rede Neural seguindo a ordem citada anteriormente, de forma que o resultado do processamento de uma camada passa para próxima e, ao final, volta da última camada

CÓDIGO 1 – Método para pré-processamento da imagem.

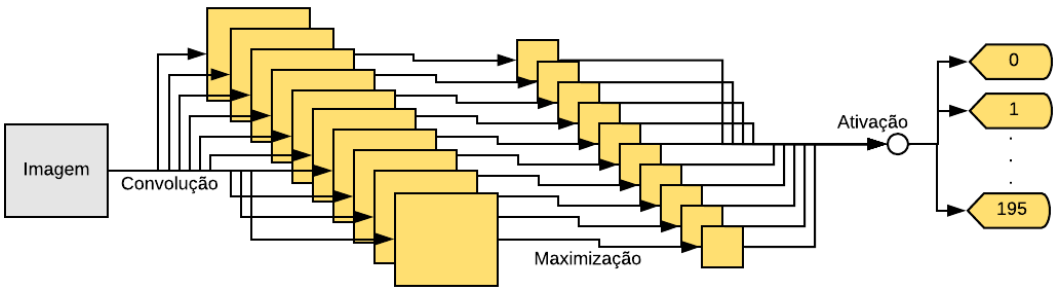
```
1
2   def ProcessImage(image):
3       resized = cv2.resize(image, (200, 200))
4       grayscale = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
5       filtered = cv2.GaussianBlur(grayscale, (3,3), 0)
6
7       return grayscale
8
9
```

FONTE: Os Próprios Autores.

em direção a primeira para que os parâmetros de cada uma sejam ajustados de acordo com o resultado obtido, nos métodos *Forward* e *Backpropagation* respectivamente (Códigos 2 e 3).

A imagem é inicialmente enviada para camada de Convolução, conforme a Figura 25, contendo dez neurônios que irão aplicar o processo de Convolução sobre a imagem, cada um utilizando um filtro único. O resultado de cada um dos neurônios convolucionais é então enviado para camada de Maximização, que irá realçar os dados obtidos na camada anterior. Por fim, os resultados da camada de Maximização são então enviados para camada de Ativação, onde serão convertidos para um vetor de probabilidades, onde cada posição representa a probabilidade da imagem pertencer a uma das 196 classes conhecidas pelo sistema.

FIGURA 25 – Modelo da CNN utilizada



Fonte: Os próprios autores.

Cada uma das camadas do modelo desenvolvido conta com um método de passagem *Forward* e outro método *Backpropagation*, responsáveis por fazer o processamento da imagem e preparar os dados para próxima camada, iniciando pela de convolução, como pode ser visto na Figura 6.

A camada de convolução consiste em uma lista de filtros, estes são representa-

CÓDIGO 2 – Método que executa a passagem adiante entre as camadas.

```
def Forward(image, label):

    out = model.convolutionLayer.Forward((image / 255) - 0.5)
    out = model.poolLayer.Forward(out)
    out = model.activationLayer.Forward(out)

    loss = -np.log(out[carsList[label]])
    acc = 0

    if np.argmax(out) == carsList[label]:
        acc = 1

   return out, loss, acc

```

FONTE: Os Próprios Autores.

CÓDIGO 3 – Método que executa a Retropropagação entre as camadas.

```
 def Backpropagation(gradient, learningRate=.005):

    gradient = model.activationLayer.
                        BackProp(gradient, learningRate)
    gradient = model.poolLayer.
                        BackProp(gradient)
    gradient = model.convolutionLayer.
                        Backpropagation(gradient, learningRate)

    return gradient

```

FONTE: Os Próprios Autores.

dos por matrizes 3x3. Nesta camada, a imagem é dividida em todos os conjuntos de *pixels* vizinhos numa área de 3x3, isso é feito para simplificar os cálculos no momento da convolução, já que fazer a iteração dos cálculos entre matrizes pode se tornar complicado e mais custoso do que o cálculo entre duas matrizes do mesmo tamanho. Para realizar os cálculos entre os conjuntos de *pixels* e os filtros, foi utilizada a biblioteca *Numpy*, onde o método *Sum* faz a soma dos produtos da multiplicação dos filtros pelo conjunto de *pixels* da imagem (Código 4), de maneira semelhante ao processo de filtragem de imagens detalhados na Seção 2.2.2.

Com a convolução realizada com todos os filtros, o resultado é enviado então para camada de maximização. Esta camada irá separar a imagem em conjuntos de *pixels* da mesma maneira que a camada anterior, porém escolhendo o maior valor dentre cada conjunto de *pixels* utilizando o método *Amax* da biblioteca *Numpy*. Este

valor é atribuído a uma posição correspondente da matriz resultante, como pode ser visto no Código 5.

CÓDIGO 4 – Método Forward da camada de convolução.

```
1
2  def Forward(self, input):
3    self.last_input = input
4
5    height = input.shape[0]
6    width = input.shape[1]
7    output = np.zeros((height - 2, width - 2, self.num_filters))
8
9    for region, i, j in self.IterateRegions(input):
10     output[i, j] = np.sum(region * self.filters, axis=(1, 2))
11
12   return output
```

FONTE: Os Próprios Autores.

CÓDIGO 5 – Método Forward da camada de maximização.

```
1
2  def Forward(self, input):
3    self.last_input = input
4
5    height, width, numFilters = input.shape
6
7    # Inicialza o array com a metade do tamanho da imagem de entrada
8    result = np.zeros((height // 2, width // 2, numFilters))
9
10   for region, i, j in self.IterateRegions(input):
11       # Realiza a maximiza  o dos valores dentro da regi o 2x2
12       result[i, j] = np.amax(region, axis=(0, 1))
13
14   return result
```

FONTE: Os Próprios Autores.

Ao fim deste processo, o resultado é então enviado para a última camada, a de ativação. A ativação é feita por meio da função *Softmax*, escolhida por ser uma das funções mais utilizadas para classificação de imagens, além de não ser afetada por resultados com valor negativo como a função ReLU, problema que poderia interferir na diferenciação entre veículos semelhantes. A camada de ativação tem como atributos os pesos e os vieses, sendo as variáveis *weights* e *biases* respectivamente, estes são inicializados de forma randômica da mesma maneira que os filtros da camada convolucional e servem para ajustar o resultado da classificação conforme o treinamento for avançando.

Nesta camada a matriz resultante dos processamentos anteriores passa por um processo de achatamento, que converte a matriz em um vetor de uma única dimensão. Feito isso, este vetor é multiplicado pela matriz de pesos e então os *biases* são somados ao resultado, também fazendo uso da biblioteca *Numpy*. Ao final empregou-se a função *Exp* para se obter o valor exponencial de cada uma das posições do vetor e então dividi-los pela soma dos valores neste vetor, conforme a Equação 2.2. Esse processo resulta em um vetor de n posições, onde cada uma representa a probabilidade daquela imagem pertencer a uma das n classes conhecidas (Código 6).

CÓDIGO 6 – Método Forward da camada de ativação.

```
1
2  def Forward(self, input):
3    self.last_input_shape = input.shape
4
5    # Transforma a entrada em um array 1d
6    input = input.flatten()
7    self.last_input = input
8
9    input_len, nodes = self.weights.shape
10
11   # Multiplica a entrada pelos pesos e ent o soma os resultados
12   totals = np.dot(input, self.weights) + self.biases
13   self.last_totals = totals
14
15   # Calcula o resultado
16   exp = np.exp(totals)
17   return exp / np.sum(exp, axis=0)
```

FONTE: Os Próprios Autores.

Agora com o vetor de probabilidades, o algoritmo calcula a perda de entropia cruzada, ou *Cross-Entropy loss*, que representa o erro de cada probabilidade calculada e pode ser encontrado através da Equação 4.1.

$$L = -ln(p_c) \tag{4.1}$$

Sendo c o valor numérico da classe correta e p_c a probabilidade da classe c ser a correta.

Com o erro calculado, este valor é utilizado para dar início ao processo de retropropagação, partindo da última camada em direção a primeira. A camada de ativação utiliza a Equação 4.2, onde out_s é o vetor com as probabilidades resultantes

desta camada, c é a classe correta e p_i é a probabilidade encontrada para classe i.

$$\partial L/\partial out_s(i) = \begin{cases} 0 \text{ se } i \neq c \\ -\frac{1}{p_i} \text{ se } i = c \end{cases} \tag{4.2}$$

Partindo dessa equação, o algoritmo calcula os gradientes de erro para fazer o ajustes dos pesos e vieses da camada de ativação. Além dos gradientes de erro, é utilizado um valor para a taxa de aprendizado do algoritmo, que controla o quanto estes pesos e vieses serão modificados a cada iteração. Esse processo pode ser visto no código 7.

CÓDIGO 7 – Método Backpropagation da camada de ativação.

```
1
2 def BackProp(self, out, learnRate):
3    for i, gradient in enumerate(out):
4      if gradient == 0:
5        continue
6
7      expTotal = np.exp(self.last_totals)
8
9      sum = np.sum(expTotal)
10
11     gradientTotals = -expTotal[i] * expTotal / (sum ** 2)
12     gradientTotals[i] = expTotal[i] * (sum - expTotal[i]) / (sum
   ** 2)
13
14     lastInput = self.last_input
15
16     gradientOfLoss = gradient * gradientTotals
17
18     inputs = self.weights @ gradientOfLoss
19
20     self.weights -= learnRate * lastInput[np.newaxis].T @
   gradientOfLoss[np.newaxis]
21     self.biases -= learnRate * gradientOfLoss
22
23     return inputs.reshape(self.last_input_shape)
```

FONTE: Os Próprios Autores.

A próxima etapa é realizar a retropropagação na camada de maximização, porém esta camada não pode ser treinada, já que apenas reduz o tamanho da imagem original pela metade escolhendo os maiores valores em regiões 2x2 sem aplicar nenhum outro peso ao cálculo. Dessa forma, o processo reverso é feito na retropropagação desta camada, dobrando o tamanho da imagem e fazendo com que esta volte ao tamanho original e mantenha os valores que foram escolhidos na passagem *Forward* (Código 8).

CÓDIGO 8 – Método Backpropagation da camada de maximização.

```
1
2  def BackProp(self, out):
3    input = np.zeros(self.last_input.shape)
4
5    for region, i, j in self.IterateRegions(self.last_input):
6      height, width, filters = region.shape
7      amax = np.amax(region, axis=(0, 1))
8
9      for i2 in range(height):
10       for j2 in range(width):
11         for f2 in range(filters):
12           if region[i2, j2, f2] == amax[f2]:
13             input[i * 2 + i2, j * 2 + j2, f2] = out[i, j, f2]
```

FONTE: Os Próprios Autores.

O último passo do treinamento é feito com os ajustes dos filtros da camada convolucional. Estes ajustes são feitos baseando-se nos gradientes calculados no início da etapa de retropropagação para gerar novos filtros para esta camada. Da mesma forma que o método *Forward*, a imagem é dividida em conjuntos de *pixels* 3x3, que são então multiplicados pelos gradientes de erros na posição equivalente a do filtro. Ao final, os filtros originais são subtraídos dos gradientes calculados (Código 9). Dessa forma, na próxima iteração do treinamento, os resultados gerados por esta camada são ligeiramente diferentes dos resultados gerados anteriormente.

Repete-se então este processo para cada uma das imagens do *Dataset*, ajustando os filtros e pesos de forma a reduzir o erro e aumentar a precisão das classificações.

CÓDIGO 9 – Método Backpropagation da camada de convolução.

```
1
2 def Backpropagation(self, output, learnRate):
3    filters = np.zeros(self.filters.shape)
4
5    for region, i, j, in self.IterateRegions(self.last_input):
6      for f in range(self.num_filters):
7        filters[f] += output[i, j, f] * region
8
9    self.filters -= learnRate * filters
```

FONTE: Os Próprios Autores.

Além do treinamento, a CNN conta com métodos para testar a precisão do algoritmo e para classificar imagens individuais, como visto nos Códigos 10 e 11 respectivamente.

CÓDIGO 10 – Método de teste da CNN.

```
1  def Test():
2      print('\n--- CNN Testando ---')
3
4      trainFiles = ImageController.PrepareImagesLabels()
5      ImageController.LoadLabelsDict()
6      ImageController.LoadLabelsDirectories()
7
8      random.shuffle(trainFiles)
9
10     num_correct = 0
11     total_loss = 0
12     total_tests = 0
13
14     for i, file in enumerate(trainFiles):
15         total_tests = total_tests + 1
16
17         label = file[0]
18         imageFile = file[1]
19         completePath = file[2]
20
21         image = ImageController.PrepareImageFromPath(completePath)
22
23         _, loss, acc = Forward(image, label)
24         total_loss += loss
25         num_correct += acc
26
27         print('[', i, '/', len(trainFiles), ']: Average Loss: ',
       loss / 100,  ' | Hits:', num_correct, ' | Accuracy: ',
28                 (num_correct / total_tests) * 100, '%')
29
30     num_tests = len(trainFiles)
31     print('Test Loss:', total_loss / num_tests)
32     print('Test Accuracy:', num_correct / num_tests)
33
34     return num_correct / num_tests
```

FONTE: Os Próprios Autores.

O método de teste executa os mesmo passos da passagem *Forward* do treinamento, porém sem retropropagação, ao final é calculada a precisão baseado no número de acertos e na quantidade de imagens classificadas. Já o método de classificação de imagens individuais consiste no mesmo processo, porém com uma única imagem, retornando ao final a *label* da classe encontrada.

Para que a Rede Neural pudesse se comunicar com as outras APIs do sistema, foi feito uso da biblioteca *Flask*, que possibilita o desenvolvimento de APIs REST (*Representational State Transfer*), tornando a comunicação com o resto do sistema

CÓDIGO 11 – Método de classificação de imagens.

```
def Classify(data):
    image = ImageController.PrepareImageFromData(data)
    out = model.convolutionLayer.Forward((image / 255) - 0.5)
    out = model.poolLayer.Forward(out)
    out = model.activationLayer.Forward(out)

    label = np.argmax(out)
    for car, key in carsList.items():
        if key == label:
            return car
```

FONTE: Os Próprios Autores.

simples e eficiente. Foram desenvolvidos quatro métodos para que a Rede Neural possa ser utilizada pelo resto do sistema, conforme o Código 12.

CÓDIGO 12 – API da CNN.

```
def SetLabels():
    labels = request.get_json()
    ImageController.SaveLabels(labels)
    return jsonify(labels)


@app.route('/api/train', methods=['POST'])
def Train():
    labels = request.get_json()
    ImageController.SaveLabels(labels)
    AIController.Train()
    return jsonify(labels)


@app.route('/api/test', methods=['GET'])
def Test():
    labels = request.get_json()
    accuracy = AIController.Test(labels)
    return accuracy

@app.route('/api/classify', methods=['POST'])
def Classify():
    return AIController.Classify(request.data)
```

FONTE: Os Próprios Autores.

O método SetLabels recebe na sua requisição uma lista contendo todas as *labels* e os diretórios das imagens que serão utilizadas pela CNN durante o treinamento. O método Train, além de salvar as *labels* e os diretórios, da início ao processo de treinamento. Já o método Test recebe uma lista da mesma maneira que o método

Train, mas incluindo os diretórios das imagens de teste, retornando ao final a precisão atingida pelo algoritmo. Por último, o método Classify recebe uma imagem no *request* e retorna o resultado da classificação da CNN, bem como a precisão da mesma.

4.2.2 Classificação

A API de classificação foi desenvolvida utilizando .NET Core e é responsável por fazer a classificação do veículo e também fornecer os dados de treinamento para a API rodando a CNN, detalhada na Seção 4.2.1.

Esta API, assim como a API de controle do estacionamento, utilizou o padrão de desenvolvimento chamado Injeção de Dependência. Isso possibilita um baixo nível de acoplamento entre as partes do sistema, facilitando a manutenção do código e melhorando sua legibilidade.

O principal método desta API é o Classificate, este recebe a imagem enviada a partir da tela de alocação de veículos, detalhada na Seção 4.2.4, e a envia para a API da CNN. Ao receber o retorno com a *label* resultante da classificação, esta faz a conversão em um veículo cadastrado no sistema (Código 13).

Na API de classificação também é possível gerar as labels que serão utilizadas pela CNN tendo como base os veículos cadastrados no banco de dados e com isso disparar o início do treinamento da mesma.

4.2.3 Controle do estacionamento

A API de controle do estacionamento foi desenvolvida de maneira semelhante a de classificação. Ela é responsável por fazer o controle das vagas e dos veículos no banco de dados. A vaga é escolhida baseando-se no tipo do veículo a ser alocado, procurando sempre pelas vagas mais próximas da porta do estacionamento que forem compatíveis exclusivamente com o tipo do veículo. Caso não haja nenhuma vaga exclusiva para aquele tipo de veículo, o sistema busca pela vaga mais próxima que contenha dentro da sua lista de tipos de veículos comportados o tipo do veículo em questão. Caso não seja encontrado nenhuma vaga exclusiva para aquele tipo de veículo nem uma vaga que comporte mais de um tipo de veículo, o sistema irá retornar uma mensagem de erro informando a situação.

O acesso ao banco de dados é dado utilizando as funcionalidades do *Entity Framework* do .NET. Com isso todo acesso ao bando é feito através da classe *ParkingContext*, nela são configurados os dados de acesso ao banco e as classes que compõem suas tabelas, bem como o controle de versão através das *migrations*, que são os dados utilizados pelo *Entity Framework* para fazer atualizações na estrutura do banco de dados, conforme os Códigos 14 e 15.

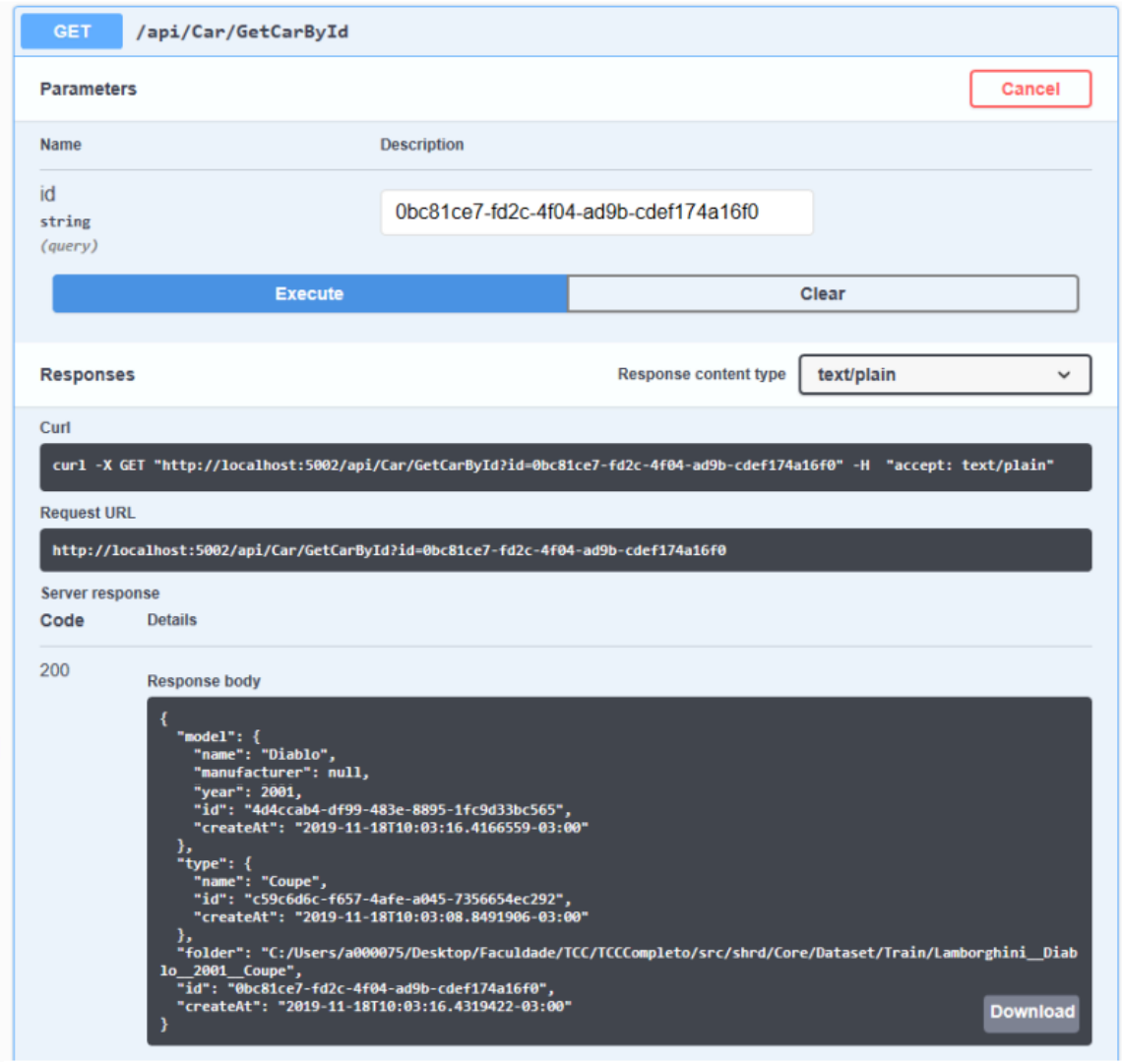
Nesta API se encontram os métodos utilizados pelo sistema para realizar consultas e alterações no banco de dados, podendo assim encontrar o veículo que se deseja, vagas disponíveis para o veículo desejado, alocar um veículo em uma vaga. Um exemplo de consulta ao banco pode ser observada no código 16.

Além disso também foram desenvolvidos métodos que realizam o controle das vagas e veículos que podem, além de serem executados pelo próprio sistema, ser utilizados através do *Swagger*, permitindo que ambos possam ser adicionados ou excluídos do banco de dados sem necessitar o acesso direto a ele, como visto nas Figuras 26 e 27.

FIGURA 26 – Exemplo de requisições possíveis pelo *Swagger*

Fonte: Os próprios autores.

FIGURA 27 – Exemplo de requisição executada com o *Swagger*



Fonte: Os próprios autores.

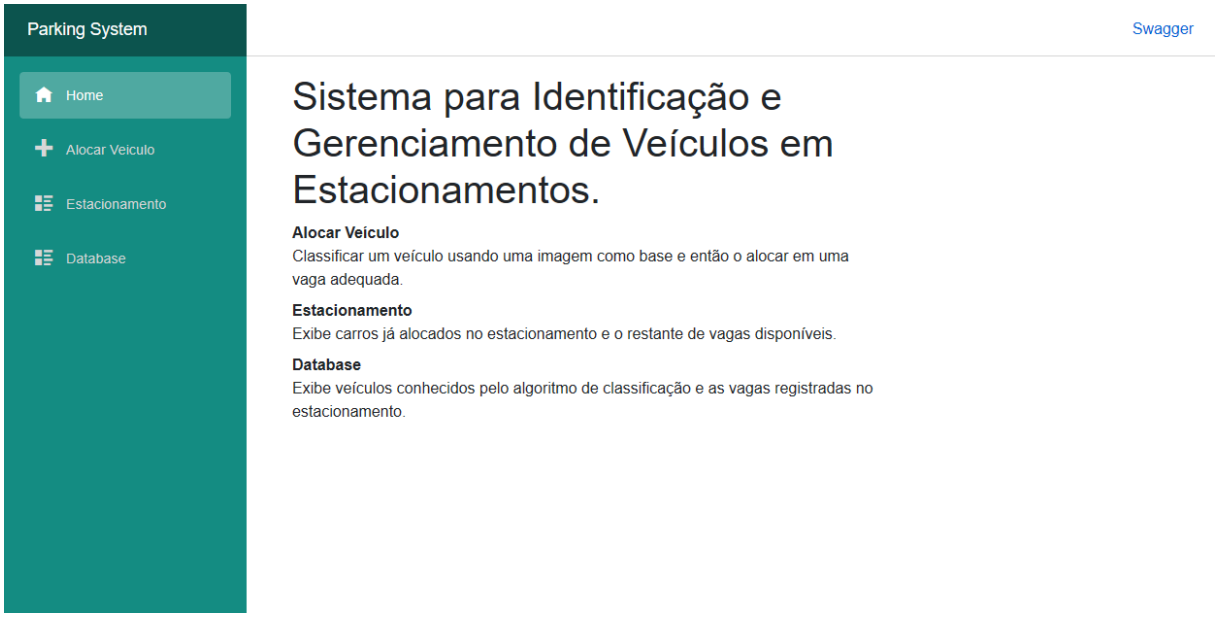
4.2.4 Interface

A interface do usuário foi criada para facilitar a utilização do sistema, bem como a visualização dos resultados obtidos na classificação e alocação do veículo, com base em *Blazor*, uma das funcionalidades mais recentes do .NET para o desenvolvimento de páginas web.

Na página inicial do sistema há uma breve introdução sobre as funcionalidades, como pode ser visto na Figura 28.

Já na página de alocação de veículo pode ser escolhida uma imagem que será enviada para o sistema e classificada utilizando a ClassificationAPI, detalhada na Seção 4.2.2. O resultado desta classificação será exibido ao lado da imagem bem como a certeza do algoritmo de que a imagem enviada contém tal veículo. Será disponibilizada

FIGURA 28 – Página inicial do sistema



Fonte: Os próprios autores.

também a opção de alocar aquele veículo em uma vaga do estacionamento, que é feito utilizando a ParkingAPI e retornará a melhor vaga disponível no momento (Figura 29).

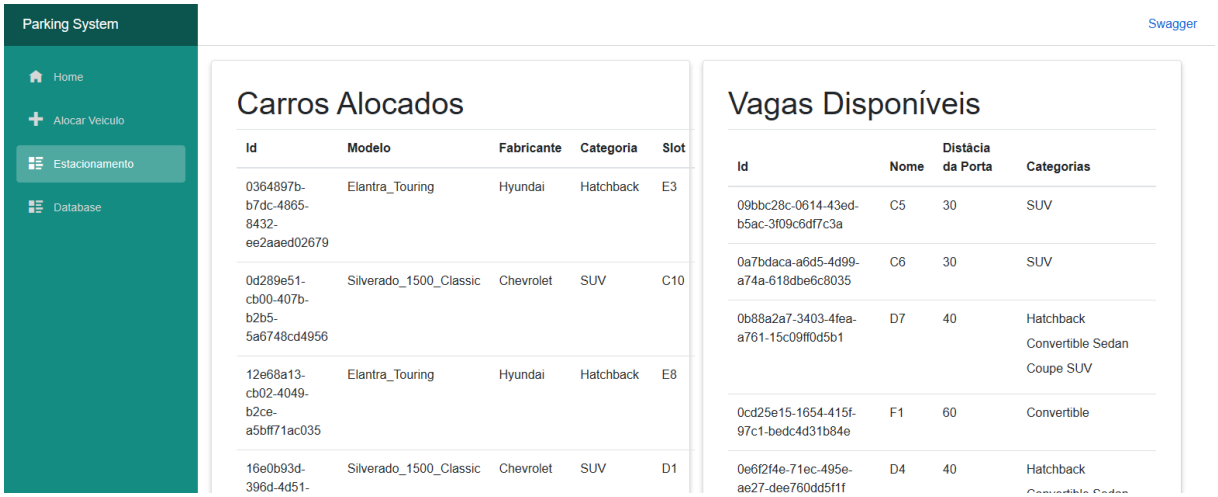
Na opção Estacionamento é possível visualizar quais vagas do estacionamento estão disponíveis e quais veículos estão ocupando vagas do estacionamento, como pode ser visto na Figura 30.

Por último, na tela *Database* é possível visualizar quais são os veículos conhecidos pelo algoritmo e quais são as vagas de estacionamento cadastradas no sistema, como mostra a Figura 31.

FIGURA 29 – Página de classificação e alocação de veículos

Fonte: Os próprios autores.

FIGURA 30 – Página de exibição das vagas do estacionamento



Fonte: Os próprios autores.

CÓDIGO 13 – Método de classificação de veículos.

```
1public async Task<ClassificationCar> ClassificateAsync(string image)
2{
3    var request = "http://127.0.0.1:5000"
4                .AppendPathSegment("api/classify")
5                .WithTimeout(TimeSpan.FromSeconds(5));
6    var response = await request
7                .PostJsonAsync(image).ReceiveString();
8
9    string[] separator = { "__" };
10   var names = response.Split(separator,
11       StringSplitOptions.RemoveEmptyEntries);
12
13   var manufacturerName = names[0].ToUpper();
14   var modelName = names[1].ToUpper();
15   var year = names[2];
16   var type = names[3].ToUpper();
17   var acc = Decimal.Parse(names[4].Replace(".", ","), new
     CultureInfo("pt-BR"));
18
19   var car = db.Cars.
20       Include(x => x.Type).
21       Include(x => x.Model).
22       ThenInclude(x => x.Manufacturer).
23       FirstOrDefault(x =>
24           x.Model.Manufacturer.Name.ToUpper()
25               .Equals(manufacturerName) &&
26           x.Model.Name.ToUpper()
27               .Equals(modelName) &&
28           x.Model.Year.ToString()
29               .Equals(year) &&
30           x.Type.Name
31               .Equals(type));
32
33   if (car == null)
34       throw new Exception("Veiculo nao encontrado na base de dados
     do sistema.");
35
36   return new ClassificationCar
37   {
38       Car = car,
39       Accuracy = decimal.Round(acc * 100, 2,
     MidpointRounding.AwayFromZero)
40   };
41}
```

FONTE: Os Próprios Autores.

CÓDIGO 14 – Contexto do banco de dados.

```
1public class ParkingContext : DbContext
2{
3    const string schema = "Parking";
4
5    public DbSet<Car> Cars { get; set; }
6
7    public DbSet<Model> Models { get; set; }
8
9    public DbSet<Manufacturer> Manufacturers { get; set; }
10
11   public DbSet<ParkedCar> Parked { get; set; }
12
13   public DbSet<Slot> Slots { get; set; }
14
15   public DbSet<SlotType> SlotTypes { get; set; }
16
17   public DbSet<Core.Models.Type> Types { get; set; }
18
19}
```

FONTE: Os Próprios Autores.

CÓDIGO 15 – Exemplo de uma *migration* criada pelo sistema.

```
1    migrationBuilder.CreateTable(
2        name: "Manufacturers",
3        schema: "Parking",
4        columns: table => new
5        {
6            Id = table.Column<string>(nullable: false),
7            CreateAt = table.Column<DateTimeOffset>(nullable: false),
8            Name = table.Column<string>(nullable: false)
9        },
10       constraints: table =>
11       {
12           table.PrimaryKey("PK_Manufacturers", x => x.Id);
13       });
```

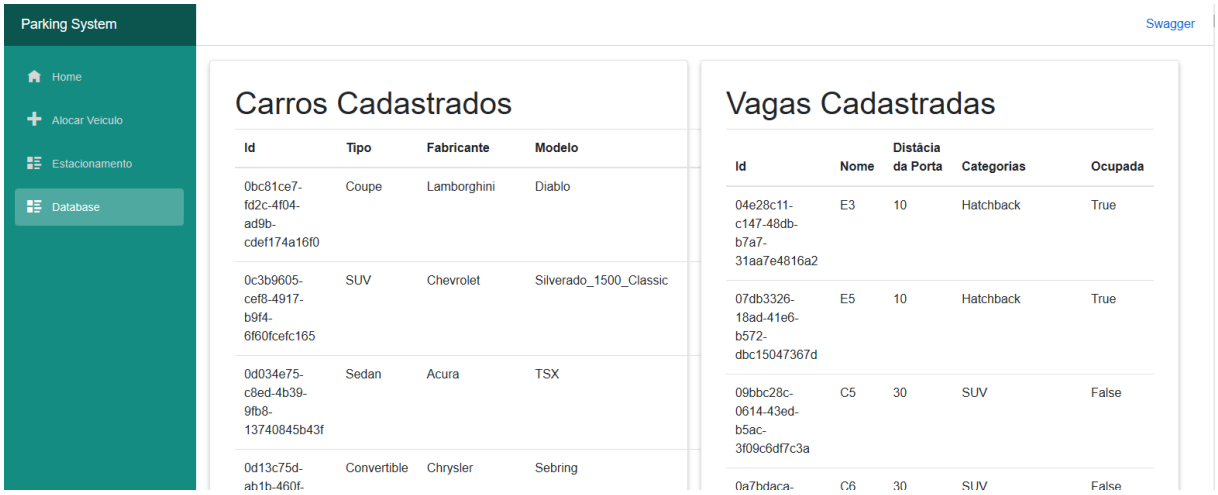
FONTE: Os Próprios Autores.

CÓDIGO 16 – Exemplo de consulta ao banco dentro do sistema.

```
1public Car GetCarById([FromQuery]string id)
2{
3    var car = db.Cars.Include(x => x.Model)
4        .Include(x => x.Type)
5        .Where(x => x.Id == id)
6        .FirstOrDefault();
7
8    return car;
9}
```

FONTE: Os Próprios Autores.

FIGURA 31 – Página de exibição do banco de dados



Fonte: Os próprios autores.

5 ANÁLISE DOS RESULTADOS

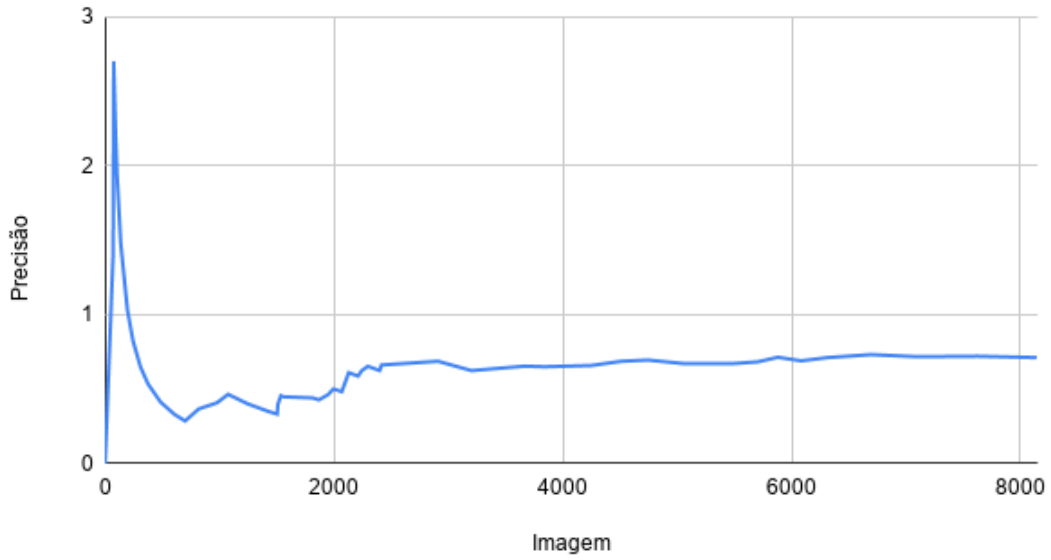
Ao final do desenvolvimento do sistema, o principal ponto a ser analisado é o funcionamento da CNN utilizada e a precisão de suas classificações. Para isso primeiramente o algoritmo foi treinado com umas seleção de 8041 imagens do *Dataset* dividas entre 196 possíveis veículos.

O modelo que melhor apresentou resultado na classificação mediante a este treinamento teve sua estrutura composta de uma camada convolucional contendo 10 filtros, seguida de uma camada de maximização e por fim a camada de ativação, conforme a Figura 25 apresentada na Seção 4.2.1.

Para percorrer todas as imagens de treinamento, o algoritmo levou cerca de 6 horas rodando em uma máquina com um Intel Core i5-7500 e com 16Gb de memória RAM, um dos fatores para utilizar uma rede com apenas uma camada convolucional já que o tempo e custo computacional para execução de uma rede composta de várias camadas aumentaria exponencialmente. Inicialmente a taxa de aprendizado utilizada foi de 0,05% com 8 filtros na camada convolucional, porém a taxa de acerto durante o treinamento não evoluiu. Isso ocorreu pelo fato da taxa de aprendizado estar muito alta, fazendo com que a variação entre as iterações do algoritmo fosse grande demais, impedindo que as camadas pudessem se adaptar a veículos semelhantes. A fim de evitar este problema, a taxa de aprendizado foi reduzida para 0.005%, o que tornou o aprendizado mais lento, porém permitindo que pequenas nuances entre as imagens fossem capturadas durante o treinamento, além de aumentar o número de filtros para 10, aumentando a quantidade de características de cada imagem que seria avaliada. Essa alteração resultou em melhoria significativa no desempenho da CNN.

Devido ao grande número de labels possíveis, a precisão final estava abaixo de 3%, como pode ser visto na Figura 32. Para que a precisão pudesse aumentar, o processo de treinamento foi repetido utilizando o mesmo *Dataset* por mais algumas vezes utilizando os mesmos parâmetros do primeiro treinamento, reforçando o que foi aprendido durante o treinamento anterior.

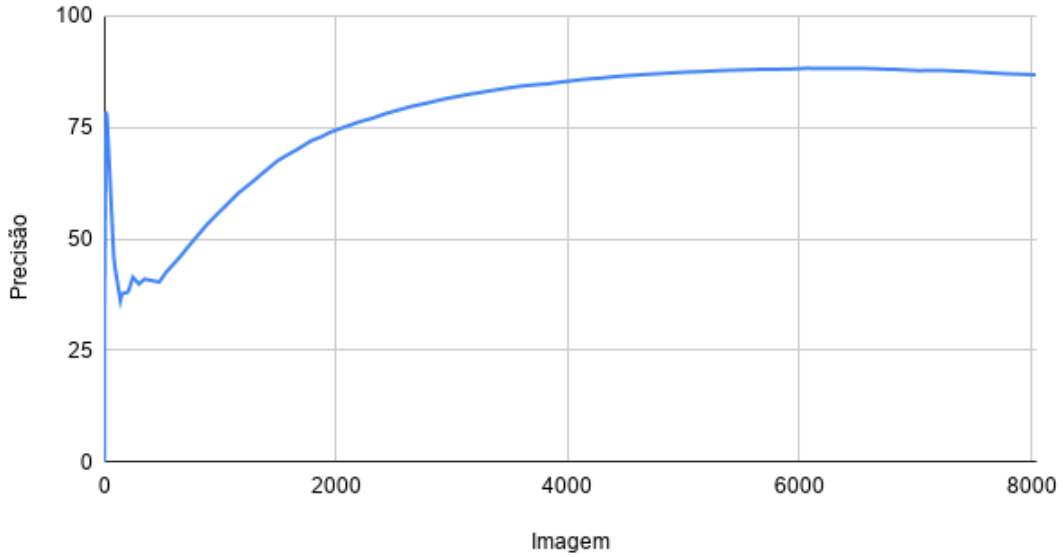
FIGURA 32 – Primeiro treinamento



Fonte: Os próprios autores.

Na execução do segundo treinamento houve uma melhora significativa na precisão do algoritmo devido aos parâmetros das camadas não terem sido inicializados aleatoriamente, mas sim com os valores da última iteração do primeiro treinamento (Figura 33).

FIGURA 33 – Segundo treinamento

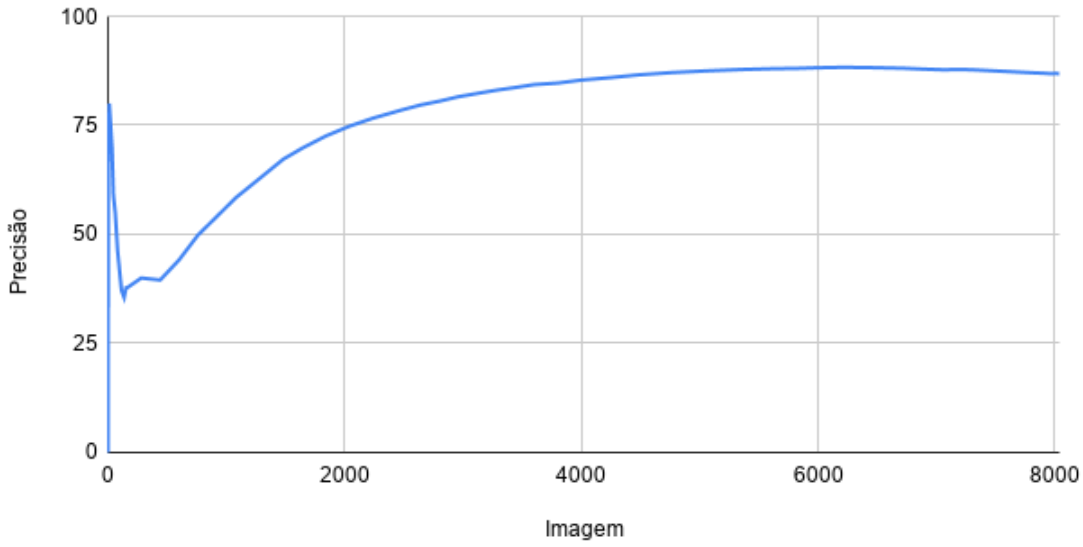


Fonte: Os próprios autores.

Como a repetição do treinamento se mostrou efetiva, o processo foi repetido

mais vezes. Na terceira iteração, o retreinamento mostrou resultados ligeiramente melhores, atingindo cerca de 86% de precisão.

FIGURA 34 – Terceiro treinamento

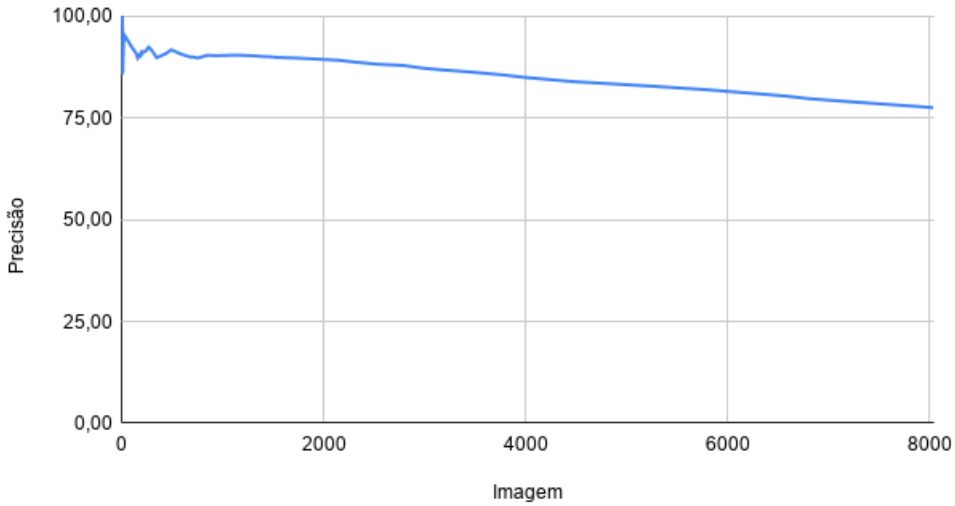


Fonte: Os próprios autores.

Na quarta iteração do treinamento (Figura 35), notou-se uma perda de precisão, baixando a taxa de acerto para 76%. Este efeito pode ter sido causado pelo excesso de treinamento, também chamado de *Overfitting*, que significa que a CNN se especializou em classificar sempre os mesmos dados, de forma que qualquer alteração nos parâmetros das camadas pode se tornar negativa.

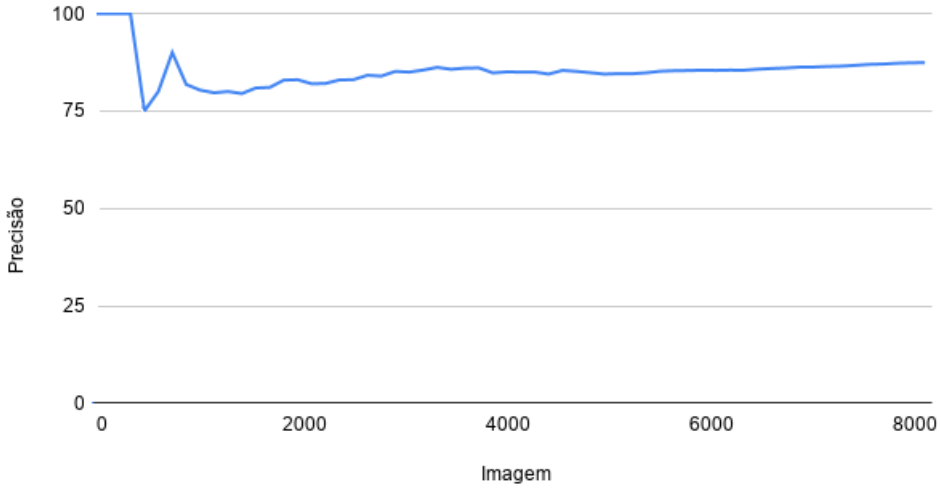
A fim de validar se houve o *overfitting*, repetiu-se o treinamento mais uma vez, dessa vez mostrando uma taxa de acerto semelhante a dos treinamentos anteriores (Figura 36), dando assim continuidade ao retreinamento.

FIGURA 35 – Quarto treinamento



Fonte: Os próprios autores.

FIGURA 36 – Quinto treinamento



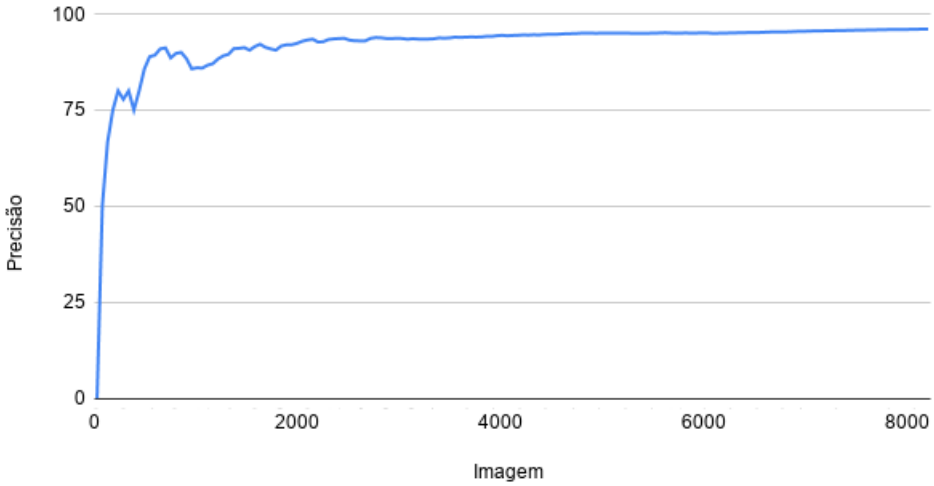
Fonte: Os próprios autores.

No sexto e último treinamento foi observada uma grande alteração nos resultados: a taxa de acerto média durante o treinamento se elevou para acima 97%, como observado na Figura 37. Este comportamento mostra que neste ponto do aprendizado foi atingido o *overfitting*, já que praticamente todos as classificações se mostraram corretas.

Mesmo que a Rede Neural esteja com uma taxa de acerto alta, o *overfit* pode ser um efeito negativo. Quando uma Rede Neural é treinada em excesso, ela se torna extremamente especializada em classificar apenas os dados utilizados durante

o treinamento, perdendo a capacidade de generalizar o que foi aprendido para dados diferentes dos dados conhecidos por ela (BROWNLEE, 2018).

FIGURA 37 – Sexto treinamento



Fonte: Os próprios autores.

Para atenuar este problema, o modelo final utilizado pelo sistema é o gerado ao final do terceiro treinamento, já que foi o menos afetado pelo *overfitting*. Como durante o treinamento erros são cometidos e corrigidos, o número de acertos ao final tende a ser menor do que a execução do algoritmo apenas para classificação. Para calcular sua taxa de acerto real, usou-se o método de teste detalhado na seção anterior, mostrando que a CNN atingiu um máximo de 94% de precisão.

Uma possível melhoria para evitar que a CNN sofra com o *overfitting* é o aumento no número de imagens utilizadas durante o treinamento, fazendo com o haja uma maior variedade de características que podem ser exploradas pela Rede Neural. É possível também adicionar novas camadas convolucionais à Rede Neural de forma que os dados que chegariam na camada de ativação estariam mais refinados e as características de cada veículo seriam realçadas, ou até mesmo dividir a classificação entre mais de uma Rede Neural, onde cada rede seria responsável por encontrar atributos individuais do veículo, como a fabricante e o tipo do veículo. No entando, cada uma das alternativas é acompanhada do aumento do custo computacional e complexidade da Rede Neural. Também há a possibilidade de utilizar uma combinação de Filtros para realce de bordas, detalhados na Seção 2.2.2, e Transformações de cores diversificadas para realçar as características dos veículos, como a utilização de uma conversão para o Negativo da imagem, detalhada na Seção 2.2.1, para realçar os detalhes em imagens mais escuras.

6 CONCLUSÃO

Visando facilitar o gerenciamento de vagas em estacionamentos, o projeto foi desenvolvido com o propósito de ser capaz de reconhecer o modelo de veículos através de imagens e então, baseando-se no tipo destes veículos, escolher uma vaga que se adéque a ele. Para que isso pudesse ser alcançado, foi utilizada uma Rede Neural Convolucional, em conjunto com técnicas de Processamento de Imagens Digitais, para fazer o reconhecimento dos veículos. Para reduzir o custo computacional e facilitar o reconhecimento dos veículos nas imagens, estas precisaram passar por processos de redimensionamento, transformações de cores e filtragem. Também foi necessário um longo tempo de treinamento da Rede Neural Convolucional, utilizando uma grande quantidade de imagens.

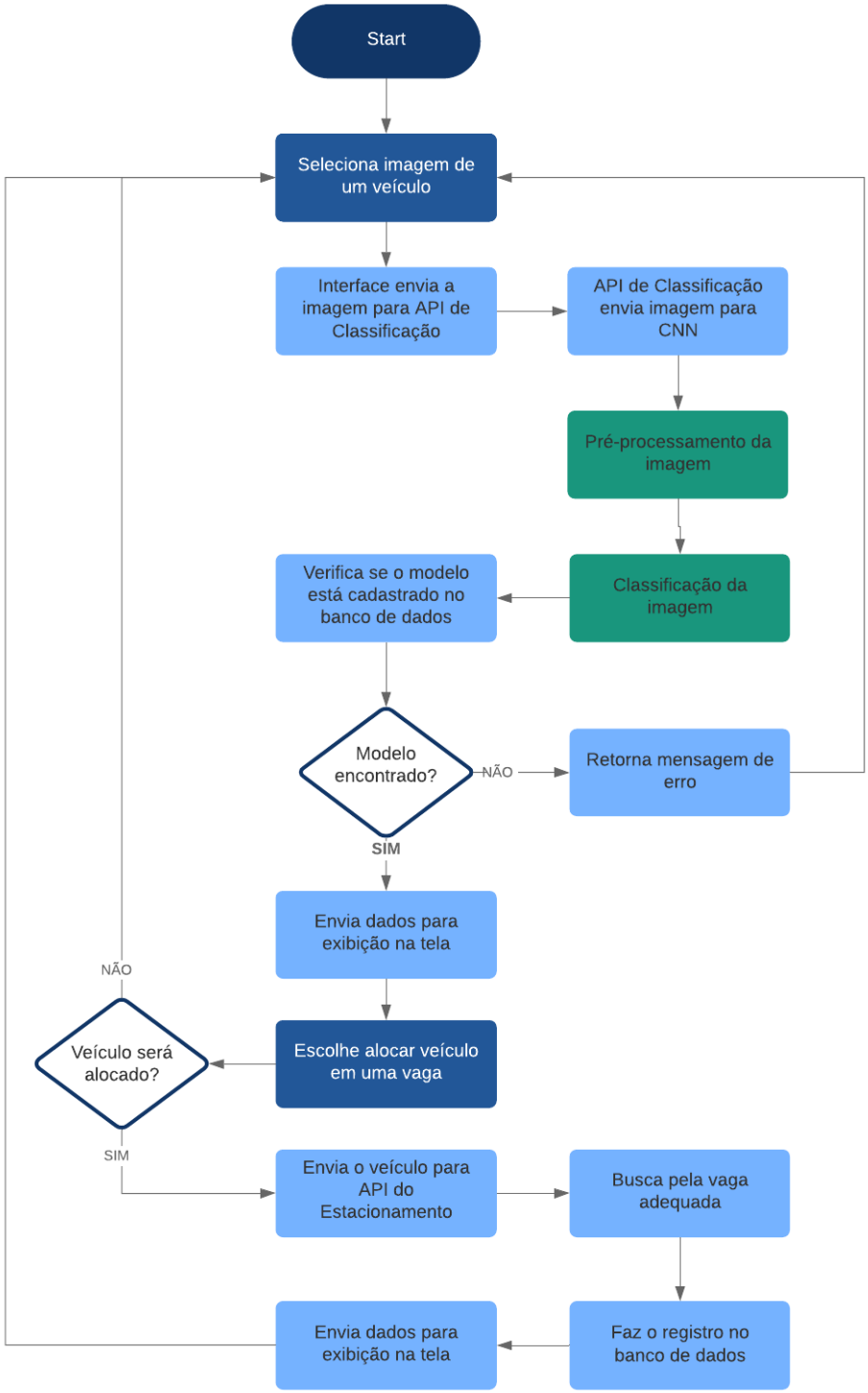
Ao final do desenvolvimento dos algoritmos, a taxa de acerto da Rede Neural Convolucional chegou a mais de 97%. No entanto, sendo este um resultado negativo. Após os testes, foi verificado que a CNN atingiu o chamado *Overfitting*, resultante do excesso de treinamento da Rede Neural. Problema este que veio a se tornar uma das maiores dificuldades encontradas no projeto, visto que não foi possível concluir um modelo de CNN que pudesse contornar o problema de mandeira eficiente.

Mesmo concluído dentro de seu propósito acadêmico, o sistema ainda pode receber melhorias, sendo a principal delas a reestruturação da CNN. Com a adição de novas camadas e uma melhor distribuição de seus neurônios, os resultados da CNN podem ser melhorados. Fazendo com que a taxa de acerto seja elevada sem sofrer com o *Overfitting*. Também há a possibilidade do desenvolvimento de uma Interface mais voltada ao usuário final, já que a interface atual foi desenvolvida apenas com o propósito de facilitar a visualização dos resultados. Além de que, também pode ser desenvolvido um método de entrada das imagens através de vídeo, eliminando a necessidade de que o usuário faça a entrada das imagens manualmente.

APÊNDICE A – TABELA DE VERSÕES

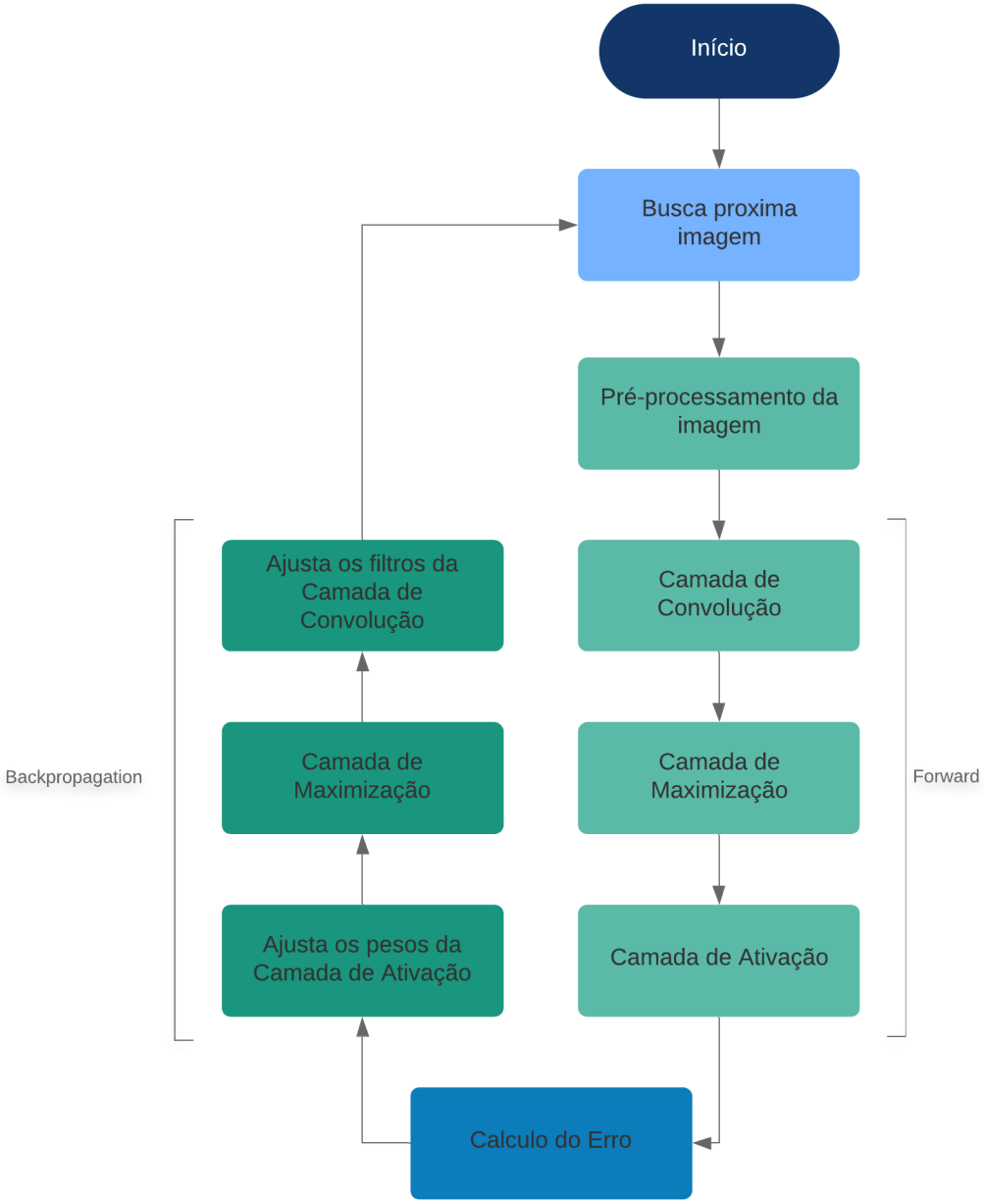
Ferramenta	Tipo	Versão	Descrição
Windows 10 Pro	Sistema Operacional	1903	Sistema operacional utilizado no desenvolvimento do projeto
Microsoft Visual Studio 2019 - Comunity	IDE	16.3.4	IDE para desenvolvimento em .NET
Jetbrains Pycharm	IDE	2018.3	IDE para desenvolvimento em Python
Google Chrome	Navegador	78.0.3904.97	Navegador utilizado durante o desenvolvimento da interface
Microsoft Edge	Navegador	44.17763.831	Navegador utilizado durante o desenvolvimento da interface
Swagger UI	Ferramenta	4.0.1	Ferramenta utilizada para testar requisições da API
C# .NET Core	Framework/Linguagem	2.0	Linguagem utilizada para desenvolvimento das APIs do estacionamento
C # .NET Core	Framework/Linguagem	3.0	Framework utilizado para desenvolvimento da interface
Pythoon	Linguagem	3.7.0	Linguagem utilizada para o desenvolvimento da CNN
AspNetCore	Framework- .NET	2.2.0	Framework utilizado para desenvolvimento da interface
AspNetCore.Blazor	Framework- .NET	3.1.0	Framework utilizado para desenvolvimento da interface
AspNetCore.Http	Framework- .NET	2.2.2	Framework utilizado para realizar requisições http
Entity Framework Core	Framework - .NET	2.2.0	Framework utilizado para acesso ao banco de dados
Flurl	Biblioteca - .NET	2.8.2	Utilizada para fazer requisições Rest
Flurl.Http	Biblioteca - .NET	2.8.2	Utilizada para fazer formatar requisições Rest
Flask	Biblioteca - Python	1.1.1	Utilizada no desenvolvimento da API em python
Numpy	Biblioteca - Python	1.17.4	Auxilia nos cálculos com matrizes
OpenCV	Framework - Python	4.1.1.26	Utilizado para fazer o processamento das imagens

APÊNDICE B – FLUXOGRAMA DE CLASSIFICAÇÃO E ALOCAÇÃO DE VEÍCULOS.



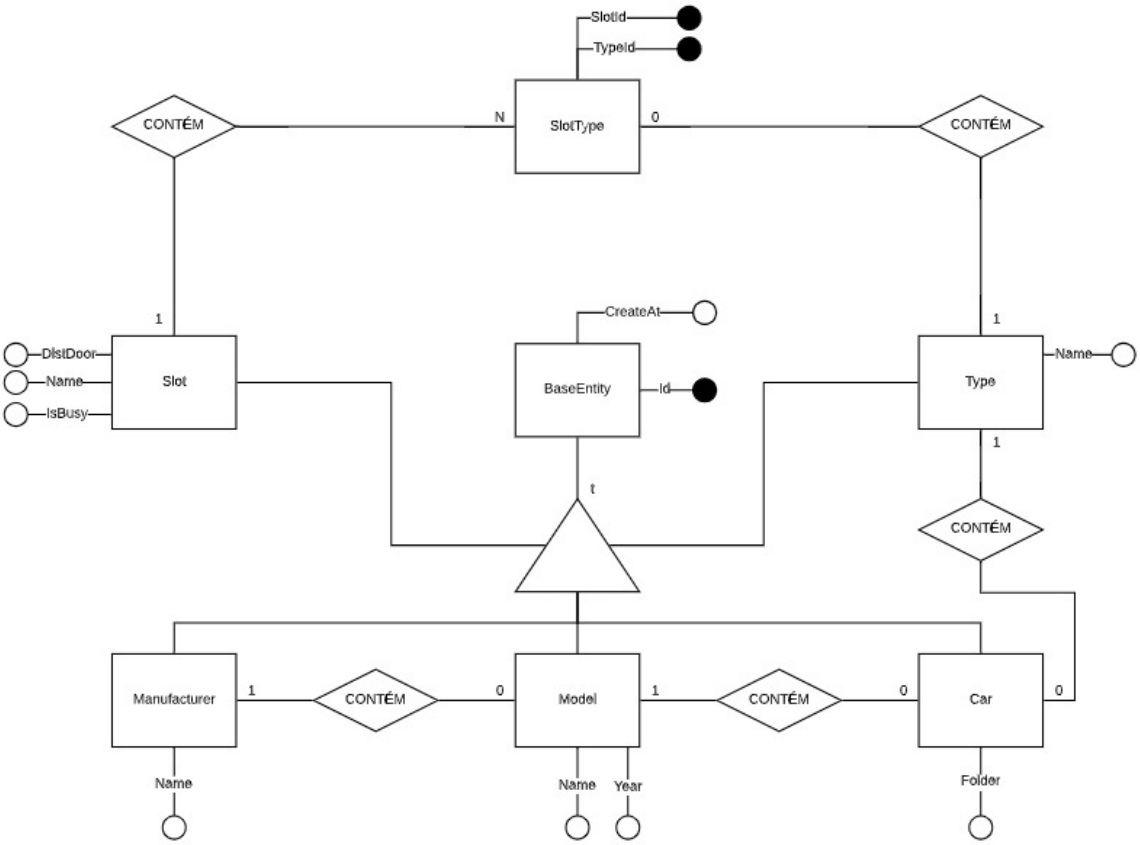
Fonte: Os próprios autores.

APÊNDICE C – FLUXO DO TREINAMENTO DA CNN.



Fonte: Os próprios autores.

APÊNDICE D – DIAGRAMA DE ENTIDADE E RELACIONAMENTO



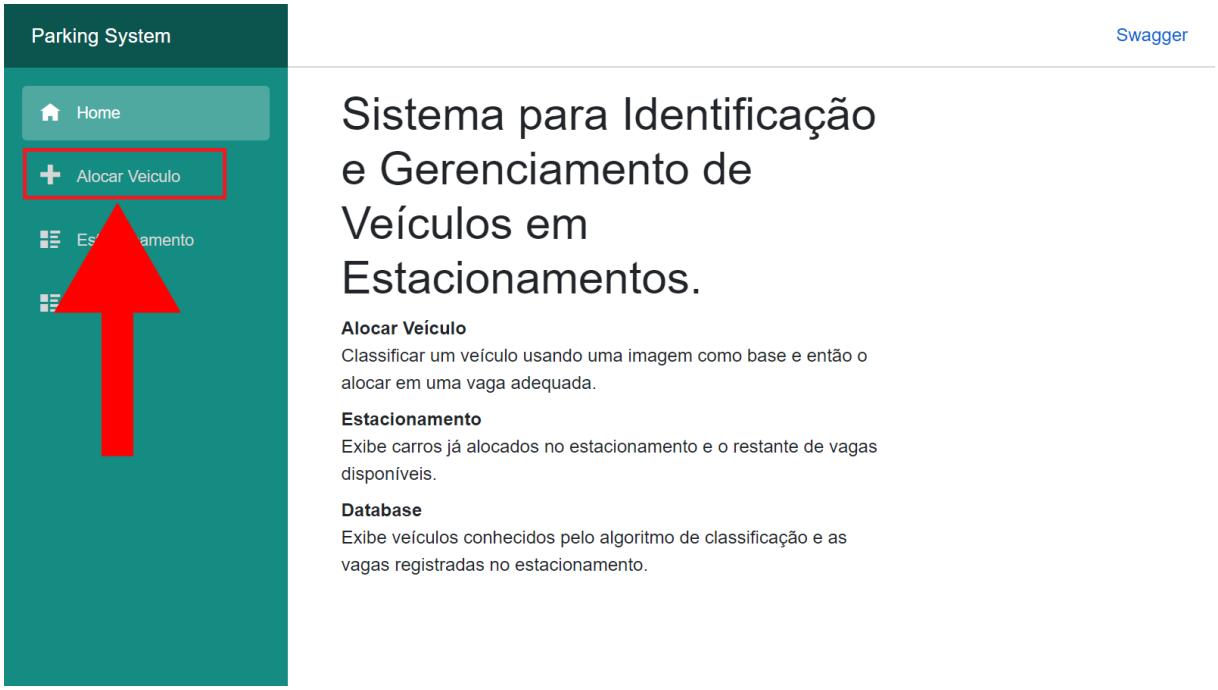
APÊNDICE E – CRONOGRAMA

Fases do Estudo	Mar/19	Abr/19	Mai/19	Jun/19	Jul/19	Ago/19	Set/19	Out/19	Nov/19	Dez/19
Relação de itens pertinentes à proposta em questão										
Desenvolvimento da pesquisa										
Definição dos fluxogramas e metodologias										
Escrita da monografia										
Apresentação e entrega TCC I										
Desenvolvimento do banco de dados e interface										
Desenvolvimento do algoritmo de processamento de imagens e CNN										
Desenvolvimento dos algoritmos de alocação de vagas										
Testes e ajustes finais										
Atualização da monografia e documentação										
Preparos para apresentação										
Apresentação										

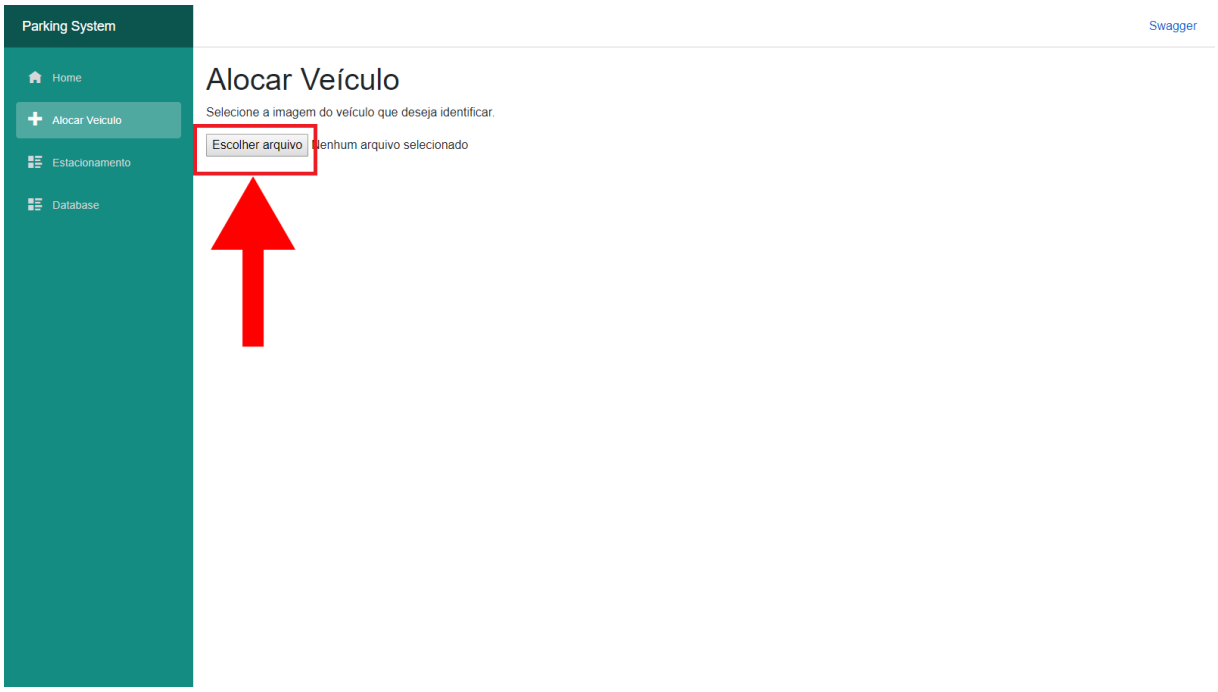
APÊNDICE F – MANUAL DO USUÁRIO

6.1 ALOCAÇÃO DE NOVO VEÍCULO

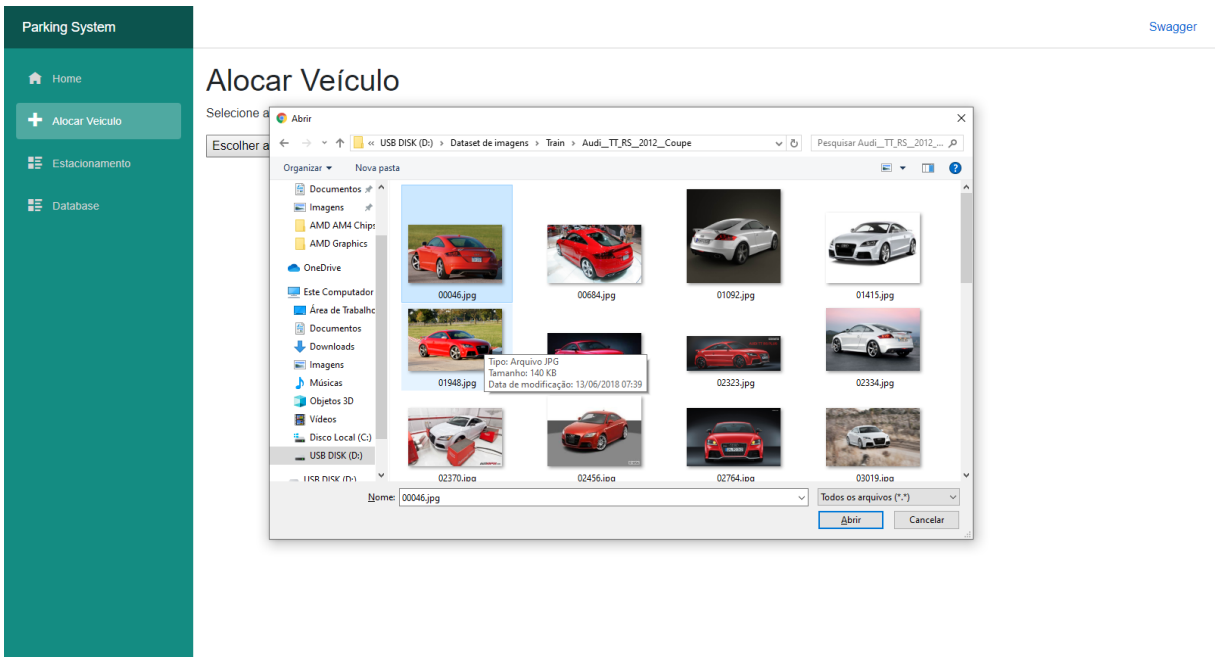
1) O usuário deverá Selecionar "Alocar Veiculo"na Aba de navegação que se encontra do canto esquerdo da página.



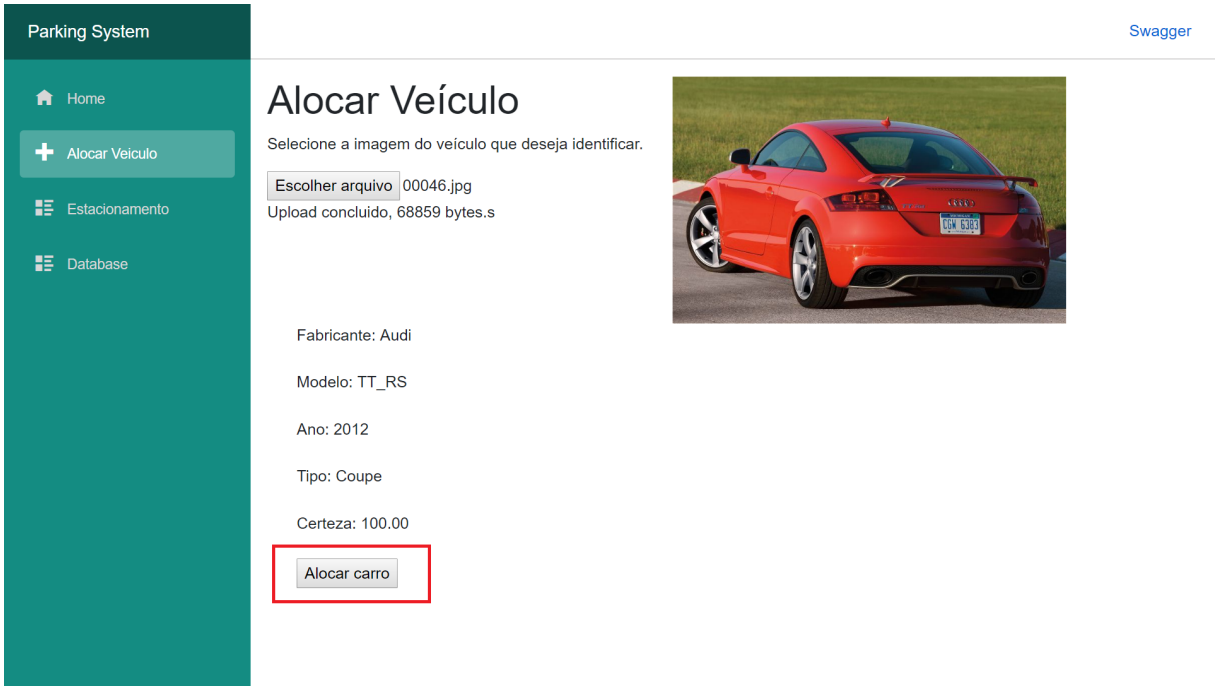
2) Clicar no botão "Escolher arquivo".



3) Selecionar em seu computador uma imagem de um veículo, que deseje fazer alocação.

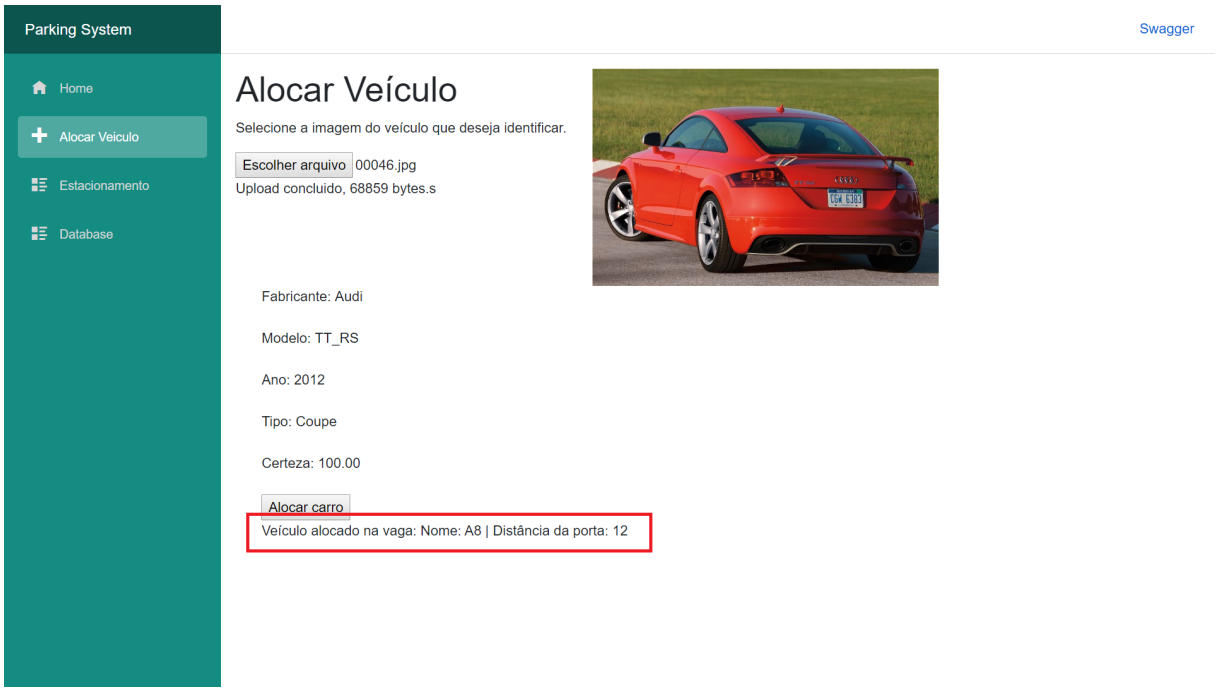


4) Após ser informado na tela os dados do veiculo contido na imagem escolhido, eh exibido um botão "Alocar"que deverá ser pressionado o usuário deseje confirmar a alocação.



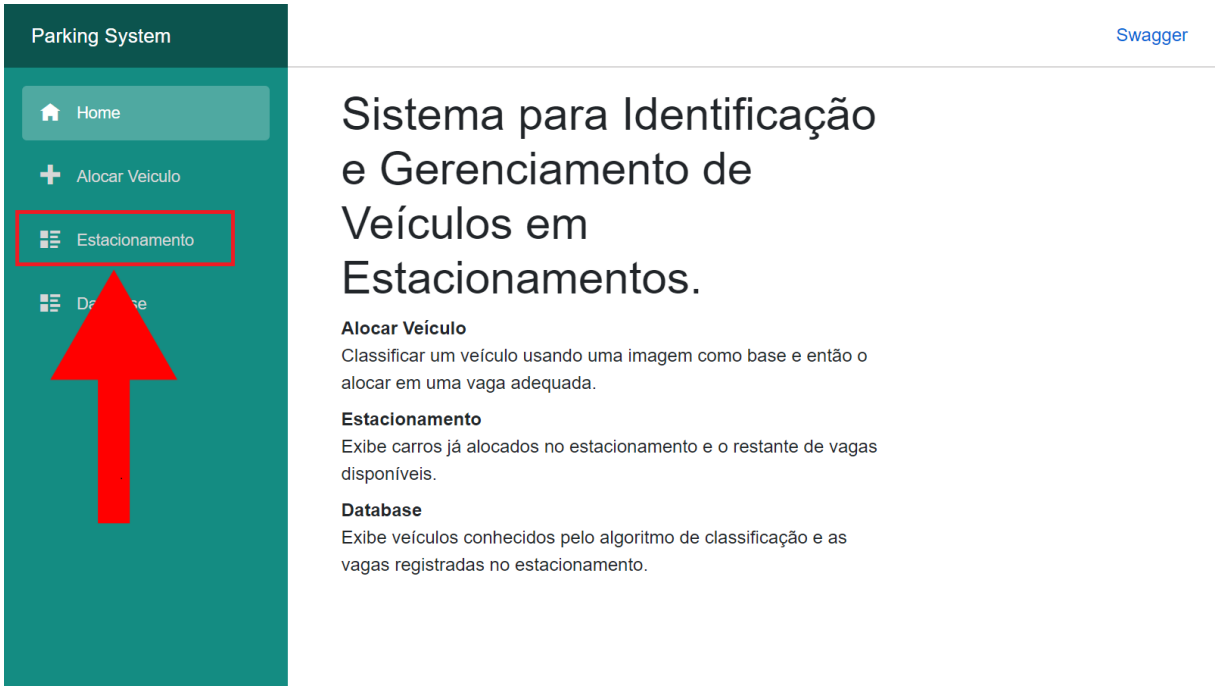
5) Será exibido se foi possível realizar a alocação do veículo em uma determi-

nada vaga, e em qual vaga foi escolhida para o mesmo.

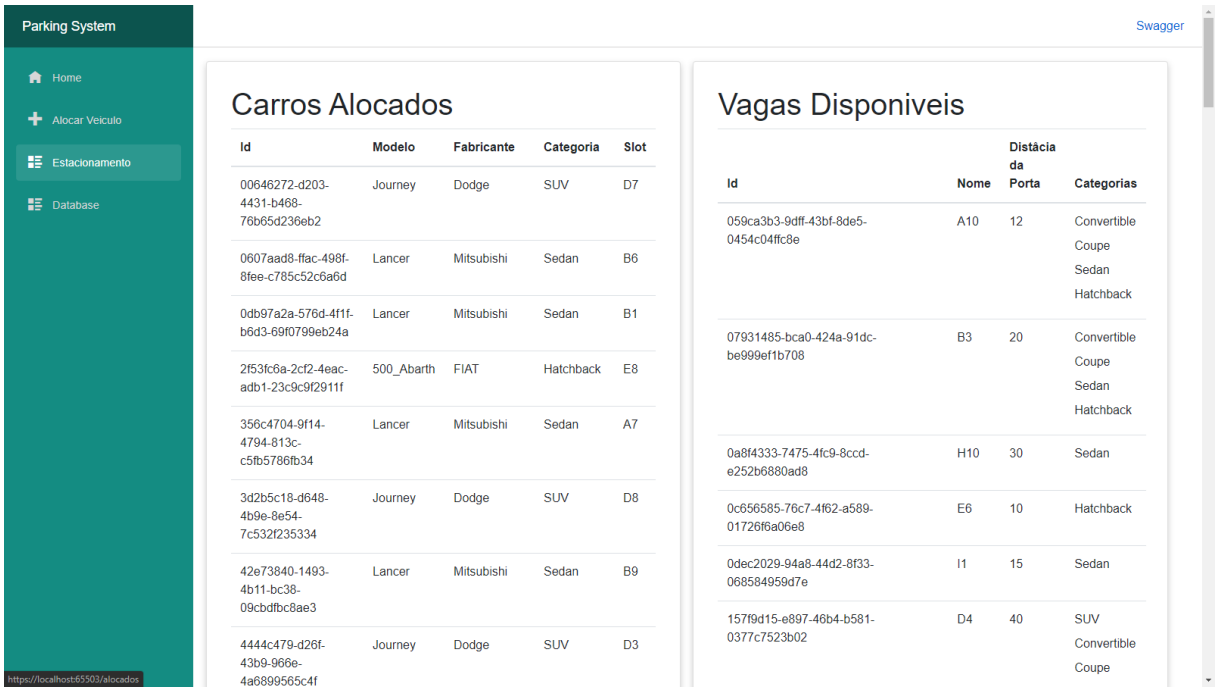


6.2 CONSULTAR CARROS ALOCADOS E VAGAS DISPONÍVEIS

1) O usuário deverá Selecionar "Database"na Aba de navegação que se encontra do canto esquerdo da página.



2) Será exibido os carros alocados e vagas disponíveis no banco, com suas respectivas informações.

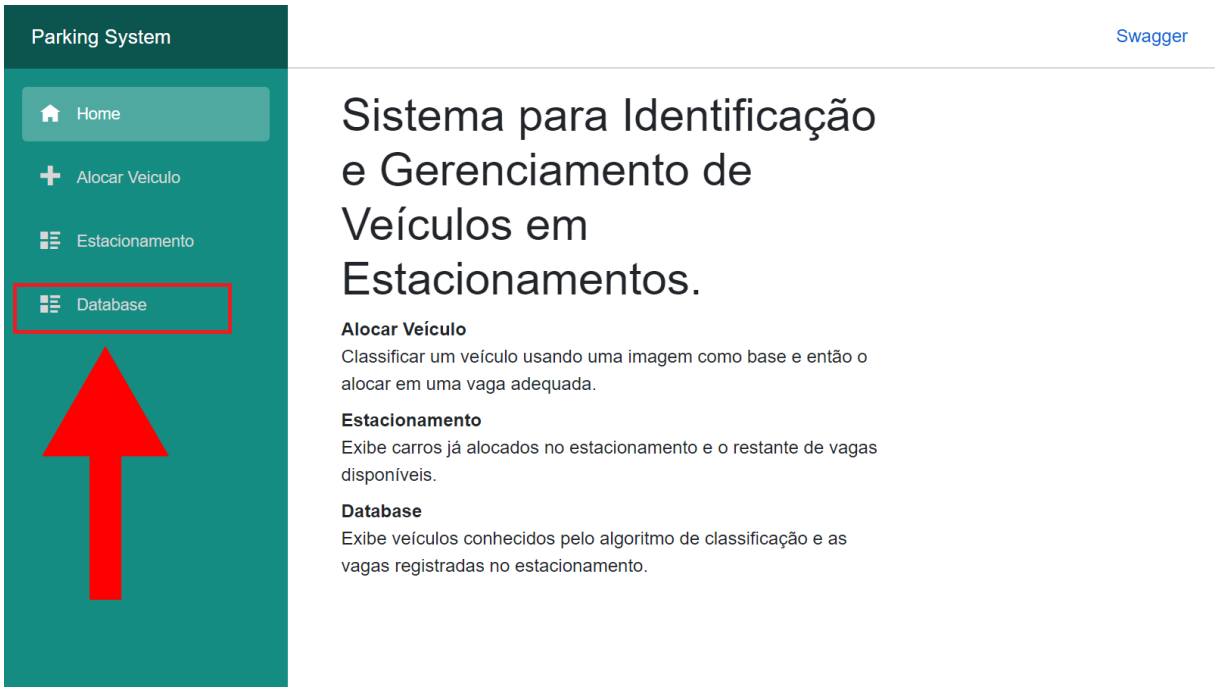


Id	Modelo	Fabricante	Categoria	Slot
00646272-d203-4431-b468-76b65d236eb2	Journey	Dodge	SUV	D7
0607aad8-ffac-498f-8fee-c785c52c6a6d	Lancer	Mitsubishi	Sedan	B6
0db97a2a-576d-4f1f-b6d3-69f0799eb24a	Lancer	Mitsubishi	Sedan	B1
2f53fc6a-2cf2-4eac-adb1-23c9c9f2911f	500_Abarth	FIAT	Hatchback	E8
356c4704-9f14-4794-813c-c5fb5786fb34	Lancer	Mitsubishi	Sedan	A7
3d2b5c18-d648-4b9e-8e54-7c532f235334	Journey	Dodge	SUV	D8
42e73840-1493-4b11-bc38-09cbdfbc8ae3	Lancer	Mitsubishi	Sedan	B9
4444c479-d26f-43b9-966e-4a6899565c4f	Journey	Dodge	SUV	D3

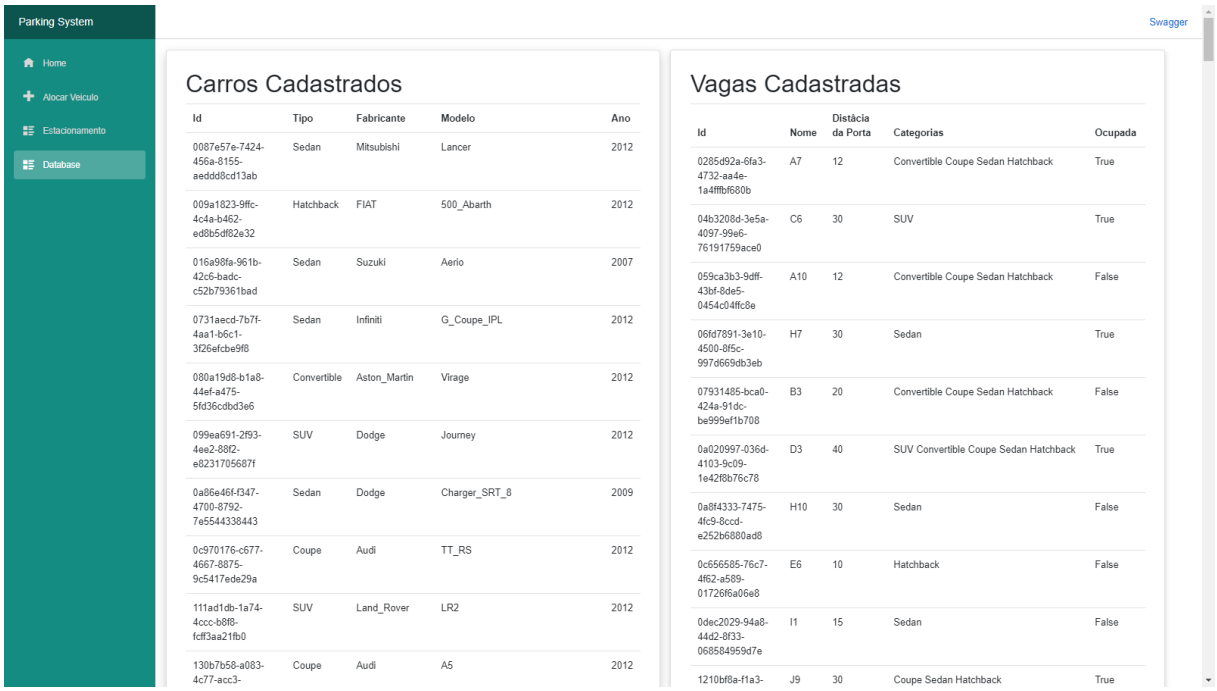
Id	Nome	Distácia da Porta	Categorias
059ca3b3-9dff-43bf-8de5-0454c04ffc8e	A10	12	Convertible Coupe Sedan Hatchback
07931485-bca0-424a-91dc-be999ef1b708	B3	20	Convertible Coupe Sedan Hatchback
0a8f4333-7475-4fc9-8ccd-e252b6880ad8	H10	30	Sedan
0c656585-76c7-4f62-a589-01726f6a06e8	E6	10	Hatchback
0dec2029-94a8-44d2-8f33-068584959d7e	I1	15	Sedan
157f9d15-e897-46b4-b581-0377c7523b02	D4	40	SUV Convertible Coupe

6.3 CONSULTAR CARROS E VAGAS CADASTRADOS NO BANCO

1) O usuário deverá Selecionar "Database"na Aba de navegação que se encontra do canto esquerdo da página.



2) Será exibido os carros e vagas cadastrados no banco, com suas respectivas informações.



Id	Tipo	Fabricante	Modelo	Ano
0087e57e-7424-456a-8155-aeddd8cd13ab	Sedan	Mitsubishi	Lancer	2012
009a1823-9ffc-4c4a-b462-ed8b5df82e32	Hatchback	FIAT	500_Abarth	2012
016a98fa-961b-42c6-badc-c52b79361bad	Sedan	Suzuki	Aerio	2007
0731aecd-7b7f-4aa1-b6c1-3f26efcbe9f8	Sedan	Infiniti	G_Coupe_IPL	2012
080a19d8-b1a8-44ef-a475-5fd36cdbd3e6	Convertible	Aston_Martin	Virage	2012
099ea691-2f93-4ee2-88f2-e8231705687f	SUV	Dodge	Journey	2012
0a86e46f-f347-4700-8792-7e5544338443	Sedan	Dodge	Charger_SRT_8	2009
0c970176-c677-4667-8875-9c5417ede29a	Coupe	Audi	TT_RS	2012
111ad1db-1a74-4ccc-b8f8-fcff3aa21fb0	SUV	Land_Rover	LR2	2012
130b7b58-a083-4c77-acc3-	Coupe	Audi	A5	2012

Id	Nome	Distàcia da Porta	Categorias	Ocupada
0285d92a-6fa3-4732-aa4e-1a4fffbf680b	A7	12	Convertible Coupe Sedan Hatchback	True
04b3208d-3e5a-4097-99e6-76191759ace0	C6	30	SUV	True
059ca3b3-9dff-43bf-8de5-0454c04ffc8e	A10	12	Convertible Coupe Sedan Hatchback	False
06fd7891-3e10-4500-8f5c-997d669db3eb	H7	30	Sedan	True
07931485-bca0-424a-91dc-be999ef1b708	B3	20	Convertible Coupe Sedan Hatchback	False
0a020997-036d-4103-9c09-1e42f6b76c78	D3	40	SUV Convertible Coupe Sedan Hatchback	True
0a8f4333-7475-4fc9-8ccd-e252b6880ad8	H10	30	Sedan	False
0c656585-76c7-4f62-a589-01726f6a06e8	E6	10	Hatchback	False
0dec2029-94a8-44d2-8f33-068584959d7e	I1	15	Sedan	False
1210bf8a-f1a3-	J9	30	Coupe Sedan Hatchback	True

REFERÊNCIAS

ACADEMY, E. D. S. O que é visão computacional? *Data Science Academy*, 2018. Disponível em: <http://datascienceacademy.com.br/blog/o-que-e-visao-computacional/>. Acesso em: 02 jun 2019.

BONNER, A. The complete beginner's guide to deep learning: Artificial neural networks. *Towards Data Science*, 2019. Disponível em: <https://towardsdatascience.com/simply-deep-learning-an-effortless-introduction-45591a1c4abb>. Acesso em: 18 ago 2019.

BROWNLEE, J. How to avoid overfitting in deep learning neural networks. 2018. Disponível em: <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>. Acesso em: 24 ago 2019.

CARVALHO, A. O que é visão computacional? *University of São Paulo*, 2009. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>. Acesso em: 17 ago 2019.

DEMURO, J. What is a neural network? *World Of Tech*, 2018. Disponível em: <https://www.techradar.com/news/what-is-a-neural-network>. Acesso em: 18 ago 2019.

FONTANA, D. R.; MARIM, L. R. Reconhecimento de caracteres através de redes neurais artificiais com aplicação a placas de automóveis. *Anais do XVI Encontro de Iniciação Científica e Pós-Graduação do ITA – XVI ENCITA*, v. 1, n. 1, 2010.

GONZALES, R. C.; WOODS, R. E. *Processamento de Imagens*. 3. ed. [S.l.]: Pearson, 2009.

GONÇALVES, A. F. Uma fotografia do problema dos estacionamentos nas metrópoles. *Revista Infra*, v. 1, n. 1, 2019.

HONDA, H.; FACURE, M.; YAOHAO, P. Os três tipos de aprendizado de máquina. *LAMFO*, 2017. Disponível em: <https://lamfo-unb.github.io/2017/07/27/tres-tipos-am/>. Acesso em: 09 sep 2019.

KARN, U. An intuitive explanation of convolutional neural networks. *the data science blog*, 2016. Disponível em: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>. Acesso em: 18 ago 2019.

LAMAS, J. Estacionamentos, os novos vilões da mobilidade urbana. *Revista EXAME*, 2014. Disponível em: <https://exame.abril.com.br/brasil/estacionamentos-os-novos-viloes-da-mobilidade-urbana/>. Acesso em: 31 mai 2019.

LIN, S.-F.; CHEN, Y.-Y.; LIU, S.-C. A vision-based parking lot management system. In: . [S.l.: s.n.], 2006. v. 4, p. 2897 – 2902. ISBN 1-4244-0100-3.

LUGER, G. F. *Inteligência Artificial*. 6. ed. [S.l.]: Pearson, 2013.

MARQUES, O.; NETO, H. V. *Processamento digital de Imagens*. 1. ed. [S.l.]: Brasport, 1999.

MARTINS, S. B. Introdução ao processamento digital de imagens, parte 1. *Universidade Estadual de Campinas (UNICAMP)*, 2014.

MEDEIROS, L. F. D. *Inteligência Artificial Aplicada: Uma Abordagem Introdutória*. 1. ed. [S.l.]: Intersaberes, 2018.

MENEZES, C. Cidade de sp tem 7,4 veículos para cada 10 habitantes, aponta levantamento da cet. *Portal G1*, v. 1, 2018. Disponível em: <https://g1.globo.com/sp/sao-paulo/noticia/cidade-de-sp-tem-74-veiculos-para-cada-10-habitantes-aponta-levantamento-da-cet.ghtml>. Acesso em: 02 jun 2019.

MITTCHEL, T. *Machine Learning*. McGraw-Hill, 1997. (McGraw-Hill International Editions). ISBN 9780071154673. Disponível em: <https://books.google.com.br/books?id=EoYBngEACAAJ>.

MONARD, M. C.; BARANAUSKAS, J. A. Aplicações de inteligência artificial: Uma visão geral. In: *Proceedings LAPTEC 2000*. São Paulo: Plêiade, 2000. ISBN 8585795. Disponível em: <http://www.icmc.usp.br/~mcmonard/public/laptecB2000.pdf>.

MONTANARI, R. Detection and classification of object in image for vehicle tracking. *USP*, v. 1, n. 1, 2015.

OZBAY, S.; ERCELEBI, E. Automatic vehicle identification by plate recognition. *International Journal of Computer and Information Engineering*, v. 1, 2017. Disponível em: <https://waset.org/publications/4283/automatic-vehicle-identification-by-plate-recognition>. Acesso em: 07 may 2019.

PARKER, J. R.; FEDERL, P. An approach to license plate recognition. *University of Calgary, Department of Computer Science*, v. 1, 1997. Disponível em: <https://www.researchgate.net/publication/256409290_An_Approach_To_License_Plate_Recognition>. Acesso em: 05 may 2019.

PATEL, N. A study of digital image filtering techniques in spatial image processing. *International Conference on Convergence of Technology*, v. 1, n. 1, 2014.

RIBEIRO, B. Suavização de imagens - image smoothing. *UFF*, n. 1, 2015. Disponível em: <http://www.icmc.usp.br/~mcmonard/public/laptecB2000.pdf>.

SAHA, S. A comprehensive guide to convolutional neural network. *Towards Data Science*, 2018. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em: 22 ago 2019.

SHARMA, S. Activation functions in neural networks. *Towards Data Science*, 2017. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. Acesso em: 18 ago 2019.

TEIXEIRA, J. d. F. *Inteligência Artificial*. 1. ed. [S.l.]: Paulus, 2009.

THOMPSON, W. *Visão Computacional - O que é e qual sua importância?* 2019. Disponível em: <https://www.sas.com/pt_br/insights/analytics/computer-vision.html>.

TOMAYKO, J. E. *Computers in Spaceflight: The NASA Experience*. 1. ed. [S.l.]: NASA, 1988.

ZHOU, V. An introduction to convolutional neural networks. *Towards Data Science*, 2019. Disponível em: <https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-bdf692352c7>. Acesso em: 05 ago 2019.